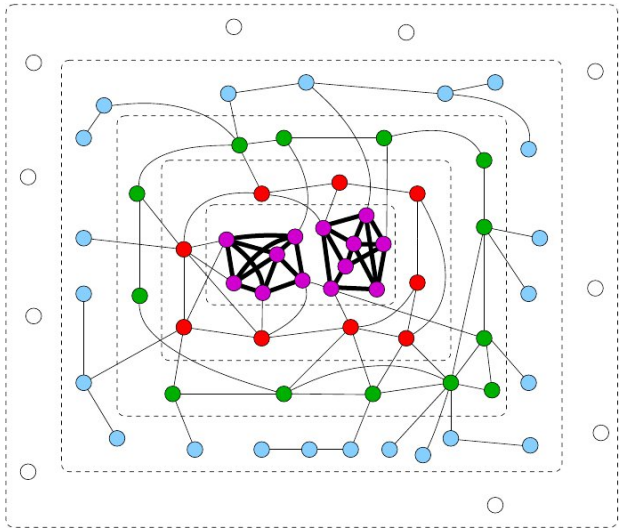
Graph Mining II

Community Detection – modularity based methods

Graph classification



Michalis Vazirgiannis

LIX @ Ecole Polytechnique

Outline

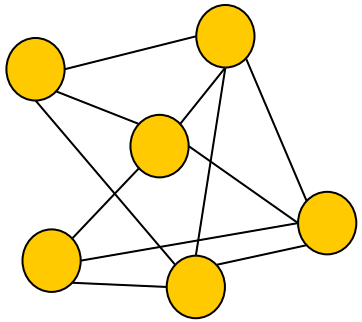
- **Modularity Based Methods**
- Louvain algorithm
- Graph kernels - classification

Basics

- Most of the community evaluation measures (e.g., conductance, cut-based measures), quantify the quality of a community based on
  - **Internal connectivity** (intra-community edges)
  - **External connectivity** (inter-community edges)
- **Question:** Is there any other way to distinguish groups of nodes with good community structure?
- **Random graphs** are not expected to present inherent community structure
- **Idea:** Compare the number of edges that lie **within a cluster** with the expected one in case of **random graphs** with the same degree distribution – **modularity measure**

Main idea

- **Modularity** function **[Newman and Girvan '04], [Newman '06]**
- Initially introduced as a measure for assessing the strength of communities
  - **Q = (fraction of edges within communities) –**
    (expected number of edges within communities)
- What is the **expected** number of edges?
- Consider a configuration model
  - **Random graph** model with the same degree distribution
  - Let P_{ij} = probability of an edge between nodes **i** and **j** with degrees k_i and k_j respectively
  - Then $P_{ij} = k_i k_j / 2m$, where $m = |E| = \frac{1}{2} \sum_i k_i$



Definition of modularity

- **Modularity Q**

$$Q = \frac{1}{2m} \sum_{ij} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \delta(C_i, C_j)$$

where

- **A** is the adjacency matrix
- k_i, k_j the degrees of nodes **i** and **j** respectively
- **m** is the number of edges
- C_i is the community of node **i**
- **δ(.)** is the Kronecker function: 1 if both nodes **i** and **j** belong on the same community ($C_i = C_j$), 0 otherwise

[Newman and Girvan '04], [Newman '06]

Properties of modularity

$$Q = \frac{1}{2m} \sum_{ij} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \delta(c_i, c_j)$$

- **Larger** modularity **Q** indicates **better** communities (more than random intra-cluster density)
  - The community structure would be better if the number of internal edges exceed the expected number
- Modularity value is always **smaller than 1**
- It can also take **negative values**
  - E.g., if each node is a community itself
  - No partitions with positive modularity → No community structure
  - Partitions with large negative modularity → Existence of subgraphs with small internal number of edges and large number of inter-community edges

[Newman and Girvan '04], [Newman '06], [Fortunato '10]

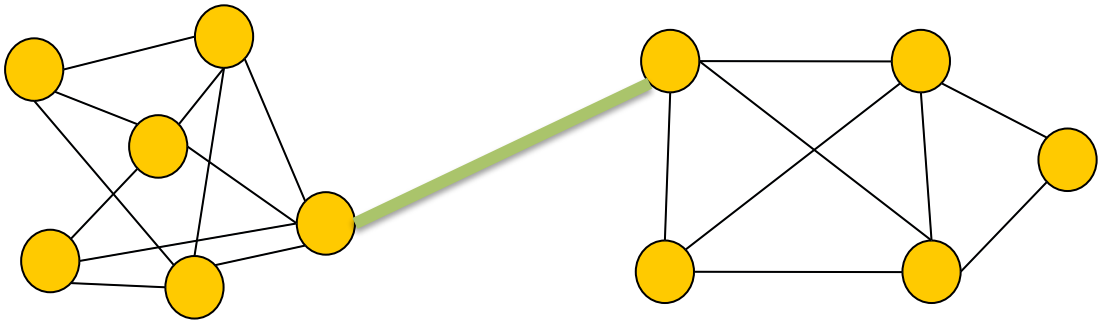
Applications of modularity

- Modularity can be applied:
  - As **quality function** in clustering algorithms
  - As **evaluation measure** for comparison of different partitions or algorithms
  - As a community detection tool itself
    - ❑ **Modularity optimization**
  - As criterion for reducing the size of a graph
    - ❑ Size reduction preserving modularity **[Arenas et al. '07]**

[Newman and Girvan '04], [Newman '06], [Fortunato '10]

Modularity-based community detection

- Modularity was first applied as a **stopping criterion** in the Newman-Girvan algorithm
- Newman-Girvan algorithm **[Newman and Girvan '04]**
  - A **divisive** algorithm (detect and remove edges that connect vertices of different communities)
  - **Idea:** try to identify the edges of the graph that are most between other vertices → responsible for connecting many node pairs
  - Select and remove edges based to the value of **betweenness centrality**
  - **Betweenness centrality:** number of **shortest paths** between every pair of nodes, that pass through an edge



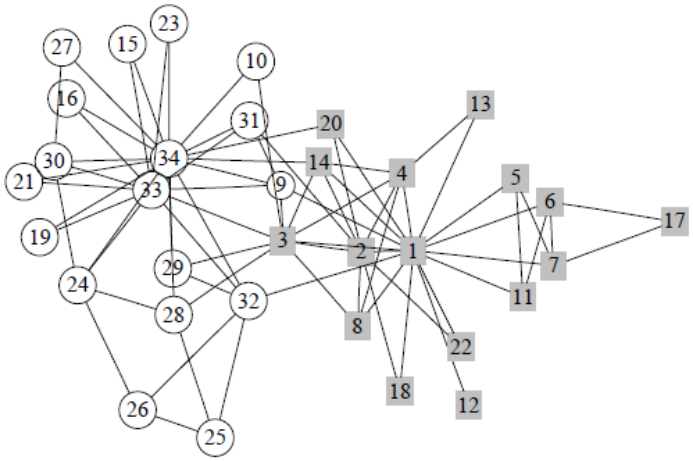
Edge betweenness is higher for edges that connect different communities

Newman-Girvan algorithm (1)

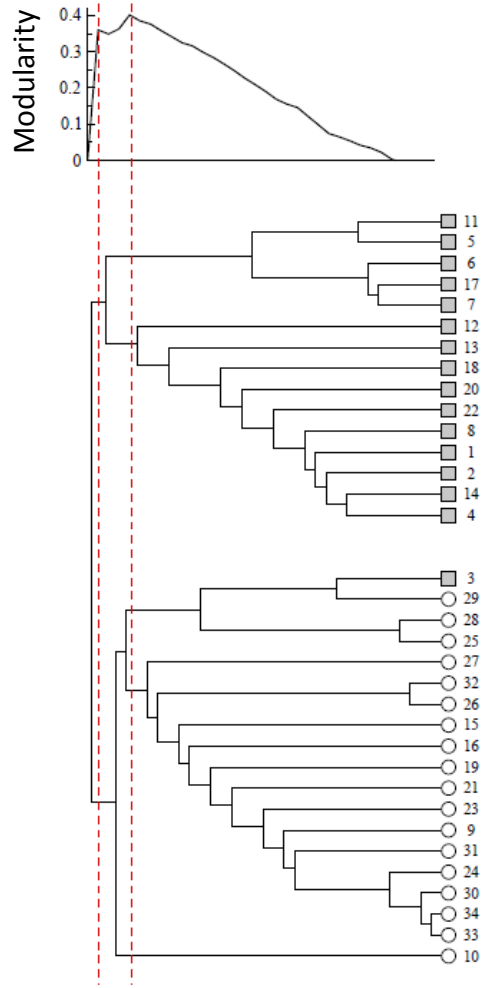
- **Basic steps:**
  1. Compute betweenness centrality for all edges in the graph
  2. Find and remove the edge with the highest score
  3. Recalculate betweenness centrality score for the remaining edges
  4. Go to step 2
- How do we know if the produced communities are **good ones** and stop the algorithm?
  - The output of the algorithm is in the form of a **dendrogram**
  - Use **modularity** as a criterion to cut the dendrogram and terminate the algorithm (Q ~= 0.3-0.7 indicates good partitions)
- Complexity: **$O(m^2n)$** (or **$O(n^3)$** on a sparse graph)

[Newman and Girvan '04], [Girvan and Newman '02]

Newman-Girvan algorithm (2)



Zachary's karate club



Community structure

[Newman and Girvan '04]

Modularity optimization

- High values of modularity indicate good quality of partitions
- **Goal:** find the partition that corresponds to the maximum value of modularity
- **Modularity maximization** problem
  - Computationally difficult problem **[Brandes et al. '06]**
  - Appoximation techniques and heuristics
- Four main categories of techniques
  1. Greedy techniques
  2. **Spectral optimization**
  3. Simulated annealing
  4. Extremal optimization

[Fortunato '10]

Spectral optimization (1)

- **Idea:** Spectral techniques for modularity optimization
- **Goal:** Assign the nodes into two communities, **X** and **Y**
- Let $s_i, \forall i \in V$ an indicator: **s_i = +1** if **i** assigned to **X** and **s_i = -1** if **i** assigned to **Y**

$$Q = \frac{1}{2m}\sum_{ij}\left(A_{ij} - \frac{k_i k_j}{2m}\right)\delta(C_i, C_j)$$

$$= \frac{1}{4m}\sum_{ij}\left(A_{ij} - \frac{k_i k_j}{2m}\right)(s_i s_j + 1)$$

$$= \frac{1}{4m}\sum_{ij} B_{ij} s_i s_j = \frac{1}{4m} s^T \mathrm{B} s$$

- **B** is the **modularity matrix**

$$B_{ij} = A_{ij} - \frac{k_i k_j}{2m}$$

[Newman '06], [Newman '06b]

Spectral optimization (2)

- Modularity matrix **B** $B_{ij} = A_{ij} - \frac{k_i k_j}{2m}$

- Vector **s** can be written as a linear combination of the eigenvectors $\mathbf{u_i}$ of the modularity matrix **B**

$$s = \sum_i a_i u_i \text{ where } \quad a_i = u_i^T s$$

- Modularity can now expressed as

$$Q = \frac{1}{4m} \sum_i a_i u_i^T \; B \; \sum_j a_j u_j^T = \frac{1}{4m} \sum_{i=1}^{n} \left(u_i^T s \right)^2 \beta_i$$

Where $\boldsymbol{\beta_i}$ is the eigenvalue of **B** corresponding to eigenvector $\mathbf{u_i}$

[Newman '06], [Newman '06b]

Spectral optimization (3)

- Spectral modularity optimization algorithm
  1. Consider the eigenvector **u_1** of **B** corresponding to the largest eigenvalue
  2. Assign nodes of the graph in one of the two communities **X** (**$s_i = +1$**) and **Y** (**$s_i = -1$**) based on the **signs** of the corresponding components of the eigenvector

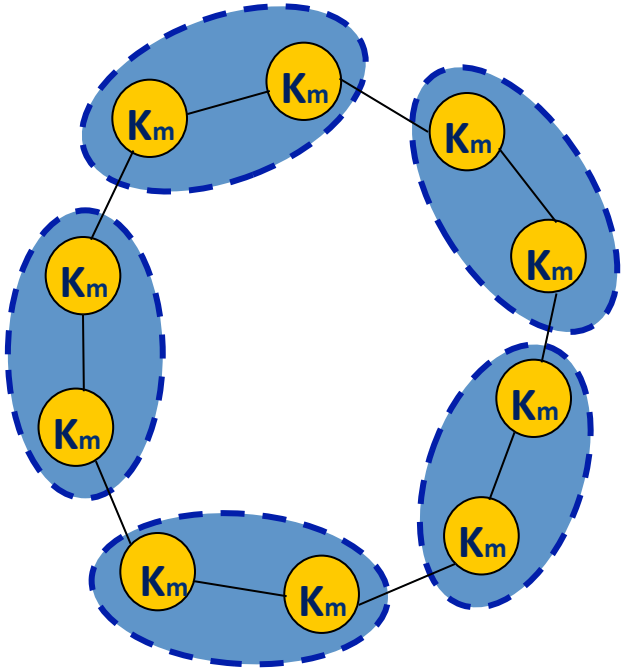
$$s_i = \begin{cases} 1 & if\ u_1(i) \geq 0 \\ -1 & if\ u_1(i) < 0 \end{cases}$$

  - More than two partitions?
  1. **Iteratively**, divide the produced partitions into two parts
  2. If at any step the split does not contribute to the modularity, leave the corresponding subgraph as is
  3. End when the entire graph has been splinted into no further divisible subgraphs
  - Complexity: **$O(n^2 \log n)$** for sparse graphs

[Newman '06], [Newman '06b]

Resolution limit of modularity

- **Resolution Limit** of modularity **[Fortunato and Barthelemy '07]**
- The method of modularity optimization may not detect communities with relatively small size, which depends on the total number of edges in the graph



- K_m are cliques with **m** edges (**$m \leq \sqrt{|E|}$**)
- K_m represent well-defined clusters
- However, the maximum modularity corresponds to clusters formed by two or more cliques
- It is difficult to know if the community returned by modularity optimization corresponds to a **single community** or a **union of smaller communities**

Outline

- Modularity Based Methods
- **Louvain algorithm**
- Graph kernels - classification

Louvain algorithm

- method to extract the community structure of networks
- heuristic method based on modularity optimization.
- Relatively low cost in terms of computation time.
- quality of the communities good, as measured by modularity.

The algorithm

- Assume a weighted undirected graph $G(E,V)$ and a splitting og G in $\{Ci\}$ communities. Then the modularity of the community splitting is:

$$Q = \frac{1}{2m}\sum_{ij}\left(A_{ij} - \frac{k_i k_j}{2m}\right)\delta\left(C_i, C_j\right)$$

- A_{ij} is the weight among nodes i and j, k_i is the sum of weight of the edges attached to vertex i, c_i the community to which i belongs and the δ function $\delta(u,v)$ is 1 if u-v and 0 otherwise. Finally $m = \frac{1}{2}\sum_{ij} A_{ij}$
- **Phase 1:**
- weighted graph. Create for each node i a community .
- For each node I consider all its neighbors j. For ach of them evaluate the gain in modularity if we place i in the community of j.
- The node i is placed in the community that maximizes the gain
- The process is repeated until there is no further gain for any reassignment.

$$\Delta Q = \left[\frac{\sum_{in} + k_{i,in}}{2m} - \left(\frac{\sum_{tot} + k_i}{2m}\right)^2\right] - \left[\frac{\sum_{in}}{2m} - \left(\frac{\sum_{tot}}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)^2\right]$$

Phase1: re-assignment of nodes – modularity gain

- Reassigning a node i to a cluster C incurs a gain in modularity:

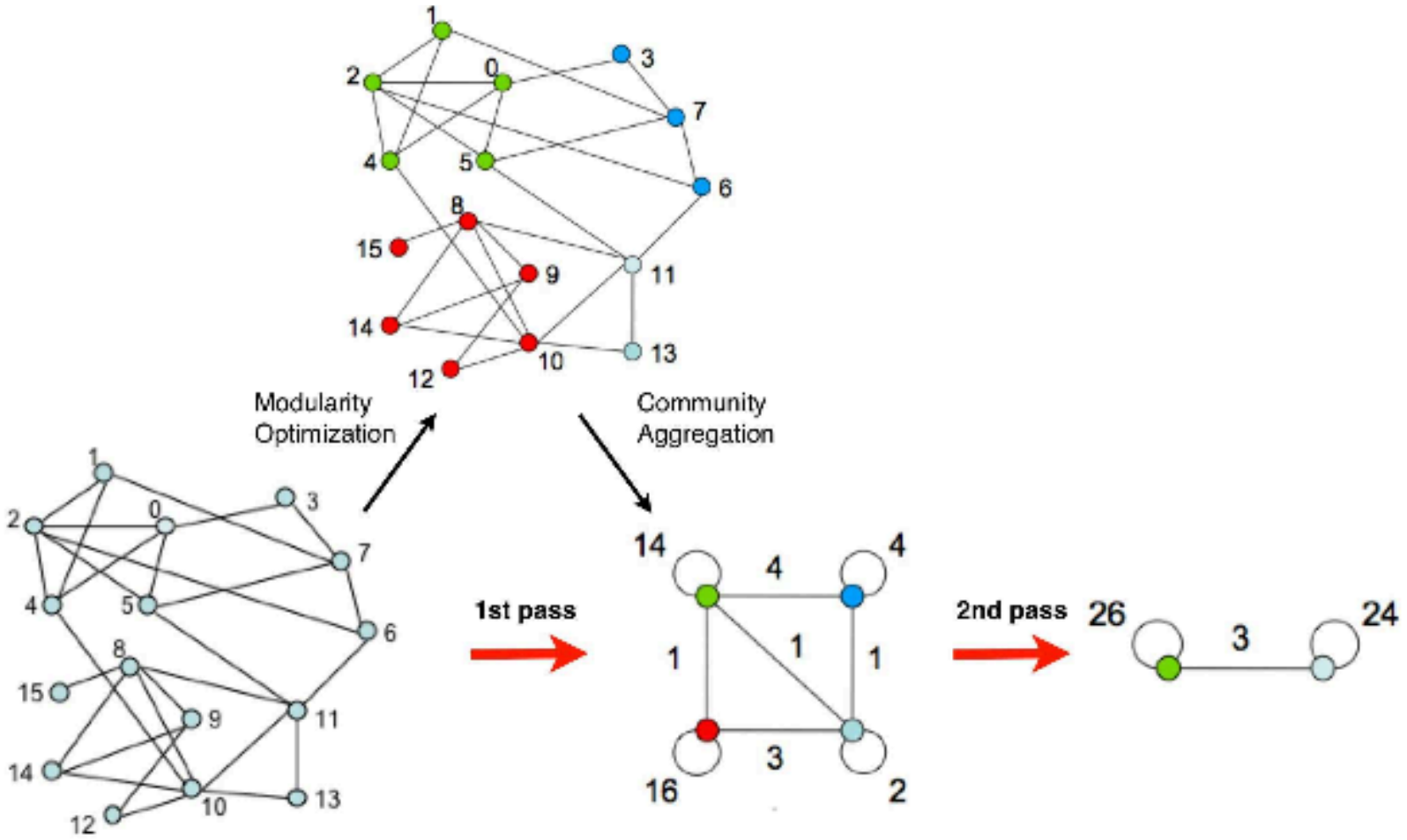
$$\Delta Q = \left[\frac{\sum_{in} + k_{i,in}}{2m} - \left(\frac{\sum_{tot} + k_i}{2m} \right)^2 \right] - \left[\frac{\sum_{in}}{2m} - \left(\frac{\sum_{tot}}{2m} \right)^2 - \left(\frac{k_i}{2m} \right)^2 \right]$$

$\sum_{in}$ sum of the weights of links in cluster C,
$\sum_{tot}$ sum of weights of the links incident to nodes in C ,
κ_i sum weights of links incident node i,
$\kappa_{i,in}$ sum weights of links from i to nodes in C
m sum of the weights of all the links in the network.

Phase 2: collapsing communities

- For each community c_i all nodes i in collapse in to one supernode c_i
- Inter community links weight: sum of the weights of the links among the communities nodes.
- Within community links collapse to a self link
- Go to phase 1

Iterative passes



Example

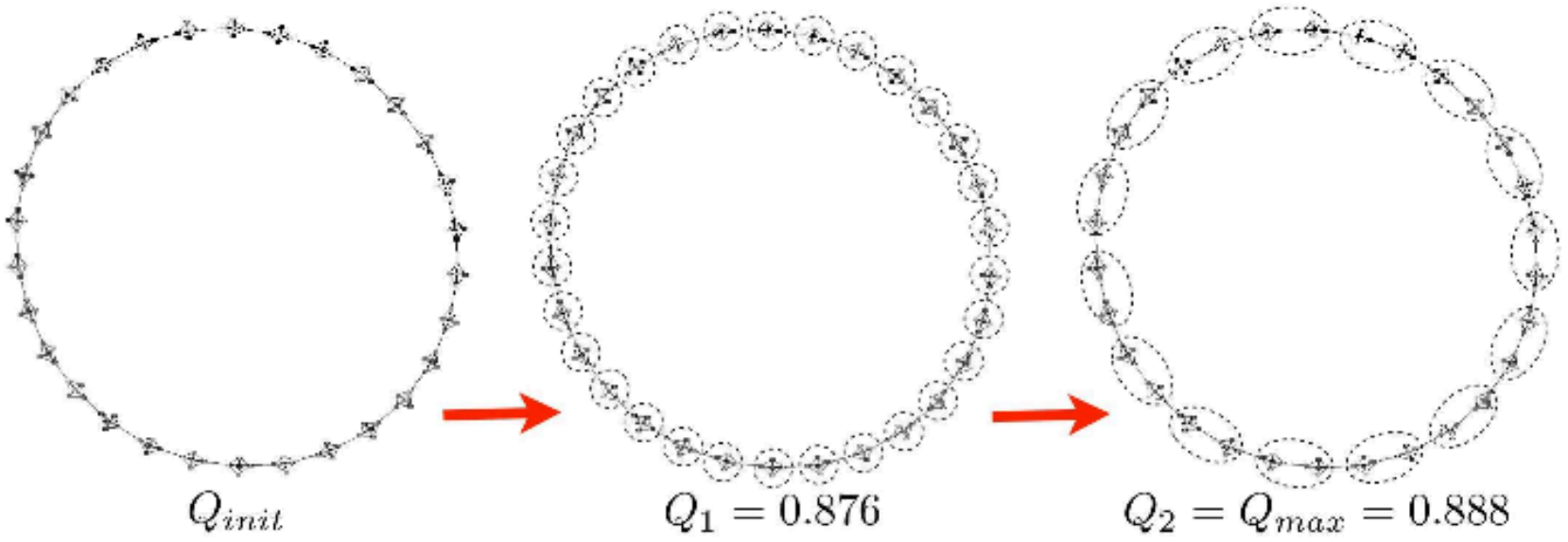


Figure 2. We have applied our method to the ring of 30 cliques discussed in [23]. The cliques are composed of 5 nodes and are inter-connected through single links. The first pass of the algorithm finds the natural partition of the network. The second pass finds the global maximum of modularity where cliques are combined into groups of two.

Performance considerations

Performance on large scale graphs (2008)

- sub-network of the .uk domain (39 M nodes, 783 M links
- Stanford WebBase crawler (118 million nodes and 1Bn links.
- Time: 12 minutes and 152 minutes respectively

Almost linear complexity for sparse data

- Number of communities decreases drastically after just a few passes
- Running time is concentrated on the first iterations.

Resolution limit problem modularity circumvented

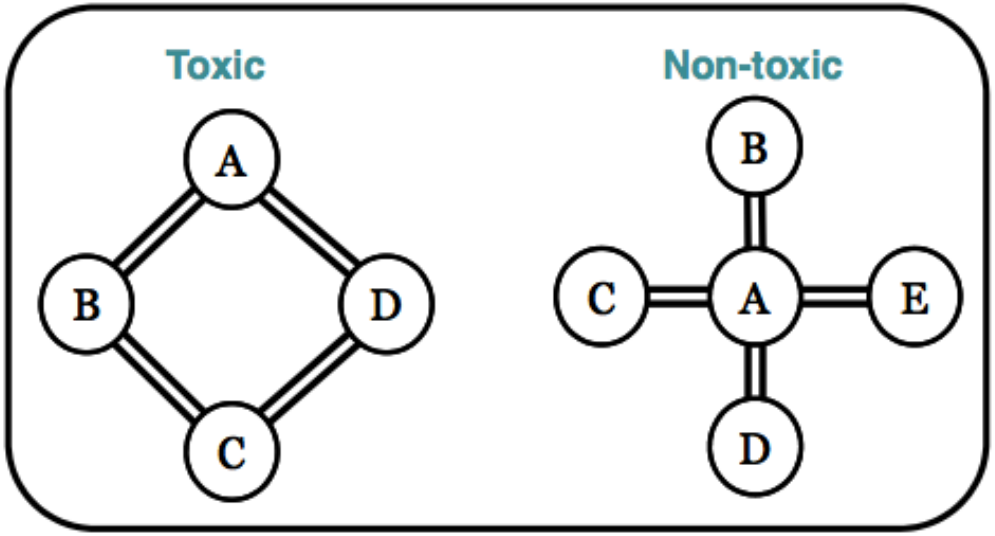
- intrinsic multi-level nature

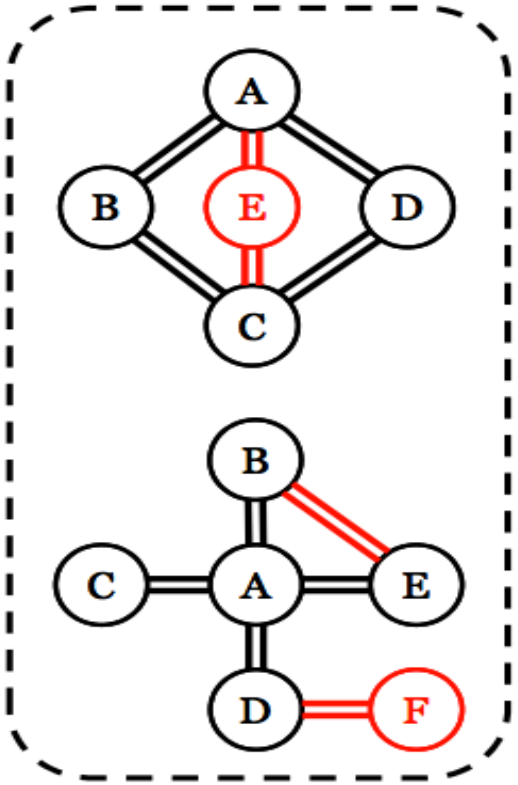
Outline

- Modularity Based Methods
- Louvain algorithm
- **Graph kernels - classification**

Graph classification

Chemical compounds





Known unknown=>

Predict toxic molecules assuming the known toxic ones

http://www.csc.ncsu.edu/faculty/samatova/practical-graph-mining-with-R/slides/pdf/Classification.pdf

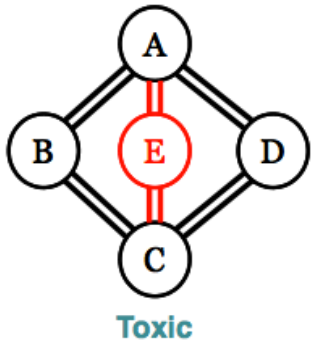
Graph classification

Graphs classification applications

- Chemical compounds
- Molecular structures
- Textual data – text categorization, opinion mining, sentiment analysis, metaphor detection …
- Social networks – community similarity

Classification based on graph structures

- Graph classification – i.e. recognize graph topologies
  - i.e. molecular graphs:



- Vertex classification – label prediction
  - i.e. predict in a social graph is the node is male/ female.

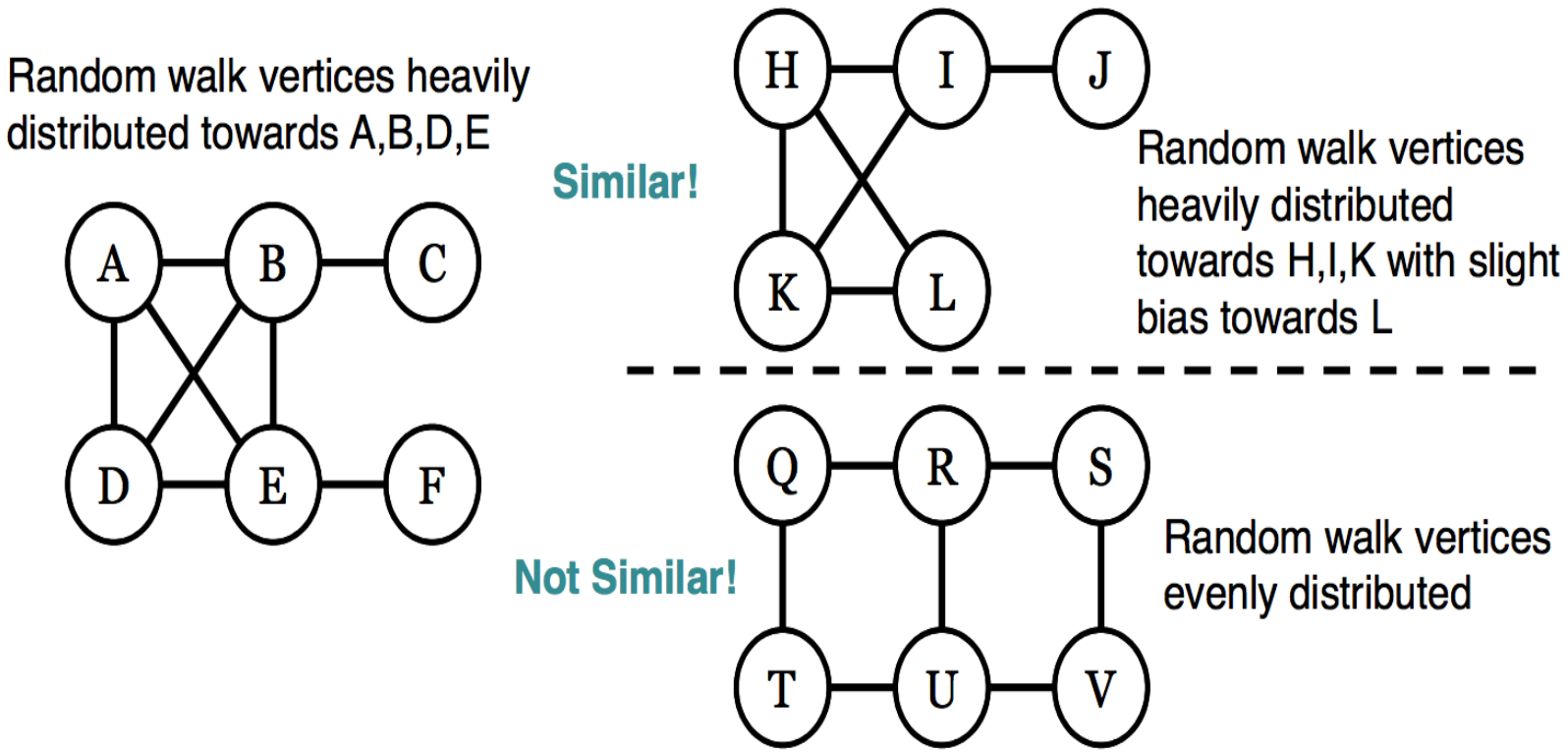
Graph Isomorphism

- $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic if
  - there exists a bijection(isomorphism) $f : V_1 \rightarrow V_2$ such that $(v_i , v_j) \in E_1$ if and only if $(f (v_i), f (v_j)) \in E_2$
- *Subgraph isompophism:*
- try sub-graph isomorphism between two graphs
- could be a potential solution to graph similarity
- the problem is NP-complete
- finding the largest isomorphic subgraphs is NP-hard

Graph kernels

- define a similarity measure between graphs computable in polynomial time
- Desired Properties of Graph Kernels
  - Symmetric, Semi-positive definite
  - Fast to compute
  - Expressive (captures topological similarity between graphs)
- Categories of Graph Kernels – based on:
  - walks and paths
  - limited-sized subgraphs
  - Sub tree patterns
  - others (based on graph edit distance etc.)

Random walk Graph kernels

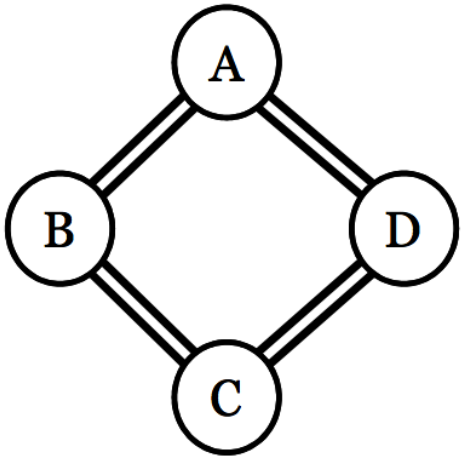


http://www.csc.ncsu.edu/faculty/samatova/practical-graph-mining-with-R/slides/pdf/Classification.pdf

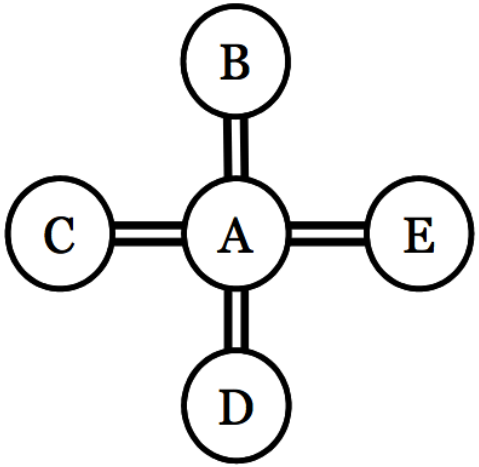
Product graph

- Assume two graphs $G_1(V_1, E_1)$, $G_2(V_2, E_2)$
- Direct Product notation: $G_x = G_1 x\ G_2$
- Direct product vertices: $V(G_x) = \{(a,b) \in (V_1 x V_2)\}$
- Direct product edges: $E(G_x) = \{(a,b),(c,d) \mid (a,c) \in E_1\ \&\ (b,d) \in E_2\}$

Graph Product - example



Type-A



Type-B

Type-A	A	B	C	D
A	0	1	1	0
B	1	0	0	1
C	1	0	0	1
D	0	1	1	0

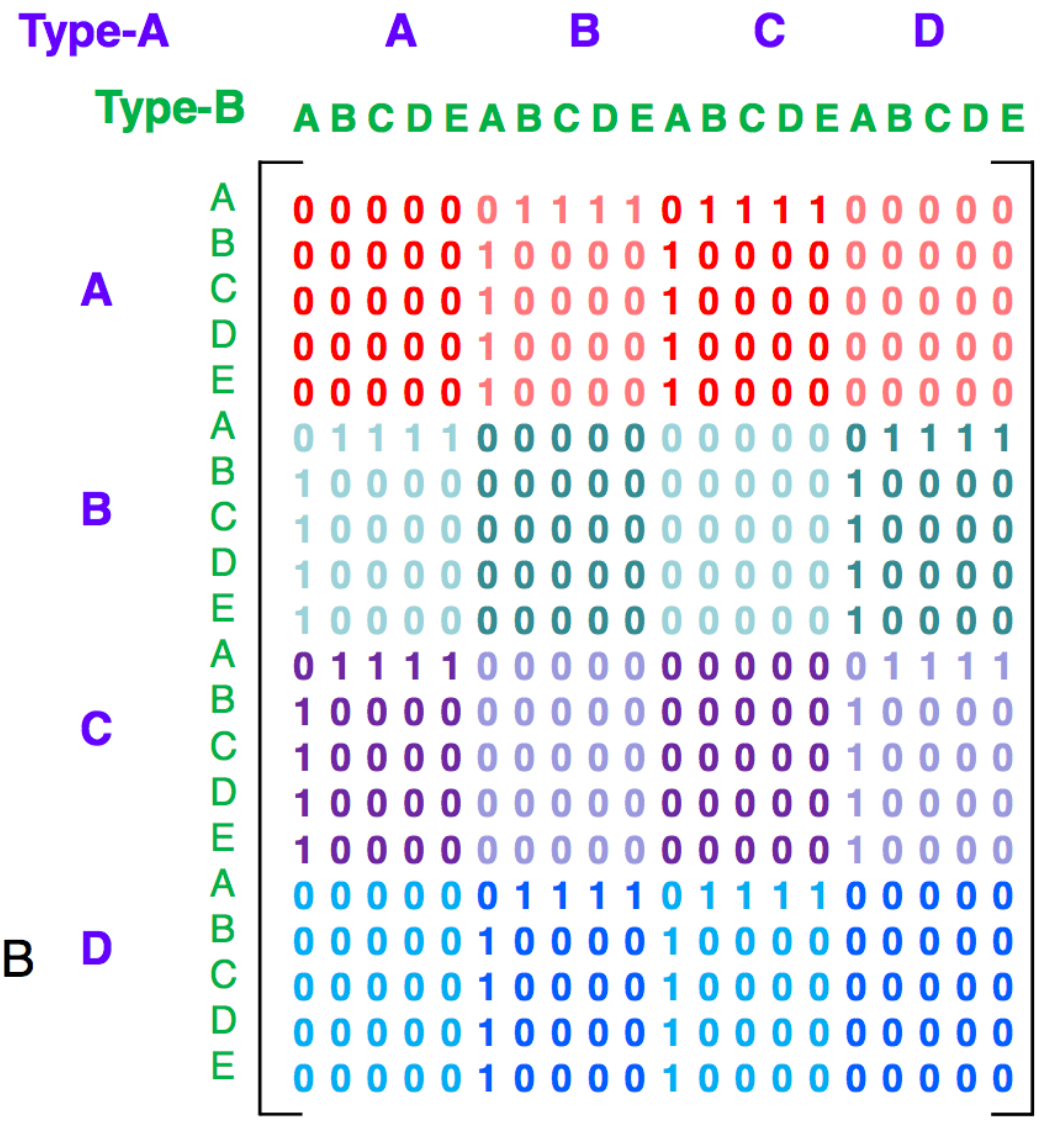
Type-B	A	B	C	D	E
A	0	1	1	1	1
B	1	0	0	0	0
C	1	0	0	0	0
D	1	0	0	0	0
E	1	0	0	0	0

Graph Product - example

Type-A	A	B	C	D
A	0	1	1	0
B	1	0	0	1
C	1	0	0	1
D	0	1	1	0

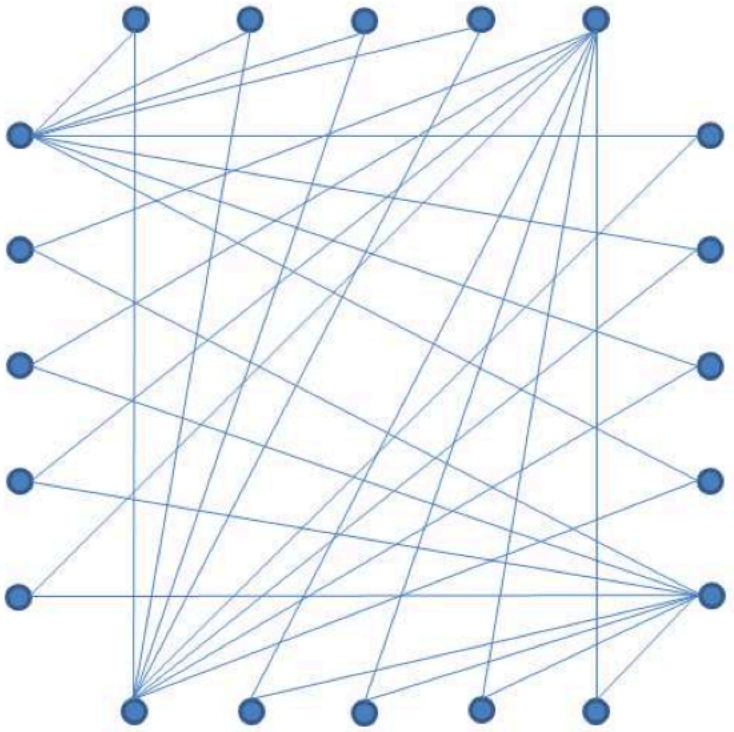
Type-B	A	B	C	D	E
A	0	1	1	1	1
B	1	0	0	0	0
C	1	0	0	0	0
D	1	0	0	0	0
E	1	0	0	0	0

Intuition: multiply each entry of Type-A by ***entire matrix*** of Type-B



Direct Product Kernel

- Compute direct product graph
- Decay constant $\gamma < 1/\min(d_i, d_o)$
  - d_i/d_o: max in/out degree in G_x
- Compute the weighted geometric series of walks (array A).
- Sum over all vertex pairs.



$$k(G_1, G_2) = \sum_{ij} \left(I - \frac{A_{ij}}{\gamma} \right)^{-1}$$

Random Walk Kernels - Preliminaries

- graph $G = (V,E)$, $V = \{v_1, v_2, \cdots, v_n\}$ ordered set of n vertices, $E \subset V \times V$ set of edges
- The adjacency matrix A of G is defined as $[A_{ij}] = \{1 \text{ if } (v_i, v_j) \in E, 0 \text{ otherwise}\}$
- A walk w on the graph is a sequence of nodes $w = (v_1, v_2, \cdots, v_n)$, $(v_i, v_{i+1},) \in E$

Random Walk Kernel

- Graphs with matching walks are similar
- Performing a random walk on the direct product graph G_x is equivalent to simultaneous random walk on the graphs G_1 and G_2

  - The A^k_X entry represents the probability of simultaneous length-k random walks on G and G'.
- Number of walks of length *k* from v_i to v_j can be computed by A^k_{ij}
- Discount longer walks by the factor of λ

How to compute common walks between two graphs?

- The product graph $G\times = (V_x, E_x)$ of G1 and G2:

  $V_x = \{(v,w) \in V_1 \times V_2 : l(v)=l(w)\}$, l(v): label of vertex v

  $E_x = \{((v_1,w_1),(v_2,w_2)) \in V_x^2 : (v1,v2) \in E1$

  and $(w_1,w_2) \in E_2$ and $l(v_1,w_1) = l(v_2,w_2)\}$
- Common walks can be thought of as walks within the product graph

Random Walk Kernel

- Random walk kernel between two graphs G1 and G2 is defined as

$$k_x\left(G_1, G_2\right) = \sum_{i,j=1}^{|V_x|} [\sum_{n=0}^{\infty} \lambda^n A_X^n]_{ij} = e^T (I - \lambda \mathrm{A}_x)^{-1} e$$

- where $e = (1,1,\cdots,1)^T$ start/termination prob distribution
- A_x, un-normalized adjacency matrix of the direct product graph.
- A_X^n represents similarity between simultaneous length *n* walks on G and G', measured via the kernel function κ

- Requires matrix inversion of $n^2 \times n^2$ matrix
- Time complexity $O(n^6)$
- Can be improved to $O(n^3)$ (Vishwanathan et al., 2010)
  - Employ Sylvester equation $M = SMT + M_0$ where $S,T,M_0 \in Rn \times n$ are given and we need to solve for $M \in Rn \times n$

Random walk kernels alternatives

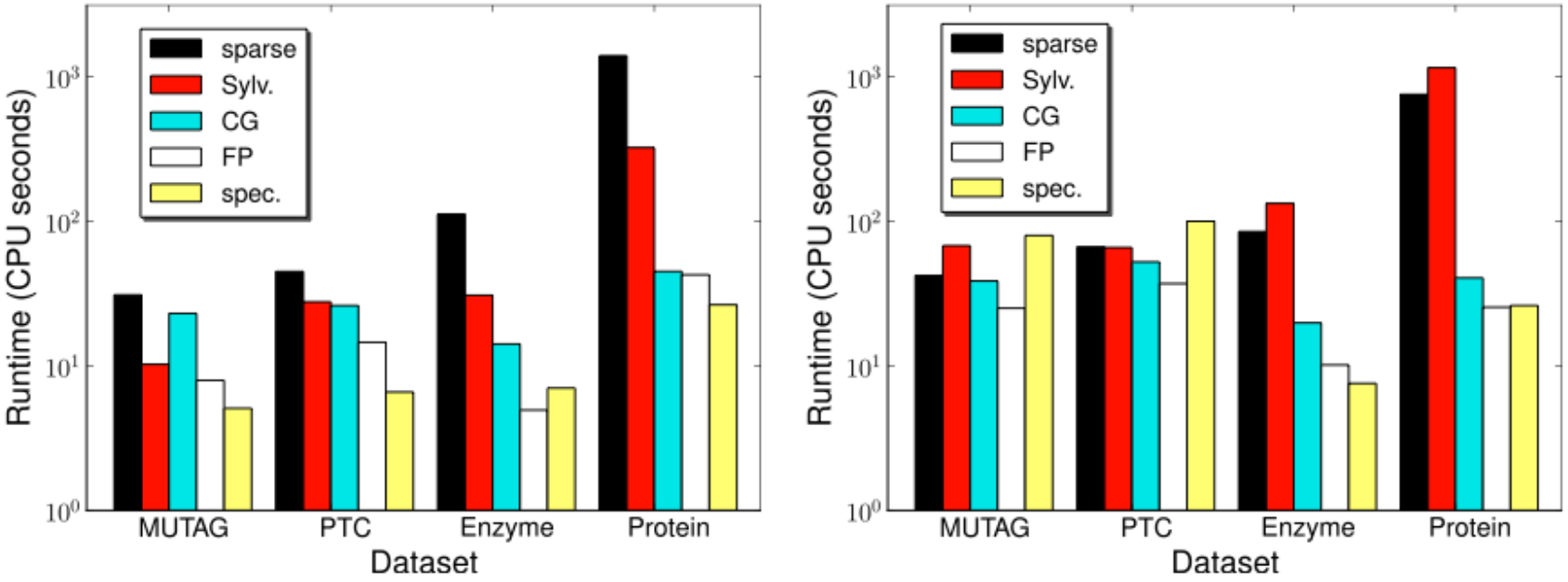


Figure 5: Time (in seconds on a log-scale) to compute 100×100 kernel matrix for unlabeled (left) *resp*. labeled (right) graphs from several data sets, comparing the conventional sparse method to our fast Sylvester equation, conjugate gradient (CG), fixed-point iteration (FP), and spectral approaches.

Kernel Matrix

- Assuming a set {G1, …Gn} graphs

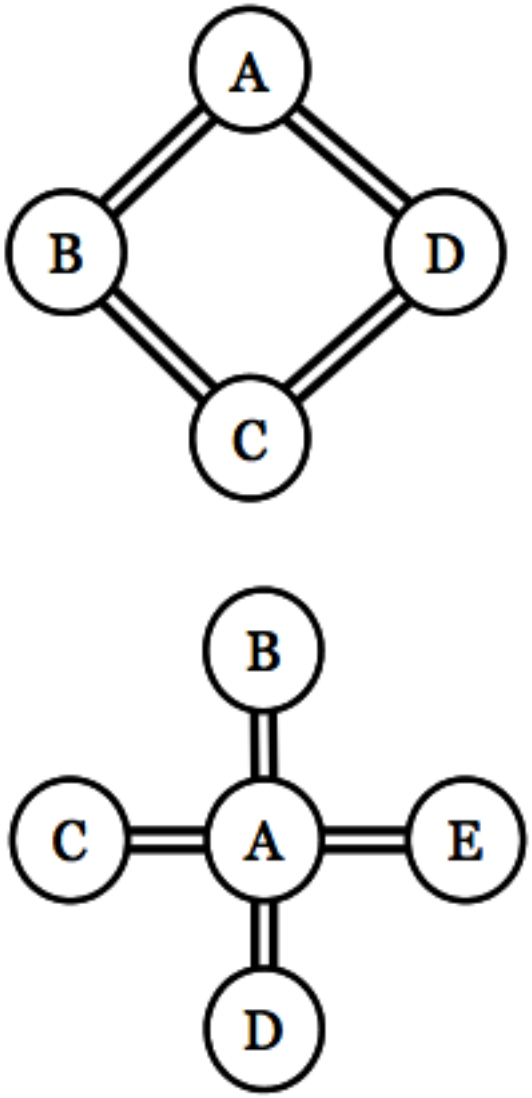
$$\begin{bmatrix} K(G_1,G_1), K(G_1,G_2), \dots, K(G_1,G_n) \\ K(G_2,G_1), K(G_2,G_2), \dots, K(G_2,G_n) \\ \dots \\ K(G_n,G_1), K(G_n,G_2), \dots, K(G_n,G_n) \end{bmatrix}$$

- This matrix is used as input to the classification learning method (i.e. SVM) function to learn the model.

Predicting toxic compounds

Assume the PTC dataset of molecules tested for positive or negative toxicit

```
# Learning pipeline (SVM ) in R
data("PTCData") # graph data
data("PTCLabels") # toxicity
information
# select 5 molecules to build model
on
sTrain= sample(1:length(PTCData),5)
PTCDataSmall<-PTCData[sTrain]
PTCLabelsSmall<-PTCLabels[sTrain]
# generate kernel matrix
K =
generateKernelMatrix(PTCDataSmall,
PTCDataSmall)
# create SVM model
model =ksvm(K, PTCLabelsSmall,
kernel='matrix')
```



References (modularity)

- M.E.J. Newman and M. Girvan. Finding and evaluating community structure in networks. Physical Review E 69(02), 2004.
- M.E.J. Newman. Modularity and community structure in networks. PNAS, 103(23), 2006.
- S.E. Schaeffer. Graph clustering. Computer Science Review 1(1), 2007.
- S. Fortunato. Community detection in graphs. Physics Reports 486 (3-5), 2010.
- M. Coscia, F. Giannotti, and D. Pedreschi. A classification for community discovery methods in complex networks. Statistical Analysis and Data Mining 4 (5), 2011.
- A. Arenas, J. Duch, A. Fernandez, and S. Gomez. Size reduction of complex networks preserving modularity. New J. Phys., 9(176), 2007.
- M. Girvan and M.E.J. Newman. Community structure in social and biological networks. PNAS 99(12), 2002.
- U. Brandes, D. Delling, M. Gaertler, R. Gorke, M. Hoefer, Z. Nikoloski, and D. Wagner. On Modularity Clustering. IEEE TKDE 20(2), 2008.
- M.E.J. Newman. Fast algorithm for detecting community structure in networks. Phys. Rev. E 69, 2004.
- A. Clauset, M.E.J. Newman, and C. Moore. Finding community structure in very large networks. Phys. Rev. E 70, 2004.

References (modularity)

- M.E.J. Newman. Finding community structure in networks using the eigenvectors of matrices. Phys. Rev. E 74, 2006.
- R. Guimera, M. Sales-Pardo, L.A.N. Amaral. Modularity from Fluctuations in Random Graphs and Complex Networks. Phys. Rev. E 70, 2004.
- J. Duch and A. Arenas. Community detection in complex networks using Extremal Optimization. Phys. Rev. E 72, 2005.
- A. Arenas, J. Duch, A. Fernandez, and S. Gomez. Size reduction of complex networks preserving modularity. New Journal of Physics 9(6), 2007.
- E.A. Leicht and M.E.J. Newman. Community structure in directed networks. Phys. Rev. Lett. 100, 2008.
- V. Nicosia, G. Mangioni, V. Carchiolo, and M. Malgeri. Extending the definition of modularity to directed graphs with overlapping communities. J. Stat. Mech. 03, 2009.
- S. Muff, F. Rao, A. Caflisch. Local modularity measure for network clusterizations. Phys. Rev. E, 72, 2005.
- S. Fortunato and M. Barthelemy. Resolution limit in community detection. PNAS 104(1), 2007.
- Finding community structure in very large networks, Aaron Clauset,M. E. J. Newman, and Cristopher Moore, http://arxiv.org/pdf/cond-mat/0408187v2.pdf
- Near linear time algorithm to detect community structures in large-scale networks, Phys. Rev. E 76, 036106 – Published 11 September 2007

References graph kernels

- P. Mahe, N. Ueda, T. Akutsu, J.-L. Perret, and J.-P. Vert, Extensions of Marginalized Graph Kernels, in Proceedings of the Twenty-first International Conference on Machine Learning, New York, NY, USA, 2004, p. 70-.
- N. Shervashidze, Scalable graph kernels, Dissertation, Universitt Tuebingen, 2012.
- J. Ramon and T. Grtner, Expressivity versus efficiency of graph kernels, in Proceedings of the First International Workshop on Mining Graphs, Trees and Sequences, 2003, pp. 65-74.
- S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt, Graph Kernels, Journal of Machine Learning Research, vol. 11, p. 1201-1242, Apr. 2010.