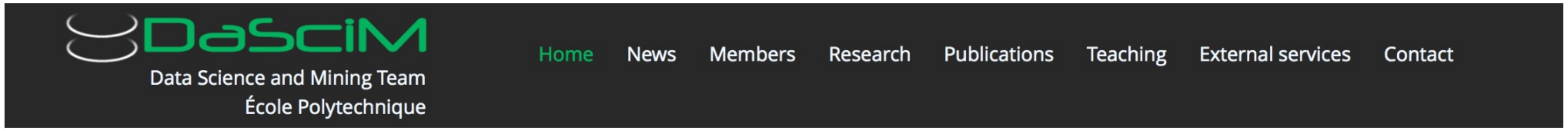


Data Science & Mining group

LIX @ Ecole Polytechnique

& AUEB

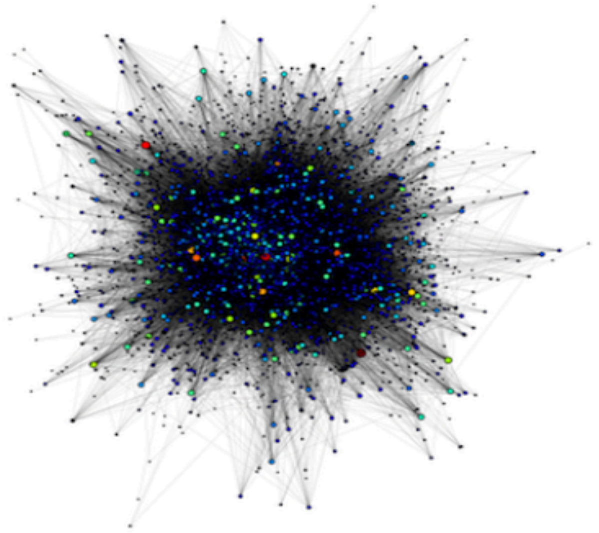
Graph Mining for fraud detection

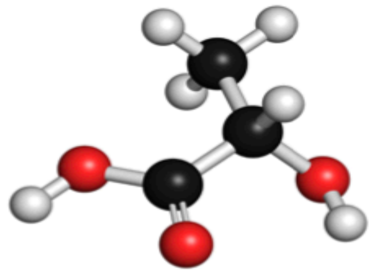
Michalis Vazirgiannis

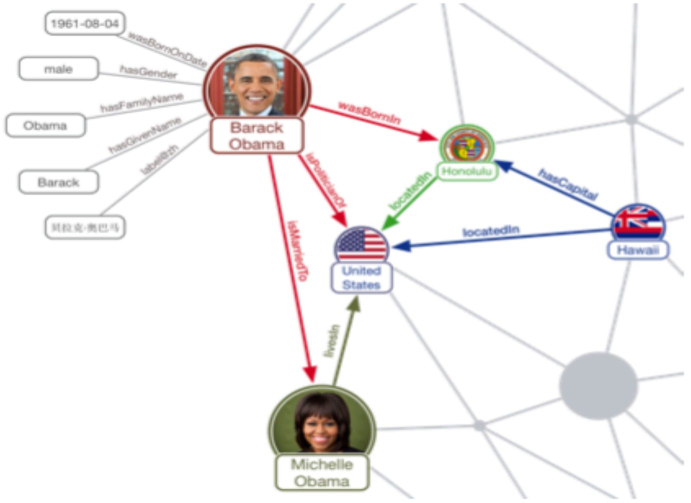
http://www.lix.polytechnique.fr/dascim/

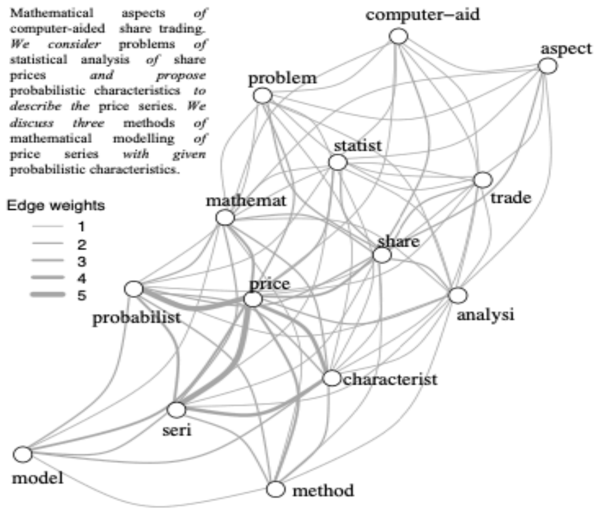
June 2020

Graphs are everywhere



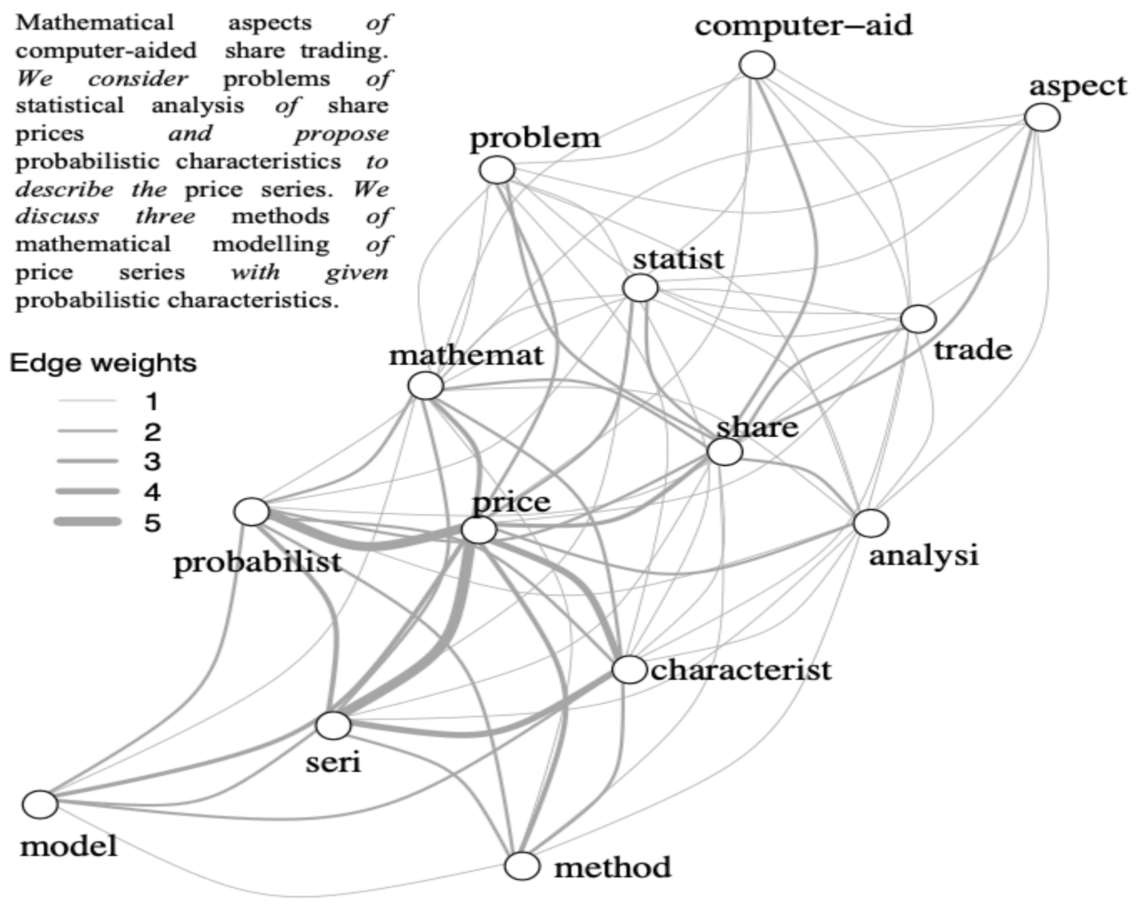






Mathematical aspects of computer-aided share trading. *We consider* problems *of* statistical analysis *of* share prices *and propose* probabilistic characteristics *to describe the* price series. *We discuss three* methods *of* mathematical modelling *of* price series *with given* probabilistic characteristics.

Motivation - Text Categorization



Mathematical aspects *of* computer-aided share trading. *We consider* problems *of* statistical analysis *of* share prices *and propose* probabilistic characteristics *to describe the* price series. *We discuss three* methods *of* mathematical modelling *of* price series *with given* probabilistic characteristics.

Given a text, create a graph where

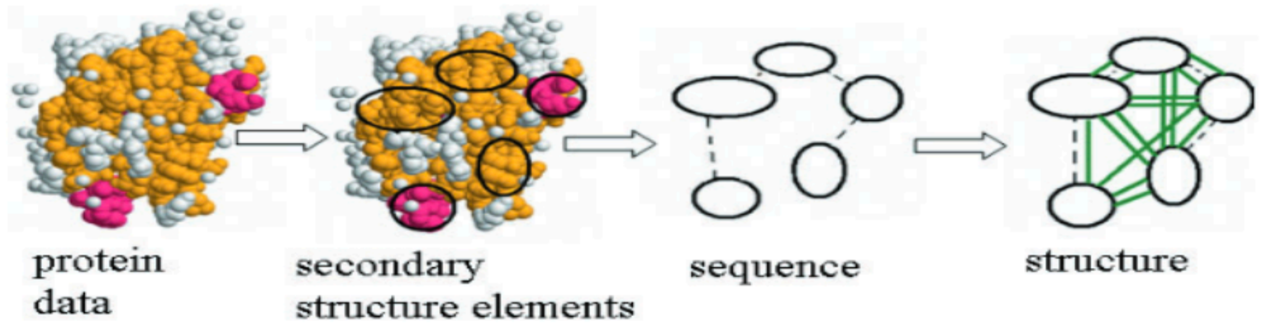
- vertices correpond to terms
- two terms are linked to each other if they co-occur within a fixed-size sliding window

Rousseau et al. "Text categorization as a graph classification problem.". ACL'15

Motivation - Protein Function Prediction

For each protein, create a graph that contains information about its

- structure
- sequence
- chemical properties



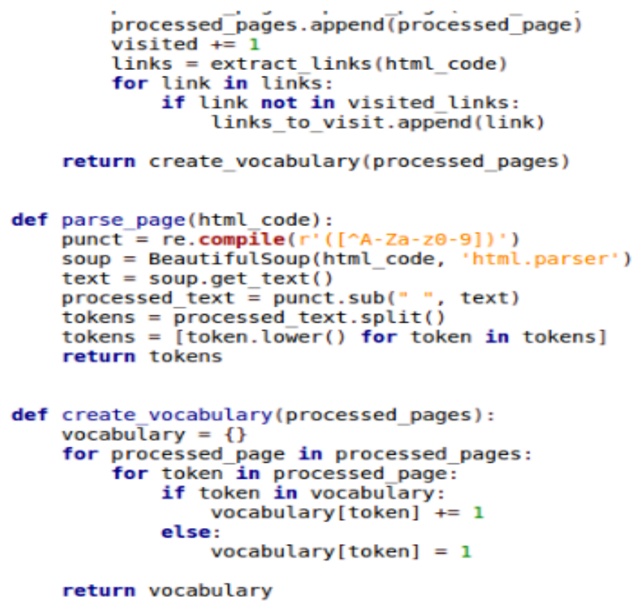
Use graph kernels to

- measure structural similarity between proteins
- predict the function of proteins

Borgwardt et al. "Protein function prediction via graph kernels". Bioinformatics 21

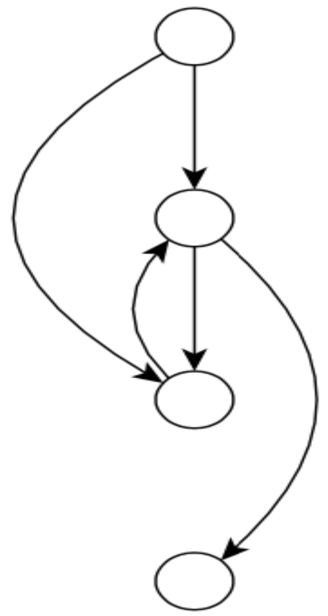
Motivation - Malware Detection

Given a computer program, create its control flow graph



```
        processed_pages.append(processed_page)
        visited += 1
        links = extract_links(html_code)
        for link in links:
            if link not in visited_links:
                links_to_visit.append(link)

    return create_vocabulary(processed_pages)


def parse_page(html_code):
    punct = re.compile(r'([^A-Za-z0-9])')
    soup = BeautifulSoup(html_code, 'html.parser')
    text = soup.get_text()
    processed_text = punct.sub(" ", text)
    tokens = processed_text.split()
    tokens = [token.lower() for token in tokens]
    return tokens


def create_vocabulary(processed_pages):
    vocabulary = {}
    for processed_page in processed_pages:
        for token in processed_page:
            if token in vocabulary:
                vocabulary[token] += 1
            else:
                vocabulary[token] = 1

    return vocabulary
```

$\rightarrow$



Compare the control flow graph of the problem against the set of control flow graphs of known malware

If it contains a subgraph isomporphic to these graphs $\rightarrow$ malicious code inside the program

Gascon et al. "Structural detection of android malware using embedded call graphs". In AISec'13

Machine Learning on Graphs

Node classification

- given a graph with labels on some nodes, provide a high quality labeling for the rest of the nodes

Graph clustering

- given a graph, group its vertices into clusters taking into account its edge structure in such a way that there are many edges within each cluster and relatively few between the clusters

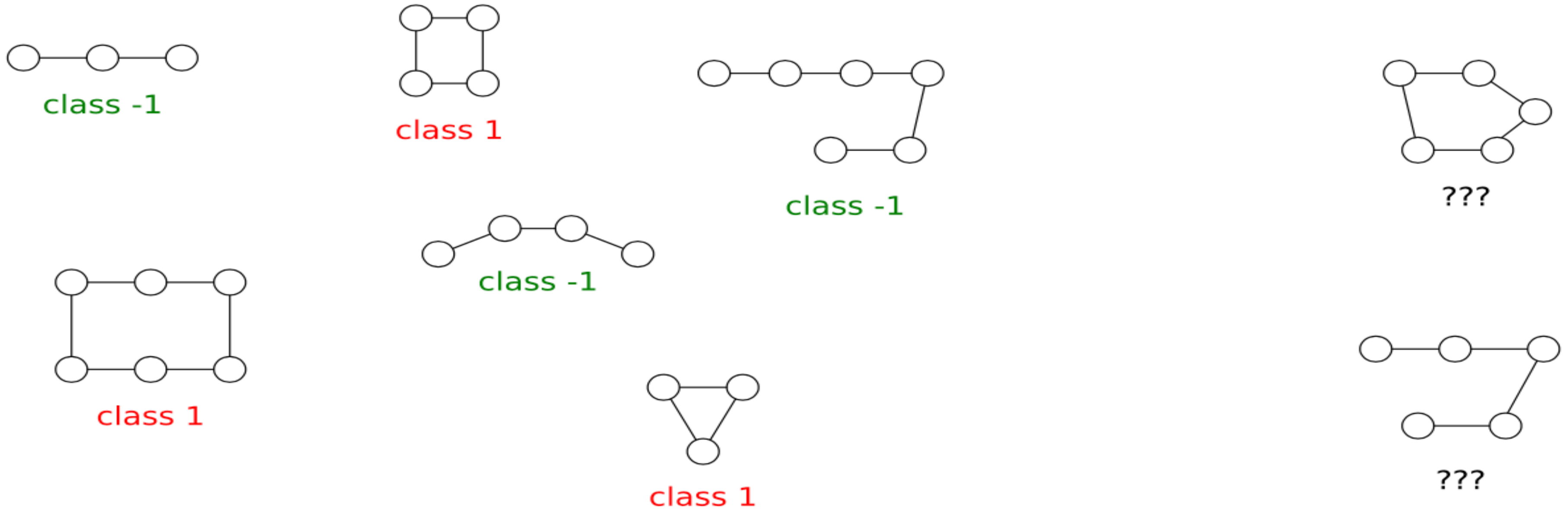
Link Prediction

- given a pair of vertices, predict if they should be linked with an edge

Graph classification

- given a set of graphs with known class labels for some of them, decide to which class the rest of the graphs belong

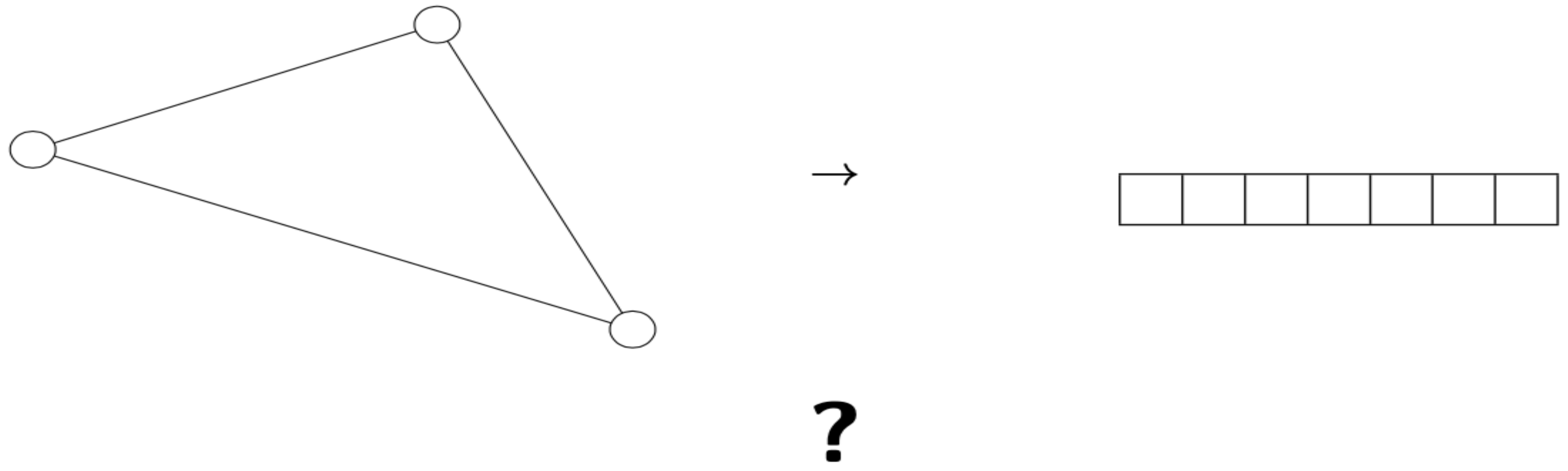
Graph Classification



- Input data $G \in \mathcal{X}$
- Output $y \in \{-1, 1\}$
- Training set $\mathcal{D} = \{(G_1, y_1), \dots, (G_n, y_n)\}$
- Goal: estimate a function $f : \mathcal{X} \to \mathbb{R}$ to predict y from $f(x)$

Graphs to vectors - kernels

- To analyze and extract knowledge from graphs, one needs to perform machine learning tasks
- Most machine learning algorithms require the input to be represented as a fixed-length feature vector
- There is no straightforward way to transform graphs to such a representation

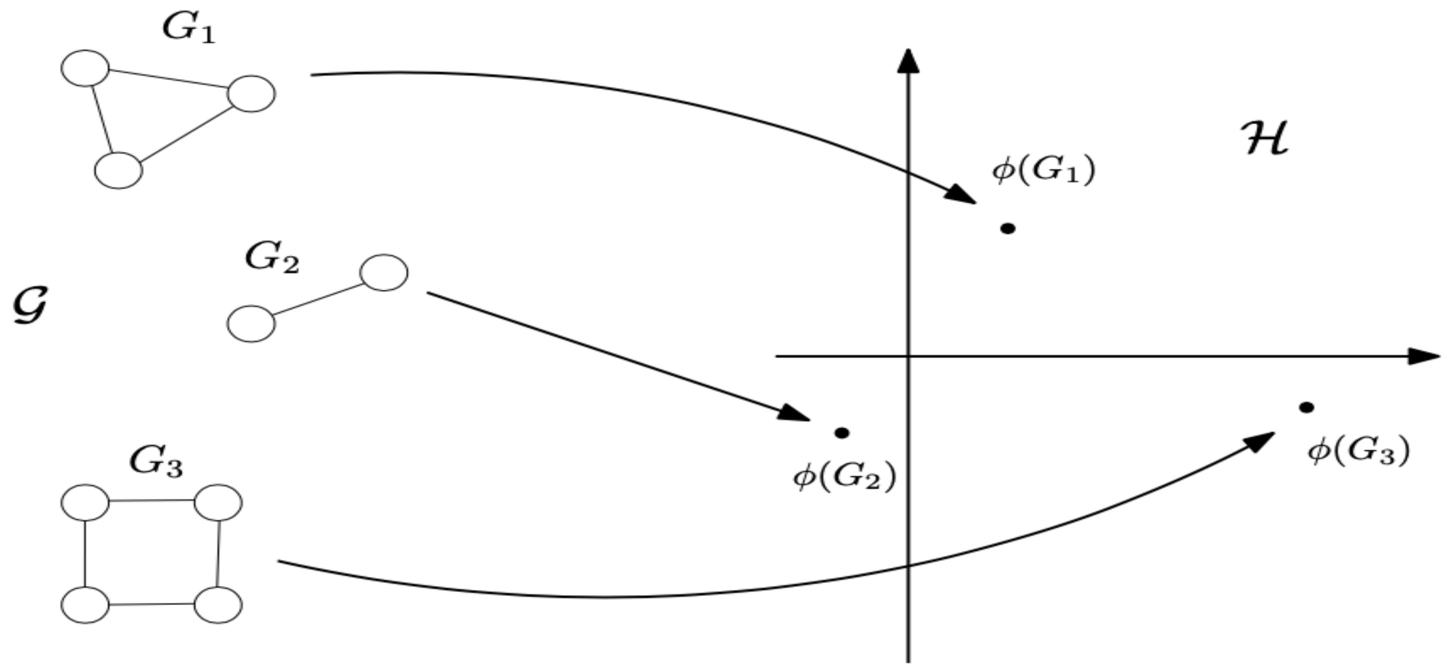


Graph Kernels

Definition (Graph Kernel)

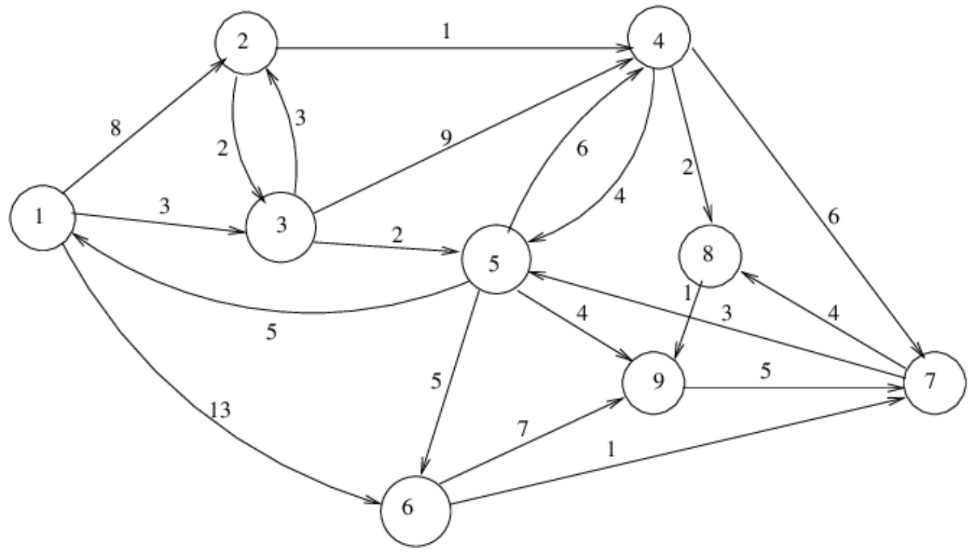
A graph kernel $k : \mathcal{G} \times \mathcal{G} \to \mathbb{R}$ is a kernel function over a set of graphs $\mathcal{G}$

- It is equivalent to an inner product of the embeddings $\phi : \mathcal{X} \to \mathcal{H}$ of a pair of graphs into a Hilbert space
- Makes the whole family of kernel methods applicable to graphs



Fraud in graph of payments

- graphs from transaction data from industry X
- nodes represent users
- edges the sum of transactions in a period of time.
- Supervised fraud communities of types t1, t2,t3
- Fraudulent nodes or communities have similar structural patterns



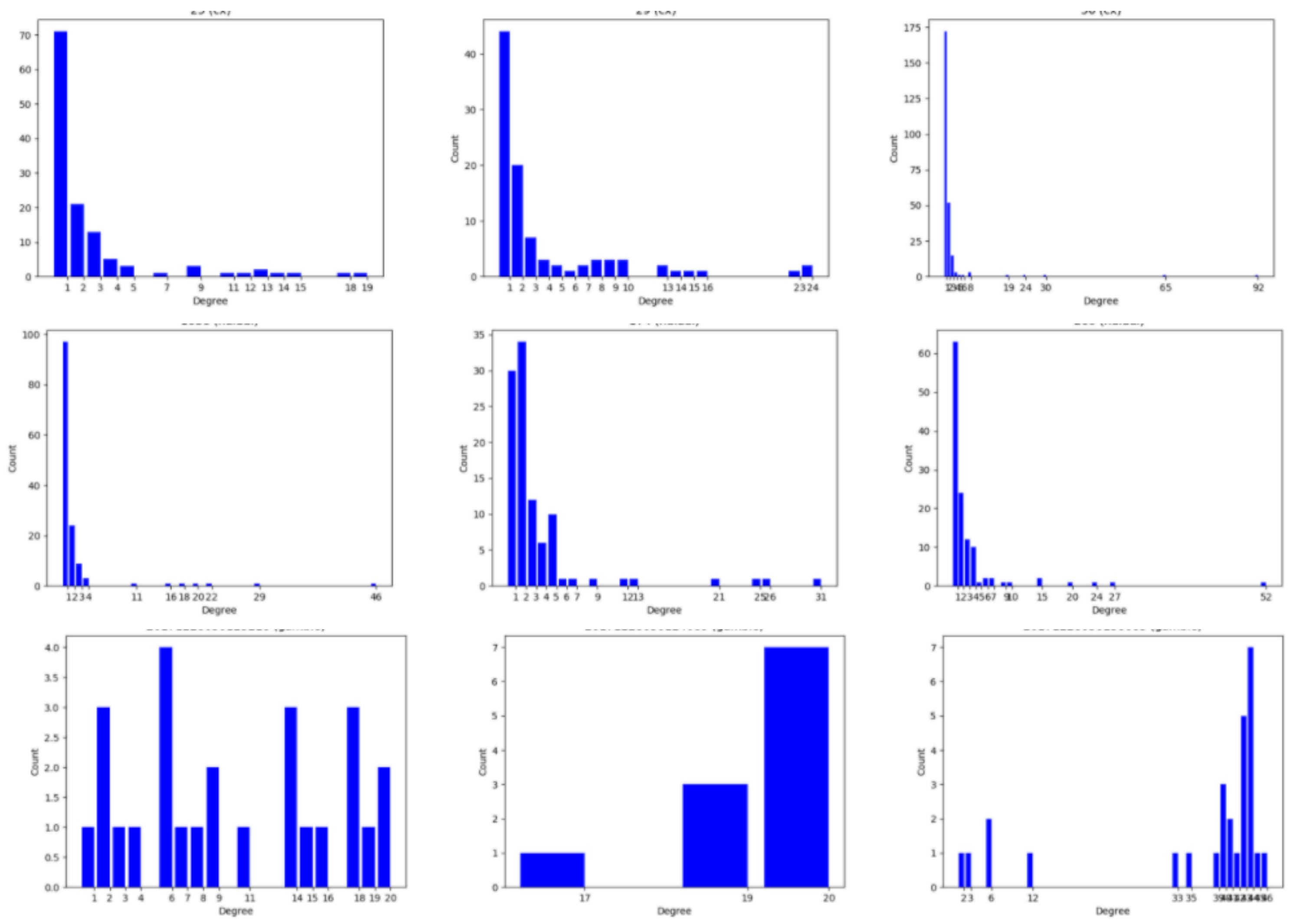
- Data set:
  - 32 fraud communities (6 t1, 6 t2, 20 t3) ~ 3K nodes
  - a sample of the user network (containing those communities) ~ 15M nodes, and ~22M edges.
  - Each fraud node in a fraud community is considered a ground-truth fraudster.
- Goal is to find (likely) fraudsters in the network.

Degree distribution in the three fraud classes

t1

t2

t3



Graph Kernel Based fraud exploration

- Fraudulent graphs: 32 directed graphs labeled as fraud with three different types: “t1”, “t2”, “t3”. Each node in every graph corresponds to a fraud account. edges between nodes represent transactions.
- two attributes: one is total transaction volume, another is the number of transactions.
- Normal graphs: 201 graphs randomly sampled from the complete network which are very unlikely to contain fraudsters.
- Capitalise on graph kernels for the similarity computations

Fraud in graphs

- Compared the four categories of graphs – t1,t2,t3 and a randomly sampled non-fraud - with respect xto a variety of graph metrics.

Observations:

- t1 and t2 are similar in terms of graph topology while t3 is quite different
- t3 graphs contain fewer nodes but are denser
- t3 subgraphs contain much fewer SCCs, as opposed to t1, t2 !
- money flow in t1, t2 graphs are mostly unidirectional (whereas in t3 it's multidirectional).
- However, in terms of transaction sums, t3 and t2 are actually more similar than t2 and t1!

Shortest path kernel for fraud graph similarity

Compares the length of shortest-paths of two graphs

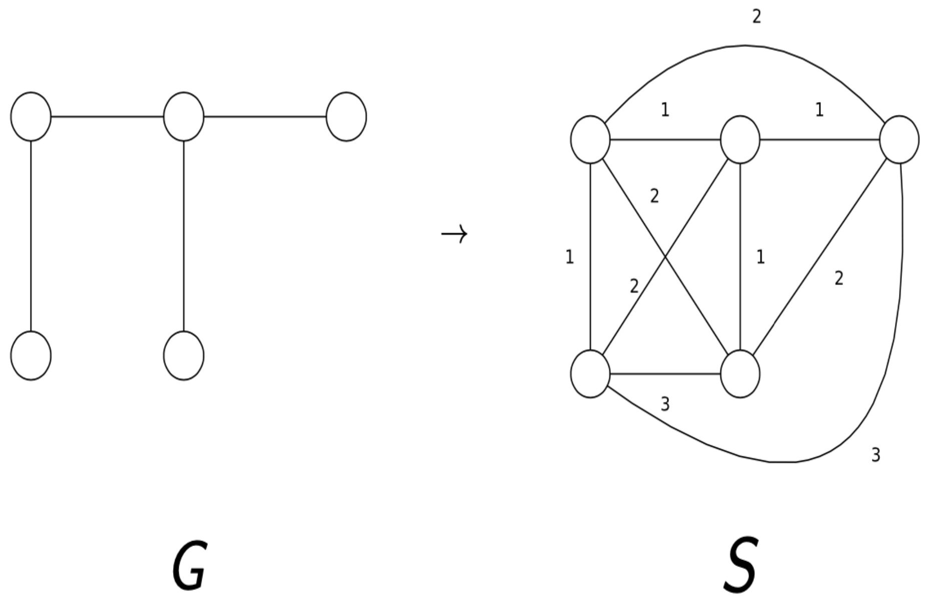
- and their endpoints in labeled graphs

Floyd-transformation

Transforms the original graphs into shortest-paths graphs

- Compute the shortest-paths between all pairs of vertices of the input graph G using some algorithm (i. e. Floyd-Warshall)
- Create a shortest-path graph S which contains the same set of nodes as th input graph G
- All nodes which are connected by a walk in G are linked with an edge in S
- Each edge in S is labeled by the shortest distance between its endpoints in G

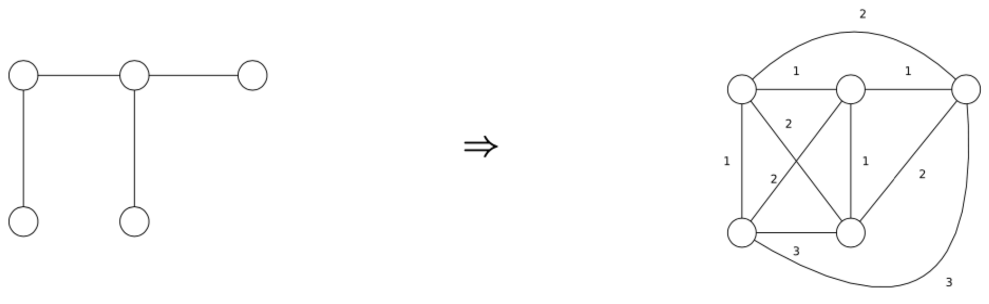
Floyd-transformation



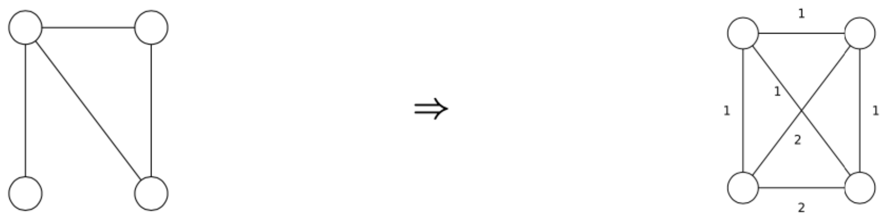
[Borgwardt and Kriegel. ICDM'05]

Shortest path kernel – an Example

Floyd-transformations



G_1 S_1



G_2 S_2

In S_1 we have:

- 4 edges with label 1
- 4 edges with label 2
- 2 edges with label 3

In S_2 we have:

- 4 edges with label 1
- 2 edges with label 2

Hence, the value of the kernel is:

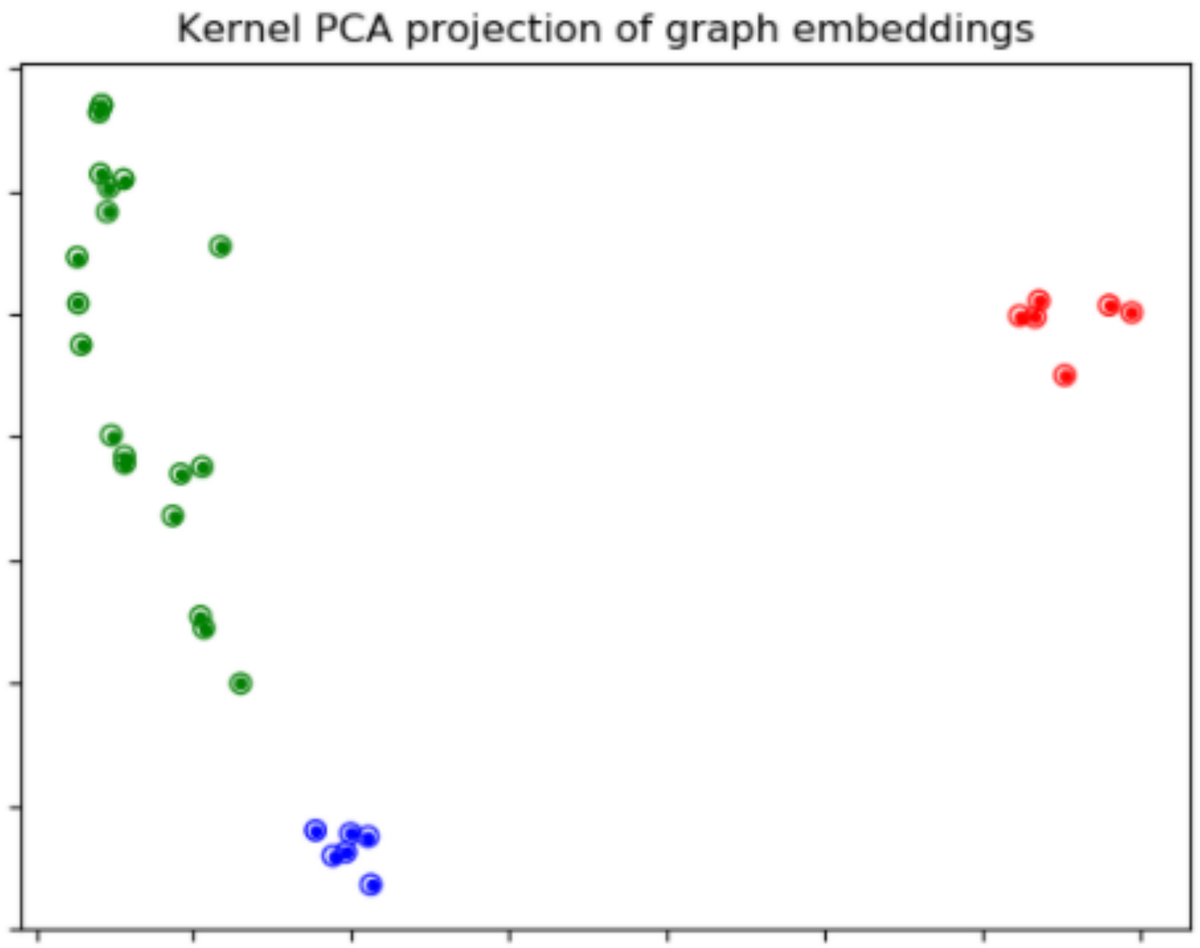
$$k(G_1, G_2) = \sum_{e_1 \in E_1} \sum_{e_2 \in E_2} k_{edge}(e_1, e_2) = 4 \cdot 4 + 4 \cdot 2 = 24$$

Fraudulent graph prediction

- We compute the kernel matrix of the graphs calculated by shortest-path kernel.
- With kernel PCA, we are able to extract the principle components (2 in our case) of them by simply using a precomputed similarity matrix without knowing the actual embeddings.
- classification is done by SVM

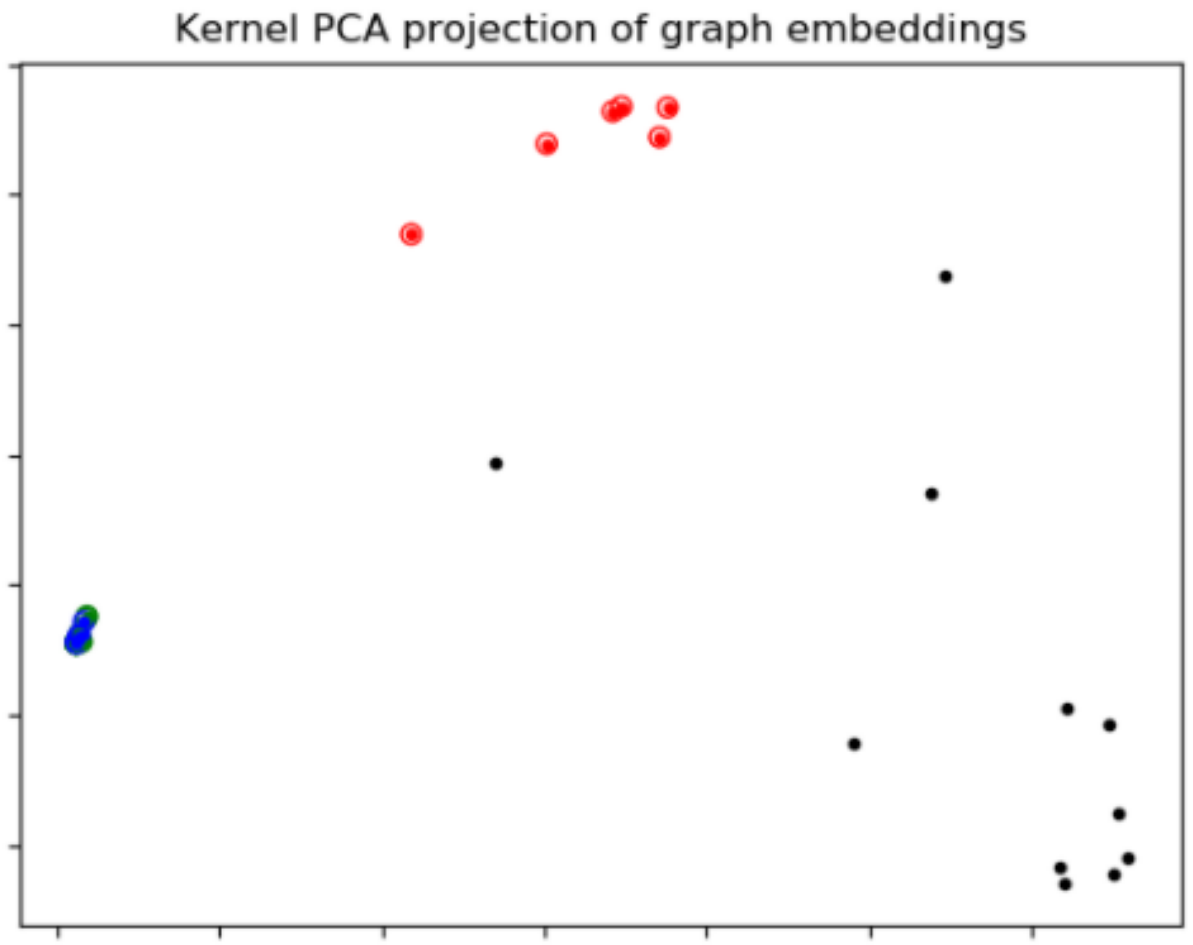
Experiments and Analytics

- *kernel PCA* to visualize our result.
- colours of points represent their types
- colours of circles around the points represent the predictions of our model. The colour-type correspondence is:
- Red: ‘t1’
- Green: ‘t3’
- Blue: ‘t2’
- Black: Non-fraudulent graphs



Fraudulent graph projection – normal graphs

- Projection of the three classes in the embedding space (with other random normal graphs)



Onging work: Community mining for fraud detection

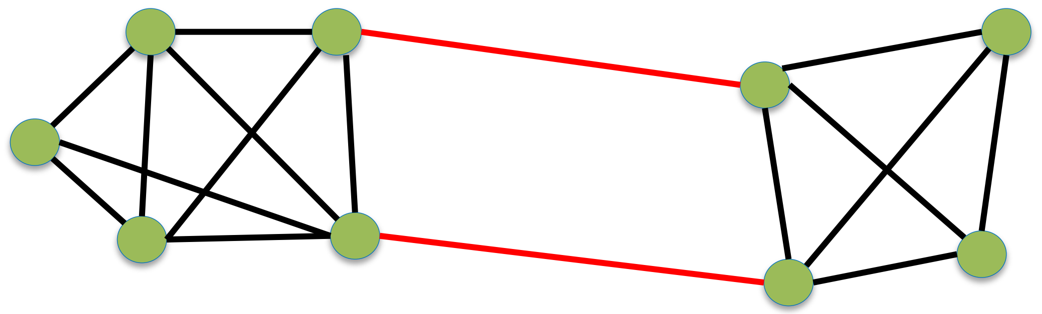
- Assume fraudulent community types in huge graphs
- **size**: The size of each subgraph should be of a manageable size. This would potentially assist in human processing of reported subgraphs.
- **overlapping clusters:** a node may have connectivity to multiple partitions.
- **Goals**
  - optimize a clustering algorithm in efficiency
  - control the cluster size
  - maintain multiple cluster assignments for each node.

A soft introduction to graph clustering

- Given Graph G=(V,E) undirected:
  - Vertex Set V={v_1,.....v_n}, Edge e_{ij} between v_i and v_j
    - we assume weight w_{ij}>0 for e_{ij}
  - |V| : number of vertices
  - d_i degree of v_i : $d_i = \sum_{v_j \in V} w_{ij}$
  - $v(V) = \sum_{v_i \in V} d_i$
  - for $A \subset V$ $\bar{A} = V - A$
  - Given $\mathrm{A, B} \subset V$ & $\mathrm{A} \cap B = \emptyset, w(A,B) = \sum_{v_i \in A, vj \in B} w_{ij}$
  - D : Diagonal matrix where $\mathrm{D(i,i)} = d_i$
  - W : Adjacency matrix $\mathrm{W(i,}j) = w_{ij}$

Graph-Cut

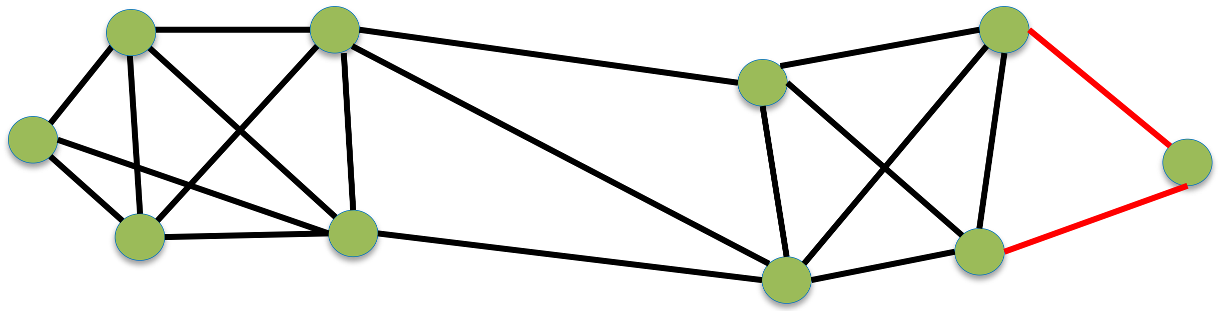
- For k clusters:
  - $cut(A_1, .., Ak) = 1/2 \sum_{i=1}^{k} w(Ai, \overline{A_i})$
    - undirected graph:1/2 we count twice each edge



- Min-cut:Minimize the edges' weight a cluster shares with the rest of the graph

Min-Cut

- Easy for k=2 : Mincut(A_1,A_2)
  - Stoer and Wagner: “A Simple Min-Cut Algorithm”
- In practice one vertex is separated from the rest
  - The algorithm is drawn to outliers



Normalized Graph Cuts

- We can normalize by the size of the cluster (size of sub-graph) :
  - number of Vertices (Hagen and Kahng, 1992):
    $Ratiocut(A_1, \dots Ak) = \sum_{i=1}^{k} \frac{cut(Ai,\overline{A_i})}{|Ai|}$
  - sum of weights (Shi and Malik, 2000) :
    $Ncut(A_1, \dots Ak) = \sum_{i=1}^{k} \frac{cut(Ai,\overline{A_i})}{v(A_i)}$
- Optimizing these functions is NP-hard
- Spectral Clustering provides solution to a relaxed version of the above

Graph Laplacian

- How is the previous useful in Spectral clustering?

$$\sum_{i,j=1}^{n} w_{ij}(f_i - fj)^2$$

$$= \sum_{i,j=1}^{n} w_{ij} f_i^2 \; -2\sum_{i,j=1}^{n} w_{ij} f_i f_j + \sum_{i,j=1}^{n} w_{ij} f_j^2$$

$$= \sum_{i,j=1}^{n} d_i f_i^2 - 2\sum_{i,j=1}^{n} w_{ij} f_i f_j + \sum_{i,j=1}^{n} d_j f_j^2$$

$$= 2\left(\sum_{i,j=1}^{n} d_{ii} f_i^2 - \sum_{i,j=1}^{n} w_{ij} f_i f_j\right)$$

$$= 2\left(\boldsymbol{f^T D f} - \boldsymbol{f^T W f}\right) = 2\boldsymbol{f^T}(\boldsymbol{D} - \boldsymbol{W})\boldsymbol{f} = 2\boldsymbol{f^T L f}$$

- **f:**a single vector with the cluster assignments of the vertices
- **L=D-W :** the Laplacian of a graph

Properties of L

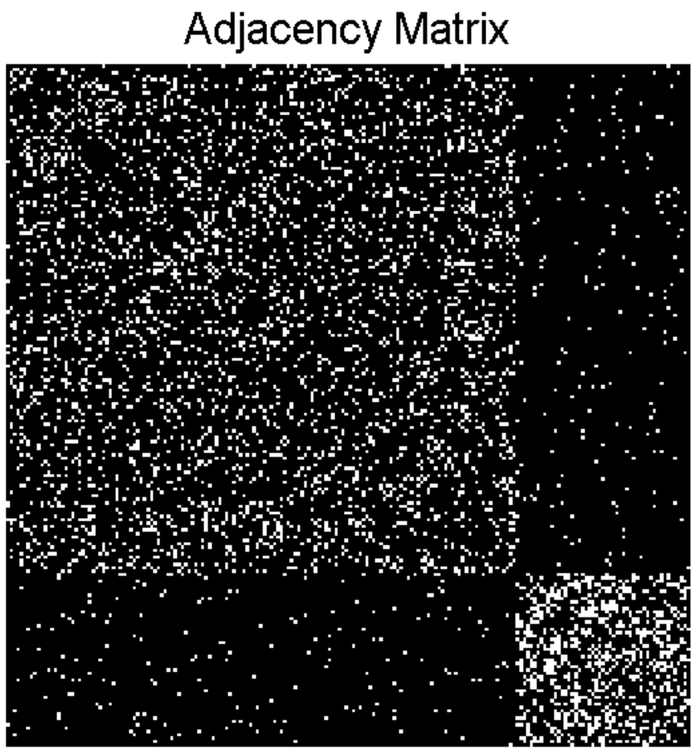
- L is
  - Symmetric
  - Positive
  - Semi-definite
- The smallest eigenvalue of L is 0
  - The corresponding eigenvector is $\mathbb{1}$
- L has n non-negative, real valued eigenvalues
  - $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$

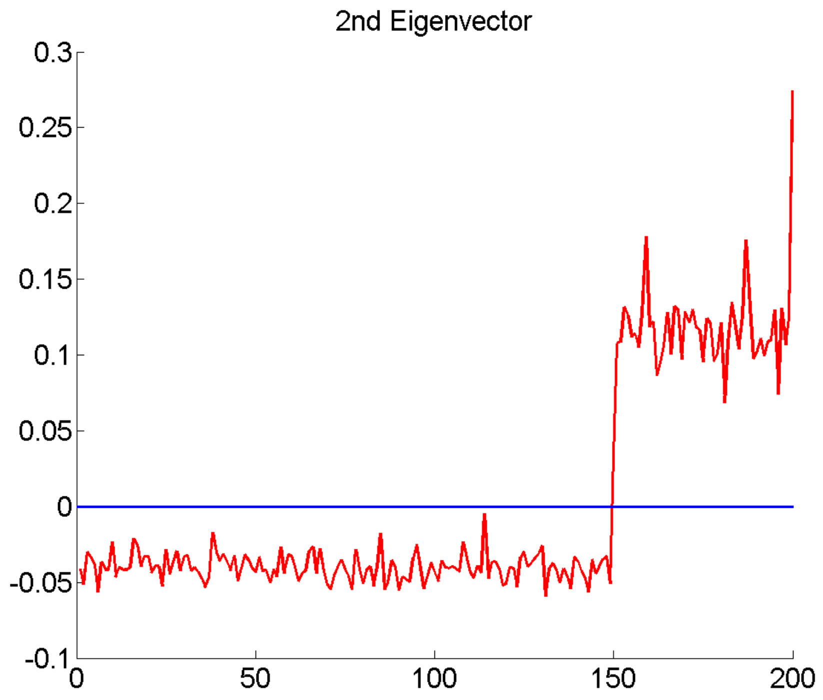
Two Way Cut from the Laplacian

- We could solve $min_f f^T Lf$ where $f \in \{-1,1\}^n$
- NP-Hard for discrete cluster assignments
  - Relax the constraint *to* $f \in R^n$:

    $min_f f^T Lf$ subject to $f^T f=n$
- The solution to this problem is given by:
  - (**Rayleigh-Ritz Theorem**) the eigenvector corresponding to smallest eigenvalue: 0 and the corresponding eigenvector (full of 1s) offers no information
- We use the second eigenvector as an approximation
  - $f_i>0$ the vertex belongs to one cluster , $fi<0$ to the other

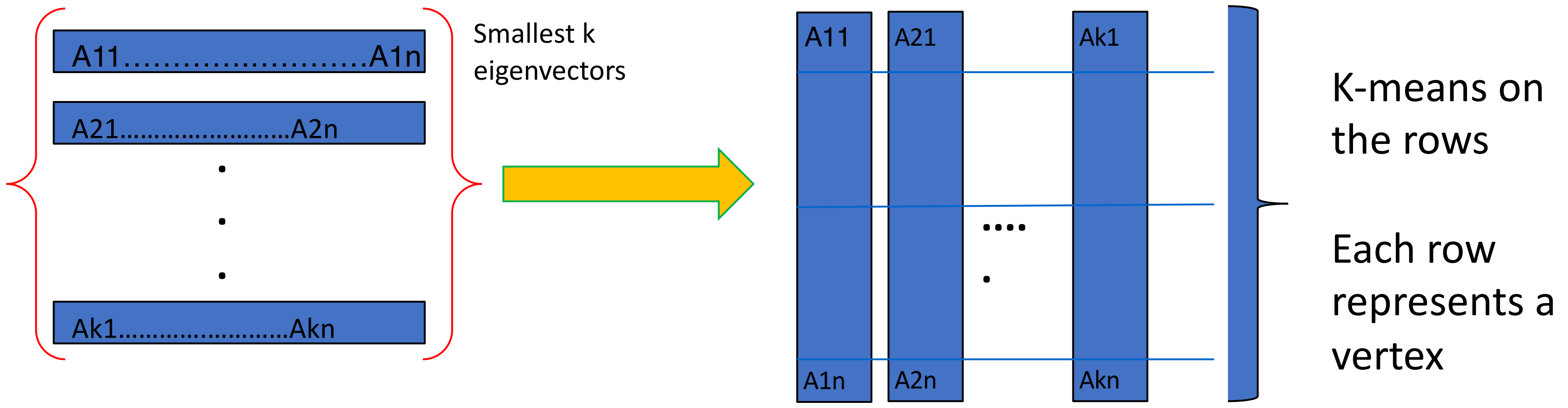
Example


Adjacency Matrix


2nd Eigenvector
0.3
0.25
0.2
0.15
0.1
0.05
0
-0.05
-0.1
0
50
100
150
200

Multi-Way Graph Partition

- The cluster assignment is given by the smallest k eigenvectors of L
- The real values need to be converted to cluster assignments
  - We use k-means to cluster the rows
  - We can substitute L with L_{sym}



Graph clustering - Modularity based methods

- **Modularity**

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

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

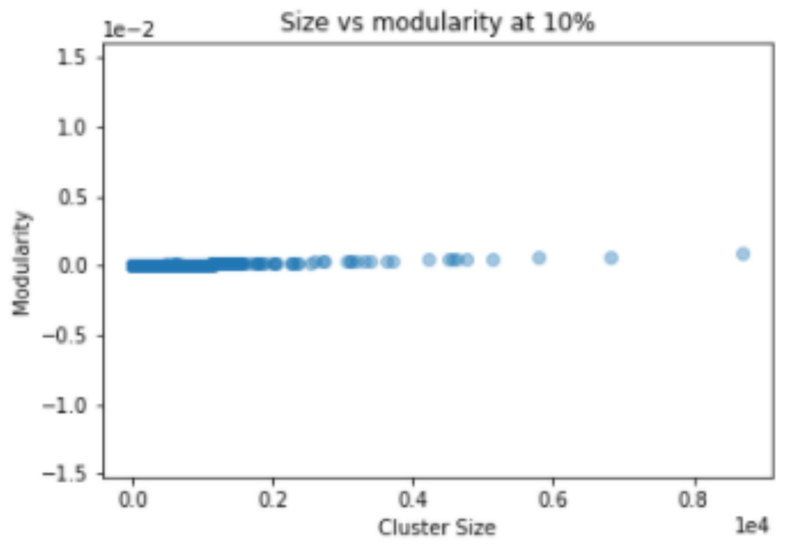
[Newman and Girvan ‘04], [Newman ‘06]

Clustering algorithms for community detection

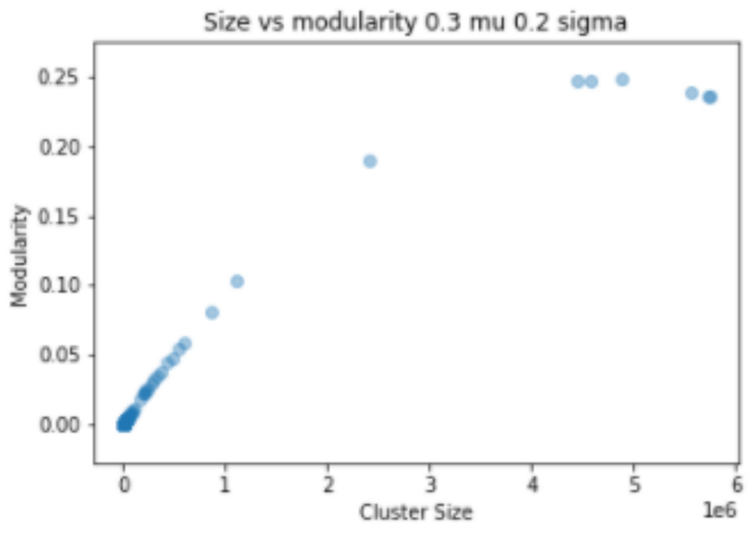
- **Louvain:** baseline/frame of reference.
  - starts each node as a single cluster and hierarchically joins the clusters while trying to optimize a clustering quality function (modularity).
  - Unfortunately, as we will see in practice, it creates some very large clusters and it does not offer overlapping clusters.
- **Markov Clustering – MCL**
  - main intuition: expands and inflates a transition matrix iteratively until it converges.
  - resulting matrix contains a graph of various connected components which are perceived as clusters.
  - does not support overlapping clusters), this can be a very demanding algorithm in resources in the Spark implementation as it requires transition matrix multiplications. Currently our experiment with MCL on Spark has shown a significantly low efficiency.
- **Label Propagation**
  - very efficient algorithm with many variations.
  - Each node starts with its own cluster/label and messages its own label to its neighbors.
  - nodes calculate their new label as an aggregation of the received messages.
  - labels converge or after a fixed number of iterations.

Design princliples of our Hybrid algorithm

- Produce overlapping clusters
- Constrain the size of the cluster



Louvain



Hybrid algorithm

Conclusions

- Use graph mininng (kernels and clustering) to explore fraud detection in graph of payments
- Encouraging results for classification
- Challenges:
  - generate clusters of controlled size and then predict fraud.
  - Huge volumes
  - Unkown fraud (unsupervised learning – autoencoders…?)

THANK YOU!!

Michalis Vazirgiannis

https://tinyurl.com/vv69dk8

Acknowledgements

C. Giatsidis, G. Nikolentzos, C. Wu, N. Steenfat, Y. Siglidis