MCL

(and other clustering algorithms)

858L

Comparing Clustering Algorithms

Brohee and van Helden (2006) compared 4 graph clustering algorithms for the task of finding protein complexes:

- MCODE
- RNSC – Restricted Neighborhood Search Clustering
- SPC – Super Paramagnetic Clustering
- MCL – Markov Clustering

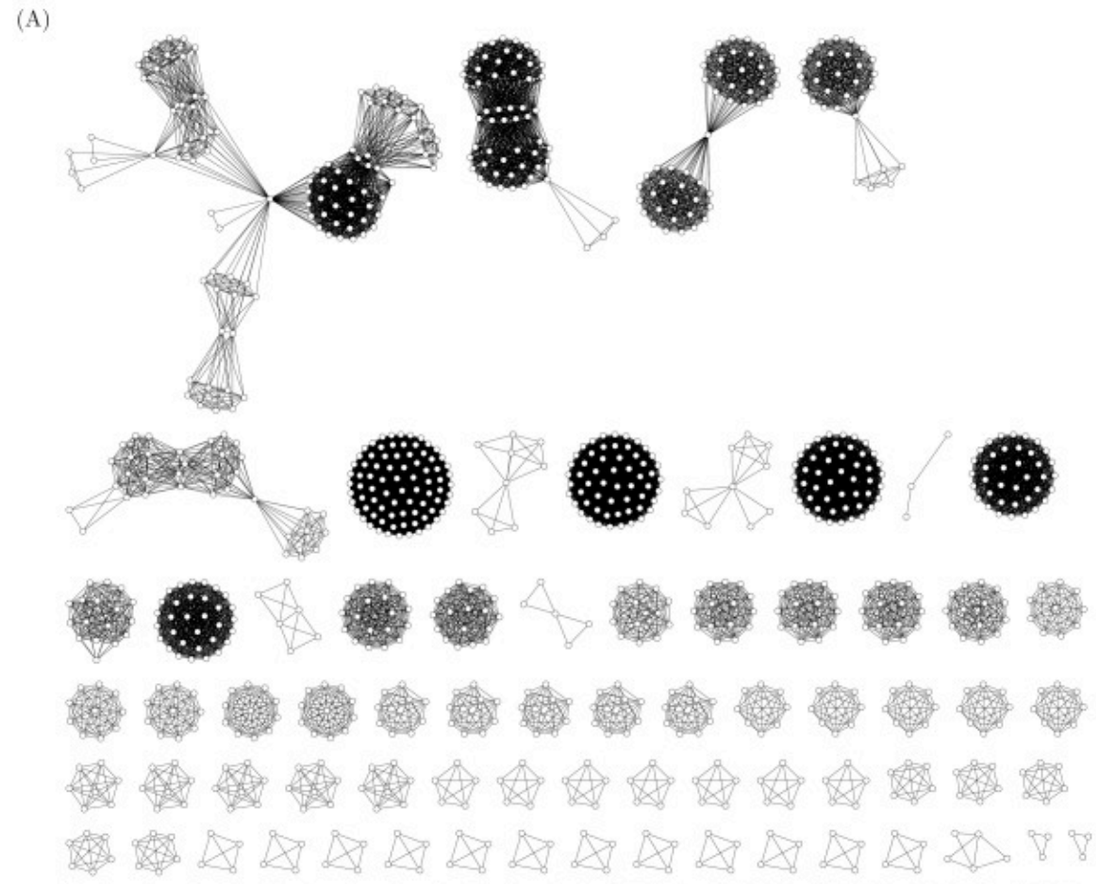
Used same MIPS complexes that we've seen before as a test set.

Created a simulated network data set.

Simulated Data Set

220 MIPS complexes (similar to the set used when we discussed VI-Cut and graph summarization).

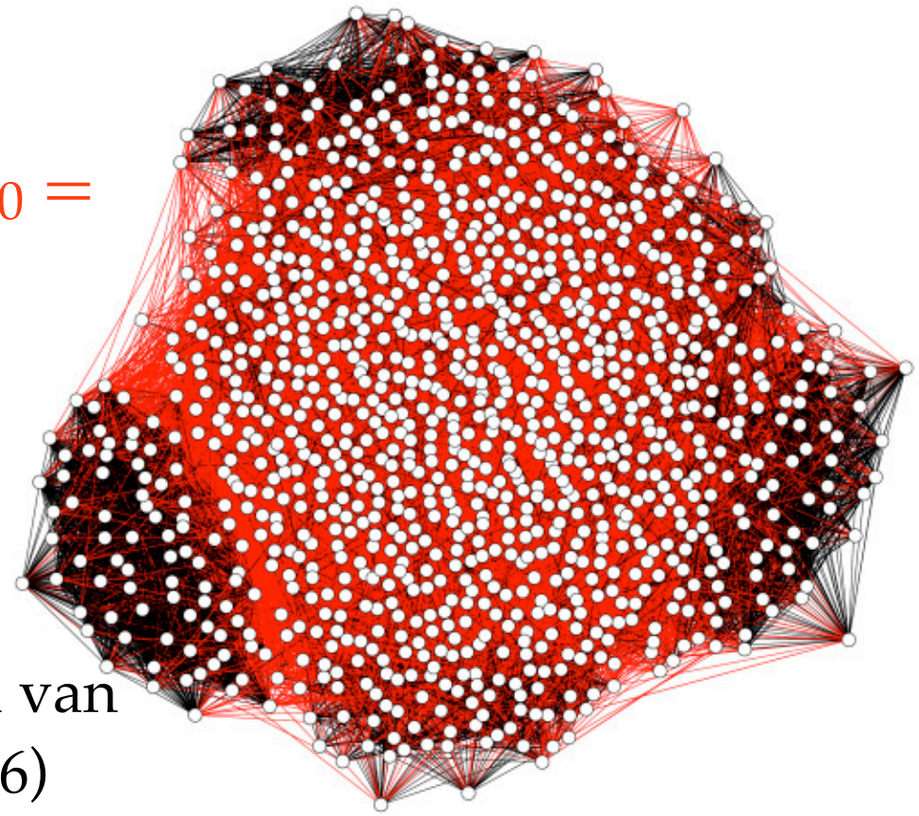
Created a clique for each complex. Giving graph A (at right)



$A_{add,del}$:= this clique graph with (add) % random edges added and (del)% edges deleted.

Also created **a** (!?) random graph R by shuffling edges and created $R_{add,del}$ for the same choices of (add) and (del).

$A_{100,40}$ =



(Brohee and van Helden, 2006)

RNSC

RNSC (King, et al, 2004): Similar in spirit to the Kernighan-Lin heuristic, but more complicated:

1. Start with a random partitioning.

2. Repeat: move a node *u* from one cluster to another cluster C, trying to minimize this cost function:

   # neighbors of u that are not in the same cluster +
   # of nodes co-clustered with u that are not its neighbors

3. Add *u* the "FIXED" list for some number of moves.

4. Occasionally, based on a user defined schedule, destroy some clusters, moving their nodes to random clusters.

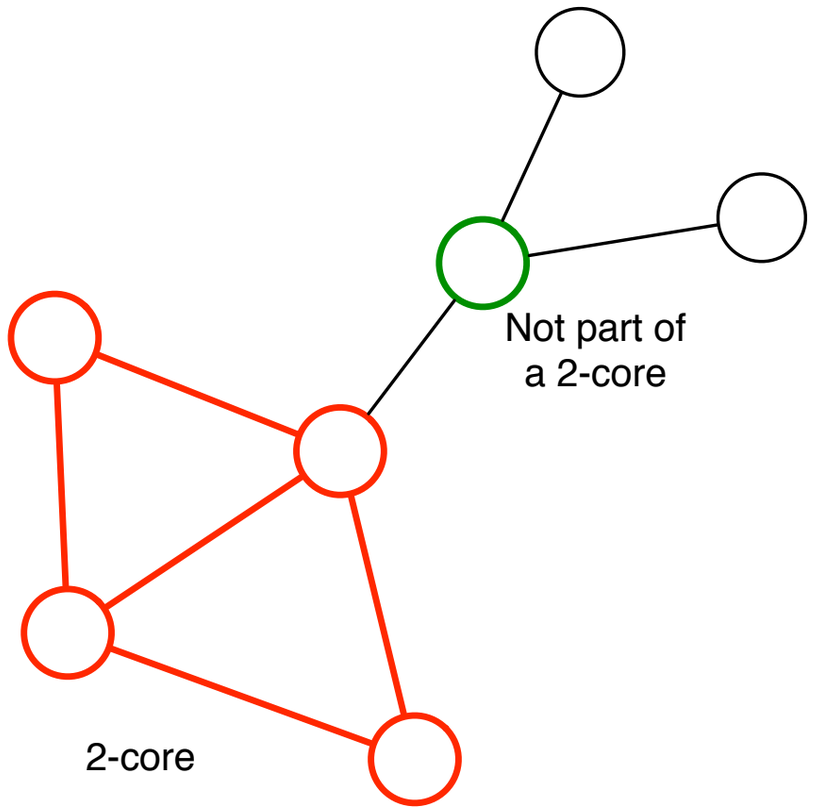
5. If no improvement is seen for *X* steps, start over from Step 2, but use a more sensitive cost function:

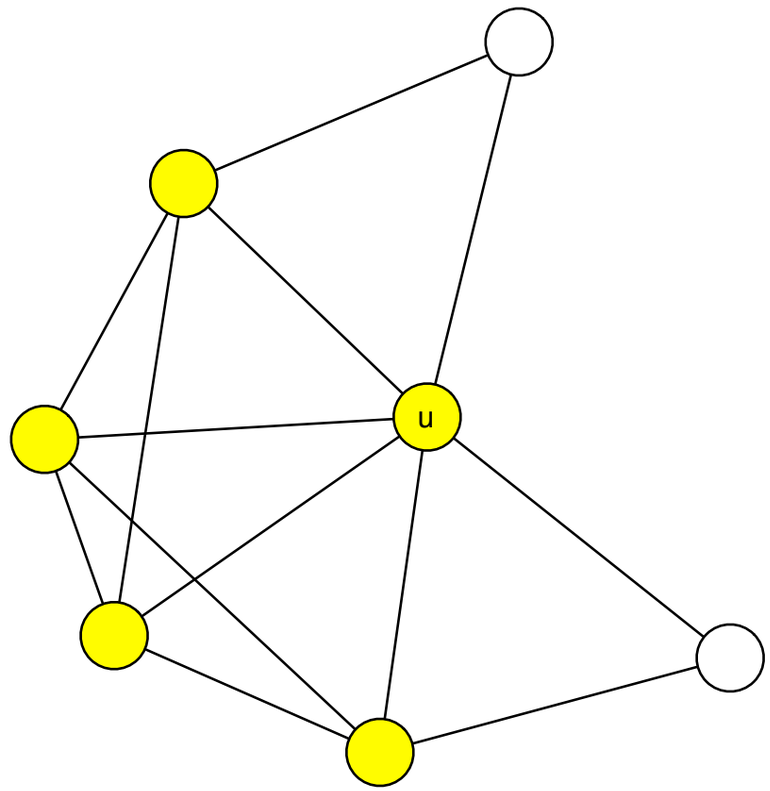
   Approximately: Naive cost function scaled
   by the size of cluster C

MCODE

Bader and Hogue (2003) use a heuristic to find dense regions of the graph.

Key Idea. A *k*-core of G is an induced subgraph of G such that every vertex has degree ≥ *k*.





A **local k-core(u, G)** is a *k*-core in the subgraph of G induced by {u} ∪ N(u).

A **highest k-core** is a *k*-core such that there is no (k+1)-core.

MCODE, continued

1. The *core clustering coefficient* CCC(*u*) is computed for each vertex *u*:

> CCC(*u*) = the density of the highest, local *k*-core of *u*.
>
> In other words, it's the density of the highest *k*-core in the graph induced by $\{u\} \cup N(u)$.
>
> "Density" is the ratio of existing edges to possible edges.

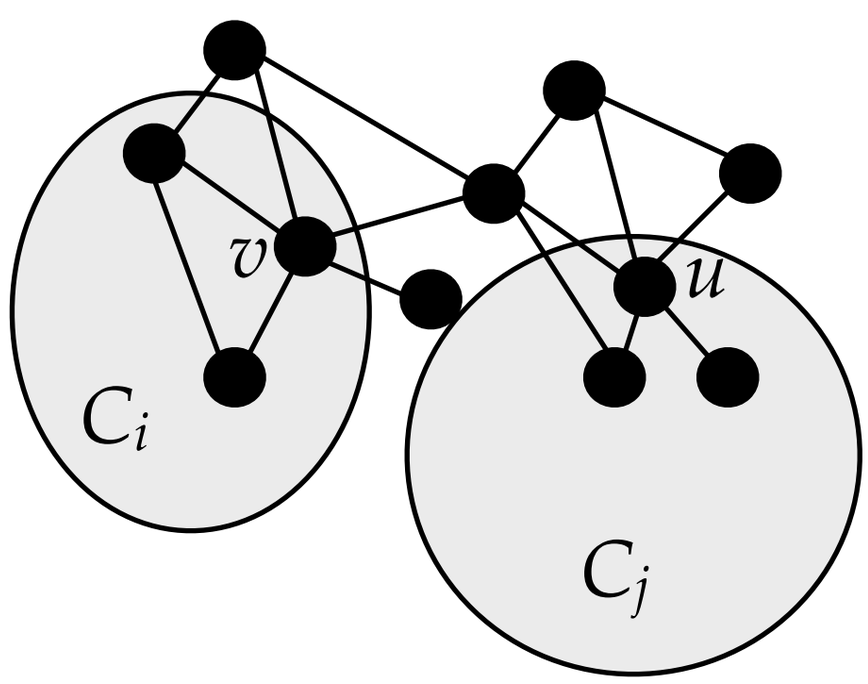
2. Vertices are weighted by $k_{\text{highest}}(u) \times \text{CCC}(u)$, where $k_{\text{highest}}(u)$ is the largest *k* for which there is a local *k*-core around *u*.
3. Do a BFS starting from the vertex *v* with the highest weight w_v, including vertices with weight $\geq$ TWP $\times w_v$.
4. Repeat step 3, starting with the next highest weighted seed, and so on.

MCODE, final step

Post-process clusters according to some options:

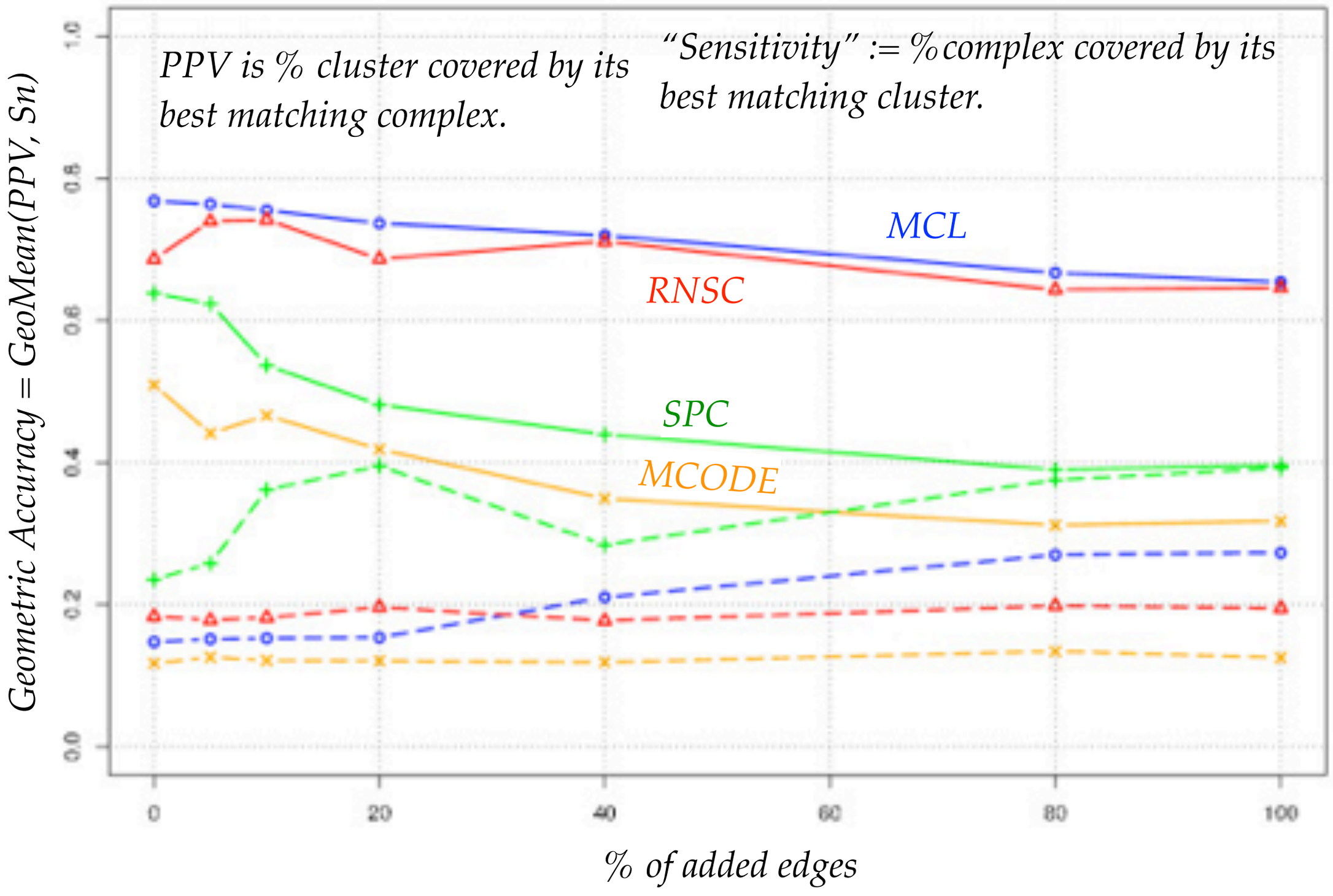
Filter. Discard clusters if the do not contain a 2-core.

Fluff. For every *u* in a cluster C, if the density of {u} ∪ N(u) exceeds a threshold, add the nodes in N(u) to C if they are not part C_1, C_2, ..., C_q. (This may cause clusters to overlap.)



Haircut. 2-core the final clusters (removes tree-like regions).

Comparison – 40% edges removed; varied % added



Representative test; MCL generally outperformed others.

MCL

Motivation

van Dongen (2000) proposes the following intuition for the *graph clustering paradigm:*

(1) Number of u-v paths of length k is larger if u,v are in the same dense cluster, and smaller if they belong to different clusters.

(2) A random walk on the graph won't leave a dense cluster until many of its vertices have been visited.

(3) Edges between clusters are likely to be on many shortest paths. ← Girvan-Newman

Think driving in a city: (1) if you're going from u to v, there are lots of ways to go; (2) random turns will keep you in the same neighborhood; (3) bridges will be heavily used.

Structural Units of a Graph

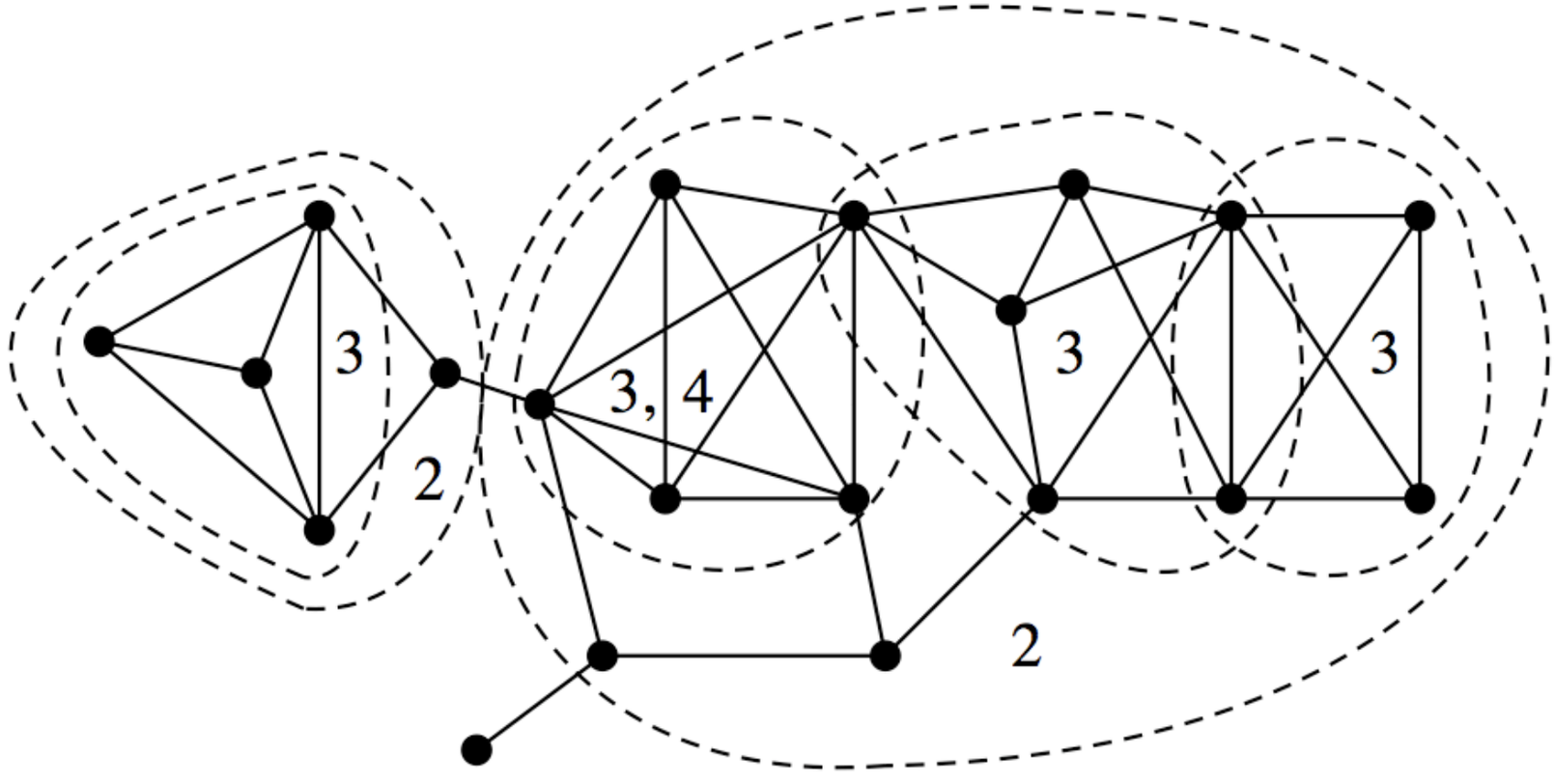
k-bond. A maximal subgraph S with all nodes having degree ≥ *k* **in *S***.

k-component. A maximal subgraph *S* such that every pair *u*, *v* ∈ *S* is connected by *k* edge-disjoint paths **in *S***.

k-block. A maximal subgraph *S* such that every pair *u*, *v* ∈ *S* is connected by k vertex-disjoint paths **in *S***.

k-blocks of a graph (van Dongen, 2000):

(k+1)-blocks nest inside k-blocks.



Structural Units of a Graph

k-bond. A *maximal* subgraph S with all nodes having degree ≥ *k* **in *S***.

k-component. A maximal subgraph *S* such that every pair *u*, *v* ∈ *S* is connected by *k* edge-disjoint paths **in *S***.

k-block. A maximal subgraph *S* such that every pair *u*, *v* ∈ *S* is connected by k vertex-disjoint paths **in *S***.

Every k-block ⊆ some k-component

k vertex-disjoint paths are all edge-disjoint.

Hence if u,v are connected by k vertex-disjoint paths in S, they are connected by k edge-disjoint paths in S.

Every k-component ⊆ some k-bond

All vertices of a k-component must have degree ≥ k in S.

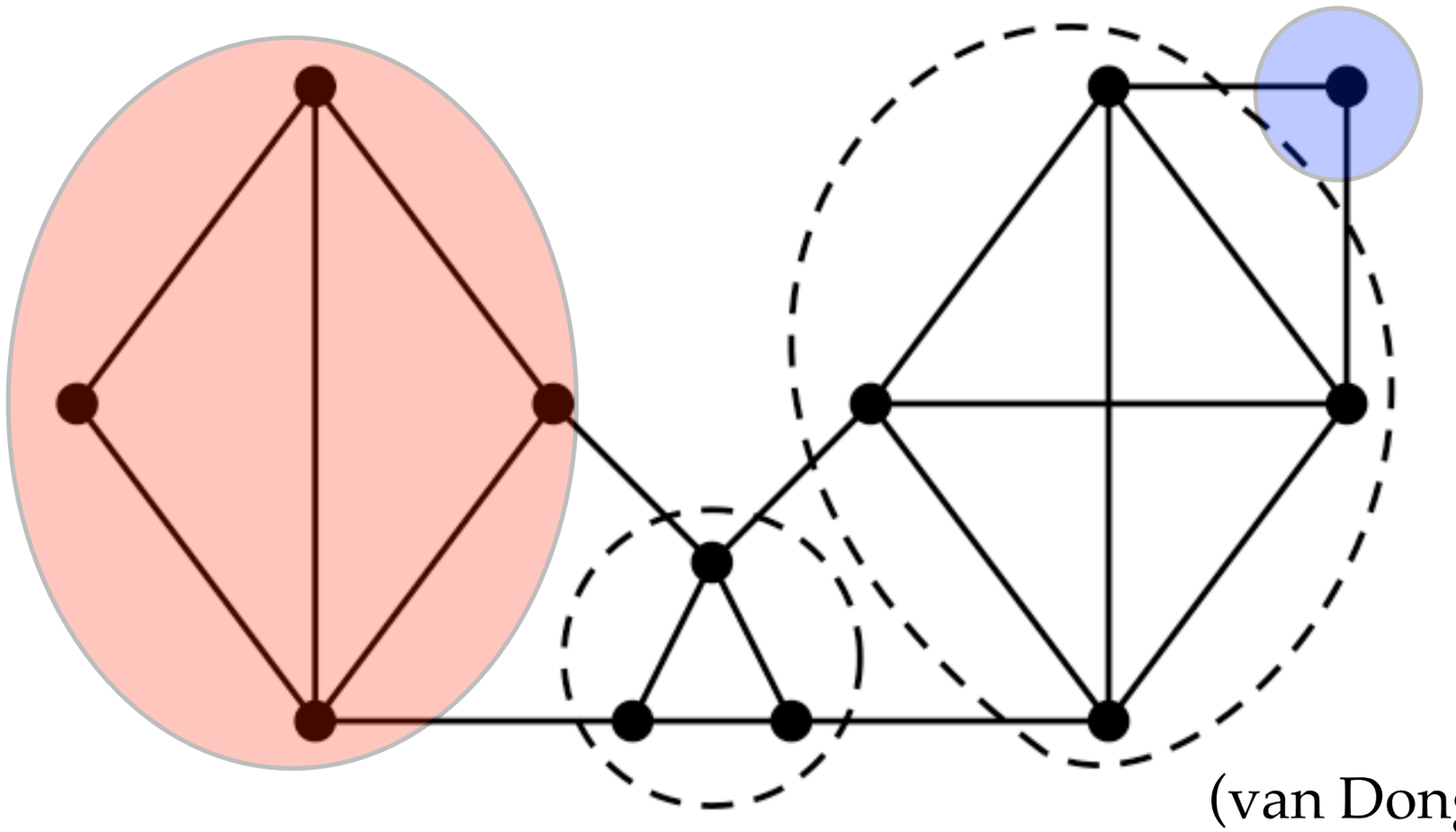
(If degree(u) < k, u couldn't have k edge disjoint paths to v in S.

Thm. (Matula) The *k*-components form equivalence classes (they don't overlap).

Problem with *k*-blocks as clusters

The clustering is very sensitive to node degree and to particular configurations of edge-disjoint paths.

Example 1. Red shaded region is nearly a complete graph (missing only one edge), yet each of its nodes is in its own 3-block.



Example 2. Blue shaded region can't be in a 3-block with any other vertex (b/c it has degree 2), but really it should be with the K_4 subgraph it is next to.

(van Dongen, 2000):

Number of Length-k Paths

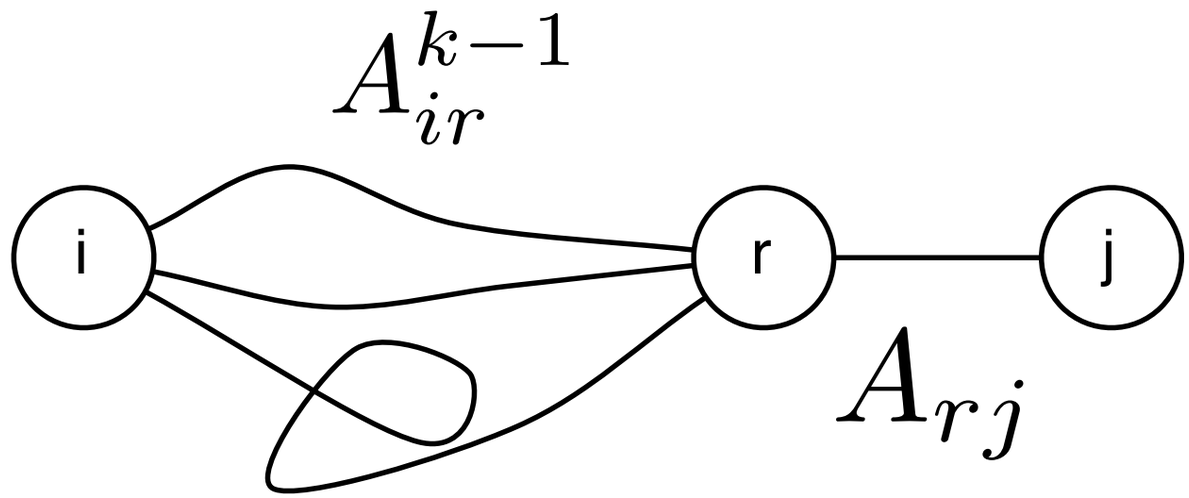
Let A be a the adjacency matrix of an unweighted simple graph G. A^k is $A \cdot A \cdot \ldots \cdot A$ (k times)

Thm. The (i,j) entry of A^k, denoted $(A^k)_{ij}$, is the number of paths of length k starting at i and ending at j.

Proof. By induction on k. When $k = 1$, A directly gives the number (0 or 1) of length 1 paths. For $k > 1$:

$$(A^k)_{ij} = (A^{k-1}A)_{ij} = \sum_{r=1}^{n} (A_{ir}^{k-1} A_{rj})$$

Note: the paths do not have to be simple.



k-Path Clustering

Idea. Use $Z_k(u,v) := (A^k)_{uv}$ as a similarity matrix.

k is an input parameter.

Given $Z_k(u,v)$, for some k, use it as a similarity matrix and perform *single-link clustering*.

Single-link clustering of matrix M:
  Throw out all entries of M that are < threshold **t**
  Return connected components of remaining edges.

Called single-link clustering because a single "good" edge can merge two clusters.

Problem with *k*-Path Clustering

(van Dongen, 2000)

Consider Z_2:

$Z_2(a,b) = 1$, and

$Z_2(a,c) = 1$

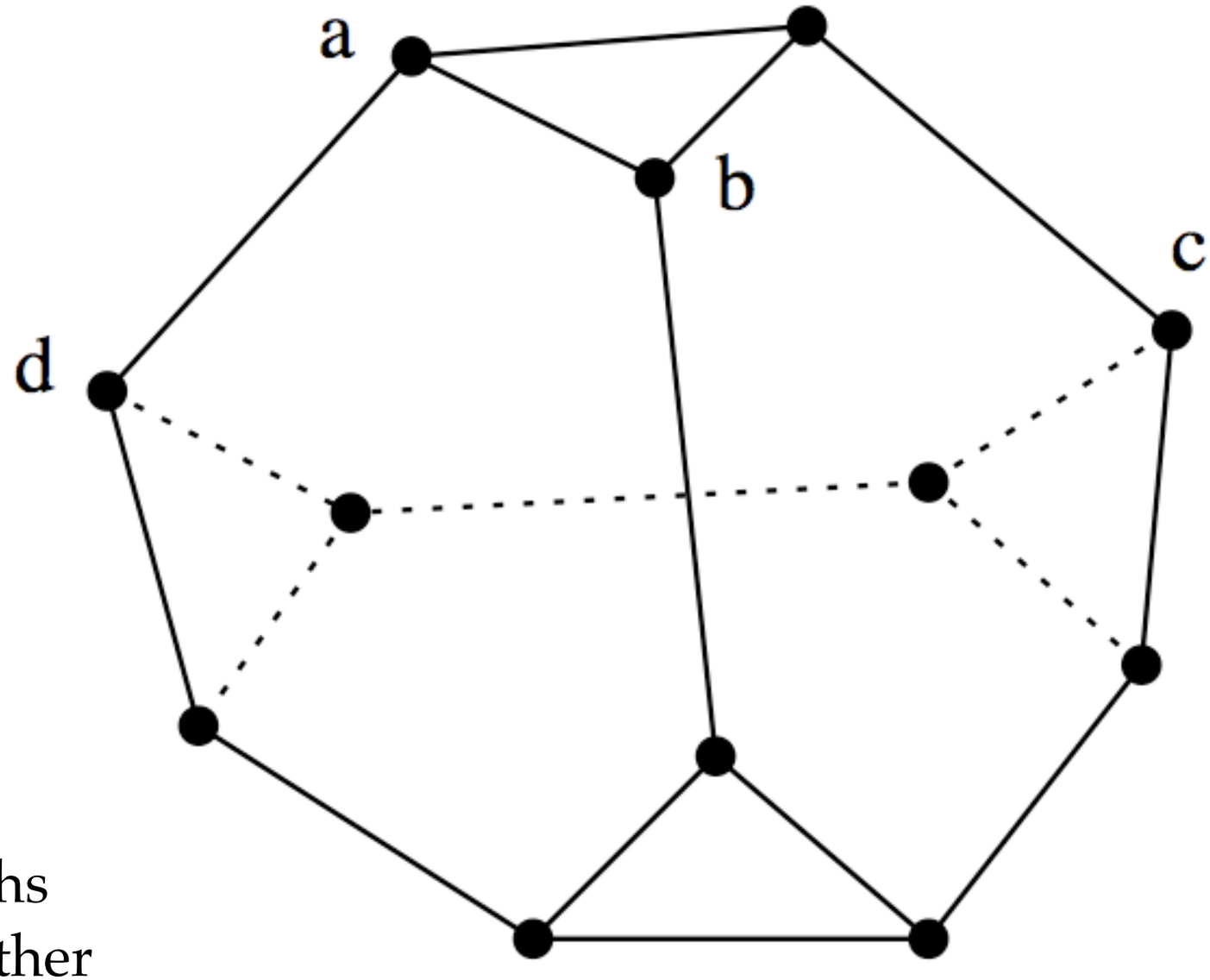
But intuitively, a,b are more closely coupled than a,c

Consider Z_3:

$Z_3(a,b) = 2$, and

$Z_3(a,d) = 2$ [Why?]

But intuitively, a,b are more closely coupled than a,d



While there *are* more short paths between a & b than between other pairs, half of the short paths are of odd length and half are of even length.

Problem with *k*-Path Clustering

(van Dongen, 2000)

Consider Z_2:

$Z_2(a,b) = 1$, and

$Z_2(a,c) = 1$

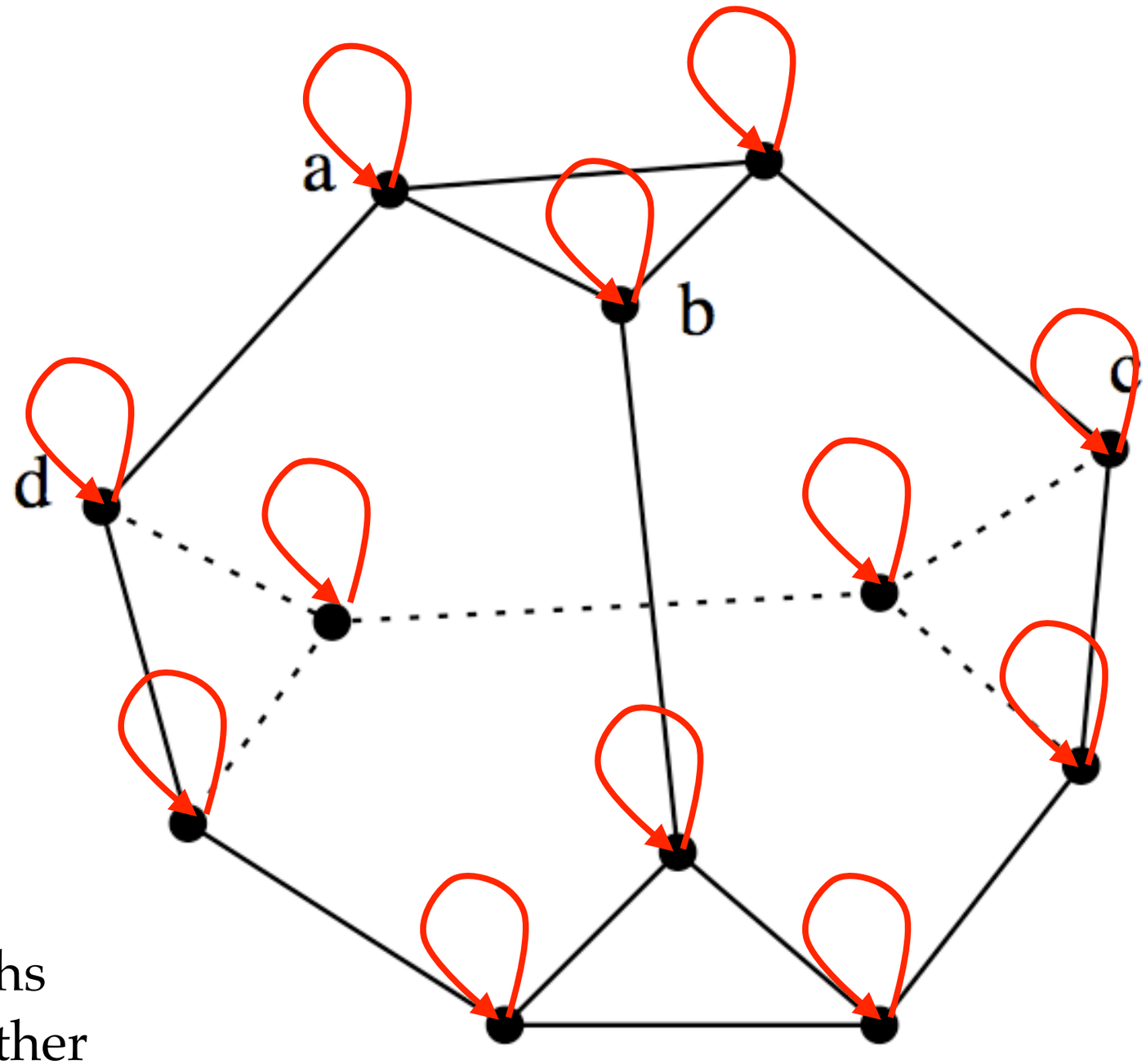
But intuitively, a,b are more closely coupled than a,c

Consider Z_3:

$Z_3(a,b) = 2$, and

$Z_3(a,d) = 2$ [Why?]

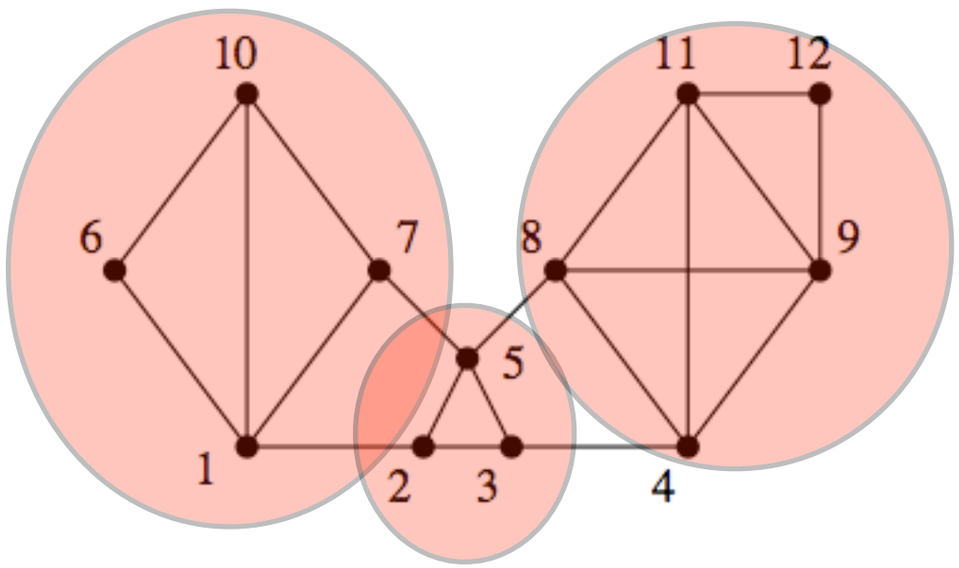
But intuitively, a,b are more closely coupled than a,d



While there *are* more short paths between a & b than between other pairs, half of the short paths are of odd length and half are of even length.

Solution. Add self-loops to every node.

Example *k*-path clustering. (van Dongen, 2000)

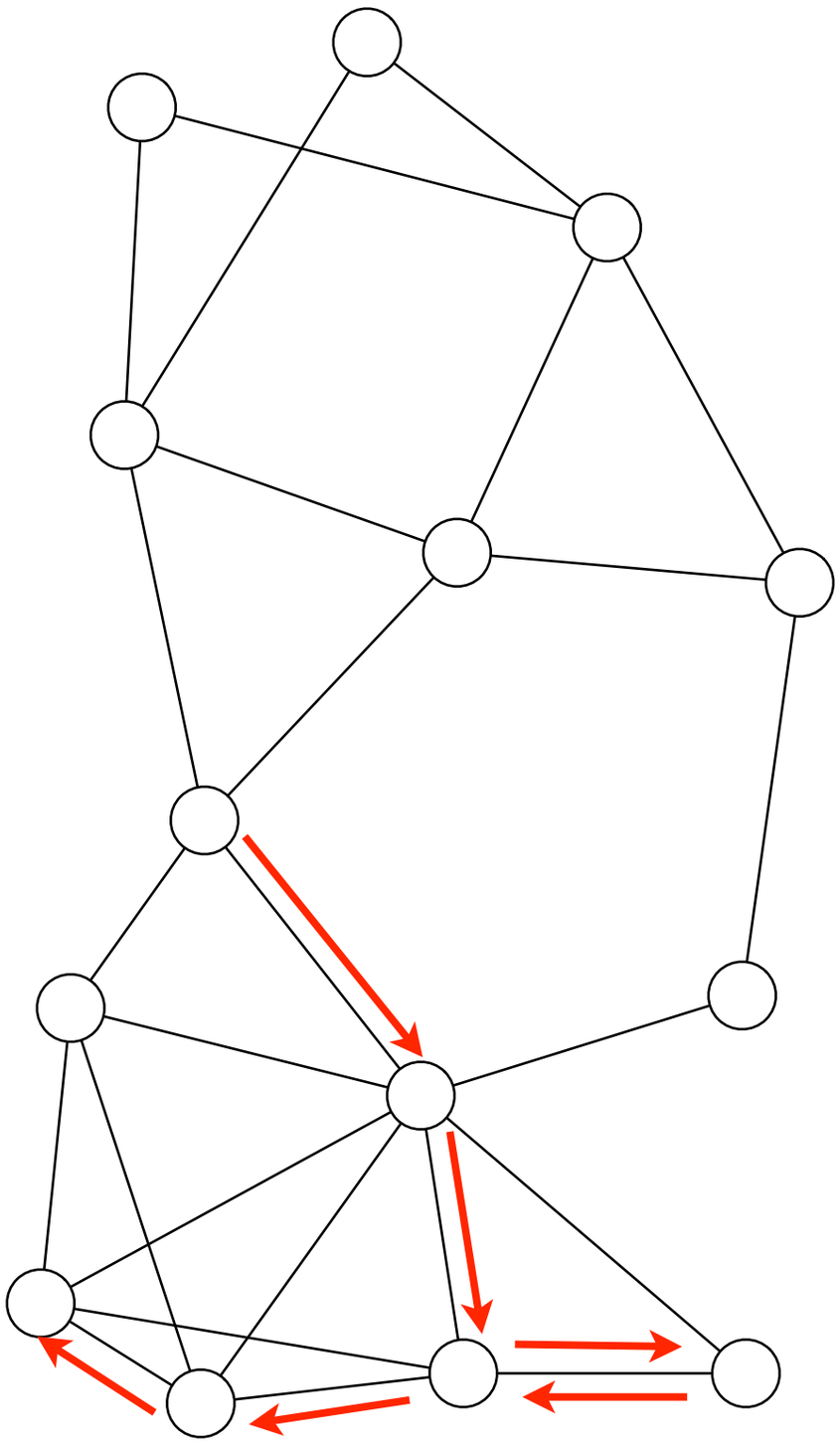


$$\begin{pmatrix}
5 & 2 & 1 & 0 & 2 & \mathbf{3} & \mathbf{3} & 0 & 0 & \mathbf{4} & 0 & 0 \\
2 & 4 & \mathbf{3} & 1 & \mathbf{3} & 1 & 2 & 1 & 0 & 1 & 0 & 0 \\
1 & \mathbf{3} & 4 & 2 & \mathbf{3} & 0 & 1 & 2 & 1 & 0 & 1 & 0 \\
0 & 1 & 2 & 5 & 2 & 0 & 0 & \mathbf{4} & \mathbf{4} & 0 & \mathbf{4} & 2 \\
2 & \mathbf{3} & \mathbf{3} & 2 & 5 & 0 & 2 & 2 & 1 & 1 & 1 & 0 \\
\mathbf{3} & 1 & 0 & 0 & 0 & 3 & 2 & 0 & 0 & \mathbf{3} & 0 & 0 \\
\mathbf{3} & 2 & 1 & 0 & 2 & 2 & 4 & 1 & 0 & \mathbf{3} & 0 & 0 \\
0 & 1 & 2 & \mathbf{4} & 2 & 0 & 1 & 5 & \mathbf{4} & 0 & \mathbf{4} & 2 \\
0 & 0 & 1 & \mathbf{4} & 1 & 0 & 0 & \mathbf{4} & 5 & 0 & \mathbf{5} & \mathbf{3} \\
\mathbf{4} & 1 & 0 & 0 & 1 & \mathbf{3} & \mathbf{3} & 0 & 0 & 4 & 0 & 0 \\
0 & 0 & 1 & \mathbf{4} & 1 & 0 & 0 & \mathbf{4} & \mathbf{5} & 0 & 5 & \mathbf{3} \\
0 & 0 & 0 & 2 & 0 & 0 & 0 & 2 & \mathbf{3} & 0 & \mathbf{3} & 3
\end{pmatrix}$$

Using k=2, $Z_2(u,v) := (A^2)_{uv}$ as the similarity matrix.

Level	Clustering
$\infty \ldots 6$	$\{\text{singletons}(V)\} = \{\ \{1\},\{2\},\{3\},\{4\},\{5\},\{6\},\{7\},\{8\},\{9\},\{10\},\{11\},\{12\}\ \}$
5	$\{\ \{9,11\},\ \{1\},\{2\},\{3\},\{4\},\{5\},\{6\},\{7\},\{8\},\{10\},\{12\}\ \}$
4	$\{\ \{1,10\},\ \{4,8,9,11\},\ \{2\},\{3\},\{5\},\{6\},\{7\},\{12\}\ \}$
3	$\{\{1,6,7,10\}\ \{2,3,5\},\ \{4,8,9,11,12\}\ \}$
$2,1,0$	$\{V\} = \{\{1,2,3,4,5,6,7,8,9,10,11,12\}\}$

Random Walks



On an unweighted graph: Start at a vertex, choose an outgoing edge uniformly at random, walk along that edge, and repeat.

On a weighted graph: Start at a vertex u, choose an incident edge e with weight w_e with probability

$$w_e \;/\; \sum_d w_d$$

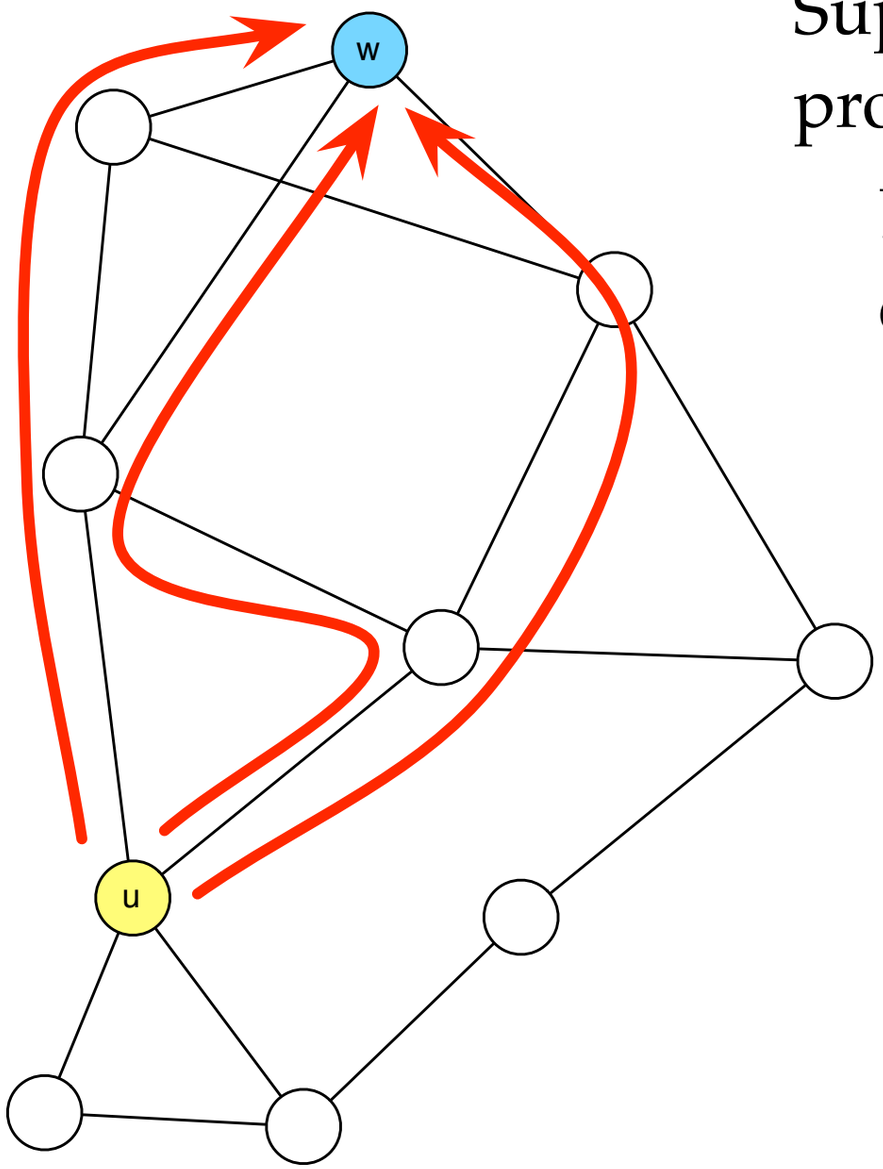
where d ranges over the edges incident to u, walk along that edge, and repeat.

Transition matrix. If A_G is the adjacency matrix of graph G, we form T_G by normalizing each row to sum to 1:

$$T_G = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

$$\sum = 1$$

Random Walks, 2



Suppose you start at u. What's the probability you are at w after 3 steps?

Let v_u be the vector that is 0 everywhere except index u.

At step 0, $v_u[w]$ gives the probability you are at node w.

After 1 step, $(T_G v_u)[w]$ gives the probability that you are at w.

After k steps, the probability that you are at w is:

$$(T_G{}^k v_u)[w]$$

In other words, $T_G{}^k v_u$ is a vector giving our probability of being at any node after taking k steps.

Random Walks for Finding Clusters

$T_G{}^k v_u$ is a vector giving our probability of being at any node after taking k steps and starting from u.

We don't want to choose a starting point. Instead of v_u we could use the vector $v_{uniform}$ with every entry $= 1/n$.

But then for clustering purposes, $v_{uniform}$ just gives a scaling factor, so we can ignore it and focus on $T_G{}^k =: T^k$

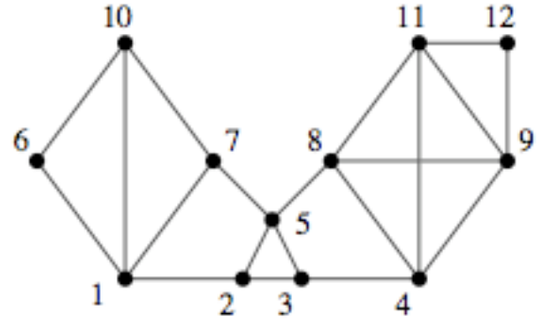
$T^k[i,j]$ gives the probability that we will cross from i to j on step k.

If i, j are in the same dense region, you expect $T^k[i,j]$ to be higher.

Example (van Dongen, 2000)

$$\begin{pmatrix}
0.200 & 0.250 & -- & -- & -- & 0.333 & 0.250 & -- & -- & 0.250 & -- & -- \\
0.200 & 0.250 & 0.250 & -- & 0.200 & -- & -- & -- & -- & -- & -- & -- \\
-- & 0.250 & 0.250 & 0.200 & 0.200 & -- & -- & -- & -- & -- & -- & -- \\
-- & -- & 0.250 & 0.200 & -- & -- & -- & 0.200 & 0.200 & -- & 0.200 & -- \\
-- & 0.250 & 0.250 & -- & 0.200 & -- & 0.250 & 0.200 & -- & -- & -- & -- \\
0.200 & -- & -- & -- & -- & 0.333 & -- & -- & -- & 0.250 & -- & -- \\
0.200 & -- & -- & -- & 0.200 & -- & 0.250 & -- & -- & 0.250 & -- & -- \\
-- & -- & -- & 0.200 & 0.200 & -- & -- & 0.200 & 0.200 & -- & 0.200 & -- \\
-- & -- & -- & 0.200 & -- & -- & -- & 0.200 & 0.200 & -- & 0.200 & 0.333 \\
0.200 & -- & -- & -- & -- & 0.333 & 0.250 & -- & -- & 0.250 & -- & -- \\
-- & -- & -- & 0.200 & -- & -- & -- & 0.200 & 0.200 & -- & 0.200 & 0.333 \\
-- & -- & -- & -- & -- & -- & -- & -- & 0.200 & -- & 0.200 & 0.333
\end{pmatrix}$$

$$M = \mathcal{T}_{G_3+I}$$



$$\begin{pmatrix}
0.096 & 0.096 & 0.096 & 0.096 & 0.096 & 0.096 & 0.096 & 0.096 & 0.096 & 0.096 & 0.096 & 0.096 \\
0.077 & 0.077 & 0.077 & 0.077 & 0.077 & 0.077 & 0.077 & 0.077 & 0.077 & 0.077 & 0.077 & 0.077 \\
0.077 & 0.077 & 0.077 & 0.077 & 0.077 & 0.077 & 0.077 & 0.077 & 0.077 & 0.077 & 0.077 & 0.077 \\
0.096 & 0.096 & 0.096 & 0.096 & 0.096 & 0.096 & 0.096 & 0.096 & 0.096 & 0.096 & 0.096 & 0.096 \\
0.096 & 0.096 & 0.096 & 0.096 & 0.096 & 0.096 & 0.096 & 0.096 & 0.096 & 0.096 & 0.096 & 0.096 \\
0.058 & 0.058 & 0.058 & 0.058 & 0.058 & 0.058 & 0.058 & 0.058 & 0.058 & 0.058 & 0.058 & 0.058 \\
0.077 & 0.077 & 0.077 & 0.077 & 0.077 & 0.077 & 0.077 & 0.077 & 0.077 & 0.077 & 0.077 & 0.077 \\
0.096 & 0.096 & 0.096 & 0.096 & 0.096 & 0.096 & 0.096 & 0.096 & 0.096 & 0.096 & 0.096 & 0.096 \\
0.096 & 0.096 & 0.096 & 0.096 & 0.096 & 0.096 & 0.096 & 0.096 & 0.096 & 0.096 & 0.096 & 0.096 \\
0.077 & 0.077 & 0.077 & 0.077 & 0.077 & 0.077 & 0.077 & 0.077 & 0.077 & 0.077 & 0.077 & 0.077 \\
0.096 & 0.096 & 0.096 & 0.096 & 0.096 & 0.096 & 0.096 & 0.096 & 0.096 & 0.096 & 0.096 & 0.096 \\
0.058 & 0.058 & 0.058 & 0.058 & 0.058 & 0.058 & 0.058 & 0.058 & 0.058 & 0.058 & 0.058 & 0.058
\end{pmatrix}$$

$$M^{\infty}$$

The probability tends to spread out quickly.

Second Key Idea

According to some schedule, apply an "inflation" operator to the matrix.

The affect will be to heighten the contrast between the existing small differences. (As in inflation in cosmology.)

Inflation(M, r) :=

$$\begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \\ p_{41} & p_{42} & p_{43} & p_{44} \end{bmatrix} \mapsto \begin{bmatrix} p_{11}^r & p_{12}^r & p_{13}^r & p_{14}^r \\ p_{21}^r & p_{22}^r & p_{23}^r & p_{24}^r \\ p_{31}^r & p_{32}^r & p_{33}^r & p_{34}^r \\ p_{41}^r & p_{42}^r & p_{43}^r & p_{44}^r \end{bmatrix} \mapsto$$

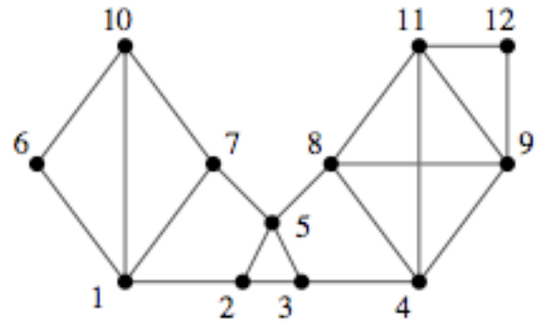
Rescale columns

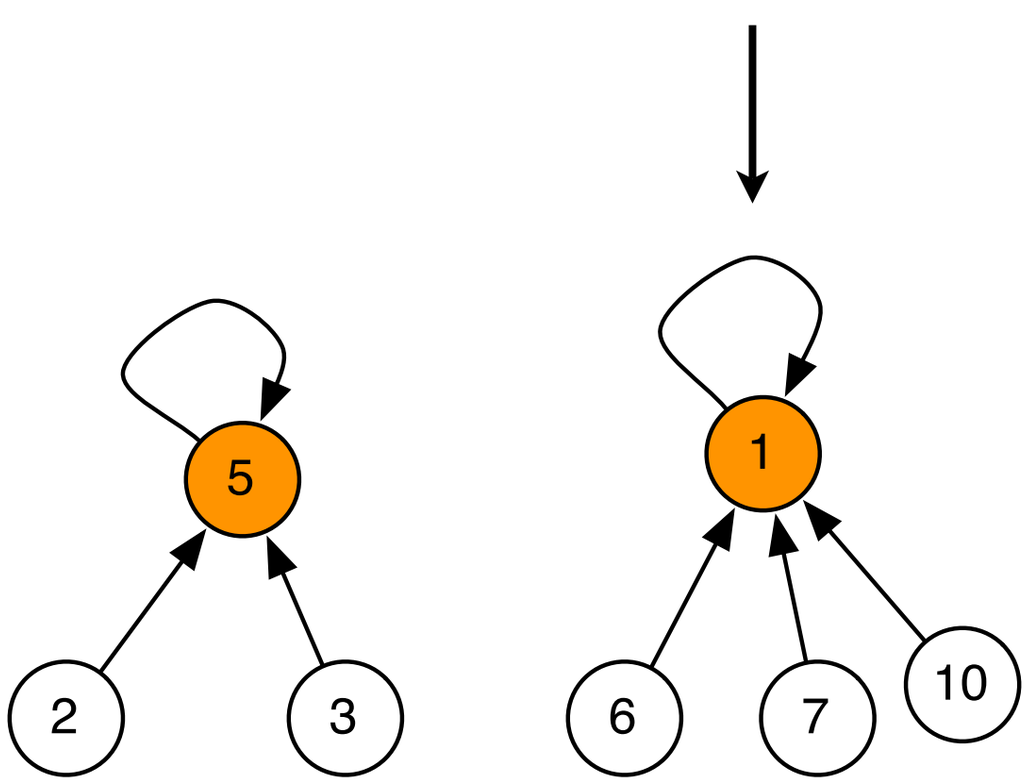
Examples. (r=2)

$$\begin{pmatrix} 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \end{pmatrix} \mapsto \begin{pmatrix} 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \end{pmatrix} \qquad \begin{pmatrix} 0.25 \\ 0.25 \\ 0.25 \\ 0 \end{pmatrix} \mapsto \begin{pmatrix} 0.25 \\ 0.25 \\ 0.25 \\ 0 \end{pmatrix} \qquad \begin{pmatrix} 0.3 \\ 0.3 \\ 0.2 \\ 0.2 \end{pmatrix} \mapsto \begin{pmatrix} 0.346 \\ 0.346 \\ 0.154 \\ 0.154 \end{pmatrix}$$

Example.

$$M_{mci}^{\infty} = \begin{pmatrix}
1.000 & -- & -- & -- & -- & 1.000 & 1.000 & -- & -- & 1.000 & -- & -- \\
-- & -- & -- & -- & -- & -- & -- & -- & -- & -- & -- & -- \\
-- & -- & -- & -- & -- & -- & -- & -- & -- & -- & -- & -- \\
-- & -- & -- & -- & -- & -- & -- & -- & -- & -- & -- & -- \\
-- & 1.000 & 1.000 & -- & 1.000 & -- & -- & -- & -- & -- & -- & -- \\
-- & -- & -- & -- & -- & -- & -- & -- & -- & -- & -- & -- \\
-- & -- & -- & -- & -- & -- & -- & -- & -- & -- & -- & -- \\
-- & -- & -- & -- & -- & -- & -- & -- & -- & -- & -- & -- \\
-- & -- & -- & 0.500 & -- & -- & -- & 0.500 & 0.500 & -- & 0.500 & 0.500 \\
-- & -- & -- & -- & -- & -- & -- & -- & -- & -- & -- & -- \\
-- & -- & -- & 0.500 & -- & -- & -- & 0.500 & 0.500 & -- & 0.500 & 0.500 \\
-- & -- & -- & -- & -- & -- & -- & -- & -- & -- & -- & --
\end{pmatrix}$$





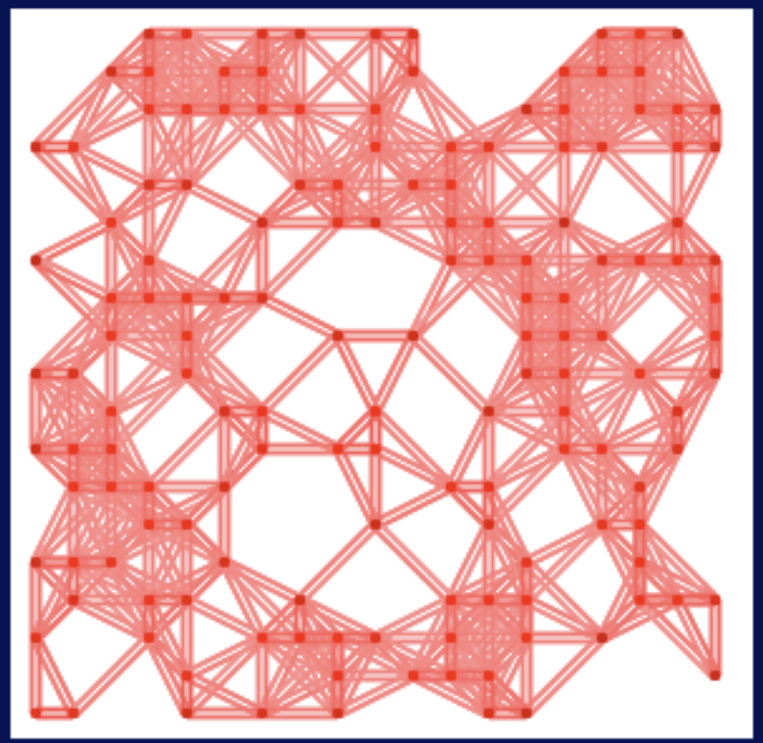
Attractors: nodes with positive return probability.

The algorithm

```
MCL(G, {e_i}, {r_i}):
  # Input:
  #   Graph G,
  #   sequence of powers e_i, and
  #   sequence of inflation parameters r_k

  Add weighted loops to G and compute T_{G,1} =: T_1
  for k = 1,...,∞:
      T := Inflate(r_k, Power(e_k, T))
      if T ≈ T^2 then break;
Treat T as the adjacency matrix of a directed graph.
return the weakly connected components of T.
```

Weakly connected components = some strongly connected component + the nodes that can reach it

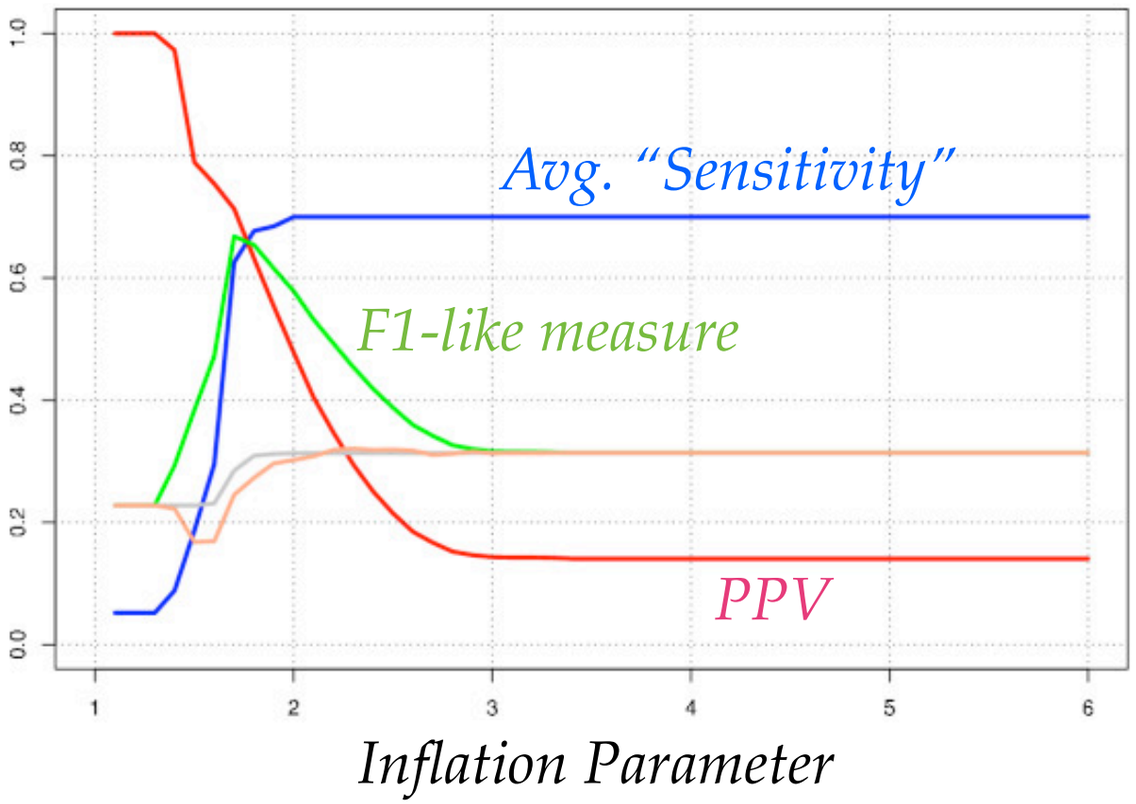
MCL animation



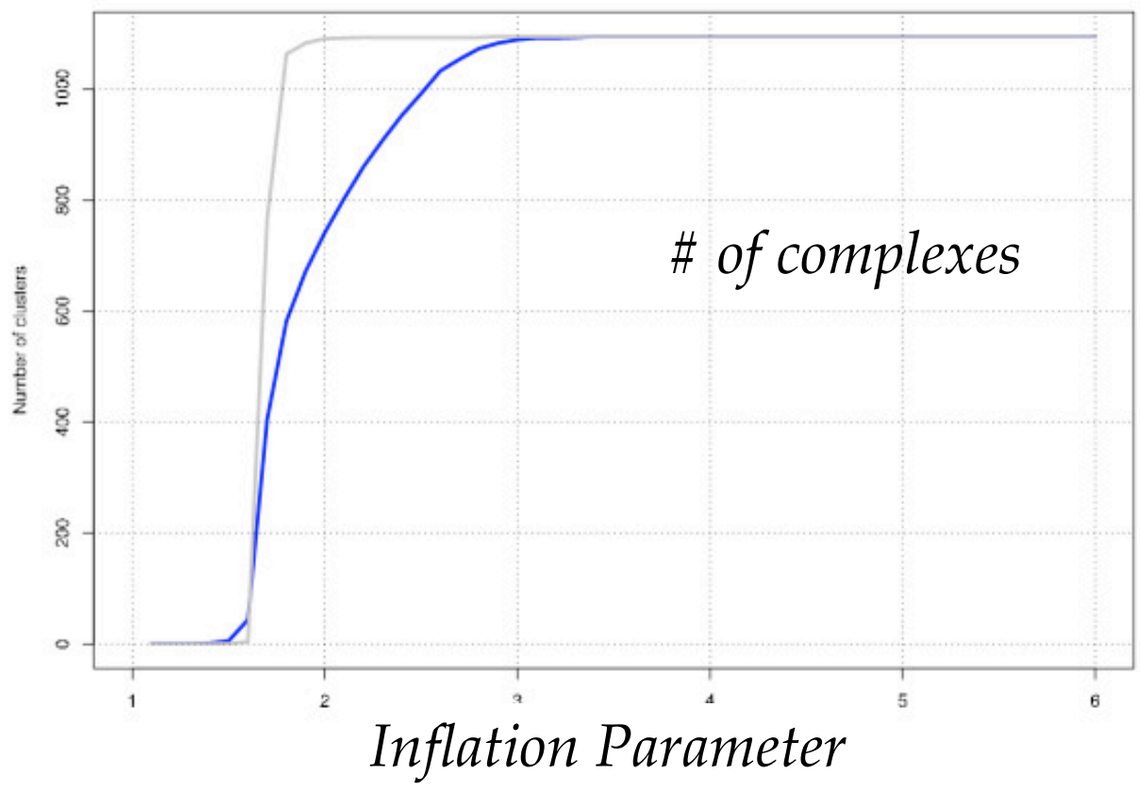
```
   s/d       step forward/back
 h/j/k       halt/play/rewind
   a/z       start/end
   8         set frame speed in milliseconds

   JavaScript:     should be enabled
typeaheadfind:     should be disabled (Firefox)

   Tested on Firefox, Safari, IE. Preferably use something
   else than IE.
```

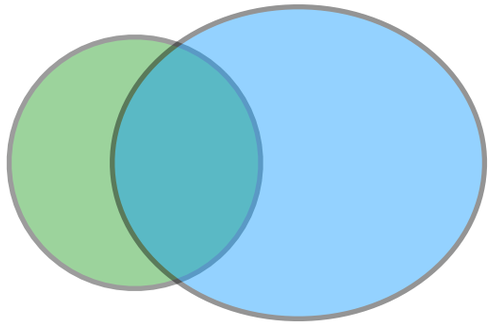
Impact of Inflation Parameter on $A_{100,40}$

(Brohee and van Helden, 2006)



(D)





(complex-wise) "Sensitivity" := % complex covered by its best matching cluster.

(cluster-wise) PPV is % cluster covered by its best matching complex.

F1-like measure: sqrt(PPV × Sensitivity)

Implementation

- As written, the algorithm requires $O(N^2)$ space to store the matrix
- It requires $O(N^3)$ time if the number of rounds is considered constant (not unreasonable in practice, as convergence tends to be fast).
- This is generally too slow for very large graphs.
- Solution: **Pruning**
  - **Exact pruning:** keep the m largest entries in each column [matrix multiplication becomes $O(Nm^2)$
  - **Threshold pruning:** keep entries that are above a threshold.
  - Threshold is faster than exact pruning in practice

Summary

- MCL is a very successful graph clustering approach.
- Draws on intuition from random walks / "flow"
- But random walks tend to spread out over time
  (The same was true for Functional Flow)
- Inflation operator inhibits hits flatten of probabilities.
- Input parameters: powers and inflation coefficients.
- Overlapping clusters *may* be produced: the weakly connected components may overlap. This tends not to happen in practice because it is "unstable."
  - What's a heuristic way to avoid overlapping clusters if you get them?

Summary of Function Prediction

- Functional flow
- Network neighborhood function prediction (majority, enrichment, etc.)
- Entropy / mutual information / variation of information
- Notion of node similarity (edge / node betweenness, dice distance, edge clustering coefficient)
- Graph partitioning algorithms:
  - Minimum multiway cut / integer programming
  - **Graph summarization**
  - Modularity, Girvan-Newman algorithm, Newman spectral-based partitioning
  - VI-CUT
  - RNSC
  - MCODE
  - **MCL (random walks, k-paths clustering)**
  - k-cores, k-bonds, k-components