Graph Characterization via Backtrackless Paths

Furqan Aziz
Richard C. Wilson
Edwin R. Hancock

Dept. of Computer Science
University of York

Motivation

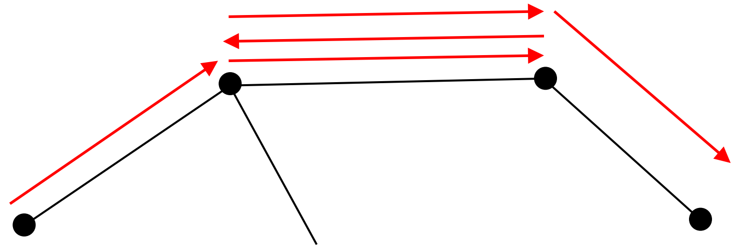
- Graph based methods are widely used in many applications like network analysis, world wide web, data mining, computer vision and complex systems.
- Graph embedding is an important because is allows statistically techniques to be applied directly
  - Embedding should reflect similarity
- Two main approaches
- Structural approach
  - Compute graph structural similarity (graph matching, edit distance, graph kernel)
  - Embed similarities or directly use kernel
  - Similarities may be non-Euclidean; pairwise comparisons
- Feature approach
  - Compare characterizations of the graphs
  - Direct feature embeddings
  - Efficiency and expressive power important
- How can we find efficient methods for characterizing graphs that do not involve exhaustive structural search?

Graph Kernels

- Random Walk Kernel (Gartner et al 2003)
  - Count the number of matching walks between two graphs

$$K(G_1, G_2) = \sum_{(i,j)\in V_\times} \sum_{k=0}^{\infty} \varepsilon_k \left[A_\times^k\right]_{ij}$$

  - k is the walk length
  - The number of walks becomes very large
  - The random walk graph kernel suffers from the problem of tottering



  - Reduces expressive power and masks structural differences

Path Kernels

- Tottering can be eliminated by comparing *paths*
- A path is a sequence of edges such that each edge neighbours the previous, and there are no repeated edges
- All-paths kernel (Borgwardt and Kriegel 2005)

$$K(G_1,G_2) = \sum_{p_1 \in \mathrm{allpaths}(G_1)} \sum_{p_2 \in \mathrm{allpaths}(G_1)} k(p_1, p_2)$$

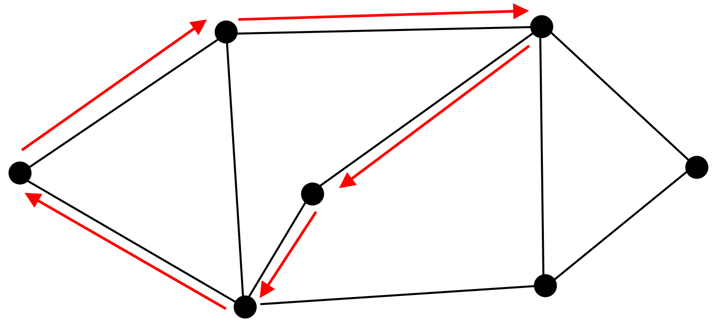
- All-paths is a true kernel, but NP-hard in general to find all paths
- Alternatives are shortest-path or *k*-shortest-path

THE UNIVERSITY *of York*

DEPARTMENT OF COMPUTER SCIENCE

Cycle Kernels

- Tottering can also be eliminated by using cycles
  - Path beginning and ending at same vertex



- Cycles and bridges kernel (Gärtner et al 2004)
- Still NP-complete to calculate all cycles
  - Some graphs have polynomial cycle complexity

Graph Characterizations

- Rather than enumerating and comparing paths, we can try to characterize graphs based on individual structure
- A *graph characterization* measures structural properties independently from the vertex labelling
  - Graph 'features'

- Spectral: use eigenvalues of adjacency matrix or Laplacian.
- Algebraic: co-efficients of characteristic polynomial.
- Topological: e.g. average degree, degree distribution, edge-density, diameter, cycle frequencies etc.

- Compare feature space to get similarity

Heat kernel

- The heat kernel is closely connected to random walks
  - Heat kernel is the kernel of a continuous-time random walk

$$\mathbf{H}(t) = \exp\left(-\mathbf{L}t\right)$$

- The heat kernel trace can be used as a random-walk characterization of a graph (Xiao, Wilson, Hancock – PR 2010).

$$\mathrm{Tr}[\mathbf{H}(t)] = \sum_i \exp(-\lambda_i t)$$

- Behaviour of trace with time gives a characterization of graph

$$\zeta(s) = \sum_{\lambda_k \neq 0} (\lambda_k)^{-s}$$

Connections

- Heat kernel related to walks

$$h_t(u,v) = \exp[-t]\sum_{k=1}^{|\infty} P_k(u,v)\frac{t^k}{k!}$$

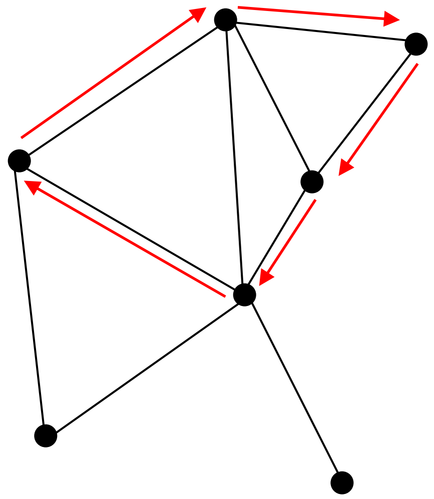
$$P_k(u,v) = \sum_{i \in V}(1-\lambda_i)^k \phi_i(u)\phi_i(v)$$

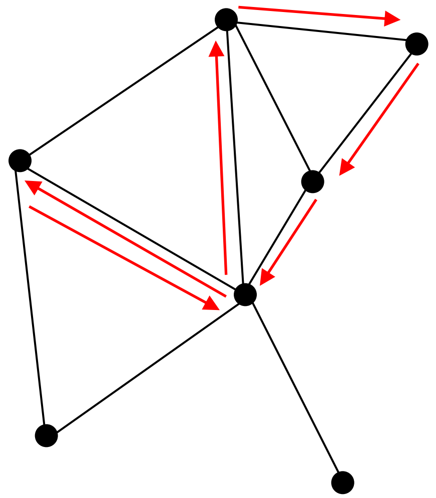
- Moments of the trace are connected to the *Rosenberg zeta function*

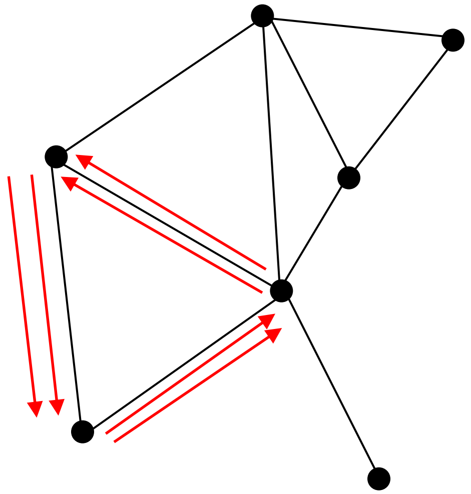
$$\zeta(s) = \sum_{\lambda_k \neq 0}(\lambda_k)^{-s}$$

Ihara Zeta Function

- Bass (1992), Kotani and Sunada (2000)
- Prime cycle of a graph:
  - A cycle which has no backtracking and is not a multiple of another cycle


Prime


Not Prime (backtracking)


Not Prime (twice round a single cycle)

Ihara zeta function

- Prime cycles eliminate some of the weaknesses of random walks (tottering)
- Can we (efficiently) characterize graphs using prime cycles? (Ren, Wilson, Hancock 2011 TNN)
- Ihara zeta function:

$$\zeta_G(u) = \prod_{p \in P} (1 - u^{l(p)})^{-1}$$

l(*p*) length of prime cycle *p*

- Depends purely on prime cycle lengths
  - So characterizes graph with prime cycles
- To evaluate, we would need to find all prime cycles

Ihara zeta function

- Three expressions for the Ihara zeta function

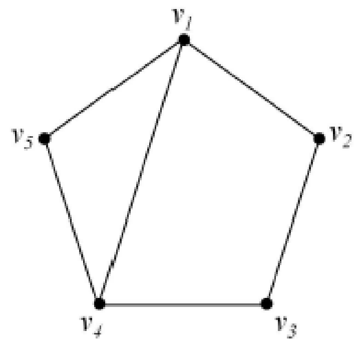
$$\zeta_G(u) = \prod_{p \in P} (1 - u^{l(p)})^{-1}$$

$$\zeta_G(u) = (1-u^2)^{|V|-|E|} \det(\mathbf{I} - u\mathbf{A} + u^2\mathbf{Q})^{-1}$$

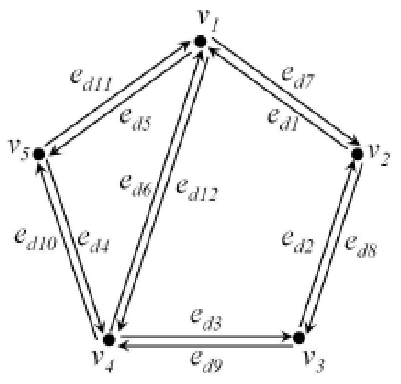
$$\zeta_G(u) = \det(\mathbf{I} - u\mathbf{T})^{-1}$$

A	adjacency matrix
Q	degree matrix
T	Perron-Frobenius operator

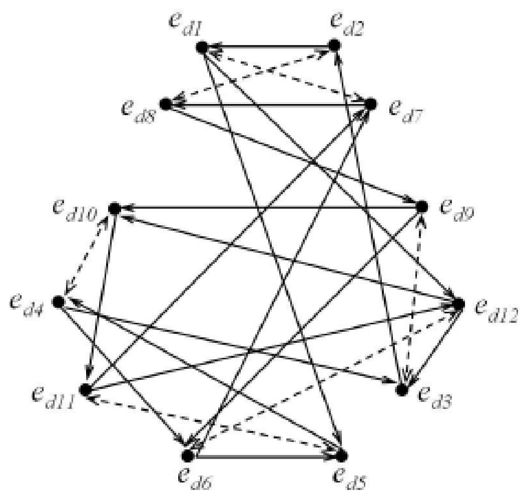
- The Perron-Frobenius operator in this case is the adjacency matrix of the oriented line graph



(a) Original Graph



(b) Digraph



(c) Oriented Line Graph

Ihara Zeta Function

- A polynomial expression for the IZF

$$\begin{aligned}\zeta_G(u) &= \det(\mathbf{I} - u\mathbf{T})^{-1} \\ &= (c_0 + c_1 u + c_2 u^2 + \ldots + c_m u^m)^{-1}\end{aligned}$$

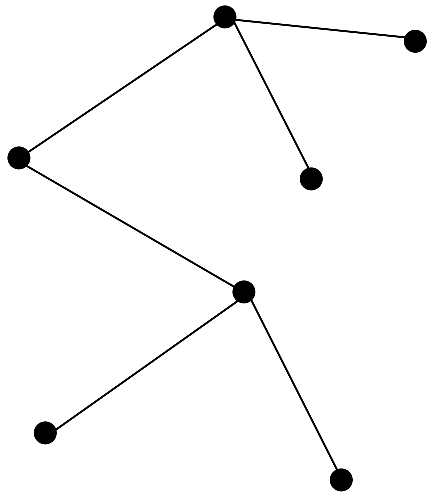
- We can characterize the graph using the coefficients *c*
- Related (non-trivially) to the number of cycles of a particular size
- Naïve implementation is expensive
  - Worse case: **T** is size $O(n^4)$ and running time $O(n^{12})$
- Showed how to evaluate efficiently using Bell polynomials (Aziz, Wilson, Hancock 2011 CAIP)

Ihara Zeta Function

- IZF is a powerful tool for representing graphs
  - IZF has proved useful for embedding graphs
- Linked to topological quantities
  - Coefficients related to number of triangles, squares etc
- ‘Edge based’ – related to oriented line graph
- Linked to another edge-based walk, the quantum walk
  - Lifting cospectrality: Emms, Hancock, Severini and Wilson showed that positive support of T-cubed can lift cospectrality of strongly regular graphs and trees (see J.Comb07 and Pattern Recognition08).
- Can also be expressed in terms of spectral polynomials (Wilson, Hancock and Luo PAMI 2005)
- Can be extended to hypergraphs

Observations

- Removing backtracking provides a richer description
- Using prime cycles (Ihara Zeta Function) avoids backtracking
  - Efficient computation possible
- IZF has limited applicability



No cycles

Cycles do not characterize the tree-like parts of a structure

Backtrackless Random Walk

- A random walk of length *k* is a sequence of vertices

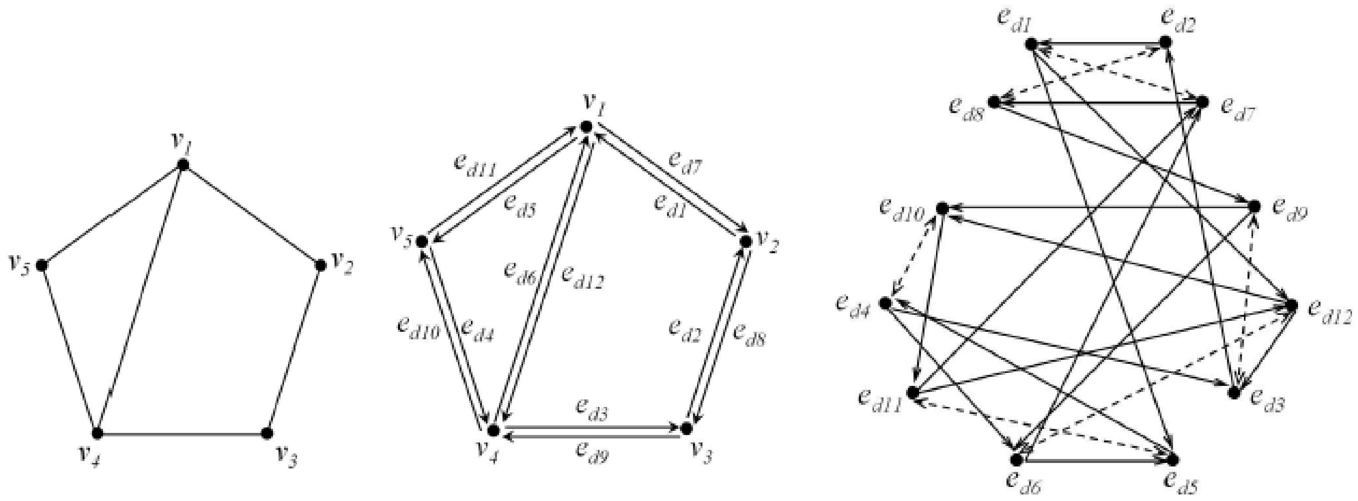
$$u_1, u_2, \ldots, u_{k+1}$$

  – Such that $e_i = (u_i, u_{i+1}) \in E$

- A backtrackless random walk has the additional condition

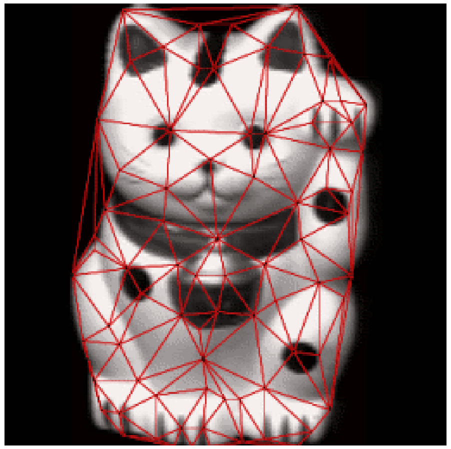
$$e_i \neq e_{i+1}$$

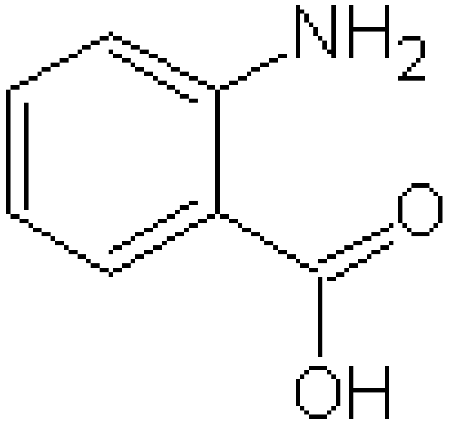
  – A sequence of oriented edges, excluding backtracking step



(a) Original Graph (b) Digraph (c) Oriented Line Graph

Labelled and Unlabelled graphs


Unlabelled vertices



Labelled (atom type)

- Unlabelled graph – two paths are the same if the sequence of vertices are the same

$$u_1, u_2, \ldots, u_{k+1} = v_1, v_2, \ldots, v_{k+1}$$

- Labelled graph – vertices and labels must be the same

$$(u_1, l_1), (u_2, l_2), \ldots, (u_{k+1}, l_{k+1}) = (v_1, m_1), (v_2, m_2), \ldots, (v_{k+1}, m_{k+1})$$

Backtrackless Random Walk Kernel

- The random walk kernel is

$$K(G_1, G_2) = \sum_{(i,j)\in V_\times} \sum_{k=0}^{\infty} \varepsilon_k \left[T_\times^k\right]_{ij}$$

- Defined on the product graph

$$V_\times\left(G_1 \times G_2\right) = \{(v_1, v_2) \in V_1 \times V_2\}$$
$$E_\times\left(G_1 \times G_2\right) = \{((u_1, u_2), (v_1, v_2))\}$$
$$(u_1, u_2) \in E_1 \wedge (v_1, v_2) \in E_2$$

- Our base graph is the OLG
  - Transform each graph into its OLG
  - Form the product graph
- By eliminating the reverse edges in the OLG, we eliminate backtracking

Efficient computation

- Complexity is a problem

$$|V_\times| = n^2 \qquad |E_\times| = n^4$$

- We can directly compute $n \times n$ matrix $\mathbf{A}_k$, defined as

$$(A_k)_{i,j} = \begin{cases} \text{number of paths in } G \text{ of length } k \text{ with no backtracking} \\ \text{starting at } i \text{ and ending at } j \end{cases}$$

here i, j run over the vertices of G.

- Recursions for the matrices $\mathbf{A}_k$
  - Let $\mathbf{A}$ be the adjacency matrix of a simple graph G and $\mathbf{Q}$ be a $n \times n$ diagonal matrix whose ith diagonal entry is the degree of the ith node minus 1. Then

$$\mathbf{A}_k = \begin{cases} \mathbf{A} & \text{if } k = 1 \\ \mathbf{A}^2 - (\mathbf{Q} + \mathbf{I}) & \text{if } k = 2 \\ \mathbf{A}_{\text{k-1}}\mathbf{A} - \mathbf{A}_{k-2}\mathbf{Q} & \text{if } k \geq 3 \end{cases}$$

Kernels and characterizations

- Using this recursion, we can compute low orders of the kernel

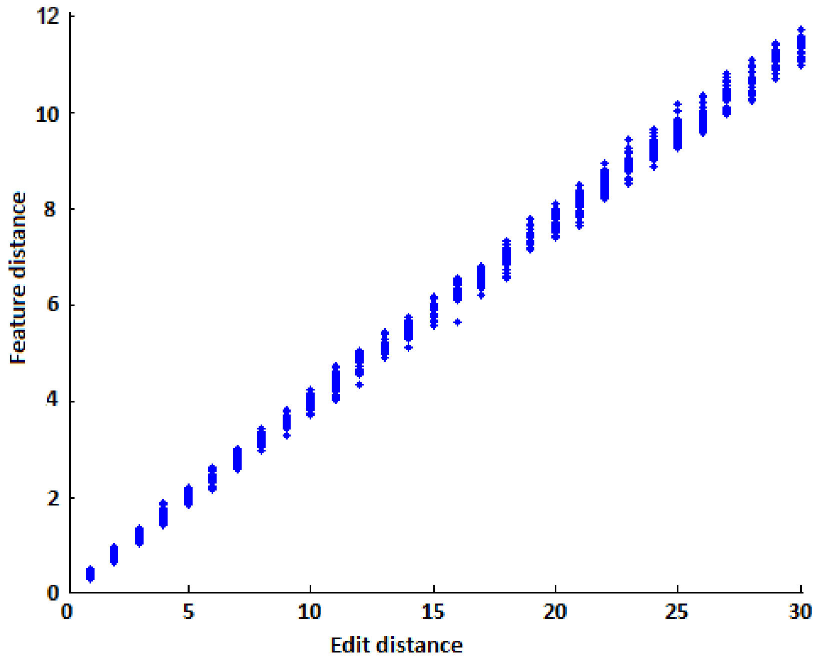
$$K(G_1,G_2)=\sum_{(i,j)\in V_\times}\sum_{k=0}^{\infty}\varepsilon_k\left[A_\times^k\right]_{ij}$$

- The coefficients ε are chosen to allow the kernel to converge quickly enough
- The kernel framework naturally allows labelled graphs
  - Edges exist in the product graph where the labels match
- We can also provide characterizations of a graph using the BRW on a graph

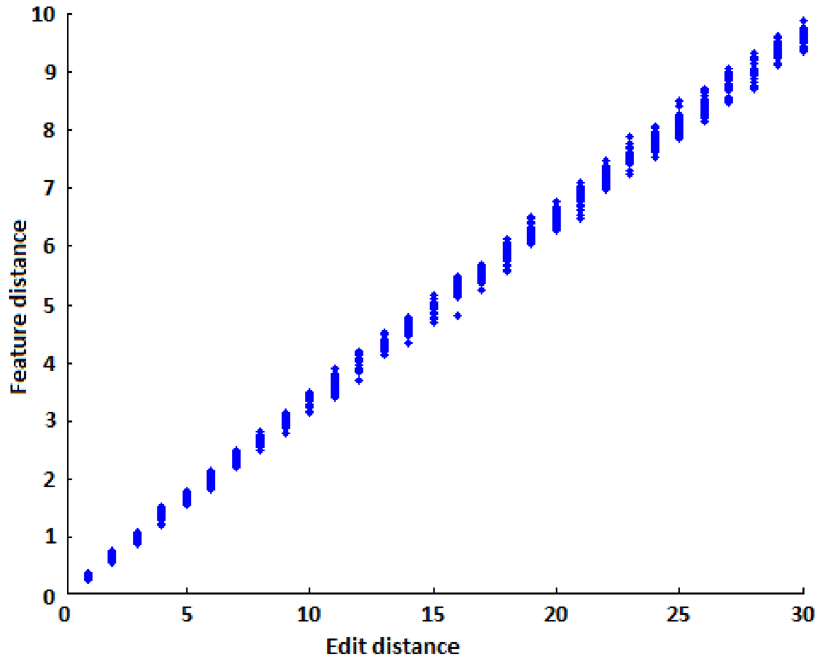
$$l_k=\sum_{i,j=1}^{|V|}\left[A_k\right]_{i,j}$$

Experiments – Synthetic Graphs

- Synthetic data: The purpose of the experiments on synthetic dataset is to evaluate whether the backtrackless walks can distinguish between different graphs under controlled structural errors.
- The edit distance between two graphs G_1 *and* G_2 *is the minimum edit cost taken* over all sequences of edit operations that transform G_1 *to* G_2.



Random Walk



Backtrackless Walk

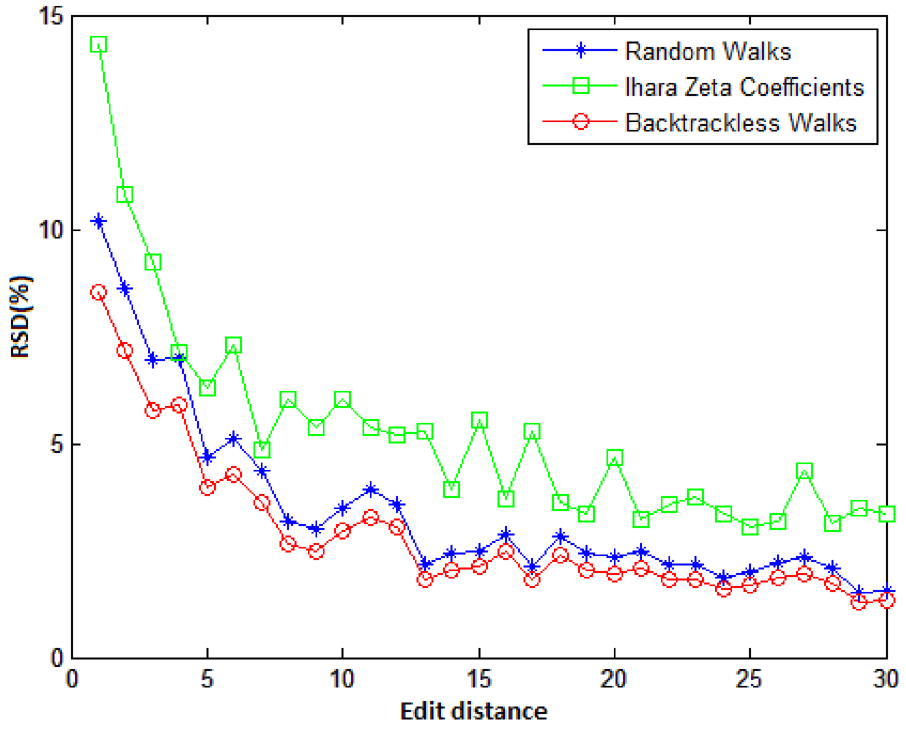
- Both walks reproduce edit-distance similarity well

THE UNIVERSITY *of York*

DEPARTMENT OF COMPUTER SCIENCE

Experiments

- Figure shows the relative percent standard deviation as a function of edit distance for different methods.
- It is clear from figure that the backtrackless walk provide a more stable representation of the graph when compared to either random walks or the Ihara Coefficients.



Real world data

- Coil
  - Consist of graphs extracted from images in coil dataset.
  - To establish graphs, feature points are extracted using Harris detector, and then Delaunay triangulation is constructed.

- MUTAG
  - Collection of 188 chemical compounds.
  - Task is to predict whether each compound has mutagenicity or not.

Results

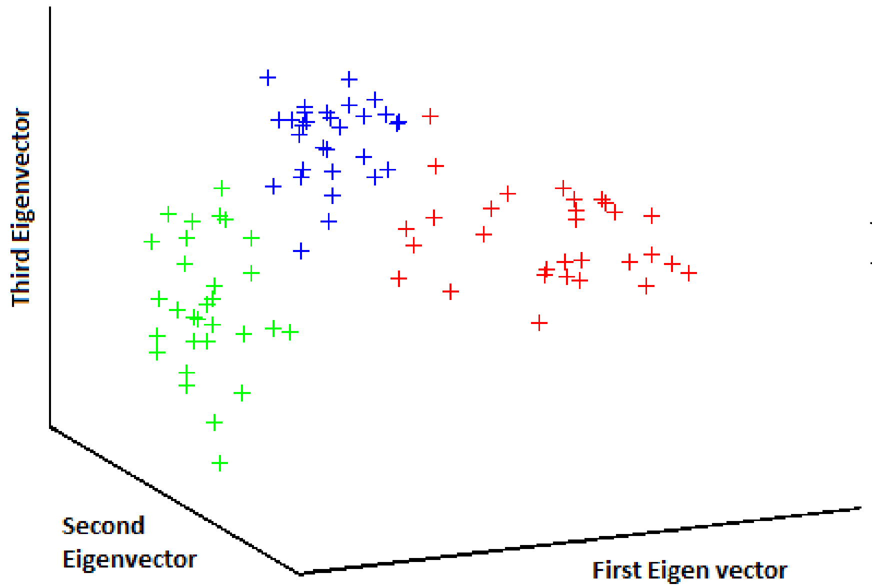


Figure: Performance of clustering

Method	Dataset	Accuracy
Random walk kernel	Mutag(labelled)	90. 0%
Backtrackless walk kernel	Mutag(labelled)	**91.1**%
Feature vector from Random walk	COIL(unlabeled)	94.4%
Feature vector from backtrackless random walk	COIL(unlabeled)	**95.5**%
Feature vector from Ihara coefficients	COIL(unlabeled)	94.4%
Shortest Path Kernel	COIL(unlabeled)	86.7%
Feature vector from Random walk	Mutag(unlabeled)	89.4%
Feature vector from backtrackless random walk	Mutag(unlabeled)	**90.5**%
Feature vector from Ihara coefficients	Mutag(unlabeled)	80.5%

Time analysis

- The worst case complexity of computing the backtrackless walk in a graph is same as that of computing the random walk.
- In practice, the execution time of computing backtrackless walk using proposed method is close to the execution time of computing the random walk.
- For comparison the following table shows the execution time of computing walks of length 10 in 1000 randomly generated graphs.

Method	**Worst case Running time**	**Execution Time (in seconds)**
Random walk	$O(n^6)$	9.98
Backtrackless walk (using proposed method)	$O(n^6)$	12.30
Backtrackless walk (by transforming graph)	$O(n^{12})$	313.14

Conclusion

- Backtrackless walks are more robust to noise compared to random walks.
- Efficient method for computing backtrackless walks for a graph.
- Efficient kernels for clustering both labelled and unlabeled graphs.
- More effective characterization of graph structure than random walks, shortest path and Ihara zeta function

Questions?