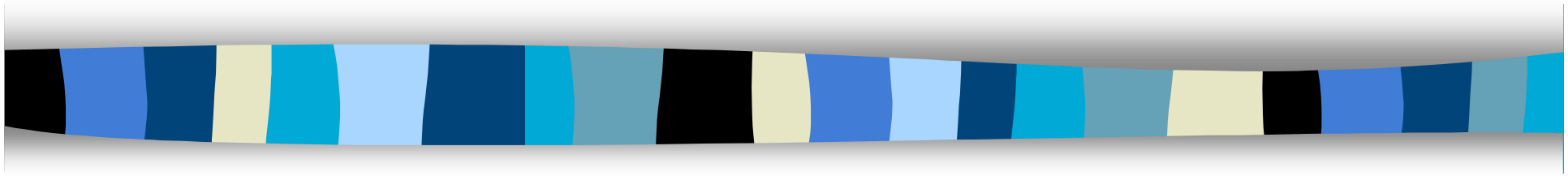
Exploratory graph analysis



Marco Saerens and Francois Fouss (UCL)

machine learning group @ UCL

UCL Université catholique de Louvain
Université partenaire de l'Académie universitaire 'Louvain'

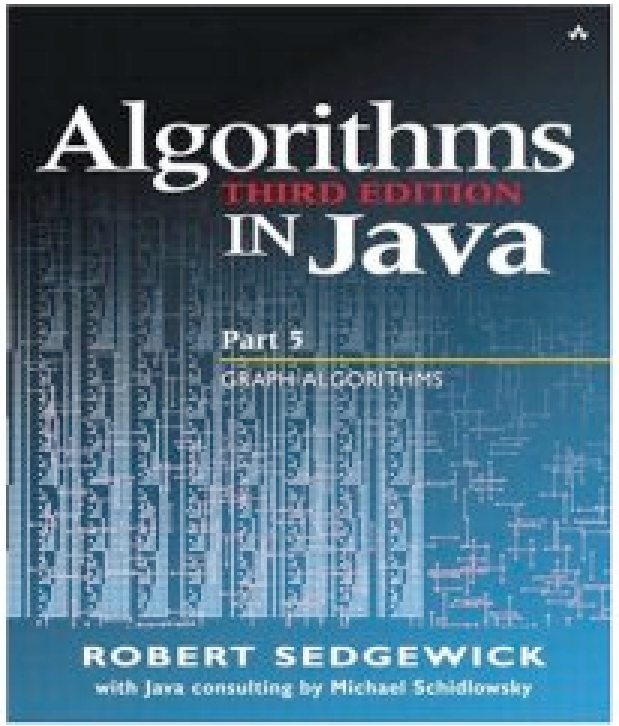
Contents

- Cutting a graph in small pieces and exploring it
- Identifying central or prestigious nodes by link analysis
- Computing similarities between nodes
- Clustering nodes of a graph
- Finding dense regions
- Graph partitioning

Cutting a graph in small pieces and exploring it

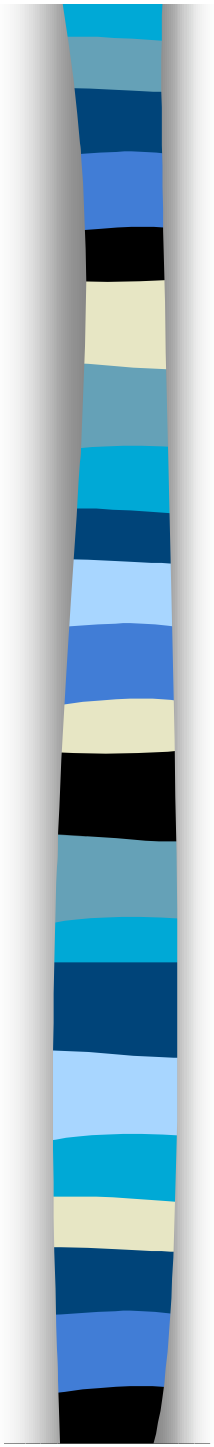
Cutting a graph in small pieces and exploring it

- A nice reference
  - Sedgewick
  - Algorithms in Java
  - Addison-Wesley



Cutting a graph in small pieces and exploring it

- Many software solutions are readily available for cuting a graph
  - See www.insna.org/INSNA/soft_inf.html
  - It provides a number of pointers to softwares

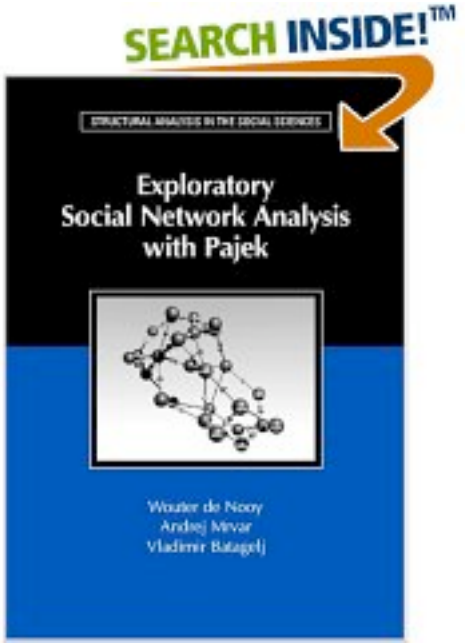


Cutting a graph in small pieces and exploring it

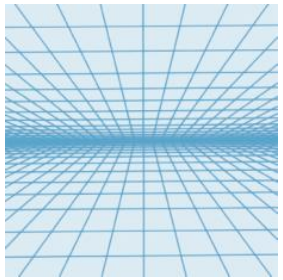
- JUNG
  - the Java Universal Network/Graph Framework
  - Open source and written in Java
  - Mainly a toolbox of methods
  - http://jung.sourceforge.net

Cutting a graph in small pieces and exploring it

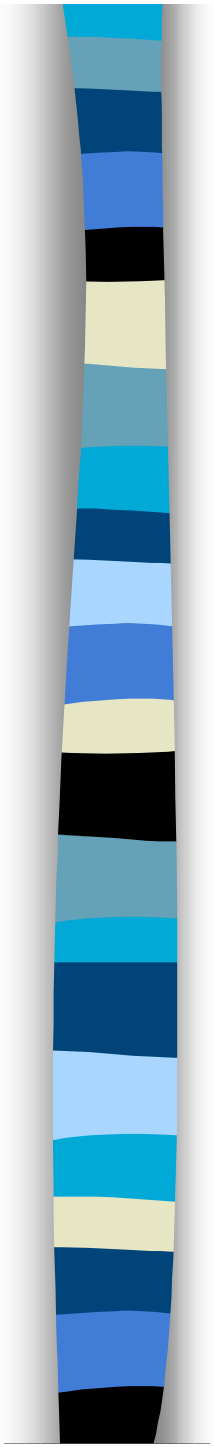
- Pajek
  - A software for large network analysis
  - There is a companion book
  - Quite powerfull
  - Free software



Cutting a graph in small pieces and exploring it



- UCINET
  - Social Network Analysis Software
  - Written by active researchers in the social network community
  - Quite complete comercial software
  - http://www.analytictech.com

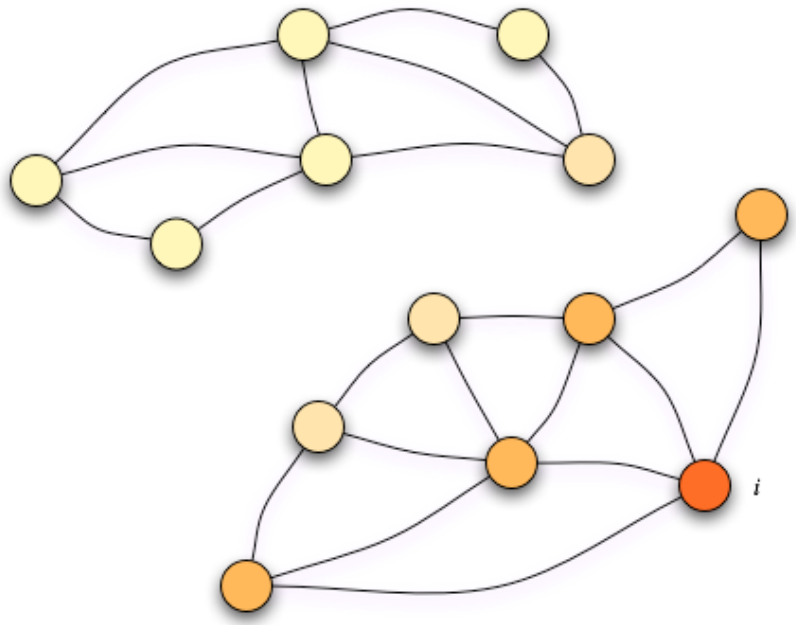


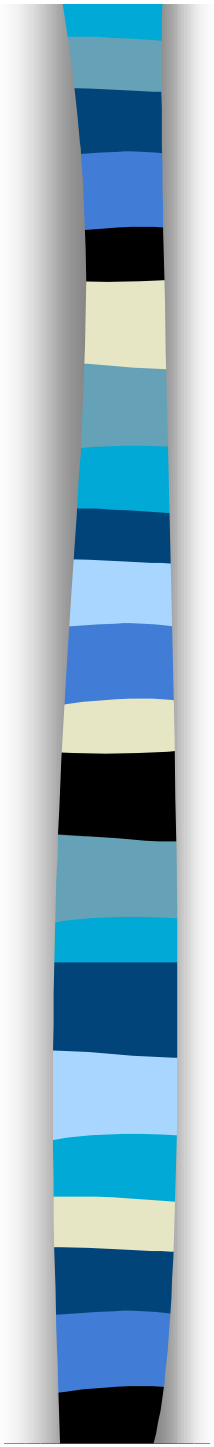
Cutting a graph in small pieces and exploring it

- The first step in analysing a graph is often to look at its connected components
  - In an undirected graph, a connected component is a maximal connected subgraph
  - A strongly connected component is the similar concept defined for directed graphs

Cutting a graph in small pieces and exploring it

- Here is a graph with two connected components



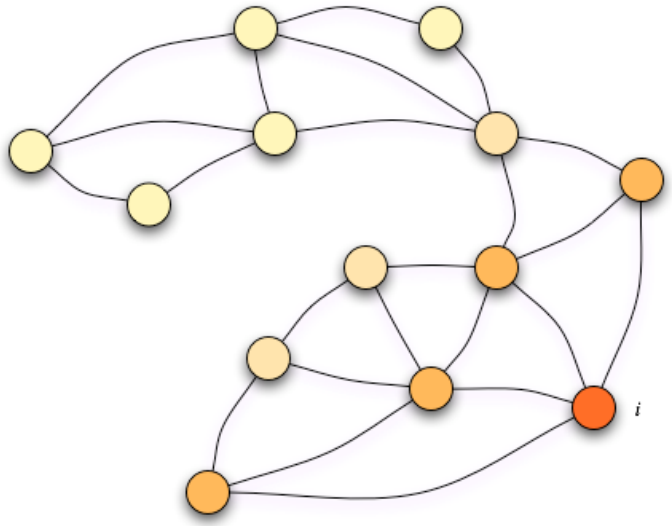


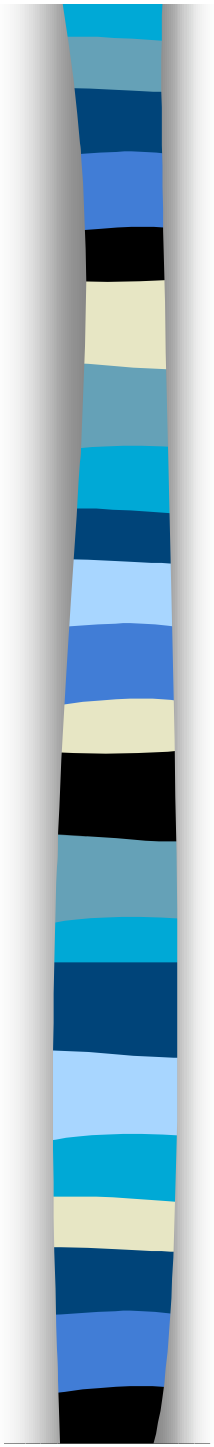
Cutting a graph in small pieces and exploring it

- There exists linear-time algorithms for solving this problem
  - Using, for instance, depth-first or breadth-first search

Cutting a graph in small pieces and exploring it

- An articulation point or vertex-cut is a node (vertex) of a graph such that
  - removal of the node causes an increase in the number of connected components



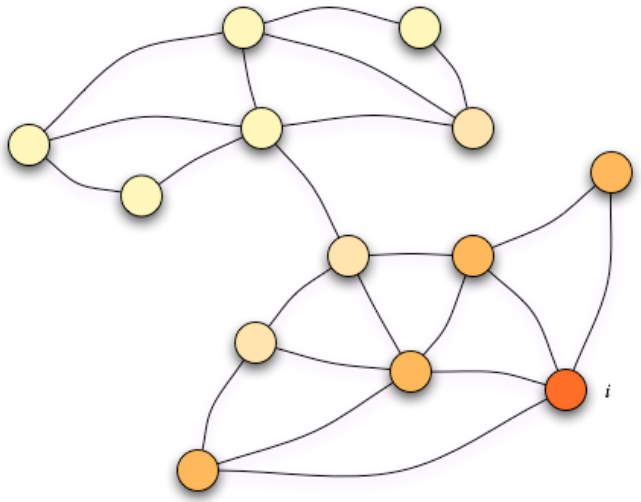


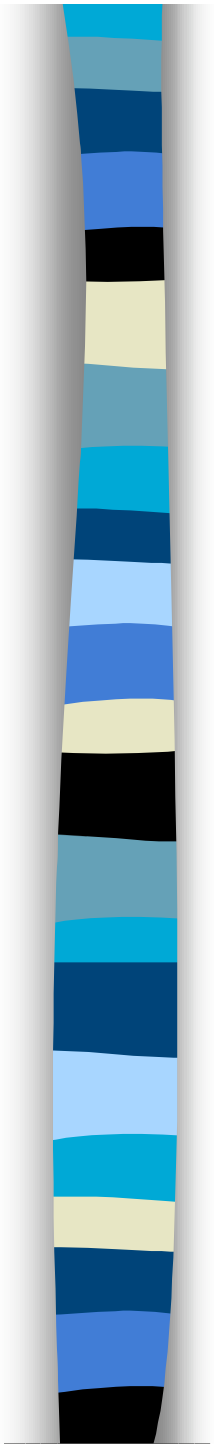
Cutting a graph in small pieces and exploring it

- Standard linear-time algorithms are available for this problem, using, for instance
  - depth-first or breadth-first search

Cutting a graph in small pieces and exploring it

- A bridge or an edge-cut is an arc (edge) of a graph such that
  - removal of the arc causes an increase in the number of connected components

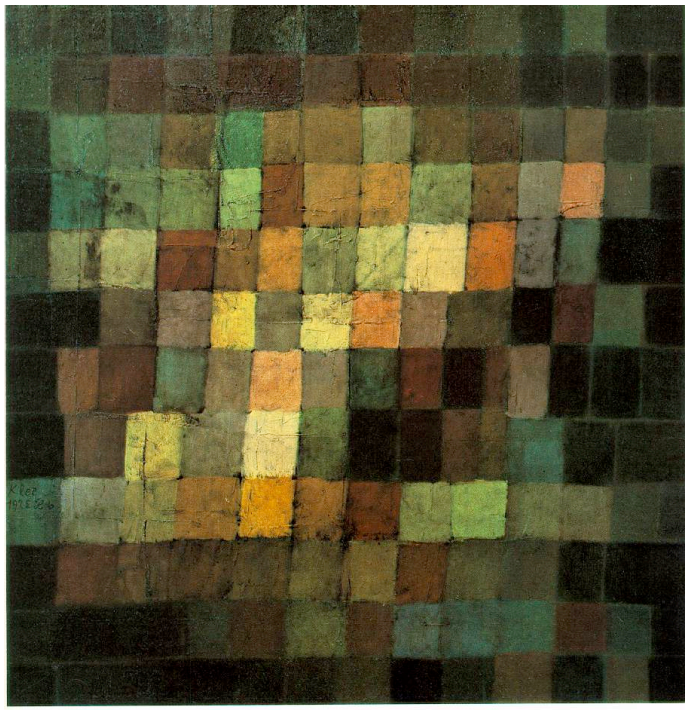




Cutting a graph in small pieces and exploring it

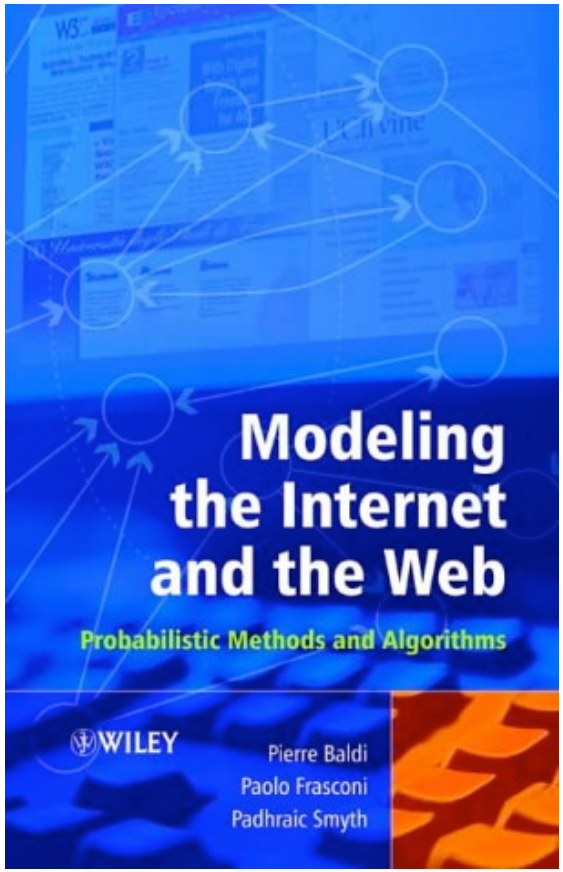
- Standard linear-time algorithms are available for this problem, using, for instance
  - depth-first or breadth-first search

Identifying central or prestigious nodes by link analysis



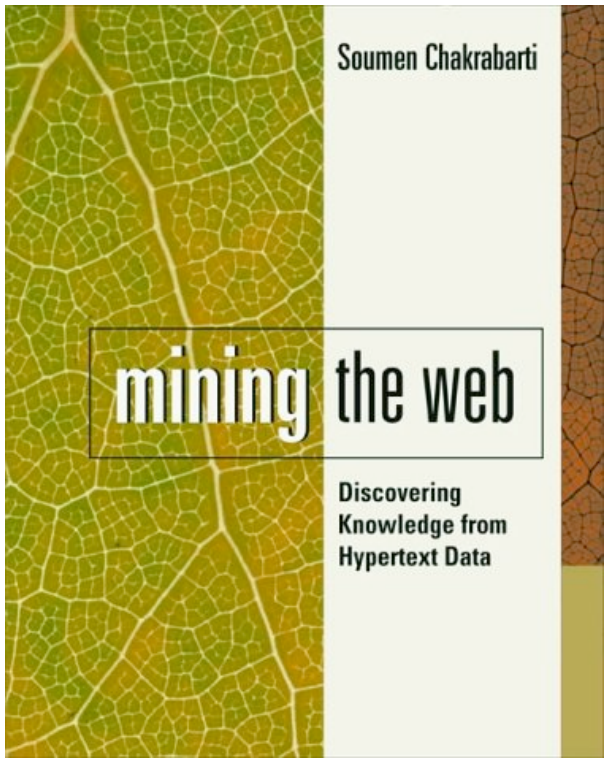
Some link analysis books

- P. Baldi, P. Frasconi & P. Smyth (2003)
  - Modeling the Internet and the Web
  - John Wiley & Sons



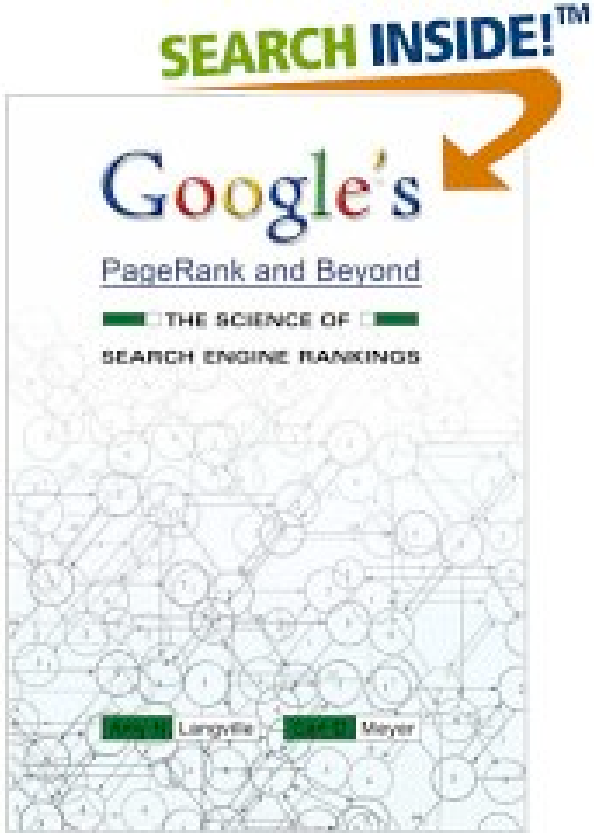
Some link analysis books

- S. Chakrabarti (2003)
  - Mining the Web
  - Morgan Kaufmann



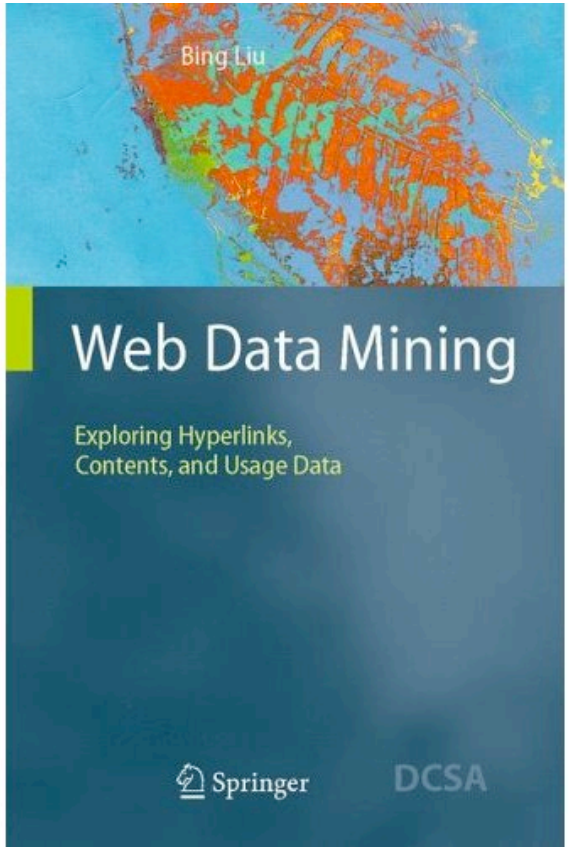
Some link analysis books

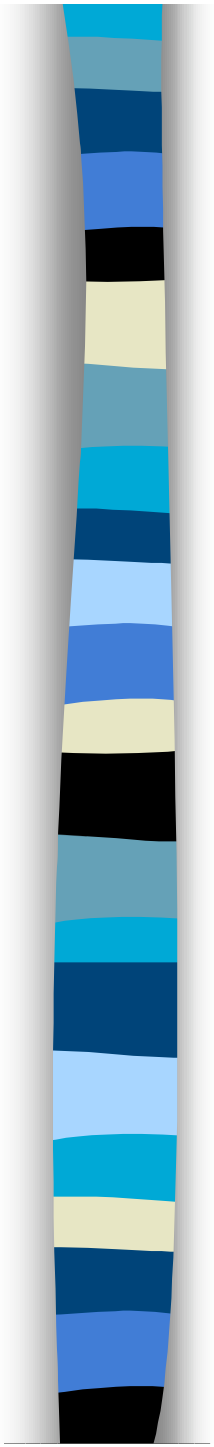
- A. Langville & C. Meyer (2006)
  - Google's PageRank and beyond
  - Princeton university



Some link analysis books

- B. Liu (2006)
  - Web data mining
  - Springer

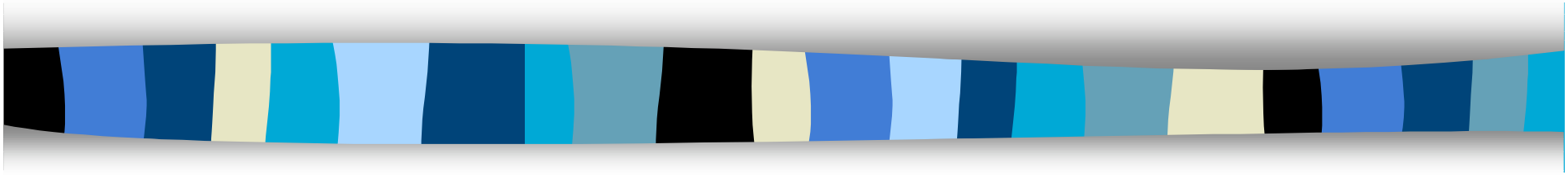




Identifying central or prestigious nodes by link analysis

- The PageRank algorithm
- The HITS algorithm
- The SALSA algorithm

The PageRank algorithm



The basic PageRank algorithm

- Introduced by Page, Brin, Motwani & Winograd in 1998
- Partly implemented in Google
- Corresponds to a measure of « prestige » in a directed graph

Web link analysis

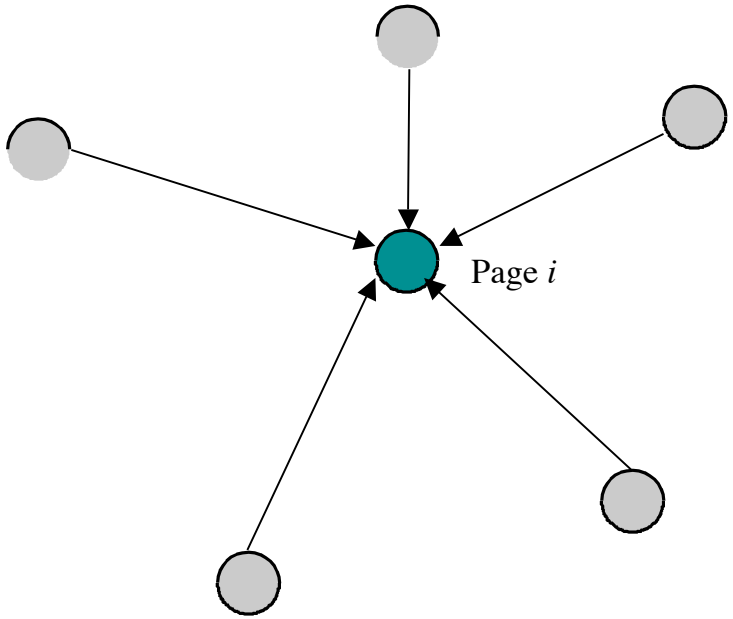
- A set of techniques
  - Applied to: Hyperlink document repositories
  - Typically web pages
- Objective:
  - To exploit the link structure of the documents
  - In order to extract interesting information
  - Viewing the document repository as a graph where
    - Nodes are documents
    - Edges are directed links between documents
  - It does not exploit the content of the pages !!

Web link analysis

- Suppose we performed a search with a search engine
- **Objective**: to improve the (content-based) ranking of the search engine
  - Based on the graph structure of the web hyperlinks
  - PageRank is computed off-line

The basic PageRank algorithm

- To each web page we associate a score, x_i
  - The score of page i, x_i, is proportional to the weighted averaged score of the pages pointing to page i



The basic PageRank algorithm

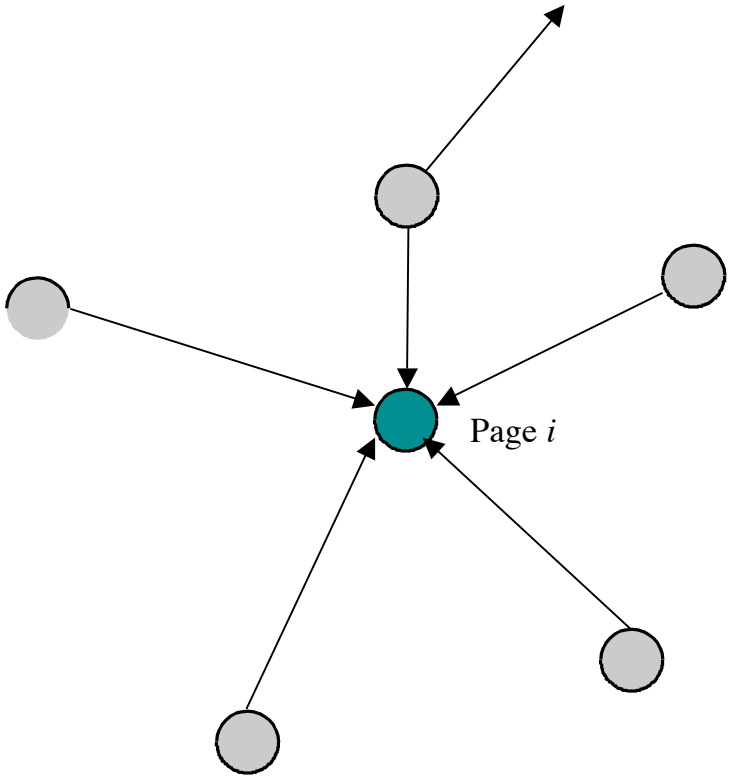
- Let w_{ij} be the weight of the link connecting page i to page j
  - Usually, it is simply 0 or 1
  - Thus, $w_{ij} = 1$ if page i has a link to page j; $w_{ij} = 0$ otherwise
- Let $\mathbf{W}$ be the matrix made of the elements w_{ij}
  - Notice that this matrix is not symmetric
  - We suppose that the graph is strongly connected

The basic PageRank algorithm

- In other words

$$x_i \propto \sum_{j=1}^{n} \frac{w_{ji}\, x_j}{w_{j.}}$$

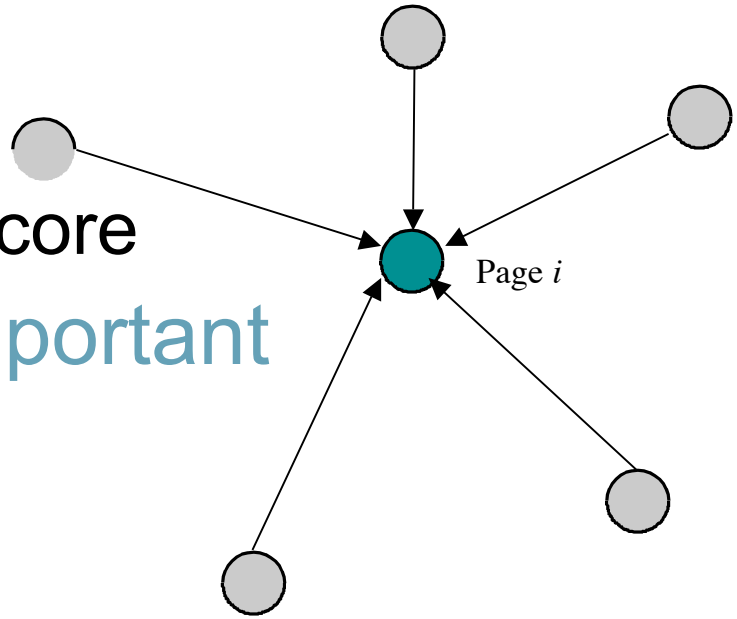
$$w_{j.} = \sum_{i=1}^{n} w_{ji}$$



– Where $w_{j.}$ is the outdegree of page j

The basic PageRank algorithm

- In other words,
- A page with a high score is a page that is pointed by
  - Many pages
  - Having each a high score
- Thus a page is an important page if
  - It is pointed by many, important, pages



The basic PageRank algorithm

- These equations can be updated iteratively until convergence
- In order to obtain the scores, x_i
  - We normalize the vector $\mathbf{x}$ at each iteration
- The pages are then ranked according to their score

The basic PageRank algorithm

- This definition has a nice interpretation in terms of random surfing
- If we define the probability of following the link from page j to page i as

$$\mathrm{P}(page(k+1) = i | page(k) = j) = \frac{w_{ji}}{w_{j.}}$$

$$w_{j.} = \sum_{i=1}^{n} w_{ji}$$

The basic PageRank algorithm

- We can write the updating equation as

$$x_i(k+1) = \mathrm{P}(page(k+1) = i)$$
$$= \sum_{j=1}^{n} \mathrm{P}(page(k+1) = i | page(k) = j)\, x_j(k)$$

- And thus we can define a random surfer following the links according to the transition probabilities

$$\mathrm{P}(page(k+1) = i | page(k) = j) = \frac{w_{ji}}{w_{j.}}$$

The basic PageRank algorithm

- This is the equation of a Markov model of random surf through the web
- This is exactly the same equation as before:

$$x_i \propto \sum_{j=1}^{n} \frac{w_{ji}\, x_j}{w_{j.}}$$

$$w_{j.} = \sum_{i=1}^{n} w_{ji}$$

The basic PageRank algorithm

- If we denote element i, j of the transition probability matrix $\mathbf{P}$ as p_{ij}
- We thus have

$$[\mathbf{P}]_{ij} = p_{ij} = \mathrm{P}(page(k+1) = j | page(k) = i)$$

- And the equation can be rewritten as

$$x_i(k+1) = \sum_{j=1}^{n} p_{ji}\, x_j(k)$$

The basic PageRank algorithm

- In matrix form, if the vector $\mathbf{x}$ has elements x_i

$$\mathbf{x}(k+1) = \mathbf{P}^{\mathrm{T}}\mathbf{x}(k)$$

- The stationary distribution is given by $\mathbf{x}(k{+}1) = \mathbf{x}(k)$, and thus

$$\mathbf{x} = \mathbf{P}^{\mathrm{T}}\mathbf{x}$$

The basic PageRank algorithm

- x_i can then be viewed as the probability of being at page i
  - The solution to these equations is the stationary distribution of the random surf
  - Which is the probability of finding the surfer on page i on the long-term behaviour
- The « most probable page » is the best ranked

The basic PageRank algorithm

- The PageRank scores can be obtained
  - By computing the left eigenvector of the matrix $\mathbf{P}$ corresponding to eigenvalue 1
  - Which is the right eigenvector of $\mathbf{P}^T$
  - Where $\mathbf{P}$ is the transition probabilities matrix of the Markov process
  - Containing the transition probabilities
- If the graph is undirected, the scores are simply the indegrees of the nodes

Adjustments to the basic model

- However, there is a problem with
  - Dangling nodes
  - That is, nodes without any outgoing link
- In this case, the $\mathbf{P}$ matrix is no more stochastic
  - Rows do not sum to one
- Moreover, the graph could have separate components

Adjustments to the basic model

- One potential solution is to allow to jump to any node of the graph
  - With some non-zero probability (teleportation)
- Thus,

$$\mathbf{G} = \alpha\mathbf{P} + (1-\alpha)\frac{\mathbf{e}\mathbf{e}^{\mathrm{T}}}{n}$$

where $\mathbf{G}$ is called the Google matrix, $\mathbf{e}$ is a column vector full of 1's and $0 < \alpha < 1$

Adjustments to the basic model

- In this case,
  - The matrix is stochastic
  - The matrix is irreducible (no separate component)
  - The matrix is aperiodic
- Then, there is a unique eigenvector associated to eigenvalue 1
- However, $\mathbf{G}$ is no more sparse !

Computing PageRank

- The problem is thus to find the left eigenvector of $\mathbf{G}$
  - corresponding to the eigenvalue 1
  - instead of $\mathbf{P}$

$$\mathbf{x}^{\mathrm{T}}\mathbf{G} = \mathbf{x}^{\mathrm{T}}$$

  - with the normalization $\|\mathbf{x}\|_1 = 1$ or $\mathbf{x}^{\mathrm{T}}\mathbf{e} = 1$
- One can use the standard power method

Computing PageRank

- Fortunately, the power method results in sparse matrix multiplication only:

$$\begin{aligned}
\mathbf{x}^{\mathrm{T}}(k+1) &= \mathbf{x}^{\mathrm{T}}(k)\mathbf{G} \\
&= \alpha\,\mathbf{x}^{\mathrm{T}}(k)\mathbf{P} + \frac{(1-\alpha)}{n}\mathbf{x}^{\mathrm{T}}(k)\mathbf{e}\mathbf{e}^{\mathrm{T}} \\
&= \alpha\,\mathbf{x}^{\mathrm{T}}(k)\mathbf{P} + \frac{(1-\alpha)}{n}\mathbf{e}^{\mathrm{T}}
\end{aligned}$$

  – where $\mathbf{x}$ is normalized after each iteration

Personalization in PageRank

- How can we favour some pages in a natural way (advertising, etc) ?

- Rather than using $\mathbf{e}\mathbf{e}^{\mathrm{T}}/n$,
- use $\mathbf{e}\mathbf{v}^{\mathrm{T}}$ where
  - $\mathbf{v} > 0$ is a probability vector $(\mathbf{v}^{\mathrm{T}}\mathbf{e} = 1)$
  - Which is called the personalization vector
- It contains the a priori probability of jumping to any page

Personalization in PageRank

- The Google matrix thus becomes

$$\mathbf{G} = \alpha\mathbf{P} + (1 - \alpha)\mathbf{e}\mathbf{v}^{\mathrm{T}}$$

  – Where $\mathbf{v}$ is provided a priori

The PageRank problem as a sparse linear system

- Here is an alternative formulation of the PageRank problem
  - As a sparse linear system

$$\mathbf{x}^{\mathrm{T}}\mathbf{G} = \mathbf{x}^{\mathrm{T}}$$

$$\Rightarrow \mathbf{x}^{\mathrm{T}}(\alpha\mathbf{P} + (1-\alpha)\mathbf{e}\mathbf{v}^{\mathrm{T}}) = \mathbf{x}^{\mathrm{T}}$$

$$\Rightarrow \alpha\mathbf{x}^{\mathrm{T}}\mathbf{P} + (1-\alpha)\mathbf{v}^{\mathrm{T}} = \mathbf{x}^{\mathrm{T}}$$

$$\Rightarrow \mathbf{x}^{\mathrm{T}}(\mathbf{I} - \alpha\mathbf{P}) = (1-\alpha)\mathbf{v}^{\mathrm{T}}$$

$$\Rightarrow (\mathbf{I} - \alpha\mathbf{P})^{\mathrm{T}}\mathbf{x} = (1-\alpha)\mathbf{v}$$

The PageRank problem as a sparse linear system

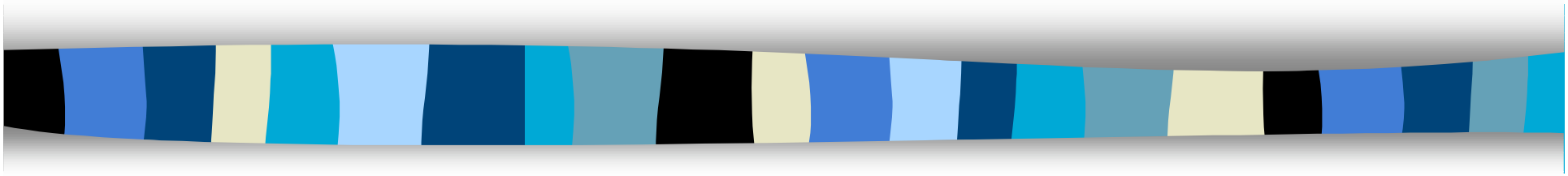
- In other words, the problem is

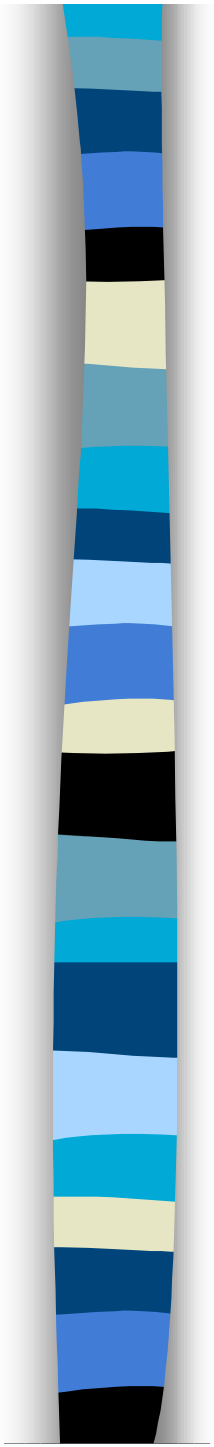
$$\text{Solve } (\mathbf{I} - \alpha\mathbf{P})^{\mathrm{T}}\mathbf{x} = (1-\alpha)\mathbf{v} \text{ with } \mathbf{x}^{\mathrm{T}}\mathbf{e} = 1$$

- Which has been shown (Del Corso, Gulli & Romani, 2005; Langville & Meyer, 2006) to be equivalent to

$$\text{Solve } (\mathbf{I} - \alpha\mathbf{P})^{\mathrm{T}}\mathbf{x}' = \mathbf{v} \text{ and compute } \mathbf{x} = \mathbf{x}' / \left\|\mathbf{x}'\right\|_1$$

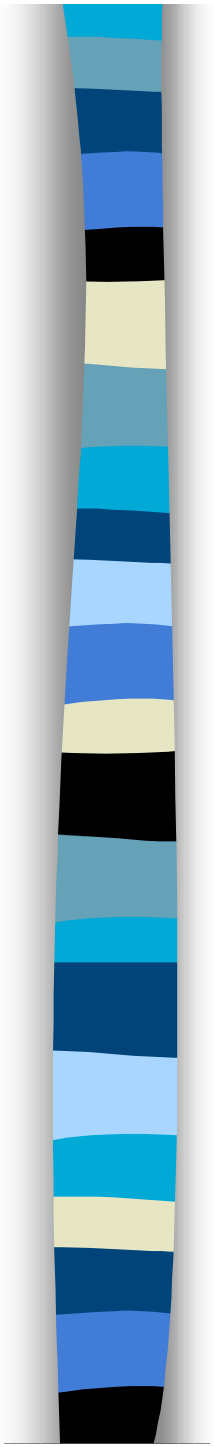
The HITS algorithm





The HITS algorithm

- Introduced by Kleinberg in 1998/1999



Web link analysis

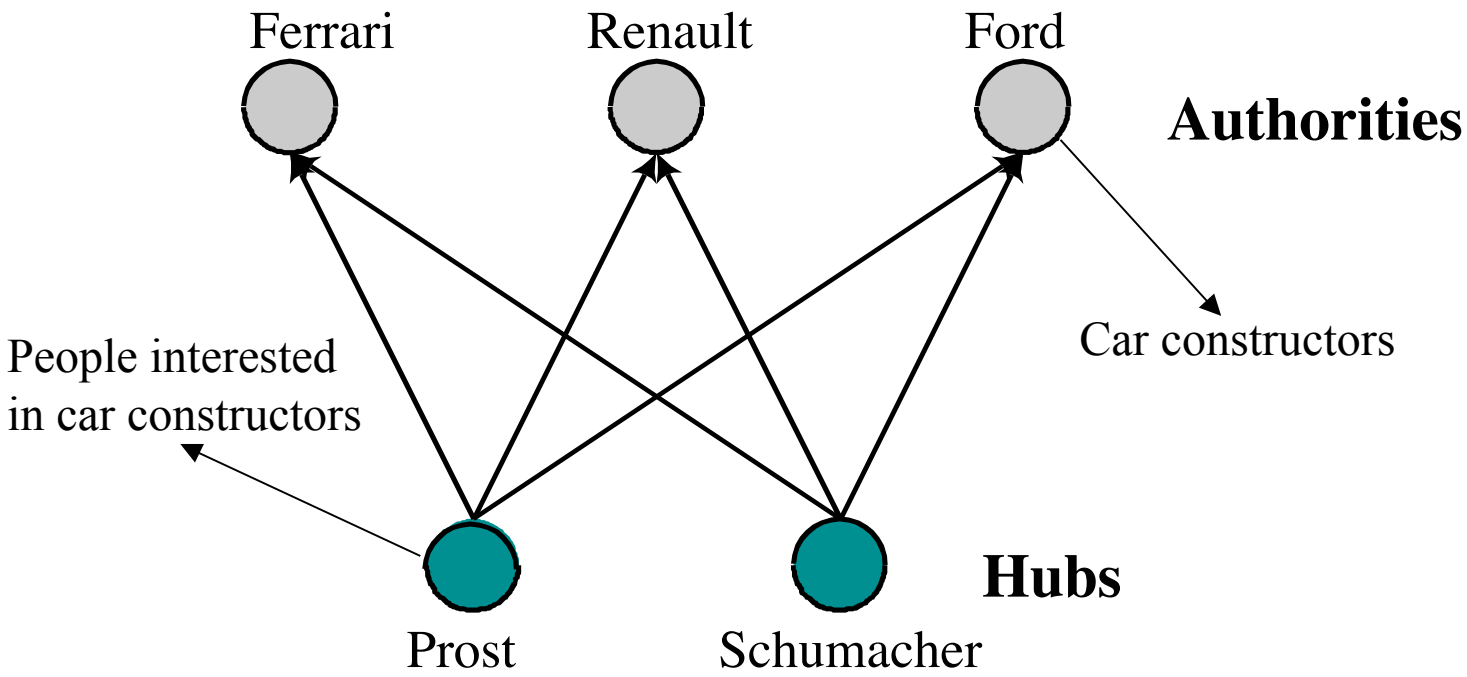
- Suppose we performed a search with a search engine
  - Compute the neighborhood graph from the retrieved documents
  - Associated to the particular query
- **Objective**: to improve the ranking provided by the search engine
  - Based on the graph structure of the web hyperlinks

The HITS algorithm

- The model proposed by Kleinberg is based on two concepts
  - **Hub** pages
  - **Authorities** pages
- These are two categories of web pages
- These two concepts are strongly connected

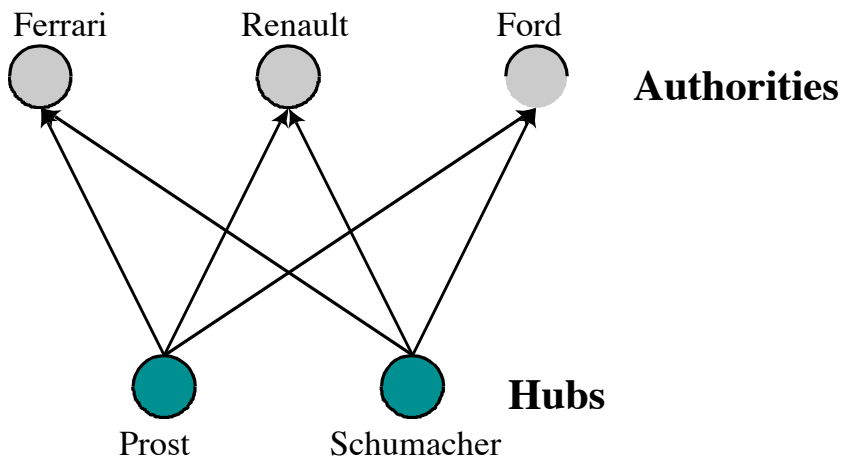
The HITS algorithm

- Example:
  - Suppose we introduced the query "Car constructors"



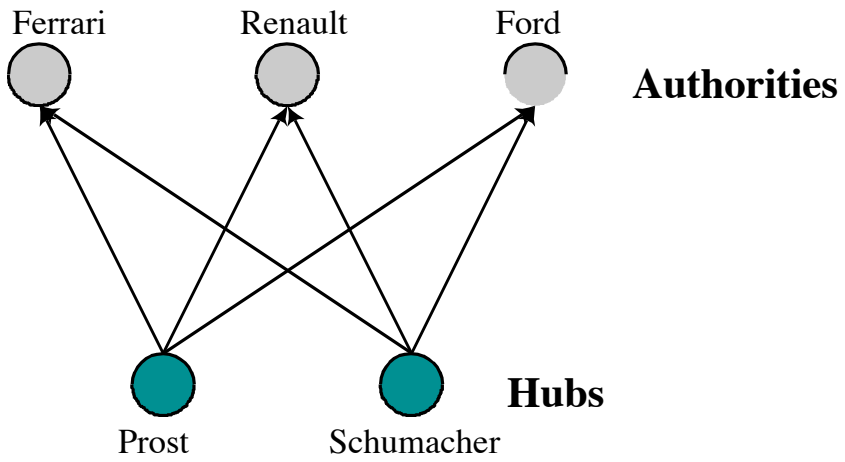
The HITS algorithm

- Hubs
  - Link heavily to authorities
  - A good hub points to many good authorities
  - Hubs have very few incoming links



The HITS algorithm

- Authorities
  - Do not link to other authorities
  - A good authority is pointed by many good hubs
  - The main authorities on a topic are often in competition with one another



The HITS algorithm

- The objective is to detect **good hubs** and **good authorities**
  - from the results of the search engine
- We therefore assign two numbers to each returned page i:
  - A **hub score,** x^h_i
  - An **authority score,** x^a_i

The HITS algorithm

- Let w_{ij} be the weight of the link connecting page i to page j
  - Usually, it is simply 0 or 1
  - Thus, $w_{ij} = 1$ if page i has a link to page j; $w_{ij} = 0$ otherwise
- Let $\mathbf{W}$ be the matrix made of elements w_{ij}
  - Notice that this matrix is not symmetric
  - We suppose that the graph is strongly connected

The HITS algorithm

- A possible procedure for computing hub/authorities scores (Kleinberg)
  - A page's **authority score** is proportional to the sum of the hub scores that link to it

$$x_j^a = \eta \sum_{i=1}^{n} w_{ij} x_i^h$$

  - A page's **hub score** is itself proportional to the sum of the authority scores that it links to

$$x_i^h = \mu \sum_{j=1}^{n} w_{ij} x_j^a$$

The HITS algorithm

- In matrix form,

$$\begin{cases} \mathbf{x}^a = \eta \mathbf{W}^{\mathrm{T}} \mathbf{x}^h \\ \mathbf{x}^h = \mu \mathbf{W} \mathbf{x}^a \end{cases}$$

- And thus,

$$\begin{cases} \mathbf{x}^a = \eta\mu \mathbf{W}^{\mathrm{T}} \mathbf{W} \mathbf{x}^a \\ \mathbf{x}^h = \eta\mu \mathbf{W} \mathbf{W}^{\mathrm{T}} \mathbf{x}^h \end{cases}$$

The HITS algorithm

- Kleinberg used this iterative procedure in order to estimate the scores
  - with a normalization at each step
  - This is equivalent to computing the eigenvectors of the following matrices

$$\mathbf{W}\mathbf{W}^{\mathbf{T}} \qquad\qquad \mathbf{W}^{\mathbf{T}}\mathbf{W}$$

  - To obtain respectively the vector of hubs scores and the vector of authorities scores

Links with principal components analysis

- This is exactly uncentered principal components analysis (PCA; Saerens et all, 2005)
  - The proof is based on the dual view of PCA
- As for multidimensional scaling
  - View the set of rows of $\mathbf{W}$ (authorities) as a cloud of points in the columns space
  - View the set of columns of $\mathbf{W}$ (hubs) as a cloud of points in the rows space

Links with principal components analysis

- Let us consider a data matrix $\mathbf{X}$
- The first PCA axis on which the data will be projected is given by the eigensystem

$$\mathbf{X}^{\mathrm{T}}\mathbf{X}\mathbf{u}_1 = \lambda_1\mathbf{u}_1$$

- Thus, the first projection axis correspond to the dominant eigenvector of

$$\mathbf{X}^{\mathrm{T}}\mathbf{X}$$

Links with principal components analysis

- Then, the first coordinate of the projected data is given by $\mathbf{X}\mathbf{u}_1$
  - Which corresponds to the data vectors projected on the first principal axis, $\mathbf{u}_1$
- These are the PCA scores for the first principal axis

Links with principal components analysis

- Here is a sketch of the proof that HITS is equivalent to uncentered PCA
- Let us consider the adjacency matrix $\mathbf{W}$ as a data matrix

$$\mathbf{X} = \mathbf{W}$$

- We simply substitute $\mathbf{X}$ by $\mathbf{W}$ for computing the first principal axis (PCA), $\mathbf{u}_1$

$$\mathbf{W}^T\mathbf{W}\mathbf{u}_1 = \lambda_1\mathbf{u}_1$$

Links with principal components analysis

- We pre-multiply this equation by $\mathbf{W}$

$$\mathbf{W}\mathbf{W}^{\mathrm{T}}(\mathbf{W}\mathbf{u}_1) = \lambda_1(\mathbf{W}\mathbf{u}_1)$$

- $\mathbf{W}\mathbf{u}_1$ is an eigenvector of $\mathbf{W}\mathbf{W}^{\mathrm{T}}$ and thus contains the hubs scores

- Since $\mathbf{W}\mathbf{u}_1$ is the projection of the data on the first principal axis (= PCA scores)

- The hubs scores are equal to the uncentered PCA scores, up to a proportionality factor, computed from the data matrix $\mathbf{W}$

Links with principal components analysis

- The same result holds for the authorities scores
- We now consider the transposed adjacency matrix $\mathbf{W}^{\mathrm{T}}$ as a data matrix

$$\mathbf{X} = \mathbf{W}^{\mathrm{T}}$$

- And proceed as before
- The authorities scores are equal to the uncentered PCA scores, up to a proportionality factor, computed from the data matrix $\mathbf{W}^{\mathrm{T}}$

Links with principal components analysis

- Thus, the situation is exactly the same as for multidimensional scaling
  - The first eigenvector of $\mathbf{W}\mathbf{W}^{\mathrm{T}}$ represents the projection of the row vectors on the first principal component (hubs scores)
  - The first eigenvector of $\mathbf{W}^{\mathrm{T}}\mathbf{W}$ represents the projection of the column vectors on the first principal component (authority scores)
- This procedure is also related to both
  - Correspondence analysis
  - A random walk (Markov) model through the graph

HITS' relationships to bibliometrics

- The HITS algorithm has also strong connections to bibliometrics research:
  - Cocitation
  - Coreference
- Cocitation occurs when two documents are both cited by the same third document
- Coreference occurs when two documents both refer to the same third document

HITS' relationships to bibliometrics

- C. Ding (2002) showed that

$$\mathbf{W}^{\mathrm{T}}\mathbf{W} = \mathbf{D}_{in} + \mathbf{C}_{cit}$$

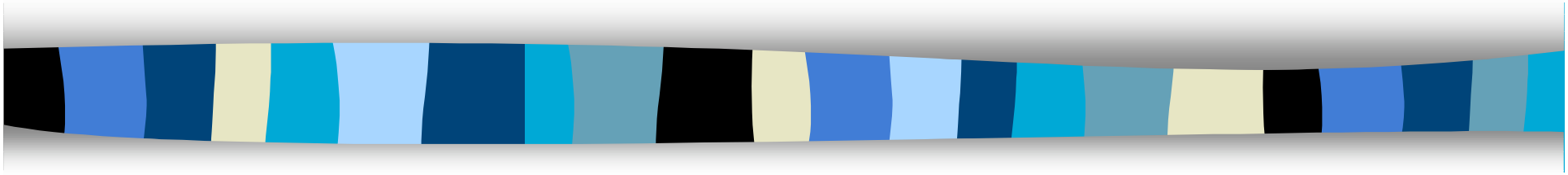
$$\mathbf{W}\mathbf{W}^{\mathrm{T}} = \mathbf{D}_{out} + \mathbf{C}_{ref}$$

  - where $\mathbf{D}_{in}$ is a diagonal matrix containing the indegree of each node $\mathbf{D}_{in} = \mathbf{Diag}(w_{\bullet j})$
  - $\mathbf{D}_{out}$ is a diagonal matrix containing the outdegree of each node $\mathbf{D}_{out} = \mathbf{Diag}(w_{i\bullet})$
  - $\mathbf{C}_{cit}$ and $\mathbf{C}_{ref}$ are the cocitation and coreference matrices

HITS' relationships to bibliometrics

- Thus,
  - The hub matrix is closely related to the coreference matrix
  - The authority matrix is closely related to the cocitation matrix

The SALSA algorithm

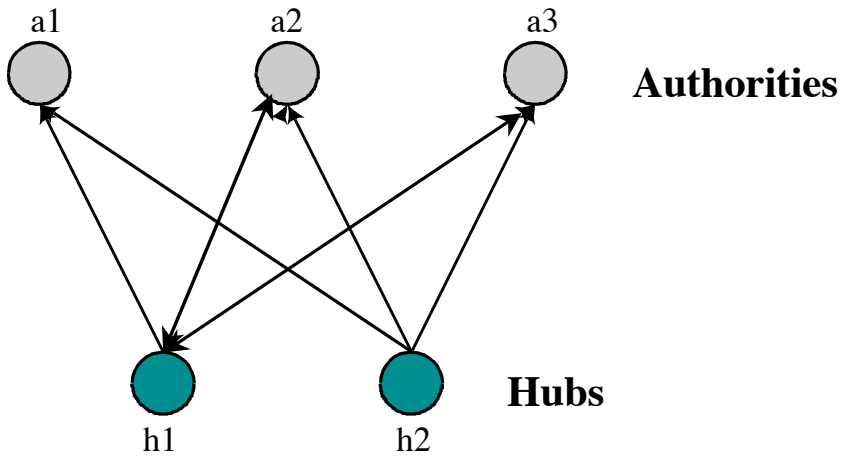


The SALSA algorithm

- Introduced by Lempel & Moran in 2000
  - « A Stochastic Approach to Link Structure Analysis »
  - Combines ideas from PageRank and HITS

The SALSA algorithm

- From the neighborhood graph, compute two sets of nodes
  - The hub nodes
  - The authority nodes
  - View this as a bipartite graph



The SALSA algorithm

- From this bipartite graph, compute a Markov chain with
  - $\mathbf{P}^h = (\mathbf{D}_{in})^{-1}\mathbf{W}^{\mathrm{T}}$: the probability of jumping from an authority node to a hub node
  - $\mathbf{P}^a = (\mathbf{D}_{out})^{-1}\mathbf{W}$: the probability of jumping from a hub node to an authority node
- Thus:

  $$\mathbf{x}^h(k+1) = (\mathbf{P}^h)^{\mathrm{T}}\,\mathbf{x}^a(k)$$

  $$\mathbf{x}^a(k+1) = (\mathbf{P}^a)^{\mathrm{T}}\,\mathbf{x}^h(k)$$

  where $\mathbf{x}^h$, $\mathbf{x}^a$ are the probability distributions for hub and authority nodes

The SALSA algorithm

- The transition probabilities matrix of the Markov chain
  - Restricted to the authority nodes is

    $\mathbf{P}^h\mathbf{P}^a$

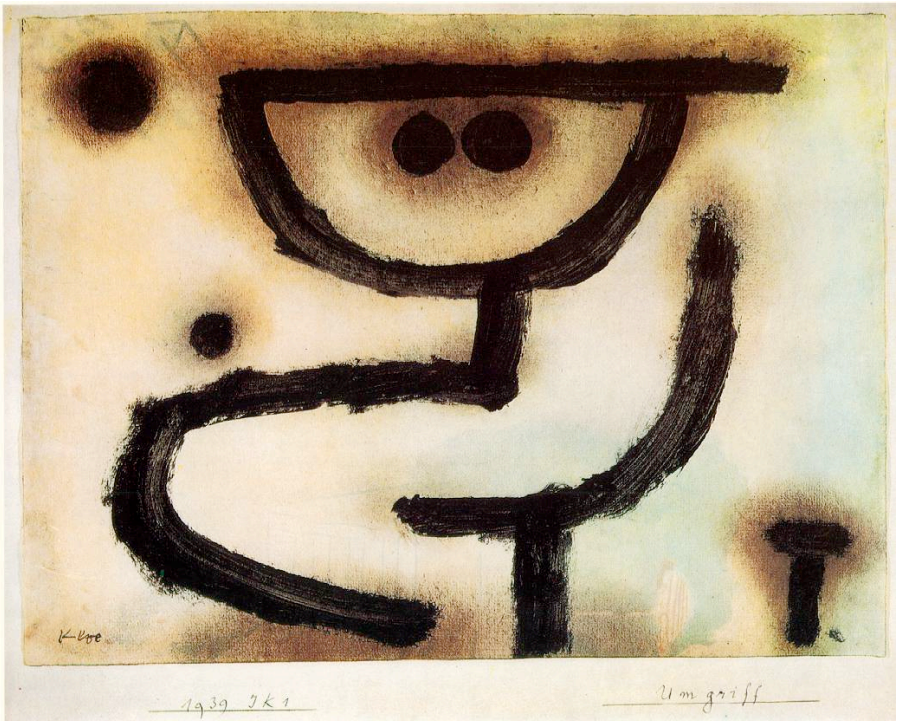
  - Restricted to the hub nodes is

    $\mathbf{P}^a\mathbf{P}^h$

The SALSA algorithm

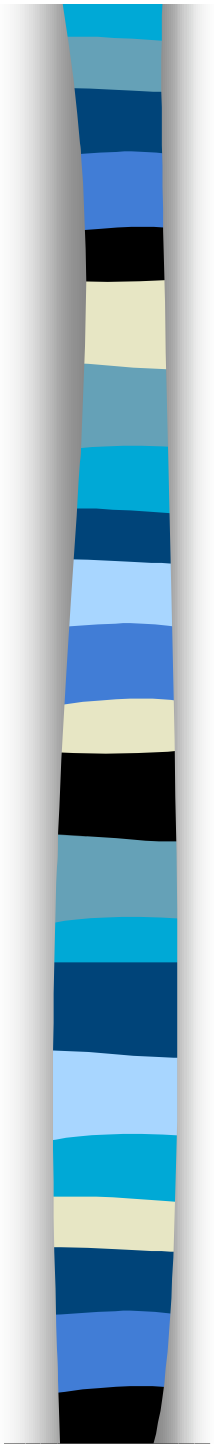
- The steady-state probability distribution of the two restricted Markov chains are the hub scores and authority scores
  - When removing dangling nodes
  - When computing the steady-state for each connected component

Computing similarities between nodes of a graph



Main goal

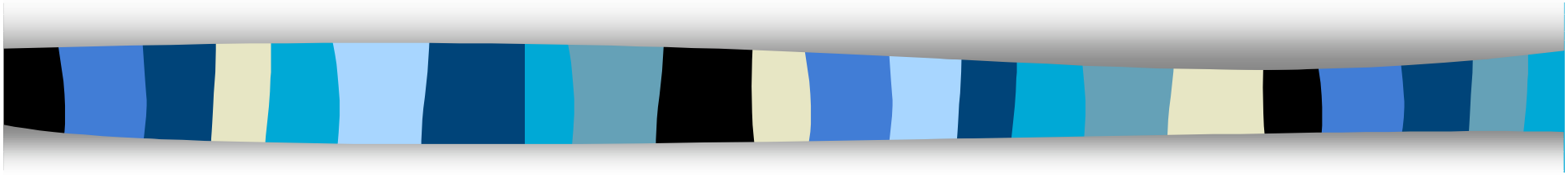
- To exploit and analyse
  - New similarity measures between the nodes of a graph
  - Which are kernels on a graph
- To use these similarities for
  - Collaborative filtering
  - Clustering
  - Finding dense regions
  - Graph visualization
  - Etc…



Three main parts

- A brief overview of kernels
  - in the « machine learning » field
- Some kernels on a graph
  - The exponential diffusion kernel
  - The Laplacian exponential diffusion kernel
  - The von Neumann diffusion kernel
  - The regularized Laplacian kernel
  - The commute-time kernel
  - The random walk with restart similarity
- Computing similarities between nodes of two graphs

A brief overview of kernels

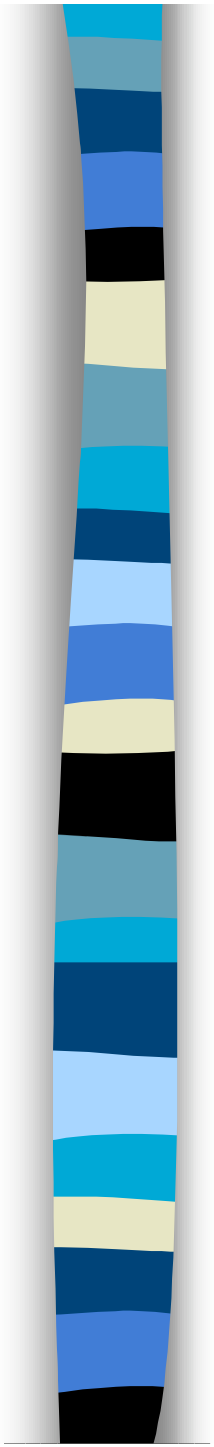


A brief overview of kernels

- In a few words, a kernel is simply
  - An inner product matrix
- That is, a matrix containing inner products as entries,

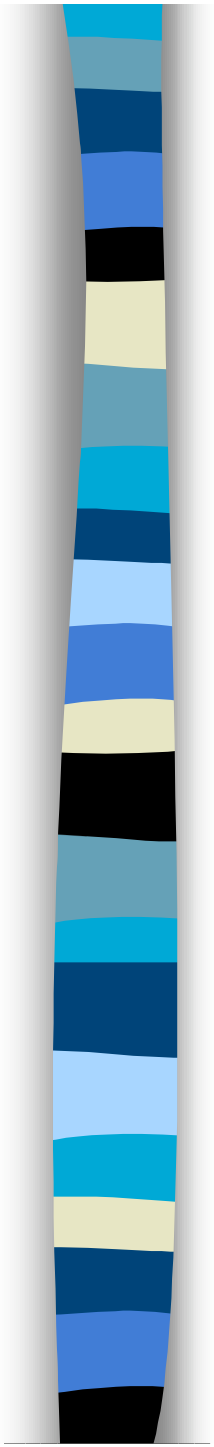
$$[\mathbf{K}]_{ij} = k_{ij} = \mathbf{x}_i^{\mathrm{T}} \mathbf{x}_j$$

  defined in some abstract inner product space, called
- The feature space



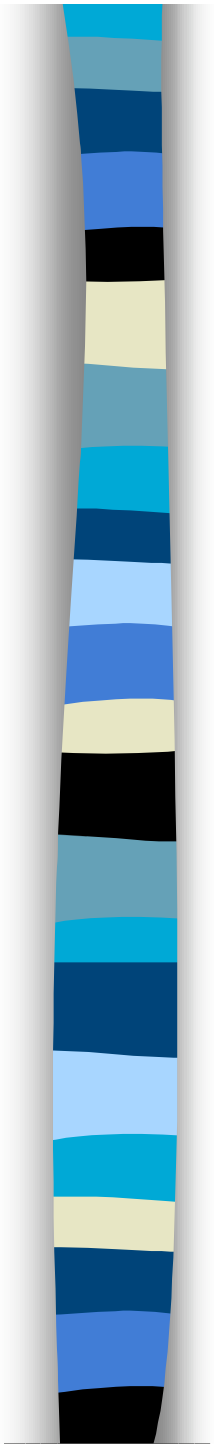
A brief overview of kernels

- The symmetric matrix $\mathbf{K}$ is called the kernel matrix
  - It contains inner products between elements, or feature vectors, $\mathbf{x}_i$ and $\mathbf{x}_j$,
  - in some feature space
- The kernel matrix is thus
  - a Gram matrix
  - Positive semi-definite
- Its entries, k_{ij}, are interpreted as similarities between elements



A brief overview of kernels

- If k_{ij} is large (positive), i and j are highly similar
- If k_{ij} is low (negative), i and j are highly dissimilar
- Most of the pattern recognition / multivariate statistics techniques can be reformulated
  - in terms of inner products, $\mathbf{K}$
  - instead of feature vectors, $\mathbf{x}_i$



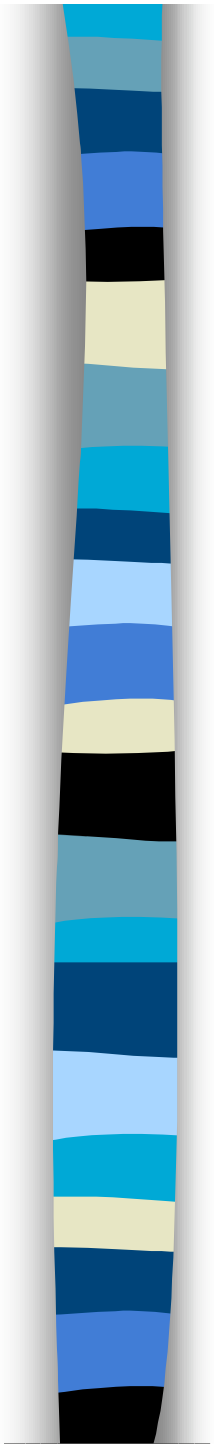
A brief overview of kernels

- This is called the « kernel trick »
- For instance,
  - Principal components analysis
    => « kernel PCA »
  - Clustering
    => « kernel clustering »
  - Logistic regression
    => « kernel logistic regression »
  - Etc…

A brief overview of kernels

- In other words,
  - We do not need the feature vectors, $\mathbf{x}_i$
  - We only need a similarity measure between the elements, k_{ij}
- Each kernel induces a Euclidean distance between the elements

$$d^2(i,j) = k_{ii} + k_{jj} - 2k_{ij}$$



A brief overview of kernels

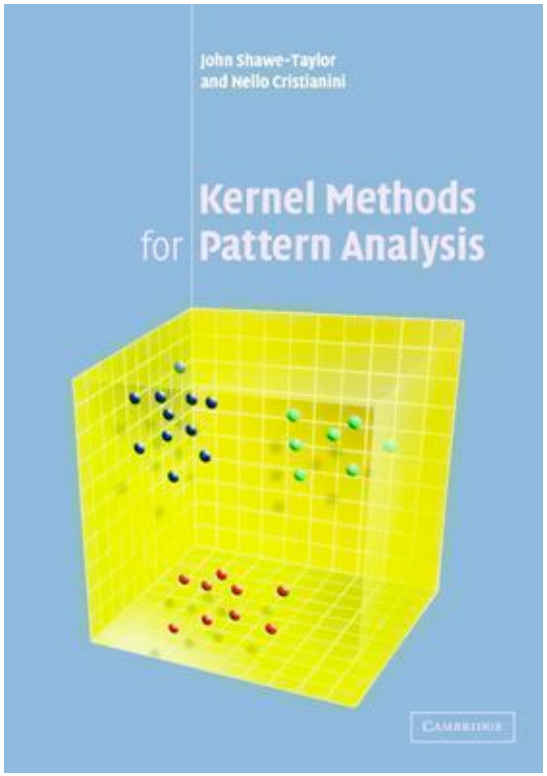
- Kernels on « structured objects » are studied in the fields of
  - pattern recognition,
  - machine learning
  - data mining…
- The idea here is to define kernel matrices (similarity matrices) on a graph
  - Defining similarities between the nodes of the graph

A brief overview of kernels

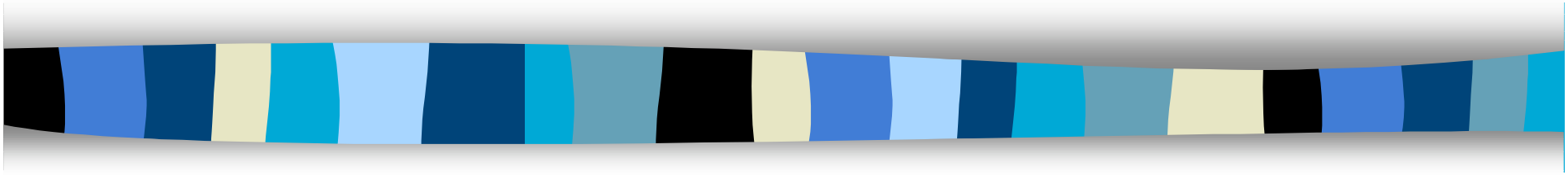
- Kernels have been defined on structures objects, such as
  - Graphs
  - Sequences of symbols
  - Probability distributions
  - Trees
  - Etc…
- See for instance Shawe-Taylor & Christianini (2005)

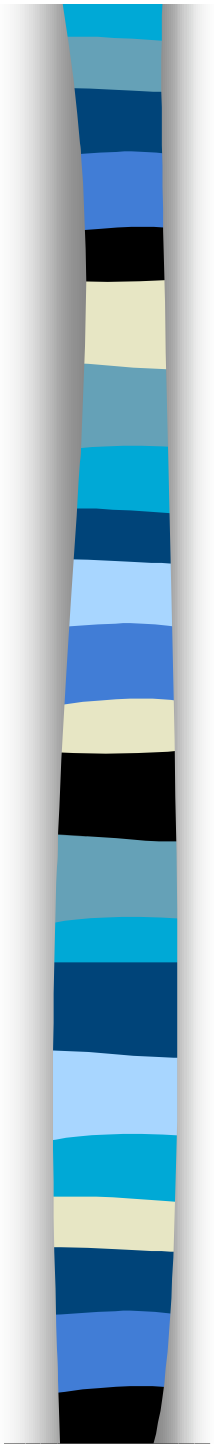
A brief overview of kernels

- See for instance Shawe-Taylor & Christianini (2004) – Cambridge University Press



Some kernel on a graph





Main point

- We will introduce several recently defined kernels on a graph
- Defining similarities between nodes of a graph

Main point

- These similarity measures between two nodes not only depend on
  - The weights of the edges
  - like the « shortest path » distance
- But also on
  - The number of paths connecting the two nodes
- They take high connectivity into account

  ≠ shortest-path or geodesic (Dijkstra) distance

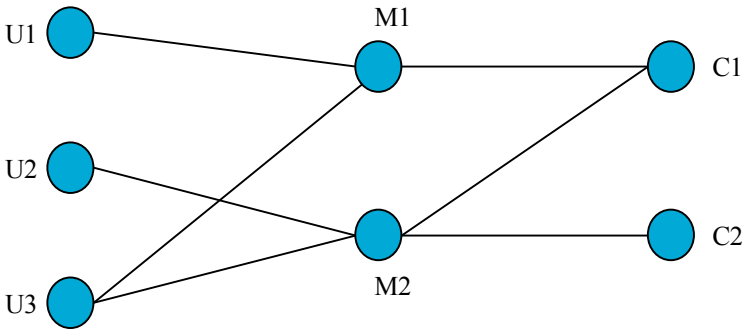
Some notations: the adjacency matrix

- The elements a_{ij} of the adjacency matrix $\mathbf{A}$ of a weighted, undirected, graph are defined as

$$a_{ij} = \begin{cases} w_{ij} \text{ if node } i \text{ is connected to node } j \\ 0 \text{ otherwise} \end{cases}$$

  where $\mathbf{A}$ is symmetric

- The $w_{ij} \geq 0$ represent the strength of relationship between node i and node j



$$\mathbf{A} = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Some notations: the Laplacian matrix

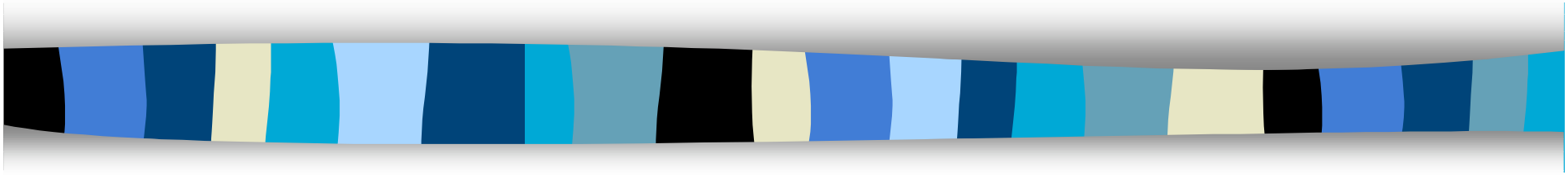
- The Laplacian matrix $\mathbf{L}$ of the graph is defined by

$$\mathbf{L} = \mathbf{D} - \mathbf{A}$$

where $\mathbf{D} = \mathrm{diag}(a_{i.})$ with $d_{ii} = [\mathbf{D}]_{ii} = a_{i.} = \sum_{j=1}^{n} a_{ij}$ (the outdegree of each node)

- $\mathbf{L}$ is doubly centered
- If the graph is connected, the rank of $\mathbf{L}$ is $n - 1$, where n is the number of nodes
- $\mathbf{L}$ is symmetric positive semidefinite

The exponential diffusion kernel



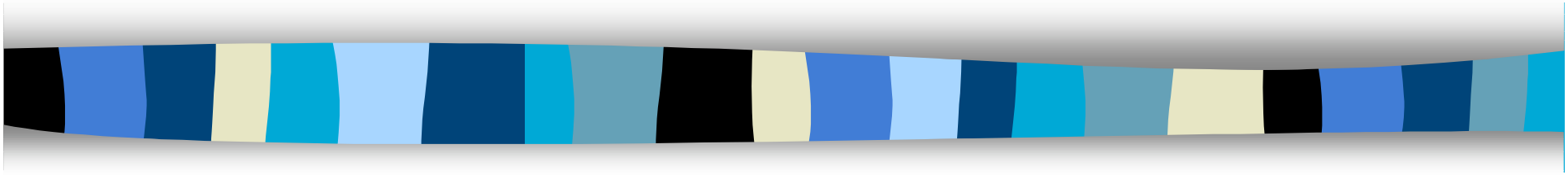
The exponential diffusion kernel

- The exponential diffusion kernel (Kondor & Lafferty, 2002; Smola & Kondor, 2003)

$$\mathbf{K}_{\mathrm{ED}} = \sum_{k=0}^{\infty} \frac{\alpha^k \mathbf{A}^k}{k!} = \exp(\alpha \mathbf{A})$$

  - If binary, $\mathbf{A}^k$ enumerates the « number of different paths » of k steps between two nodes,
  - Discounted with respect to the number of steps (k)
  - It is a kernel matrix

The Laplacian exponential diffusion kernel



The Laplacian exponential diffusion kernel

- The laplacian exponential diffusion kernel

$$\mathbf{K}_{\mathrm{LED}} = \exp(-\alpha \mathbf{L})$$

  - substitute $\mathbf{A}$ by $-\mathbf{L}$
  - It is a kernel matrix
- It has a nice interpretation in terms of a diffusion process

The Laplacian exponential diffusion kernel

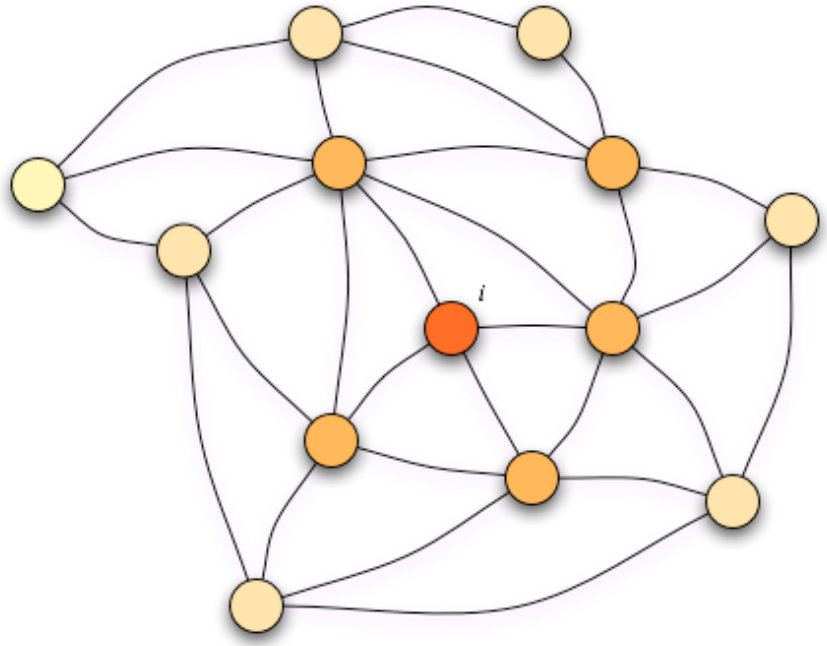
- Suppose we have a quantity x_i at each node i
- This quantity diffuses to neighbouring node j with a rate $a_{ij}x_i\ \delta t$
- This diffusion model leads to the equation

$$\mathbf{x}(t) = \exp(-\mathbf{L}t)\,\mathbf{x}_0$$

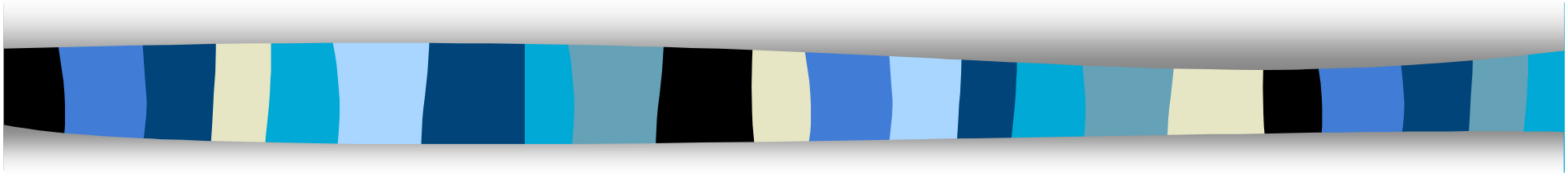
  – It corresponds to some diffusion process

The Laplacian exponential diffusion kernel

- Thus, if we have one source node i
  - The ith column of $\mathbf{K}_{\mathrm{LED}} = \exp(-\mathbf{L}t)$ will contain the diffused quantity at each node



The von Neumann diffusion kernel



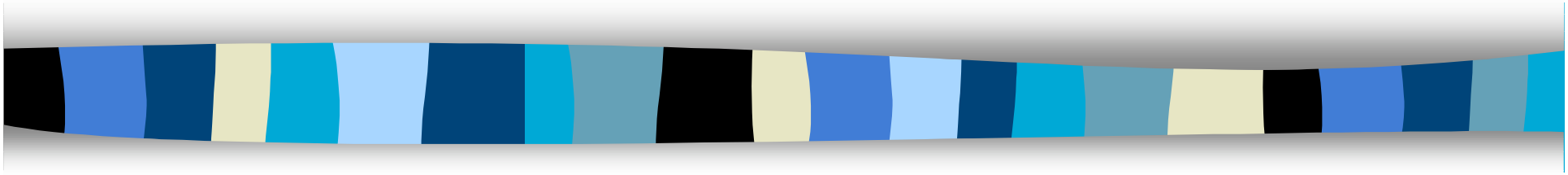
The von Neumann diffusion kernel

- The von Neumann diffusion kernel (Kandola, Shawe-Taylor & Christianini, 2002)

$$\mathbf{K}_{\text{VND}} = \sum_{k=0}^{\infty} \alpha^k \mathbf{A}^k = (\mathbf{I} - \alpha\mathbf{A})^{-1}$$

  - If $\mathbf{A}$ is binary, enumerates and sums the number of different paths between two nodes
  - discounted according to some discounting factor $0 < \alpha < 1$
  - It is a kernel matrix

The regularized Laplacian kernel



The regularized Laplacian kernel

- The regularized Laplacian kernel (Chebotarev & Shamis, 1998; Ito et al., 2004)

$$\mathbf{K}_{\mathrm{RL}} = (\mathbf{I} + \alpha\mathbf{L})^{-1}$$

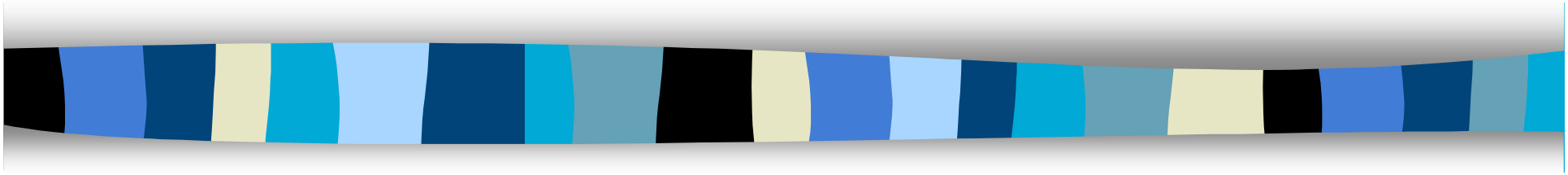
  – substitute $\mathbf{A}$ by $-\mathbf{L}$

  – It is a kernel matrix

The regularized Laplacian kernel

- It has a nice interpretation in terms of the matrix forest theorem
- Element k_{ij} corresponds to the ratio of
  - the total weight of spanning forests rooted at node i for which node i and j belong to the same tree
  - On the total weight of spanning forests rooted at node i

The commute-time kernel



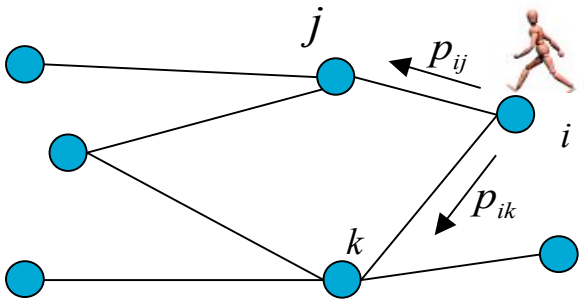
The commute-time kernel

- Introduced by Saerens et al (2004); see also Qiu & Hancock (2005) and Brand (2005)
- Every node is associated to a state of a Markov chain
- The Markov chain is defined by the single-step transition probabilities

$$\mathrm{P}(s(t+1) = j | s(t) = i) = p_{ij} = \frac{a_{ij}}{a_{i.}}$$

The commute-time kernel

- From this Markov chain, we then compute :
  - The average commute time, $n(i,j)$
  - Average number of steps a random walker, starting in state $i \neq j$, will take before entering a given state j for the first time, and go back to i



The commute-time kernel

- If we further define $\mathbf{e}_i$ as the ith column of $\mathbf{I}$

$$\mathbf{e}_i = [\underset{1}{0}, \ldots, \underset{i-1}{0}, \underset{i}{1}, \underset{i+1}{0}, \ldots, \underset{n}{0}]^{\mathrm{T}}$$

- we obtain the remarkable form

$$\boxed{n(i,j) = 2N_e(\mathbf{e}_i - \mathbf{e}_j)^{\mathrm{T}}\mathbf{L}^{+}(\mathbf{e}_i - \mathbf{e}_j)}$$

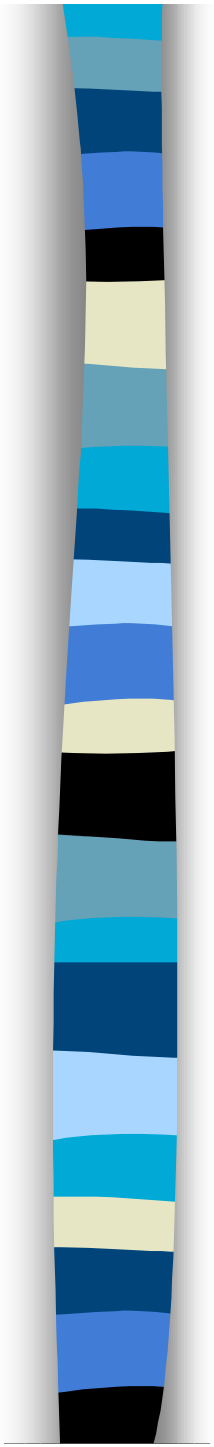
where each node i is represented by a unit basis vector, $\mathbf{e}_i$, in the node space

- $\mathbf{L}^{+}$ is the Moore-Penrose pseudoinverse of the Laplacian matrix of the graph

The commute-time kernel

- Thus, $n(i,j)$ is a Mahalanobis distance
  = Commute Time Distance

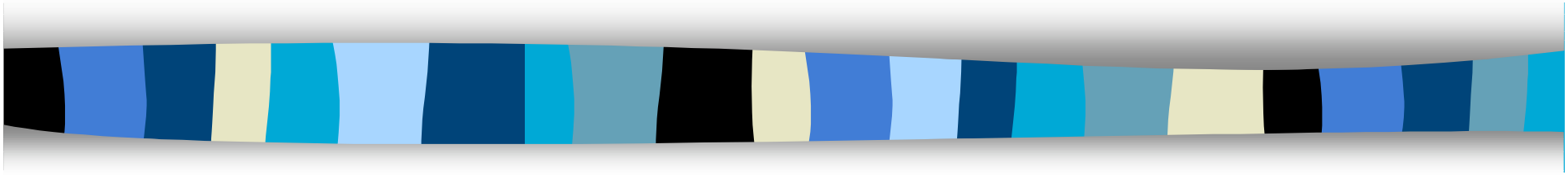
- Indeed, one can show that $\mathbf{L}^+$ is
  - (1) Symmetric
  - (2) Positive semidefinite
  - (3) Doubly centered



The commute-time kernel

- Thus $\mathbf{L}^{+}$ is a kernel matrix
  - A Gram matrix
  - Indeed, it is positive semidefinite

The Markov diffusion kernel



The Markov diffusion kernel

- The Markov diffusion kernel, inspired by Nadler et al. (2006) and Lafon & Lee (2006), is introduced in Fouss et al. (2006)

$$\mathbf{K}_{\mathrm{MD}}(t) = \mathbf{P}^t(\mathbf{P}^{\mathrm{T}})^t$$

  - where $\mathbf{P}$ is the transition probabilities matrix
  - of the associated Markov chain

The Markov diffusion kernel

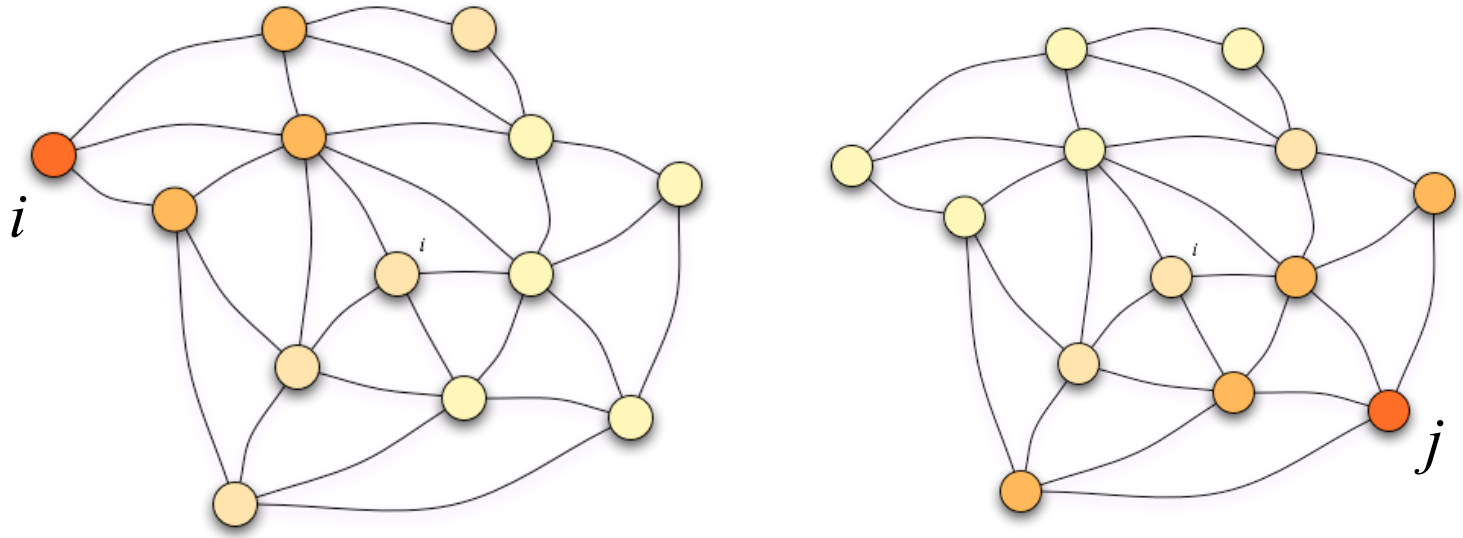
- A meaningful distance between node i and j, proposed by Nadler et al. (2005) as well as Latapy et al. (2005) is

$$d^2(i,j) = \sum_{k=1}^{n} \left[\mathrm{P}(s(t) = k | s(0) = i) - \mathrm{P}(s(t) = k | s(0) = j) \right]^2$$

- It aims to compute the distance between the distribution of presence rate when starting from two different nodes i and j

The Markov diffusion kernel

- From two source nodes i and j
- We compute the difference of densities



The Markov diffusion kernel

- We easily obtain

$$
\begin{aligned}
d^2(i,j) &= \sum_{k=1}^{n} \left[\mathrm{P}(s_t = k | s_0 = i) - \mathrm{P}(s_t = k | s_0 = j) \right]^2 \\
&= \sum_{k=1}^{n} \left[\mathbf{e}_k^{\mathrm{T}} (\mathbf{P}^{\mathrm{T}})^t \mathbf{e}_i - \mathbf{e}_k^{\mathrm{T}} (\mathbf{P}^{\mathrm{T}})^t \mathbf{e}_j \right]^2 \\
&= \left\| (\mathbf{P}^{\mathrm{T}})^t \mathbf{e}_i - (\mathbf{P}^{\mathrm{T}})^t \mathbf{e}_j \right\|^2 \\
&= (\mathbf{e}_i - \mathbf{e}_j)^{\mathrm{T}} \mathbf{P}^t (\mathbf{P}^{\mathrm{T}})^t (\mathbf{e}_i - \mathbf{e}_j)
\end{aligned}
$$

  - where $\mathbf{P}$ is the transition probability matrix
  - This is a Mahalanobis distance

The Markov diffusion kernel

- The associated kernel matrix is

$$\mathbf{K}_{\mathrm{MD}}(t) = \mathbf{P}^t(\mathbf{P}^{\mathrm{T}})^t$$

  – which is the Markov diffusion kernel

- Its natural embedding space is called the diffusion map
  – Exactly as originally proposed by Nadler et al. (2005)

The Markov diffusion kernel

- In the diffusion map, the nodes have, as first coordinate,
  - The largest non-trivial eigenvector of the transition probabilities matrix, $\mathbf{P}$
  - Multiplied by its corresponding eigenvalue
- The second coordinate of the nodes is provided by
  - The second non-trivial eigenvector, etc

The Markov diffusion kernel

- This will be shown equivalent to
  - The Laplacian eigenmap (Belkin & Nirogi, 2001, 2003)
  - A weighted one-dimensional mapping of the graph (Zien et al., 1999)
  - A relaxation of the normalized cut of the graph (Shi & Malik)
  - A relaxation of the MinMaxCut of the graph (Ding et al., 2001)

The integrated Markov diffusion kernel

- If we sum up the distance

$$
\begin{aligned}
d_{ij}^2(t) &= \left[\left(\mathbf{P}^{\mathrm{T}}\right)^t \mathbf{e}_i - \left(\mathbf{P}^{\mathrm{T}}\right)^t \mathbf{e}_j\right]^{\mathrm{T}} \mathbf{W} \left[\left(\mathbf{P}^{\mathrm{T}}\right)^t \mathbf{e}_i - \left(\mathbf{P}^{\mathrm{T}}\right)^t \mathbf{e}_j\right] \\
&= \left[\mathbf{e}_i - \mathbf{e}_j\right]^{\mathrm{T}} \mathbf{P}^t \mathbf{W} \left(\mathbf{P}^{\mathrm{T}}\right)^t \left[\mathbf{e}_i - \mathbf{e}_j\right] \\
&= \left(\mathbf{e}_i - \mathbf{e}_j\right)^{\mathrm{T}} \mathbf{K}_{\mathrm{DM}} \left(\mathbf{e}_i - \mathbf{e}_j\right)
\end{aligned}
$$

  - For $t = 1$ to *infinity*, with a damping factor α and a weighting diagonal matrix $\mathbf{W}$

The integrated Markov diffusion kernel

- We obtain the integrated Markov diffusion kernel (Yen et al., 2007)

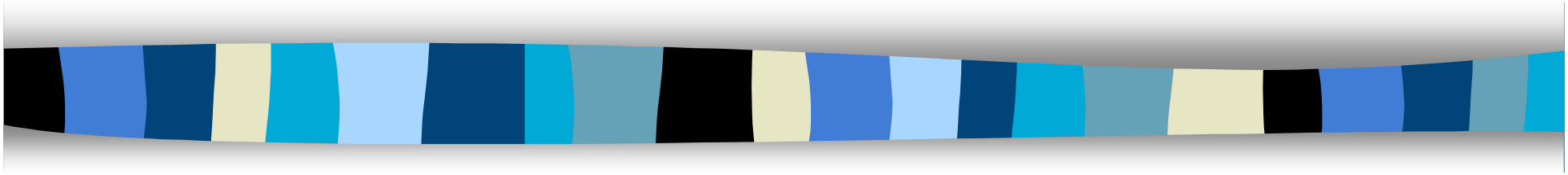
$$d_{ij}^2 = [\mathbf{e}_i - \mathbf{e}_j]^{\mathrm{T}} \left[\sum_{t=1}^{\infty} \alpha^t \mathbf{P}^t \mathbf{W} \left(\mathbf{P}^{\mathrm{T}}\right)^t \right] [\mathbf{e}_i - \mathbf{e}_j]$$

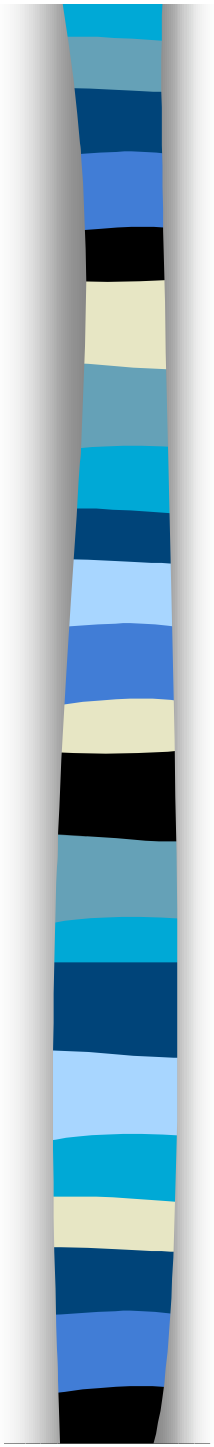
$$= [\mathbf{e}_i - \mathbf{e}_j]^{\mathrm{T}} \mathbf{K}_{\mathrm{I}} [\mathbf{e}_i - \mathbf{e}_j]$$

- with

$$\mathbf{vec}(\mathbf{K}_{\mathrm{I}}) = \alpha \left[\mathbf{I} - \alpha(\mathbf{P} \otimes \mathbf{P})\right]^{-1} \mathbf{vec}(\mathbf{P}\mathbf{W}\mathbf{P}^{\mathrm{T}})$$

The random walk with restart similarity





The random walk with restart similarity

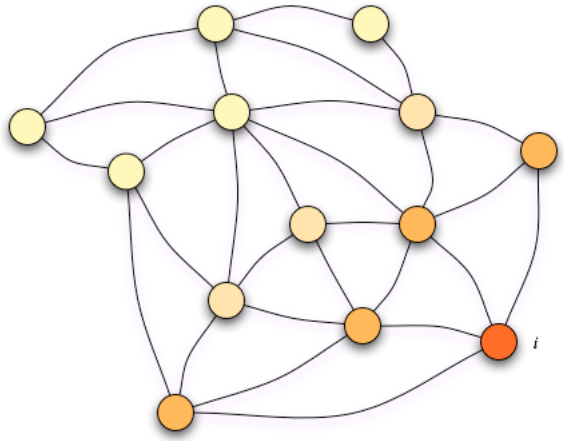
- Introduced by Tong et al. (2007)
- Inspired by PageRank

The random walk with restart similarity

- It introduces a random walk
  - With a restart at node i

$$\mathbf{x}_i(t+1) = \alpha \mathbf{P}^{\mathrm{T}} \mathbf{x}_i(t) + (1-\alpha)\mathbf{e}_i$$

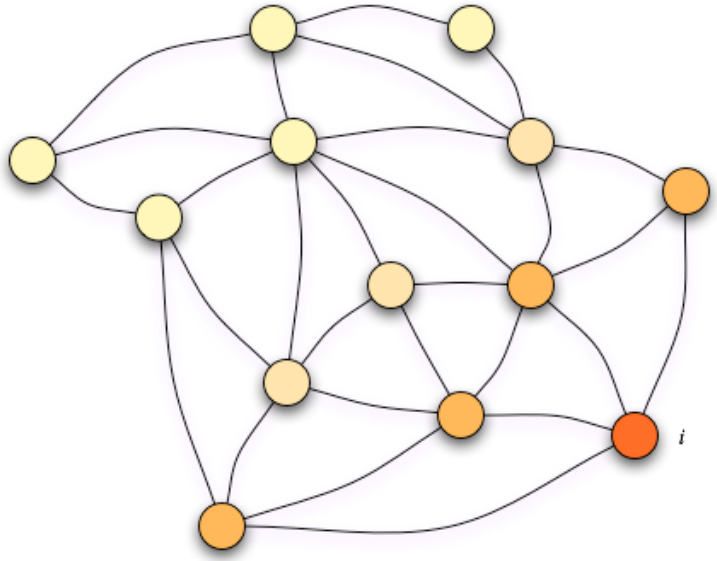
  - Which produces a similarity to node i



The random walk with restart similarity

- The long-term solution (steady-state) is

$$\mathbf{x}_i = (1-\alpha)(\mathbf{I} - \alpha \mathbf{P}^{\mathrm{T}})^{-1}\mathbf{e}_i$$



The random walk with restart similarity

- Thus,

$$\mathbf{K} = (\mathbf{I} - \alpha \mathbf{P}^{\mathrm{T}})^{-1}$$

  – Is a similarity matrix,
  – but not a kernel

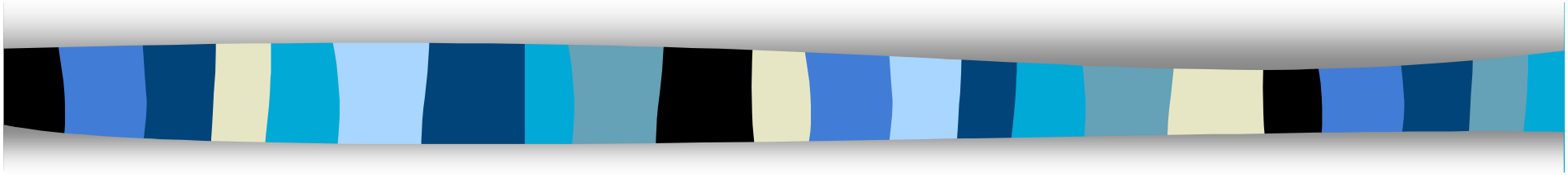
Applications of kernels on a graph

- Now that we have a similarity measure between the nodes,we can use it for
  - Clustering the nodes
  - Finding dense regions in the graph
  - Finding outlier nodes
  - Representing the graph in a low-dimensional space (principal components analysis)
  - Representing the graph in function of the similarity with some reference nodes (discriminant analysis)
  - Finding central nodes in the graph
  - Find the most similar node (nearest neighbours)
  - Etc…

Applications of kernels on a graph

- Which kernel to choose
  - Depends on the problem at hand
  - And should be stated on empical grounds
- However, we found that
  - Laplacian-based kernels perform better than adjacency-matrix based kernels
  - The diffusion map, commute-time and regularized Laplacian kernels perform best
  - for collaborative recommendation
- At least one our the datasets we tested

A similarity measure between nodes of two graphs



A similarity measure between nodes of two graphs

- Let us mention an interesting extension of Kleinberg's HITS algorithms
  - It defines a similarity measure between nodes of two graphs
  - It therefore performs some graph matching
  - Developed by Blondel et al. (2004)

A similarity measure between nodes of two graphs

- Suppose we are given two graphs G_A and G_B
  - Where $\mathbf{A}$ (n_A x n_A) and $\mathbf{B}$ (n_B x n_B) are the adjacency matrices of G_A and G_B
  - Define a (n_B x n_A) similarity matrix $\mathbf{K}$ between the nodes of the two graphs
  - Which needs not to be symmetric (not a kernel)

A similarity measure between nodes of two graphs

- The similarity matrix $\mathbf{K}$ can be computed through a power method

$$\mathbf{K}(t+1) = \mathbf{B}\mathbf{K}(t)\mathbf{A}^{\mathrm{T}} + \mathbf{B}^{\mathrm{T}}\mathbf{K}(t)\mathbf{A}$$

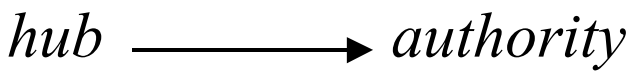
A similarity measure between nodes of two graphs

- Intuitively, two nodes, i of G_B and j of G_A are similar,
  - that is, k_{ij} is large,
  - if node i of G_B is linked to similar nodes of G_B
  - and node j of G_A is linked to similar nodes of G_A

$$k_{ij}(t+1) = \sum_k \sum_l b_{ik} k_{kl} a_{jl} + \sum_k \sum_l b_{ki} k_{kl} a_{lj}$$

A similarity measure between nodes of two graphs

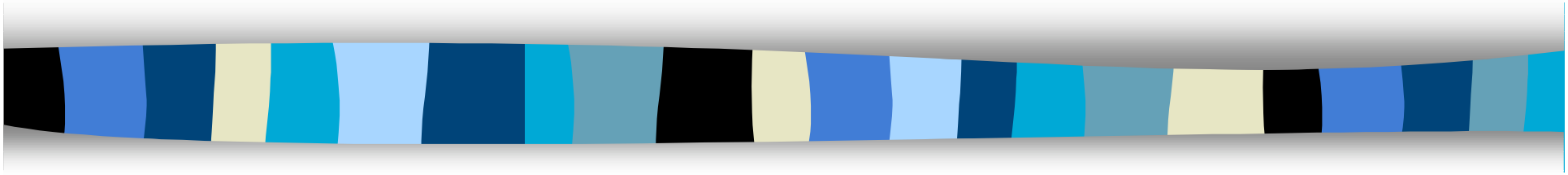
- It reduces to Kleinberg's HITS procedure for some particular form of graph G_B

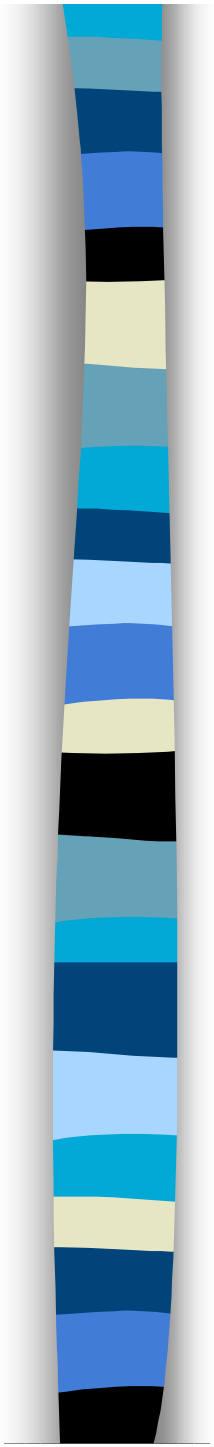


  – See the paper of Blondel et al. (2004)

Application to clustering

Clustering in the embedded space

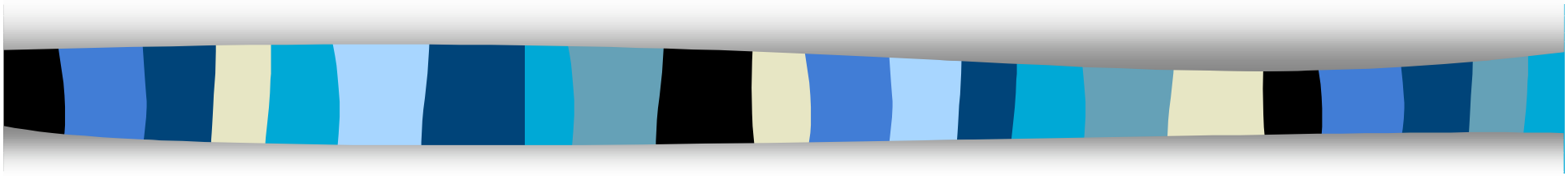


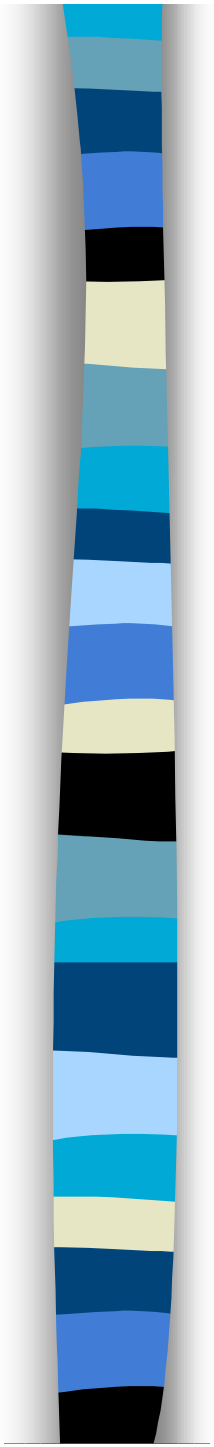


Clustering in the embedded space

- The kernels induce some embedding space
  - The nodes of the graph are embedded into this space
  - A clustering algorithm is used to group the nodes
  - See for instance Donetti & Munoz (2004)
  - This is linked to spectral clustering

Kernel clustering





Kernel clustering

- Since we have a kernel, a kernel clustering technique is quite natural
- We introduce a version of a kernel clustering method:
  - Kernel k-means
- Applied to graph nodes clustering
  - Based on the commute-time kernel

Kernel k-means

- Kernel k-means has first been introduced by Zhang & Chen (2002, 2004) and Girolami (2002)

- We introduce an intuitive version of kernel k-means (Yen et al., 2007)
  - Which easily generalizes to other clustering algorithms
  - And which is prototype-based
  - In the sample space

Kernel k-means

- We want to minimize the total within-class inertia

$$J(\mathbf{g}_1, \ldots, \mathbf{g}_m) = \sum_{k=1}^{m} \sum_{i \in C_k} ||\mathbf{x}_i - \mathbf{g}_k||^2$$

- We introduce the « kernel trick », where $\mathbf{X}$ is the data matrix

$$\mathbf{g}_k \rightarrow \mathbf{X}^{\mathrm{T}} \boldsymbol{\gamma}_k$$

  - Which aims to express the prototypes γ_k as
  - A linear combination of the observations in the feature space

Kernel k-means

- We easily obtain

$$J(\boldsymbol{\gamma}_1,\ldots,\boldsymbol{\gamma}_m) = \sum_{k=1}^{m}\sum_{i\in C_k}(\mathbf{x}_i-\mathbf{g}_k)^{\mathrm{T}}(\mathbf{x}_i-\mathbf{g}_k)$$

$$= \sum_{k=1}^{m}\sum_{i\in C_k}(\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_i - 2\mathbf{x}_i^{\mathrm{T}}\mathbf{g}_k + \mathbf{g}_k^{\mathrm{T}}\mathbf{g}_k)$$

$$= \sum_{k=1}^{m}\sum_{i\in C_k}(k_{ii} - 2\mathbf{k}_i^{\mathrm{T}}\boldsymbol{\gamma}_k + \boldsymbol{\gamma}_k^{\mathrm{T}}\mathbf{K}\boldsymbol{\gamma}_k)$$

Kernel k-means

- The k-means iteratively minimizes J by iteratively
  - Assigning cluster labels, l_i , to nodes
  - Recomputing the cluster prototypes $\boldsymbol{\gamma}_k$

- The cluster assignment that minimizes J is

$$l_i = \arg\min_k \left\{ \boldsymbol{\gamma}_k^{\mathrm{T}} \mathbf{K} \boldsymbol{\gamma}_k - 2\mathbf{k}_i^{\mathrm{T}} \boldsymbol{\gamma}_k \right\}$$

Kernel k-means

- The cluster prototypes $\boldsymbol{\gamma}_k$ that minimize J are solutions of

$$\mathbf{K}\boldsymbol{\gamma}_k = \frac{1}{n_k}\sum_{i \in C_k} \mathbf{k}_i$$

  - where $\mathbf{k}_i$ is the ith column of $\mathbf{K}$ and n_k is the number of nodes assigned to cluster k
  - But $\mathbf{k}_i = \mathbf{K}\mathbf{e}_i$

Kernel k-means

- Since, in both sides of the equation, we have a linear combination of the columns of $\mathbf{K}$, one solution is

$$\boldsymbol{\gamma}_k = \frac{1}{n_k} \sum_{i \in C_k} \mathbf{e}_i$$

$$\gamma_{ki} = [\boldsymbol{\gamma}_k]_i = \begin{cases} 1/n_k \text{ if node } i \in C_k \\ 0 \text{ otherwise} \end{cases}$$

Kernel k-means

- Thus, γ_k is a membership vector containing the membership value
  - of each node to the cluster k
- It is therefore a kind of « prototype » for the cluster k
  - In the sample space

Kernel k-means

- Thus, we simply iterate
  - For all observations i:

$$l_i = \arg\min_k \left\{ \boldsymbol{\gamma}_k^{\mathrm{T}} \mathbf{K} \boldsymbol{\gamma}_k - 2\mathbf{k}_i^{\mathrm{T}} \boldsymbol{\gamma}_k \right\}$$

  - For all clusters k:

$$\boldsymbol{\gamma}_k = \frac{1}{n_k} \sum_{i \in C_k} \mathbf{e}_i$$

Other kernel clustering methods

- Other standard clustering algorithms have been « kernelized » in the same way:
  - Iterative k-means
  - The fuzzy k-means
  - The entropy-based fuzzy k-means
  - The gaussian mixture model

Approximate prototype embedding

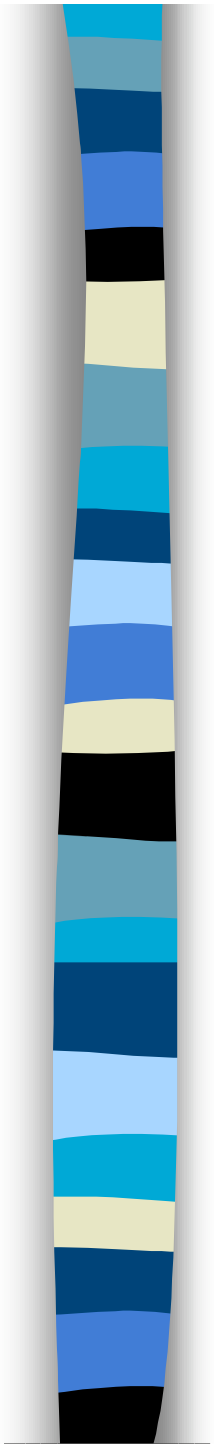
- Notice that if we want to avoid the comparisons of all pairs of nodes, we could use

$$\mathbf{g}_k \rightarrow \widetilde{\mathbf{X}}^{\mathrm{T}} \widetilde{\boldsymbol{\gamma}}_k$$

  - where $\widetilde{\mathbf{X}}$ is now a $\widetilde{n} \times p$ reduced data matrix
  - containing a small set of selected nodes, called the representatives
- The parameter vector $\mathbf{g}_k$ is restricted to lie in the subspace spanned by the representatives

Approximate prototype embedding

- In that case, only similarities between nodes and the representatives need to be computed
  - For the commute-time kernel, only sparse linear systems need to be computed
- The kernel k-means algorithm can easily be adapted to this case



Application to document clustering

- We show the results on the newsgroup database
  - Contains about 20000 documents
  - From 20 different newsgroups
  - About 1000 documents for each newsgroup

Application to document clustering

- A preprocessing step was performed:
  - Remove stop words
  - Use Porter's stemming algorithm
  - Compute the mutual information between documents and terms and remove terms with too few mutual information
  - Compute the term-document matrix $\mathbf{W}$ containing the tf.idf factors
  - Compute the document-document adjacency matrix

$$\mathbf{A} = \mathbf{W}^T\mathbf{W}$$

Application to document clustering

- Thus, we have a large graph where
  - Nodes are documents
  - Link weights are computed from the tf.idf factors

Sigmoid commute-time kernel

1. From this adjacency matrix, compute the commute-time kernel matrix

$$\mathbf{K} = (\mathbf{D} - \mathbf{A})^{+} = \mathbf{L}^{+}$$

2. Then take the sigmoid kernel

$$[\mathbf{K}_{\mathrm{CT}}]_{ij} = 1/(1 + \exp[a\ l_{ij}^{+}/\sigma])$$

where a is fixed to 1.26 based on informal tests

3. Perform a kernel k-means on $\mathbf{K}_{\mathrm{CT}}$

Newsgroups data sets

– Here are the different data sets

G-2cl-A	politics/general, sport/baseball
G-2cl-B	computer/graphics, motor/motorcycles
G-2cl-C	space/general, politics/mideast
G-3cl-A	sport/baseball, space/general, politics/mideast
G-3cl-B	computer/windows, motor/autos, religion/general
G-3cl-C	sport/hockey, religion/atheism, medicine/general
G-5cl-A	computer/graphics, motor/motorcycles, politics/mideast, space/general, sport/baseball
G-5cl-B	computer/graphics, computer/pchardware, motor/autos, religion/atheism, politics/mideast
G-5cl-C	computer/machardware, sport/hockey, medicine/general, religion/general, forsale/general

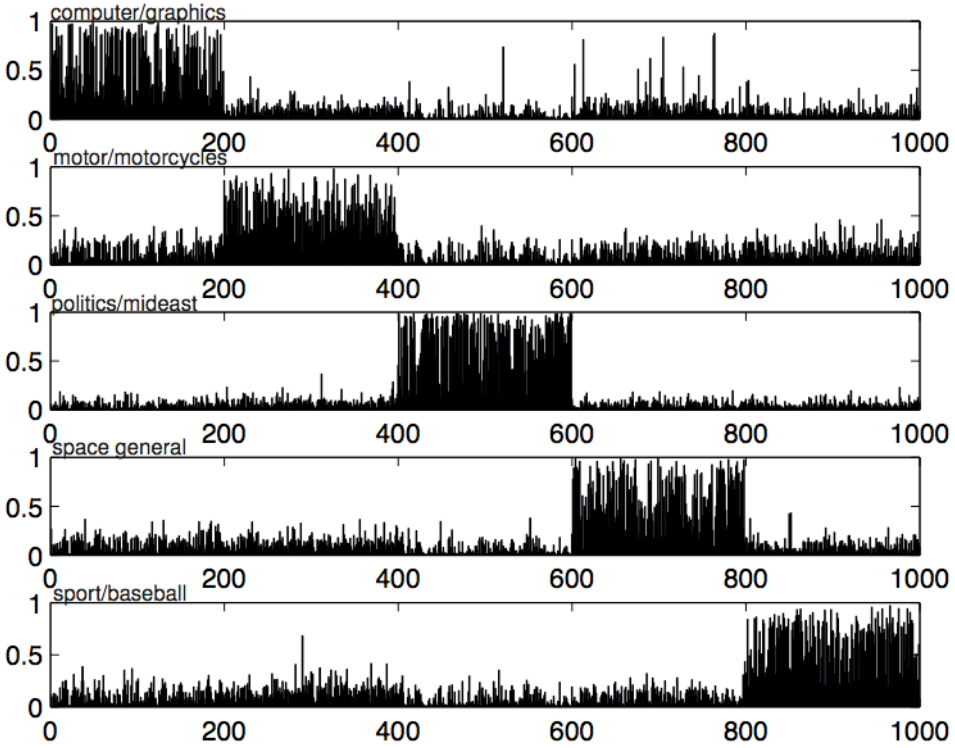
Results obtained on the newsgroups data sets

- We also report the results obtained by the spherical k-means algorithm (Dhillon et al, 2002)

	$\mathbf{K}_{CT}$ k-means		$\mathbf{K}_{CT}$ fuzzy k-means		Sph. k-means	
	class. rate	adj. Rand	class. rate	adj. Rand	class. rate	adj. Rand
G-2cl-A	97.25 %	0.95	97.25 %	0.95	91.76 %	0.85
G-2cl-B	91.23 %	0.84	91.46 %	0.84	81.46 %	0.70
G-2cl-C	95.71 %	0.92	95.99 %	0.92	94.82 %	0.90
G-3cl-A	94.42 %	0.92	94.83 %	0.93	89.23 %	0.85
G-3cl-B	93.37 %	0.91	93.14 %	0.90	86.71 %	0.82
G-3cl-C	93.65 %	0.91	92.44 %	0.89	87.35 %	0.83
G-5cl-A	82.64 %	0.80	86.49 %	0.84	75.29 %	0.74
G-5cl-B	70.75 %	0.75	77.08 %	0.78	64.43 %	0.69
G-5cl-C	77.12 %	0.76	79.54 %	0.78	65.23 %	0.69

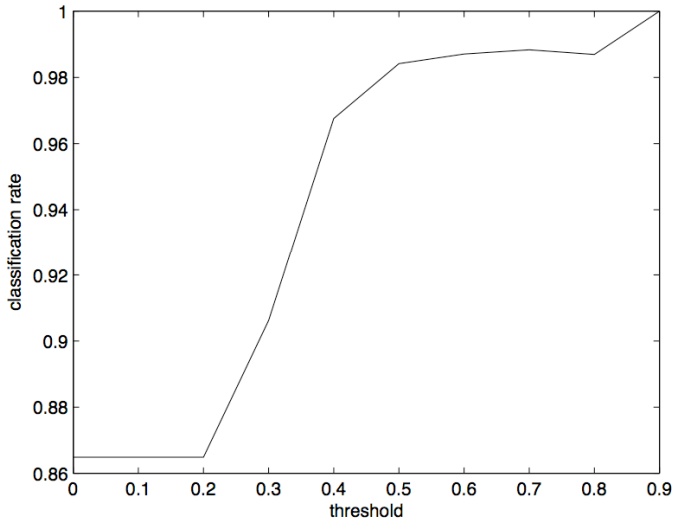
Example of kernel fuzzy clustering

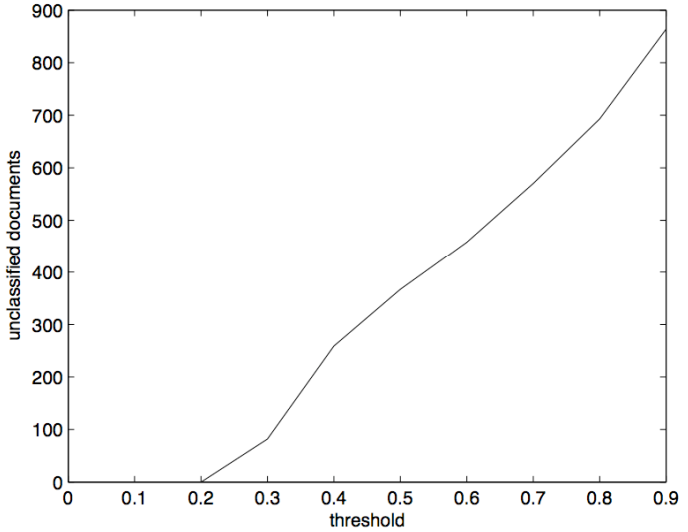
- Here are some results for the kernel fuzzy clustering



Preliminary results obtained on the newsgroups data sets

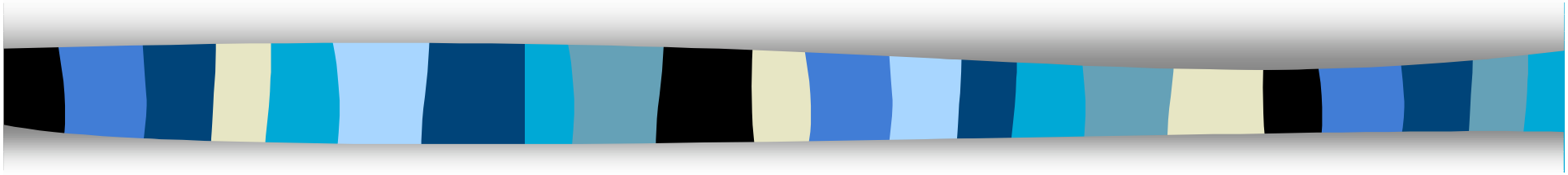
- Fuzzy clustering
  - If we change the membership treshold:

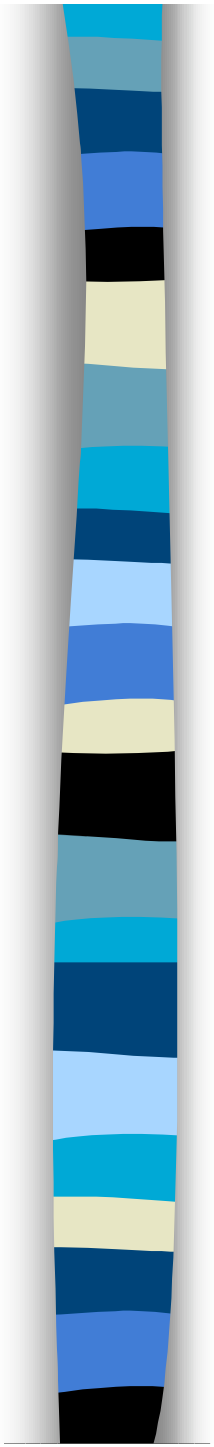




Finding dense regions

Greedy clustering based on modularity





Greedy clustering based on modularity

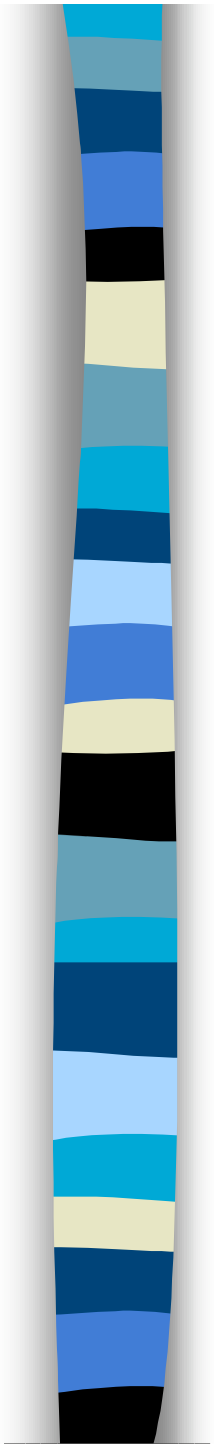
- Newman (2004) introduced the modularity Q
  - Which is a measure of the quality of a partition of the graph
  - Suppose we have a partition into three clusters A, B, C

Greedy clustering based on modularity

- The idea behind the modularity is the following

	A	B	C
A	$p(A,A)$ $p(A,.)\ p(.,A)$	$p(A,B)$ $p(A,.)\ p(.,B)$	$p(A,C)$ $p(A,.)\ p(.,C)$
B	$p(B,A)$ $p(B,.)\ p(.,A)$	$p(B,B)$ $p(B,.)\ p(.,B)$	$p(B,C)$ $p(B,.)\ p(.,C)$
C	$p(C,A)$ $p(C,.)\ p(.,A)$	$p(C,B)$ $p(C,.)\ p(.,B)$	$p(C,C)$ $p(C,.)\ p(.,C)$

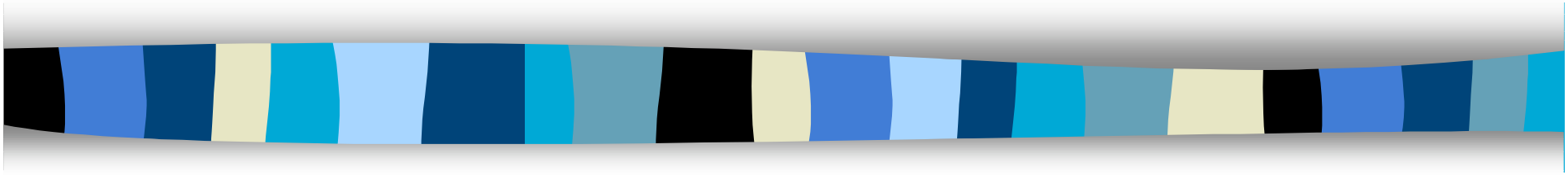
$$Q = (p(A,A) - p(A,.)p(.,A)) + (p(B,B) - p(B,.)p(.,B)) + (p(C,C) - p(C,.)p(.,C))$$



Greedy clustering based on modularity

- The modularity should be maximized
  - Meaning that the sample is very far from independence
- Newman designed a greedy approach
  - Initially, every node is a cluster
  - Try merging every couple of clusters and compute the resulting modularity
  - Merge the couple of clusters that results in the largest increase in modularity
- Modularity is very popular today !

Kernel hierarchical clustering

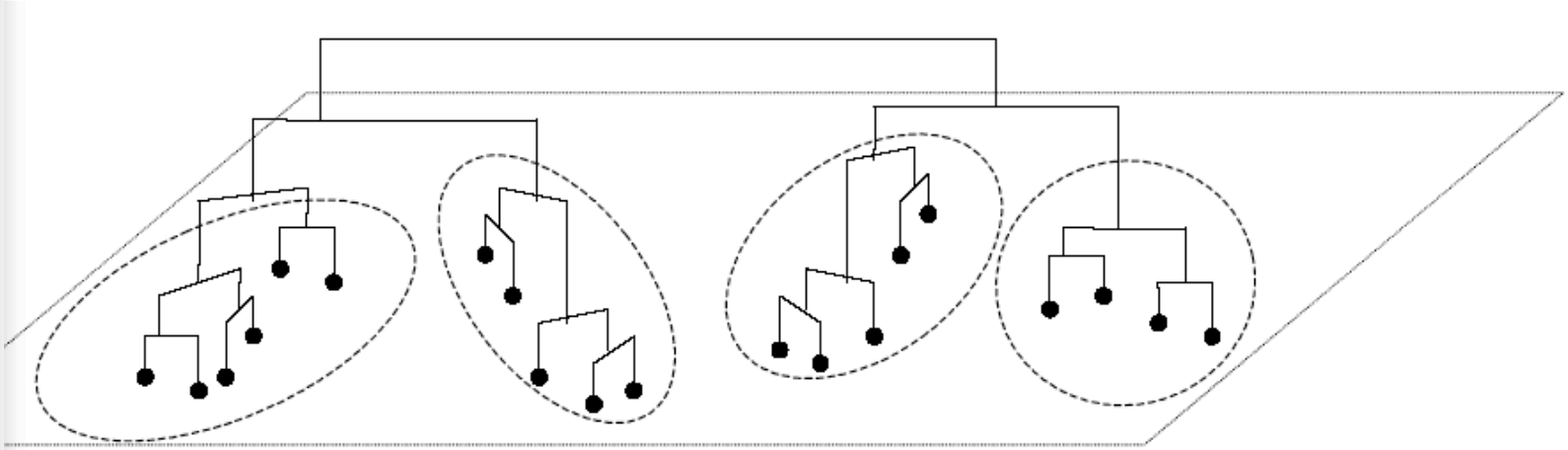


Kernel hierarchical clustering

- We define an agglomerative procedure
  - Which is a kernel version of Wald's algorithm
  - It agglomerates nodes which lead to the smallest decrease in total within-class inertia

Kernel hierarchical clustering

- We begin with one group by node
  - Then, we merge groups that lead to the smallest increase in within-class inertia
  - That is, we expand most dense groups



Kernel hierarchical clustering

- At the beginning of the process, each observation is a group
  - And the centroid $\mathbf{g}_k$ is the observation itself:

$$\mathbf{g}_k = \mathbf{x}_k, \text{ for } k \in \{1, 2, \ldots, n\}$$

  - By applying the transformation (the kernel trick),

$$\mathbf{g}_k \rightarrow \mathbf{X}^{\mathrm{T}}\mathbf{h}_k$$

Kernel hierarchical clustering

- We easily obtain by pre-multiplying this equation by $\mathbf{X}$:

$$\mathbf{K}\mathbf{h}_k = \mathbf{k}_k = \mathbf{K}\mathbf{e}_k$$

- And thus, initially,

$$\mathbf{h}_k = \mathbf{e}_k, \text{ for each } k \in \{1, 2, \ldots, n\}$$

- Thus, $\mathbf{h}_k$ is a prototype vector in the sample space

Kernel hierarchical clustering

- Now, when merging two groups, say group k and group l, into group m,
- The new centroid $\mathbf{g}_k$ in the feature space is

$$\mathbf{g}_m = \frac{n_k \mathbf{g}_k + n_l \mathbf{g}_l}{n_k + n_l}$$

- By applying the kernel trick, the update for the $\mathbf{h}_k$ is

$$\mathbf{h}_m = \frac{n_k \mathbf{h}_k + n_l \mathbf{h}_l}{n_k + n_l}$$

Kernel hierarchical clustering

- We measure the density of the groups by the within-cluster inertia
- Moreover, it is well-known that merging cluster k and cluster l
  - Results in an increase of total within-cluster inertia of

$$\Delta J = J(\text{after merge}) - J(\text{before merge})$$
$$= \frac{n_k n_l}{n_k + n_l} ||\mathbf{g}_k - \mathbf{g}_l||^2$$

Kernel hierarchical clustering

- In the sample space, we obtain

$$\Delta J = \frac{n_k n_l}{n_k + n_l} (\mathbf{h}_k - \mathbf{h}_l)^{\mathrm{T}} \mathbf{K} (\mathbf{h}_k - \mathbf{h}_l)$$

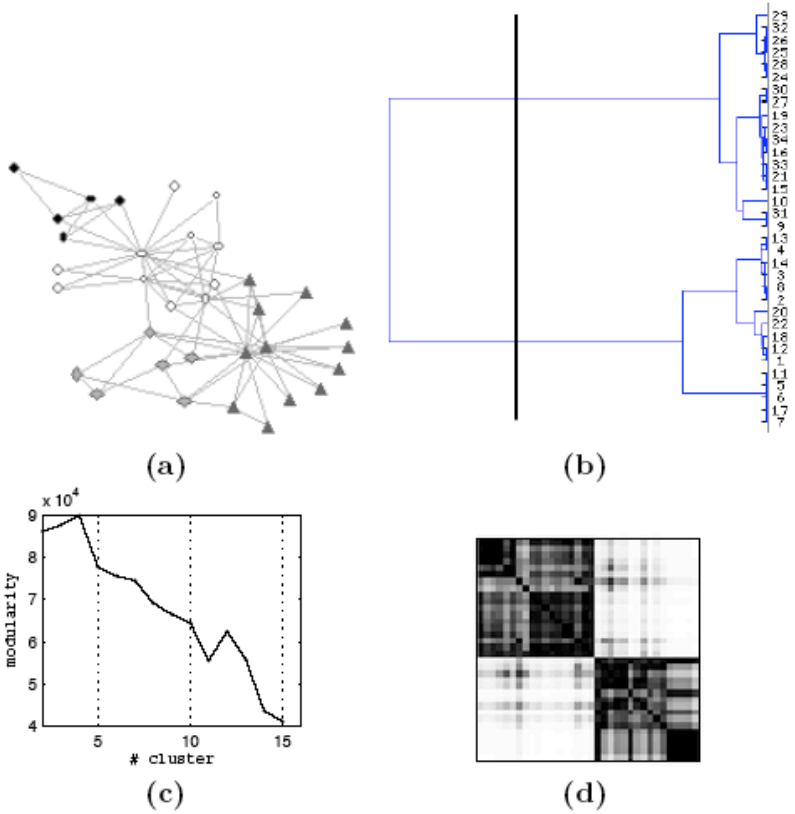
- The idea is thus
  - To try any couple of merge
  - To merge the two groups k, l that result in the smallest increase in total within-class inertia, ΔJ

Kernel hierarchical clustering

- This mimics Ward's grouping method in the sample space
- Here are two examples on social science networks (Yen et al., 2007)

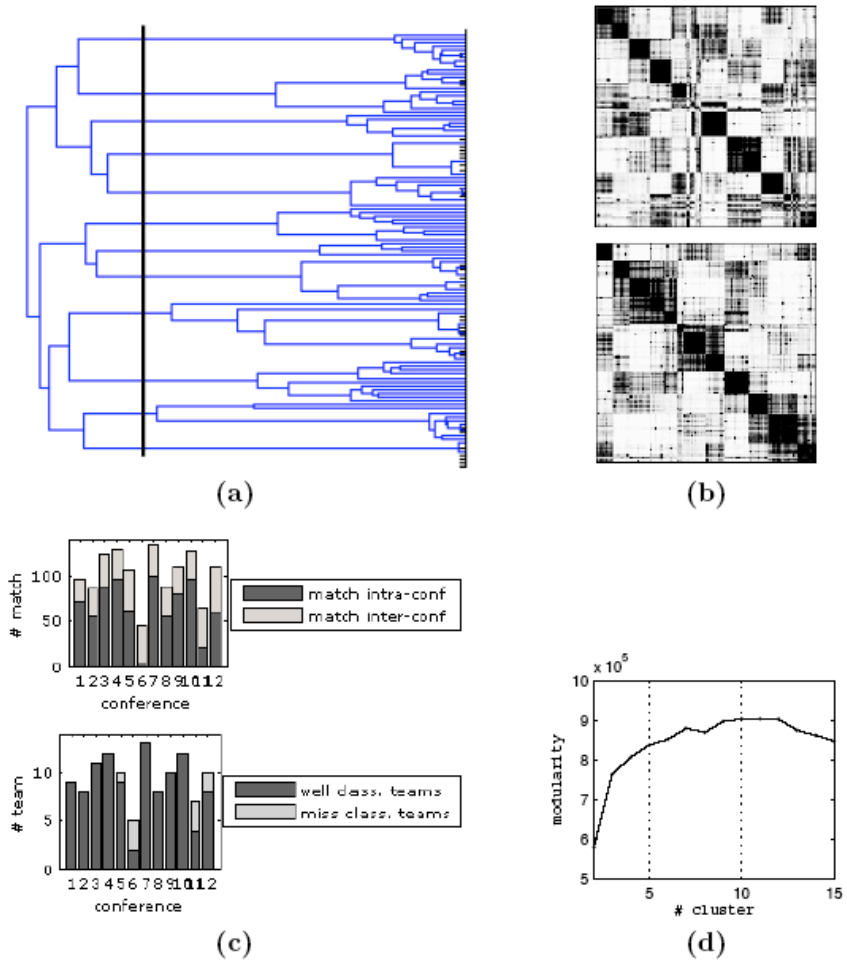
Kernel hierarchical clustering

- Zachary's karate club

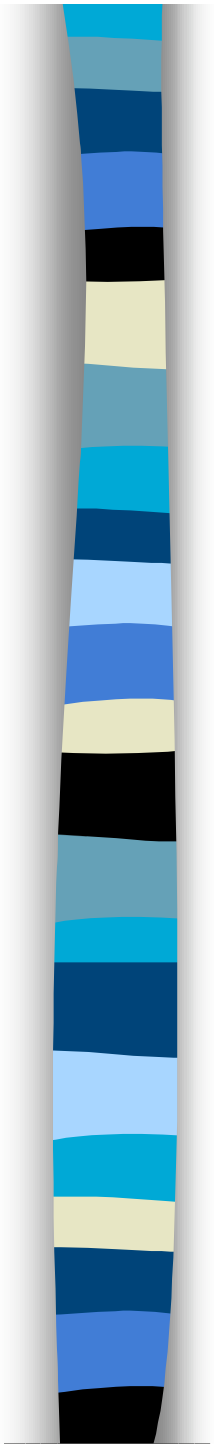


Kernel hierarchical clustering

- College football network



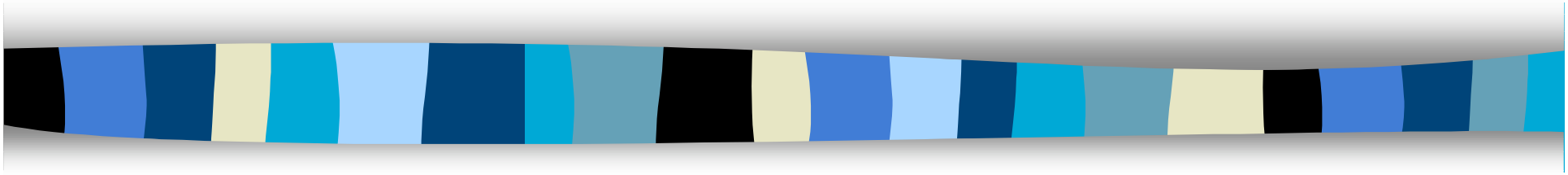
Graph partitioning



Basic models

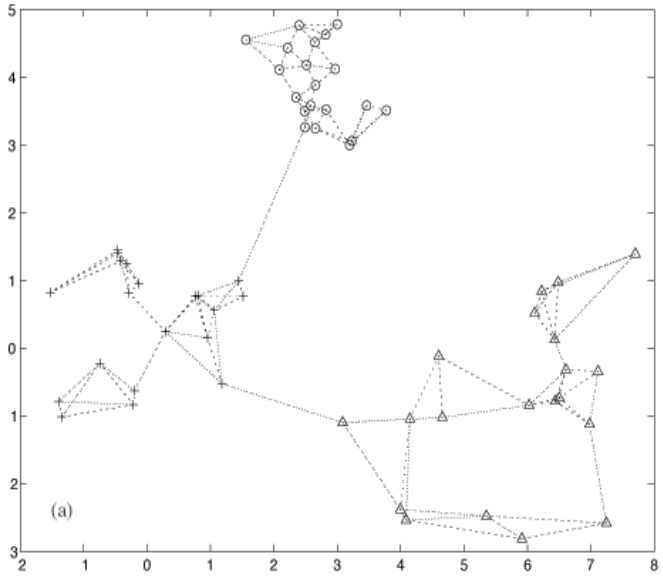
- Three basic models illustrating
- The many faces of the Laplacian matrix
- Leading to the computation of the Fiedler vector
  - The multiplicity of the eigenvalue 0 in the Laplacian matrix is equal to the number of connected components in the graph
  - Thus, small eigenvalues are indicative of two quasi-disconnected components

A first basic model



A first basic model

- Suppose we have a graph structure



A first basic model

- The elements a_{ij} of the adjacency matrix $\mathbf{A}$ of a weighted, undirected, graph are defined as

$$a_{ij} = \begin{cases} w_{ij} \text{ if node } i \text{ is connected to node } j \\ 0 \text{ otherwise} \end{cases}$$

  where $\mathbf{A}$ is symmetric

- The $w_{ij} \geq 0$ represents a similarity, an affinity or a strength of relationship between node i and node j

A first basic model

- We define a similarity index between partition C_1 and partition C_2 as

$$\text{sim}(C_1, C_2) = \sum_{\substack{i \in C_1 \\ j \in C_2}} w_{ij}$$

- Define $\mathbf{h}_1$ as an indicator vector containing 1 if the node belongs to C_1 and 0 if it belongs to C_2
- Define $\mathbf{h}_2$ as the equivalent vector for C_2
- Define $\mathbf{e}$ as a column vector made of 1s

A first basic model

- We obtain

$$\begin{aligned}
\mathrm{sim}(C_1, C_2) &= \sum_{\substack{i \in C_1 \\ j \in C_2}} w_{ij} = \sum_{\substack{i \in C_1 \\ j \in C_2}} a_{ij} \\
&= \sum_{i,j} \delta(i \in C_1) a_{ij} \delta(j \in C_2) \\
&= \sum_{i,j} [\mathbf{h}_1]_i a_{ij} [\mathbf{h}_2]_j \\
&= \mathbf{h}_1^{\mathrm{T}} \mathbf{A} \mathbf{h}_2
\end{aligned}$$

$$\mathbf{h}_1 + \mathbf{h}_2 = \mathbf{e}$$

A first basic model

- And thus

$$\begin{aligned}\mathrm{sim}(C_1, C_2) &= \mathbf{h}_1^{\mathrm{T}}\mathbf{A}(\mathbf{e} - \mathbf{h}_1) \\ &= \mathbf{h}_1^{\mathrm{T}}\mathbf{A}\mathbf{e} - \mathbf{h}_1^{\mathrm{T}}\mathbf{A}\mathbf{h}_1\end{aligned}$$

- Now,

$$\mathbf{h}_1^{\mathrm{T}}\mathbf{A}\mathbf{e} = \mathbf{h}_1^{\mathrm{T}}\mathbf{D}\mathbf{e} = \mathbf{h}_1^{\mathrm{T}}\mathbf{D}\mathbf{h}_1$$
$$\mathbf{D} = \mathrm{diag}(\mathbf{A}\mathbf{e})$$

- Thus $[\mathbf{D}]_{ij} = 0$ for $i \neq j$ and $[\mathbf{D}]_{ii} = \sum_{j=1}^{n} a_{ij}$

A first basic model

- We finally obtain

$$\begin{aligned}\mathrm{sim}(C_1, C_2) &= \mathbf{h}_1^{\mathrm{T}}\mathbf{A}\mathbf{e} - \mathbf{h}_1^{\mathrm{T}}\mathbf{A}\mathbf{h}_1 \\ &= \mathbf{h}_1^{\mathrm{T}}\mathbf{D}\mathbf{h}_1 - \mathbf{h}_1^{\mathrm{T}}\mathbf{A}\mathbf{h}_1 \\ &= \mathbf{h}_1^{\mathrm{T}}(\mathbf{D} - \mathbf{A})\mathbf{h}_1 \\ &= \mathbf{h}_1^{\mathrm{T}}\mathbf{L}\mathbf{h}_1\end{aligned}$$

- Since $\mathbf{L}$ is centered,
  - Notice that $\mathbf{h}_1$ is defined up to an additive constant $(\mathbf{h}_1 + \mathbf{c})$

A first basic model

- By relaxing the fact that $\mathbf{h}_1$ and $\mathbf{h}_2$ contain binary values,
  - This leads to the following optimization problem

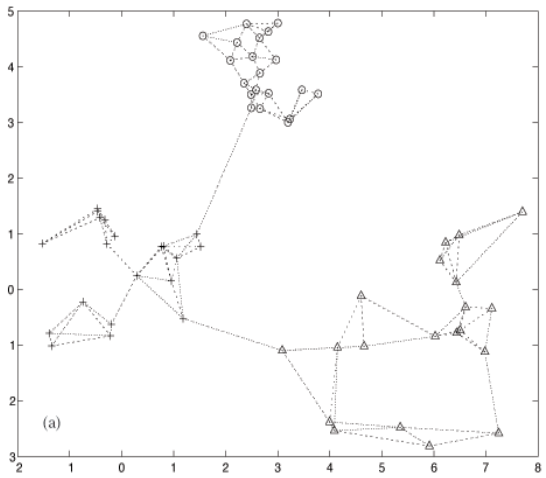
$$\min_{\mathbf{h}_1} (\mathbf{h}_1^{\mathrm{T}} \mathbf{L} \mathbf{h}_1) \text{ subject to } ||\mathbf{h}_1|| = 1$$

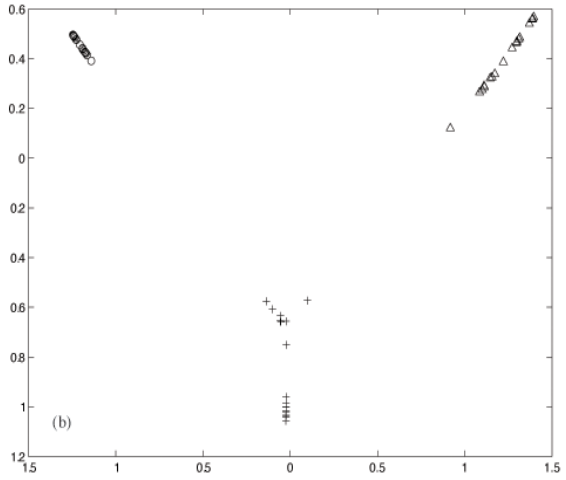
A first basic model

- This leads to an eigenvector problem
  - Compute the smallest non-trivial ($\mathbf{L}$ is not of full rank) eigenvector of

$$\mathbf{L}\mathbf{h}_1 = \lambda \mathbf{h}_1$$

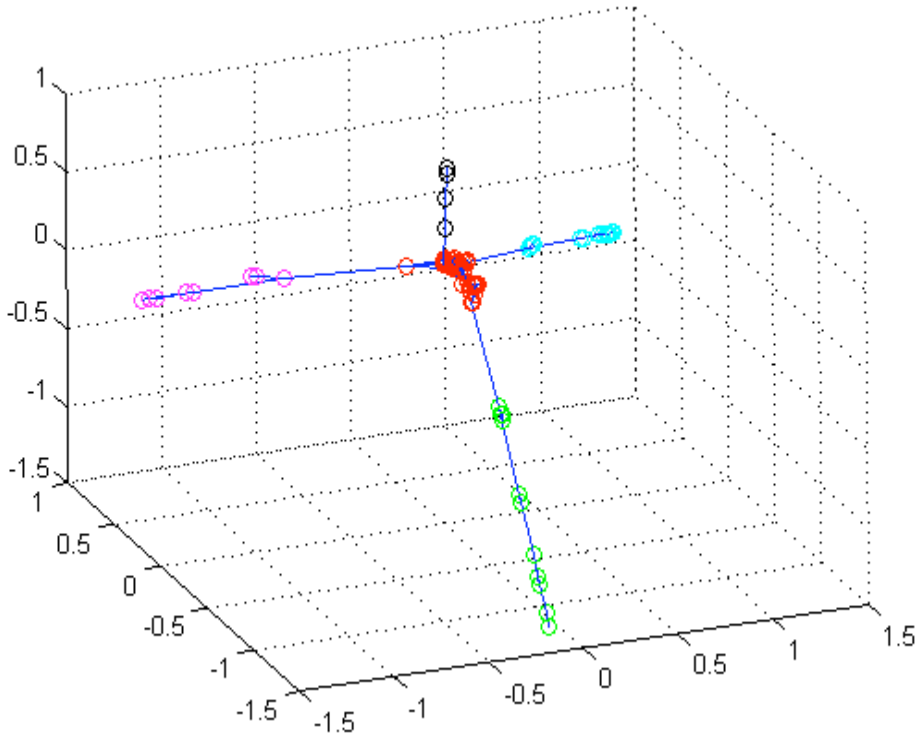
  - which is called the Fiedler vector



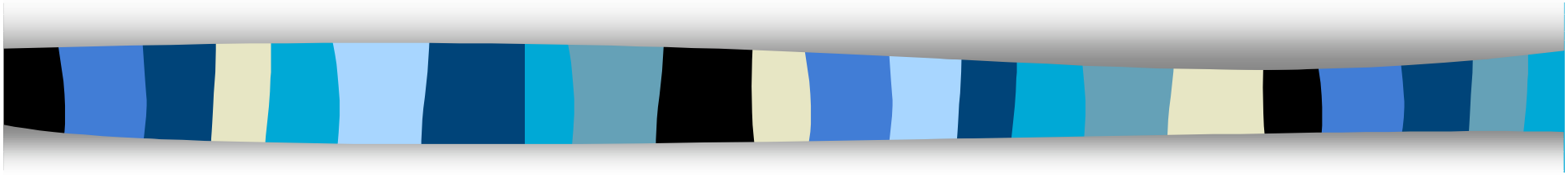


Graph partitioning: one example

- Visualization of a network of criminals



A second basic model



A second basic model

- Remember that

$$\Delta J = J(\text{after merge}) - J(\text{before merge})$$

$$\Delta J = \frac{n_k n_l}{n_k + n_l}(\mathbf{h}_k - \mathbf{h}_l)^{\mathrm{T}}\mathbf{K}(\mathbf{h}_k - \mathbf{h}_l)$$

- Here, we want to find the split that results in the
  - highest decrease in within-class inertia

A second basic model

- Now, define a new vector

$$\mathbf{h} = \sqrt{n_k n_l / (n_k + n_l)}(\mathbf{h}_k - \mathbf{h}_l)$$

  – Assuming that the kernel $\mathbf{K}$ is centered
  – Relaxing the fact that $\mathbf{h}$ has a special structure

- We obtain the following problem

$$\mathbf{h}^* = \max_{\mathbf{h}}(\mathbf{h}^T \mathbf{K} \mathbf{h}), \text{ subject to } \mathbf{h}^T \mathbf{e} = 0 \text{ and } ||\mathbf{h}||^2 = 1$$

A second basic model

- We obtain the eigensystem problem:
  - The partitioning vector $\mathbf{h}$ is the first eigenvector of $\mathbf{K}$

- If the kernel $\mathbf{K}$ is the commute-time kernel, $\mathbf{L}^+$, we obtain the Fiedler vector !

A second basic model

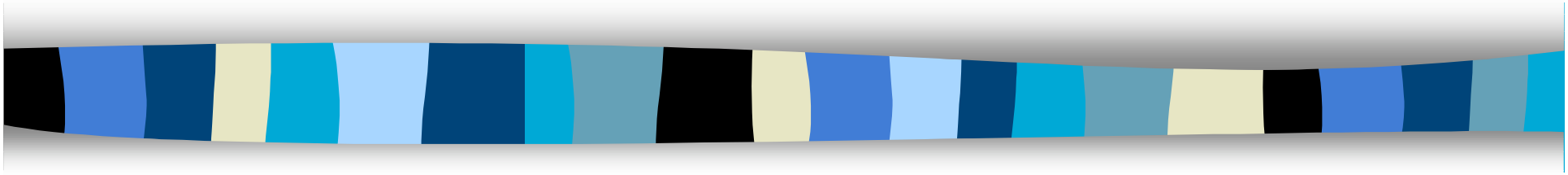
- Remember that

$$\Delta J = J(\text{after merge}) - J(\text{before merge})$$

$$\Delta J = \frac{n_k n_l}{n_k + n_l}(\mathbf{h}_k - \mathbf{h}_l)^{\mathrm{T}}\mathbf{K}(\mathbf{h}_k - \mathbf{h}_l)$$

- Here, we want to find the split that results in the
  - highest decrease in within-class inertia

A third basic model



A third basic model

- Let us consider we want to represent the nodes of the undirected graph
  - On a one-dimensional line (one-dimensional mapping)
  - Such that most similar nodes are near together
  - Each node i having coordinate z_i

A third basic model

- One criterion that can be used (Hall, 1970) is

$$
\begin{aligned}
C &= \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}(z_i - z_j)^2 \\
&= \sum_{i=1}^{n} a_{i.} z_i^2 - \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij} z_i z_j \\
&= \mathbf{z}^{\mathrm{T}}(\mathbf{D} - \mathbf{A})\mathbf{z} = \mathbf{z}^{\mathrm{T}}\mathbf{L}\mathbf{z}
\end{aligned}
$$

  - which has to be minimized

A third basic model

- If we impose the constraint $\mathbf{z}^{\mathrm{T}}\mathbf{z} = 1$,
- The solution is the smallest non-trivial eigenvector of the Laplacian matrix $\mathbf{L}$
- That is, the Fiedler vector !

A third basic model

- If, instead, we use $\mathbf{z}^{\mathrm{T}}\mathbf{D}\mathbf{z} = 1$,
  - Therefore penalizing the nodes with a large outdegree (see Zien et al., 1999)
- The solution is the smallest eigenvector of

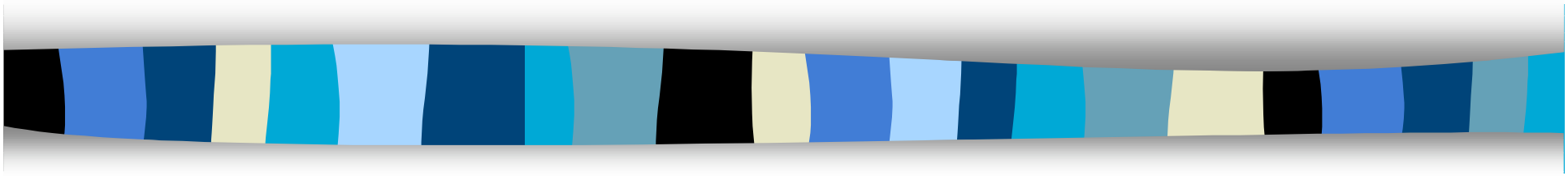
$$(\mathbf{I} - \mathbf{P})\mathbf{z} = \lambda\mathbf{z}$$

  - Where $\mathbf{P} = \mathbf{D}^{-1}\mathbf{A}$ is the associated transition probabilities matrix
- It corresponds to the largest, non-trivial, right eigenvector of $\mathbf{P}$

A third basic model

- This model corresponds exactly to the diffusion map of Nadler et al. (2005)
- As well as to the Laplacian eigenmap of Belkin & Nirogi (2001, 2003)

A fourth basic model



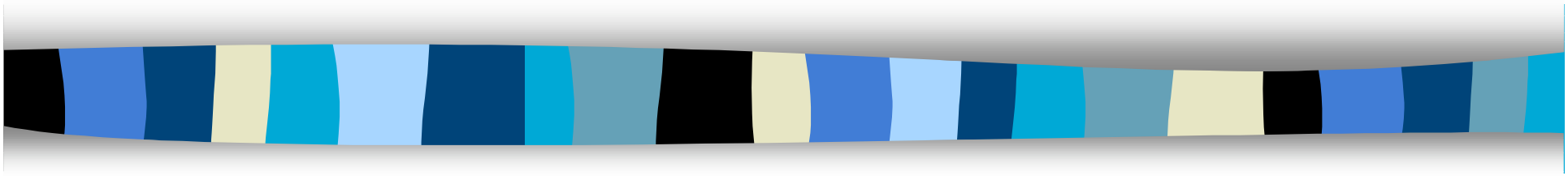
A fourth basic model

- Compute, as squared distance measure,
  - the commute-times (CT) between the nodes
- Performing a multidimensional scaling based on this distance matrix
  - That is, find the one-dimensional projection of the nodes
  - For which the CT distances between nodes are best preserved

A fourth basic model

- This aims to compute
  - The largest eigenvector of the pseudoinverse of the Laplacian matrix $\mathbf{L}^+$
  - Which corresponds to the smallest non-trivial eigenvector of $\mathbf{L}$
  - And thus to the Fiedler vector

2-way partitioning



2-way partitioning

- Various partitioning criterion have been defined
- The Ratio cut to be minimized

$$C_{Rcut} = \frac{\text{sim}(C_1, C_2)}{|C_1|} + \frac{\text{sim}(C_1, C_2)}{|C_2|}$$

$$= n\frac{\text{sim}(C_1, C_2)}{n_1 n_2}$$

2-way partitioning

- Spectral relaxation introduced by Hagen et al. (1992)
  - Naive relaxation of the problem
  - Aims to compute the smallest non-trivial eigenvector of the Laplacian matrix $\mathbf{L}$
  - Thus, computes the Fiedler vector

2-way partitioning

- Let us redefine the class membership vector
  - Into one single vector $\mathbf{x}$

$$\mathbf{x} = \mathbf{H}\mathbf{h}_1 = -\mathbf{H}\mathbf{h}_2$$

  - With $\mathbf{H}$ being the centering operator

$$\mathbf{H} = (\mathbf{I} - \frac{\mathbf{e}\mathbf{e}^{\mathrm{T}}}{n})$$

  - So that $\mathbf{x}^{\mathrm{T}}\mathbf{e} = 0$

2-way partitioning

- We therefore find

$$\mathbf{x} = q\mathbf{h}_1 - p\mathbf{h}_2$$

- With $p = \frac{n_1}{n}; q = \frac{n_2}{n}$

- Thus, by construction,

$$\mathbf{x}^{\mathrm{T}}\mathbf{e} = n_1 q - n_2 p = 0$$

2-way partitioning

- We easily show that:
  - If $i \in C_1$ and $j \in C_2$:

$$(x_i - x_j) = \frac{n_2}{n} - (-\frac{n_1}{n}) = 1$$

  - If $i \in C_1$ and $j \in C_1$:

$$(x_i - x_j) = 0$$

2-way partitioning

- Let us compute the square norm of $\mathbf{x}$

$$\begin{aligned} \|\mathbf{x}\|^2 &= n_1 \left(\frac{n_2}{n}\right)^2 + n_2 \left(\frac{n_1}{n}\right)^2 \\ &= \frac{n_1 n_2}{n^2}(n_1 + n_2) \\ &= \frac{n_1 n_2}{n} \end{aligned}$$

2-way partitioning

- The similarity between C_1 and C_2 can be rewritten as

$$\begin{aligned}
\text{sim}(C_1, C_2) &= \sum_{i \in C_1} \sum_{j \in C_2} a_{ij} \\
&= \sum_{i \in C_1} \sum_{j \in C_2} (x_i - x_j)^2 a_{ij} \\
&= \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} (x_i - x_j)^2 \\
&= \frac{1}{2} \mathbf{x}^{\mathrm{T}} \mathbf{L} \mathbf{x}
\end{aligned}$$

2-way partitioning

- Thus, the Ratio cut criterion can be rewritten as

$$C_{Rcut} = \frac{1}{2}\frac{\mathbf{x}^{\mathrm{T}}\mathbf{L}\mathbf{x}}{\|\mathbf{x}\|^2}$$

  - It thus aims to find the smallest non-trivial normalized eigenvector of the Laplacian matrix
  - That is, the Fiedler vector

2-way partitioning

- The normalized cut criterion
  - To be minimized

$$C_{Ncut} = \frac{\text{sim}(C_1, C_2)}{d_1} + \frac{\text{sim}(C_1, C_2)}{d_2}$$

$$= \frac{\text{sim}(C_1, C_2)}{\text{sim}(C_1, C_1) + \text{sim}(C_1, C_2)} + \frac{\text{sim}(C_1, C_2)}{\text{sim}(C_2, C_2) + \text{sim}(C_1, C_2)}$$

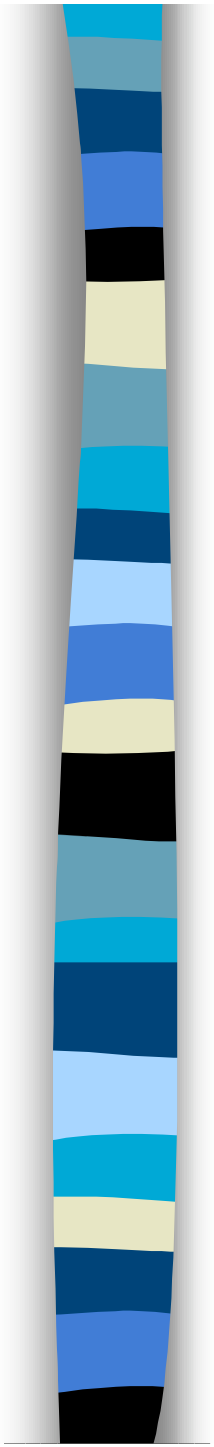
2-way partitioning

- By constraint relaxation,
  - it has been shown equivalent to the following eigenvector problem

$$(\mathbf{D} - \mathbf{A})\mathbf{x} = \lambda \mathbf{D}\mathbf{x}$$

- Which is equivalent to

$$(\mathbf{I} - \mathbf{P})\mathbf{z} = \lambda \mathbf{z}$$



2-way partitioning

- This model corresponds exactly to the diffusion map of Nadler et al. (2005)
- As well as to the Laplacian eigenmap of Belkin & Nirogi (2001, 2003)

2-way partitioning

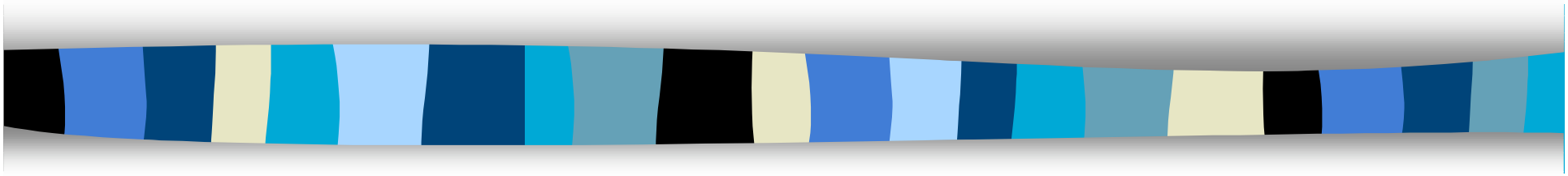
- The Min-max cut

$$C_{Mcut} = \frac{\text{sim}(C_1, C_2)}{\text{sim}(C_1, C_1)} + \frac{\text{sim}(C_1, C_2)}{\text{sim}(C_2, C_2)}$$

- By constraint relaxation,
  - once more, it has been shown equivalent to the following eigenvector problem

$$(\mathbf{D} - \mathbf{A})\mathbf{x} = \lambda \mathbf{D}\mathbf{x}$$

Link removal method

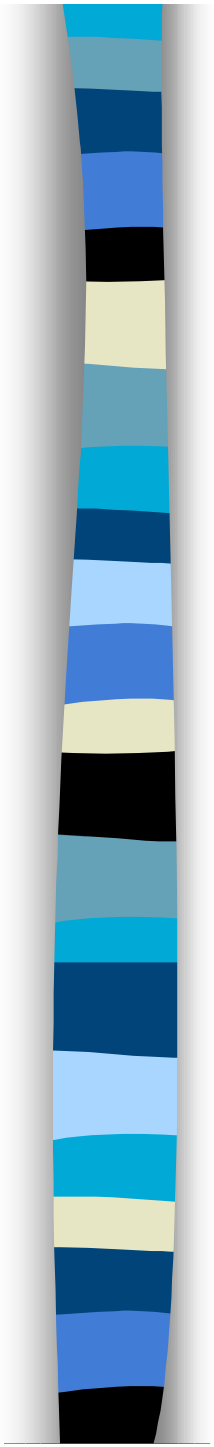


Link removal method

- Yet another method is based on greedy link removal (Girvan & Newman, 2002)
  - Based on shortest-path centrality
  - It is measured as the number of shortest paths between pairs of nodes
  - That pass through a certain link
  - Links with a large shortest-path centrality are progressively removed

Link removal method

- Thus, the method proceeds as follows
  1) Compute shortest-path centralities for all links
  2) Remove the link with the largest centrality
  3) Recalculate all link centralities
  4) Repeat from step 3 until the graph is split into two connected components



Thank you !!!