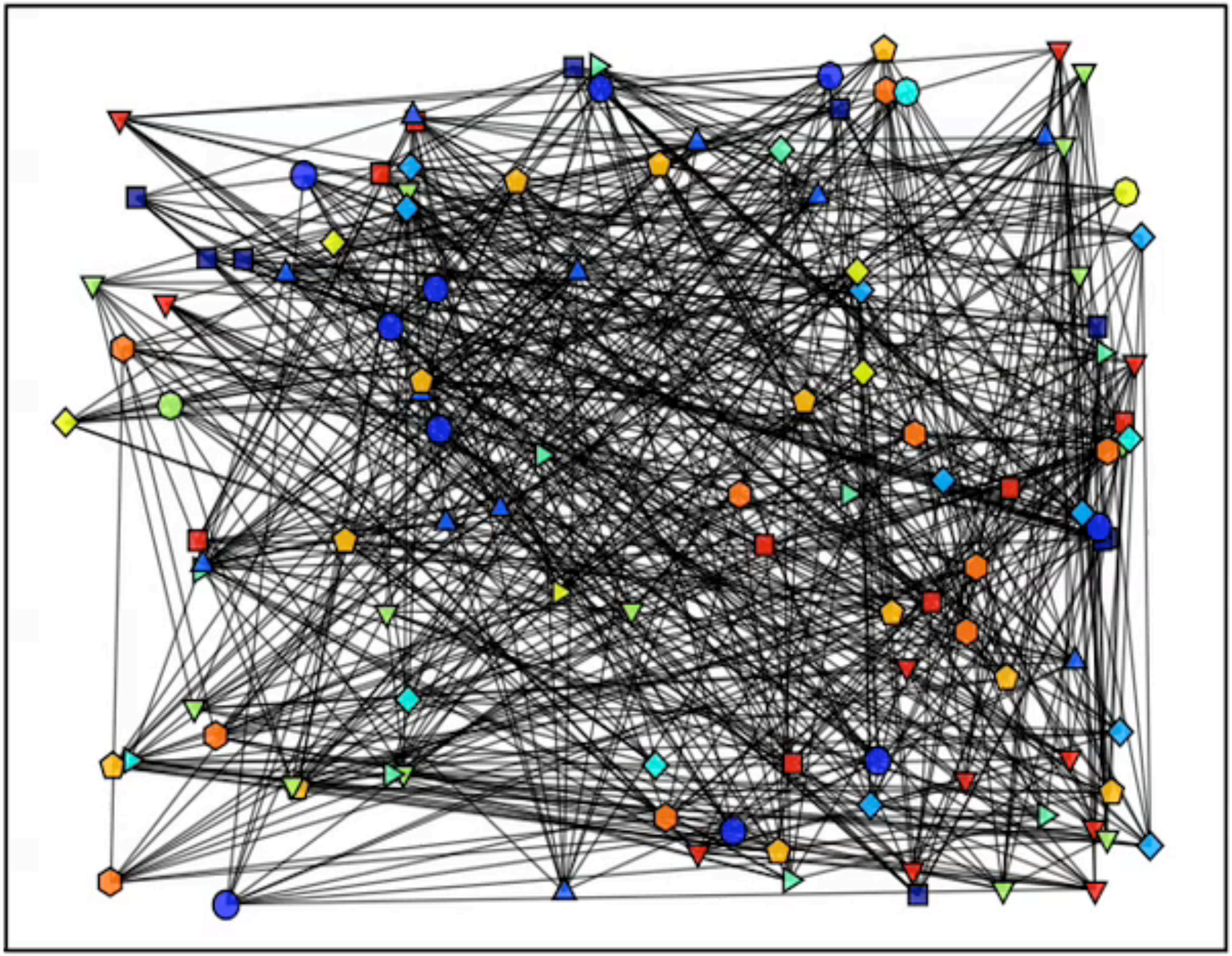
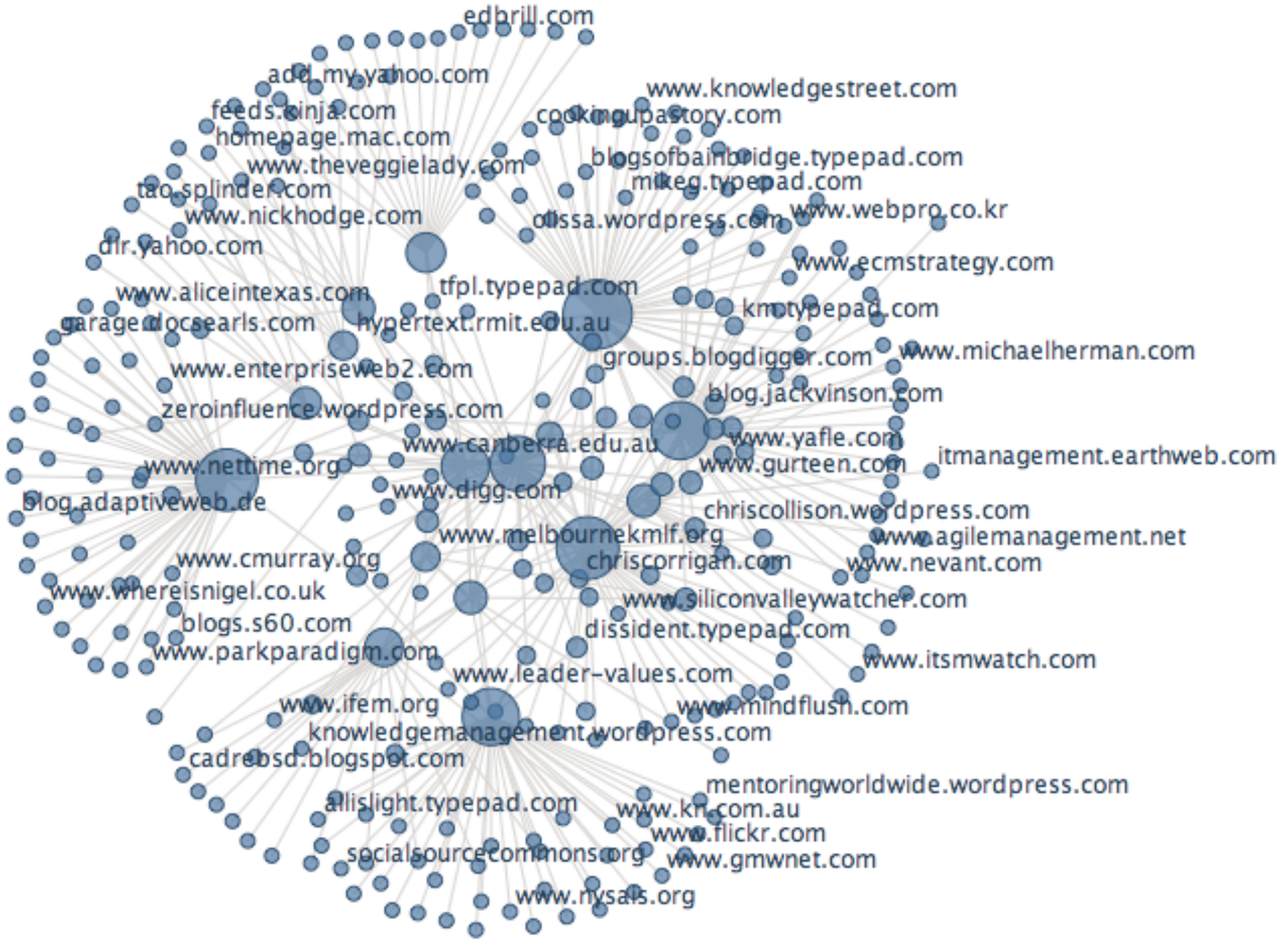


A Bayesian approach to network modularity

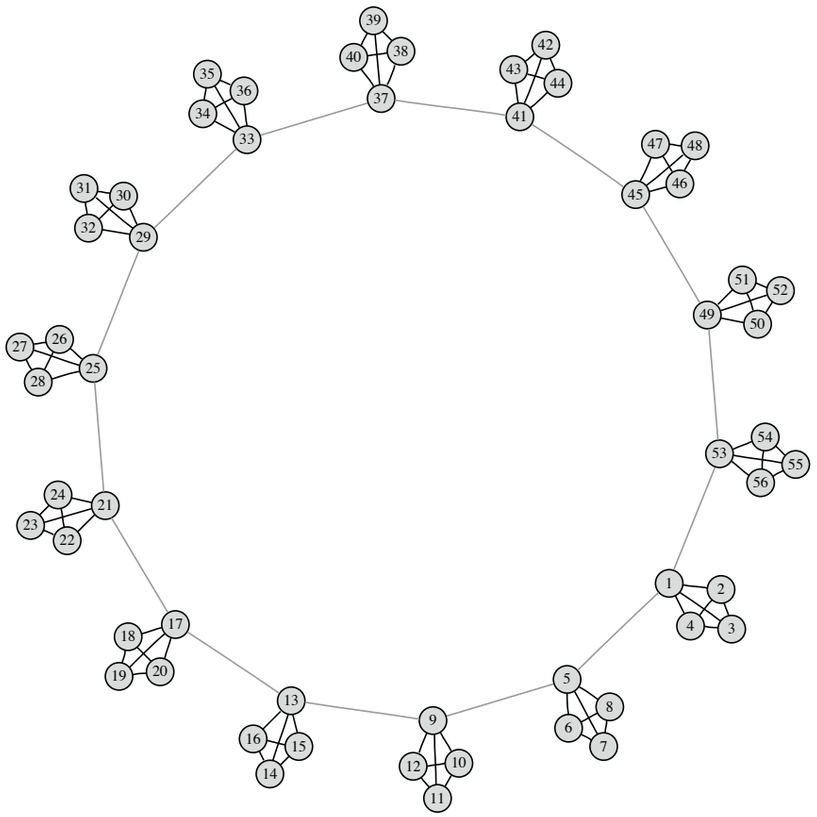
Jake Hofman
Wiggins Group
Columbia University
2008.03.12

Network modularity (community detection)



Assign nodes to modules?

The “resolution limit” problem

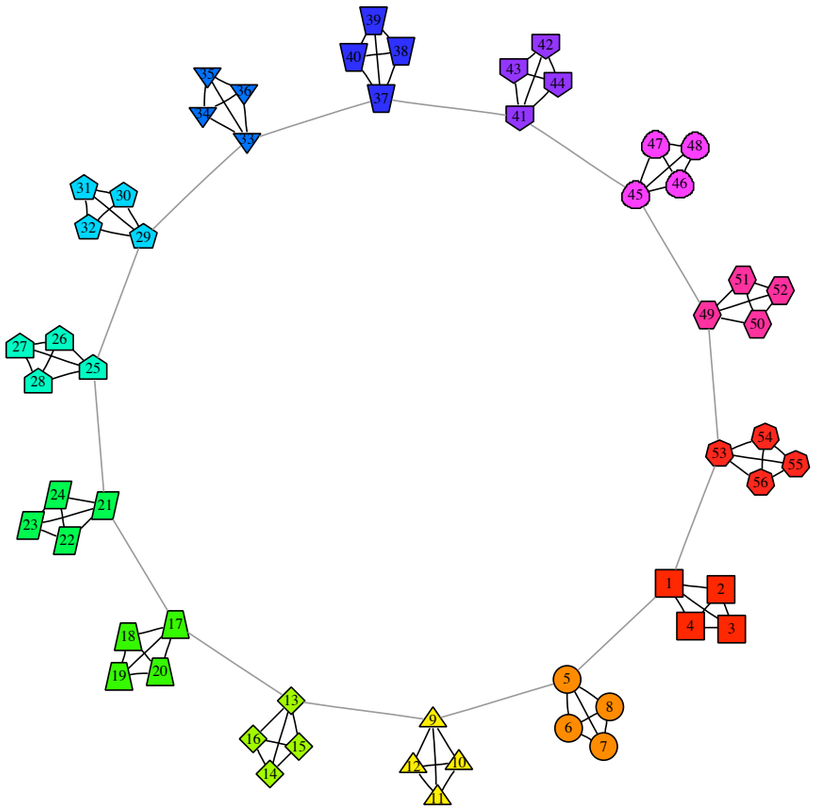


$$\mathcal{H}_{\mathrm{RB}} = -\sum_{ij}(A_{ij} - \gamma p_{ij})\delta_{z_i,z_j}$$

Girvan & Newman (2004), Reichardt & Bornholdt (2006)

Fortunato et. al. (2007), Kumpula et. al. (2007)

The “resolution limit” problem

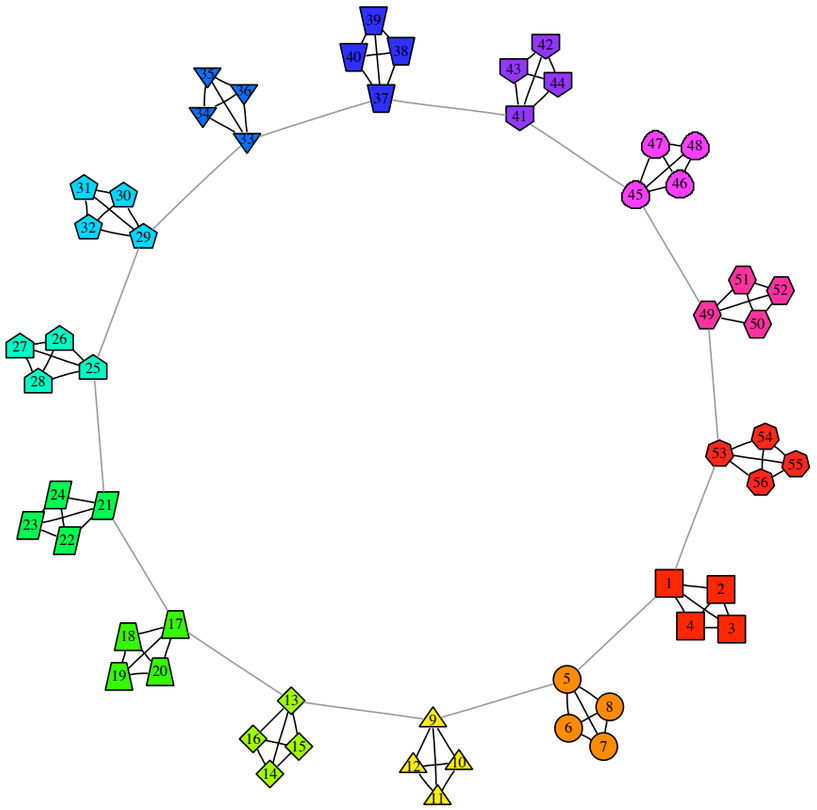


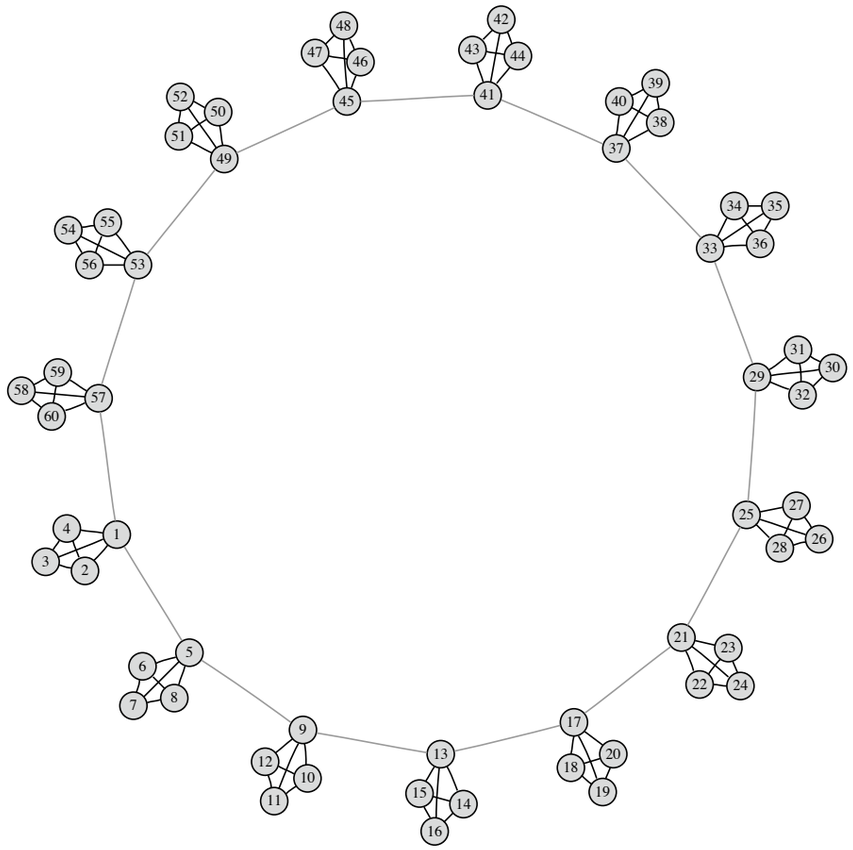
$$\mathcal{H}_{\mathrm{RB}} = -\sum_{ij}(A_{ij} - \gamma p_{ij})\delta_{z_i,z_j}$$

Girvan & Newman (2004), Reichardt & Bornholdt (2006)

Fortunato et. al. (2007), Kumpula et. al. (2007)

The “resolution limit” problem





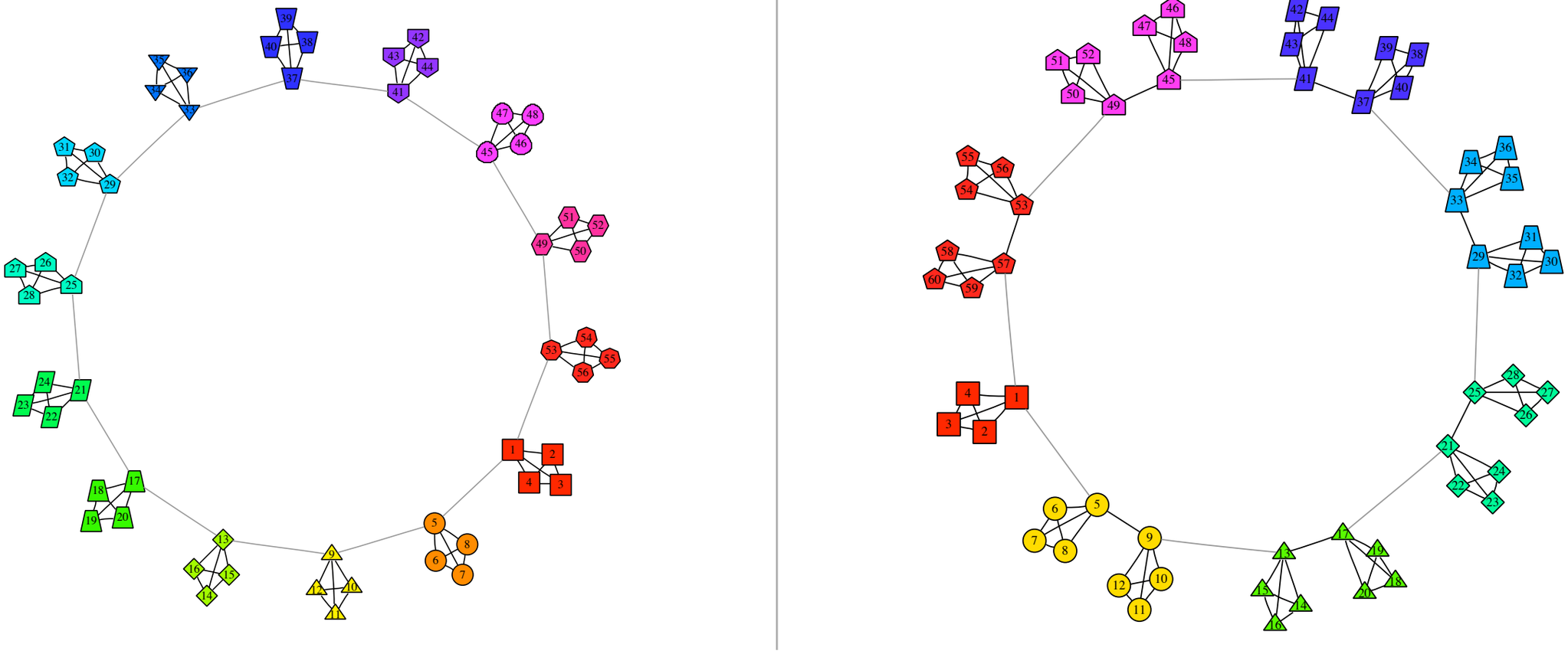
$$\mathcal{H}_{\mathrm{RB}} = -\sum_{ij}(A_{ij} - \gamma p_{ij})\delta_{z_i,z_j}$$

Girvan & Newman (2004), Reichardt & Bornholdt (2006)

Fortunato et. al. (2007), Kumpula et. al. (2007)

The “resolution limit” problem

Fixed parameters -> fixed resolution or *complexity*



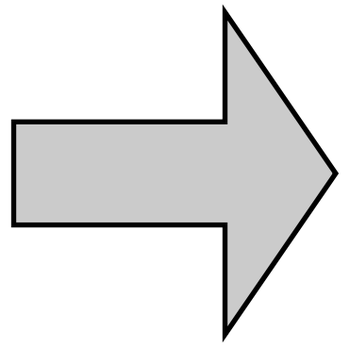
$$\mathcal{H}_{\mathrm{RB}} = -\sum_{ij}(A_{ij} - \gamma p_{ij})\delta_{z_i,z_j}$$

Girvan & Newman (2004), Reichardt & Bornholdt (2006)

Fortunato et. al. (2007), Kumpula et. al. (2007)

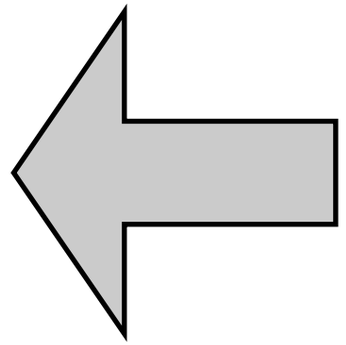
Community detection as inference

Know ensemble (parameters, assignment variables, complexity)



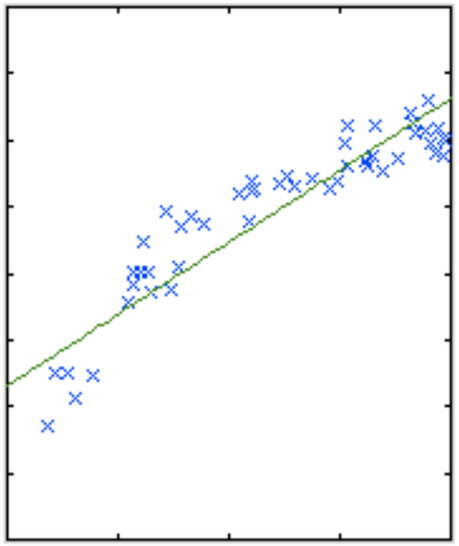
Sample microstates

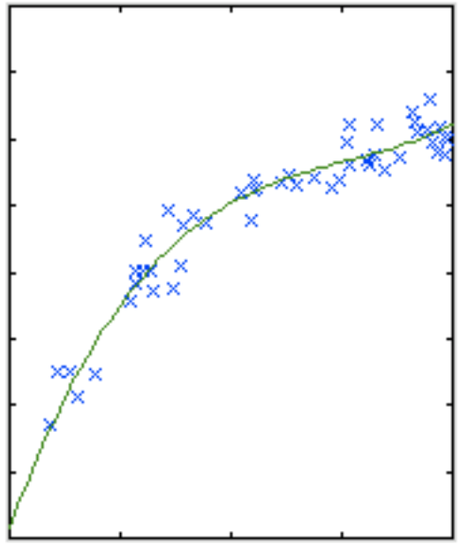
Infer ensemble (parameters, latent variables, complexity)

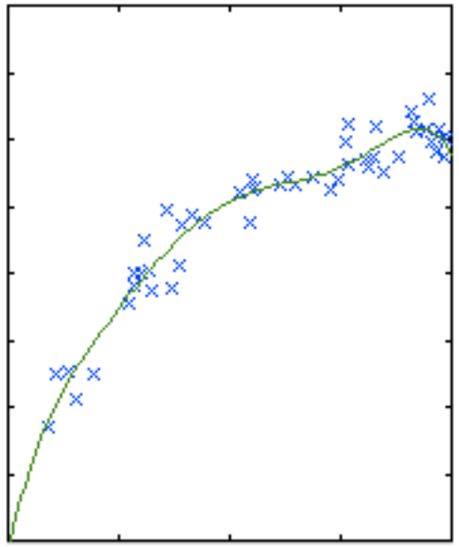


Observe microstate

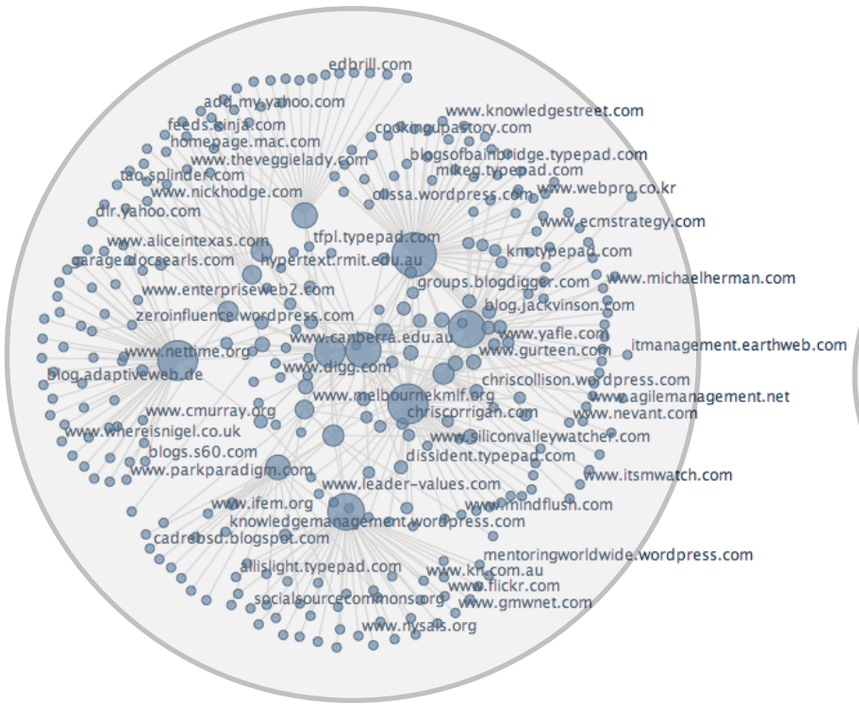
Complexity control in probabilistic models

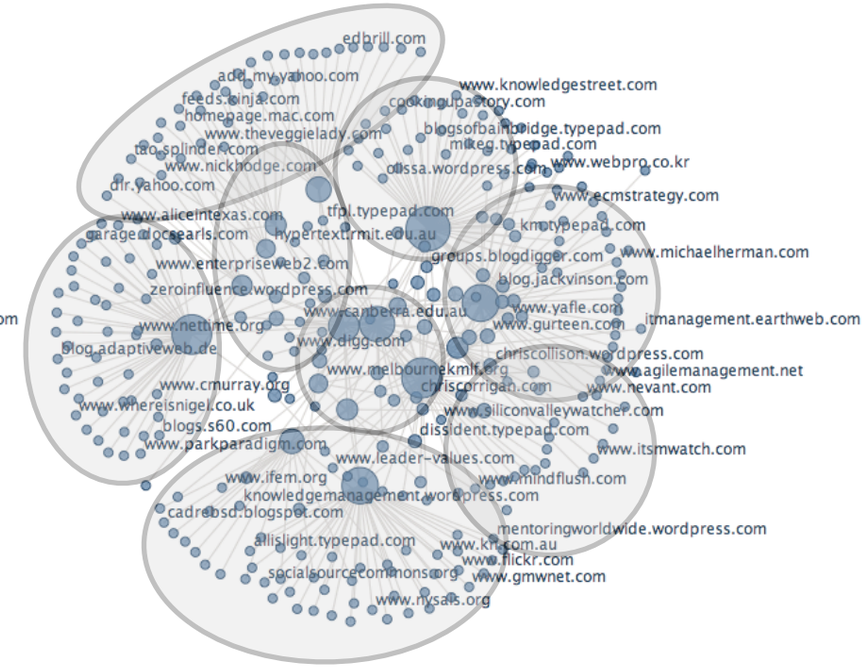


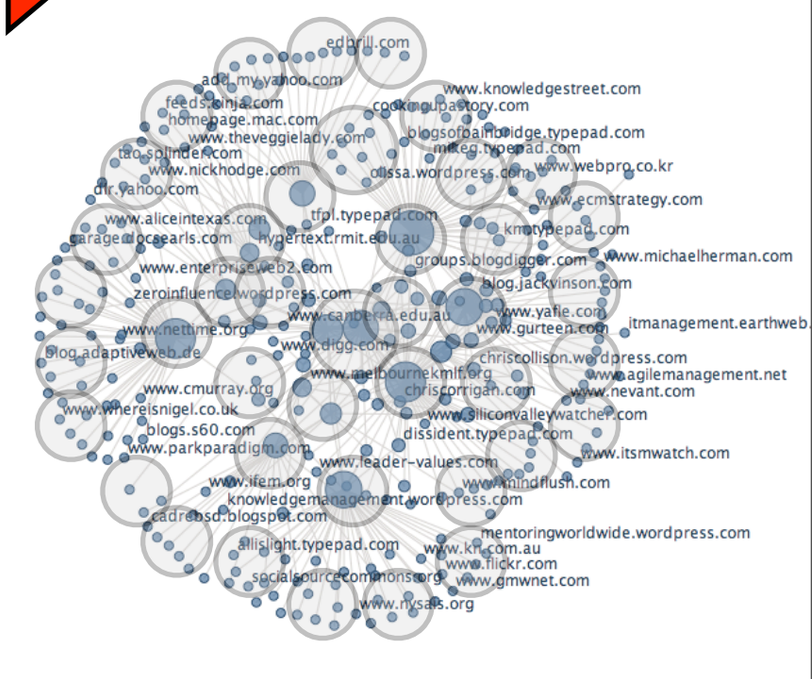




Increasing complexity

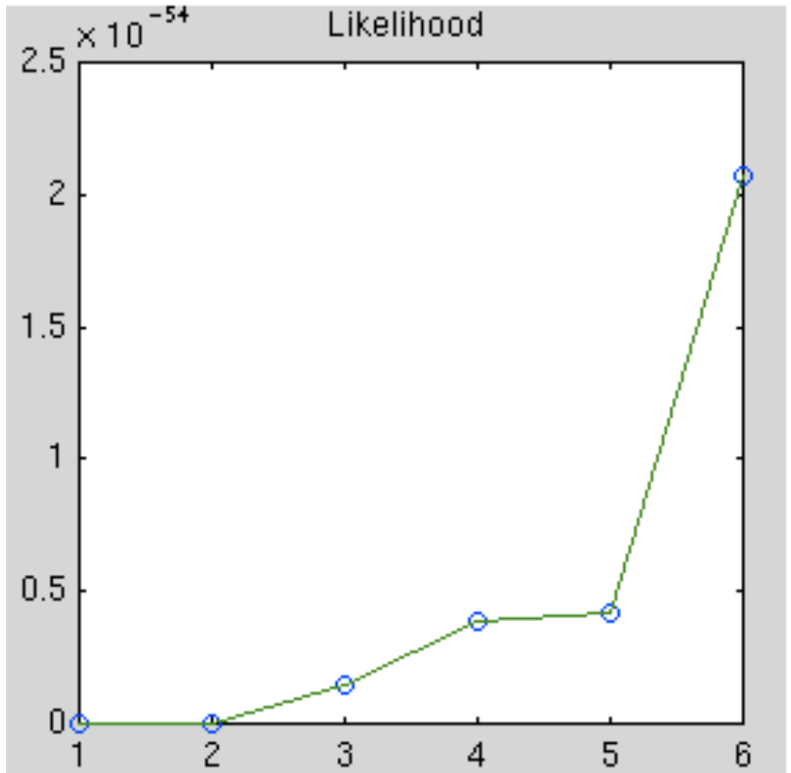




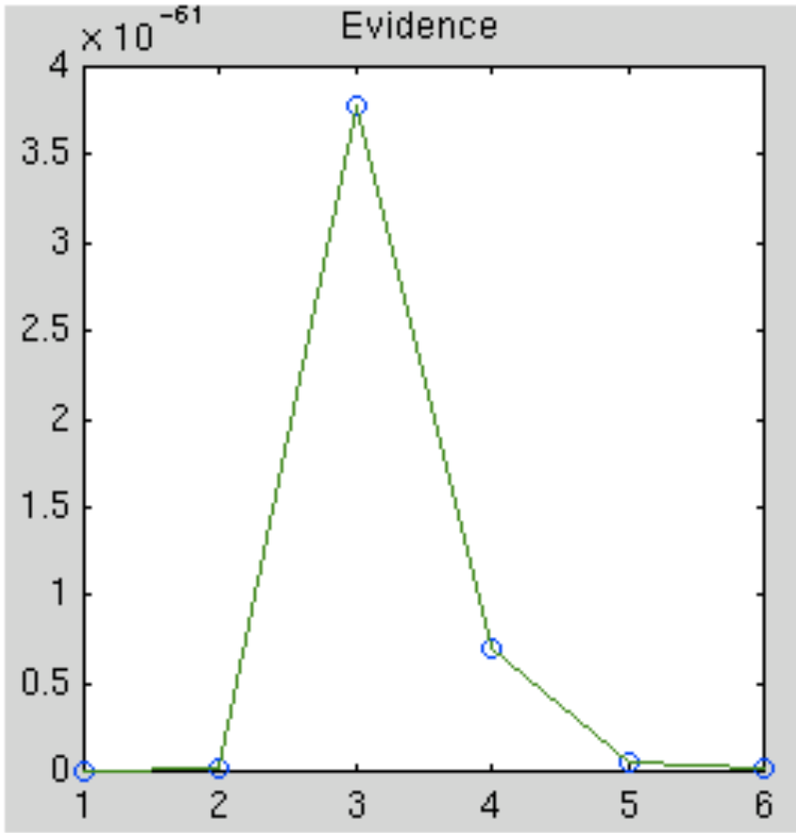


Complexity control in probabilistic models

- Maximize evidence (integrating over unknown parameters) to infer most probable model complexity



$$p(\mathcal{D}|\hat{\theta}, K)$$



$$p(\mathcal{D}|K) = \int d\theta \; p(\mathcal{D}|\theta, K)p(\theta|K)$$

http://research.microsoft.com/~minka/statlearn/demo/

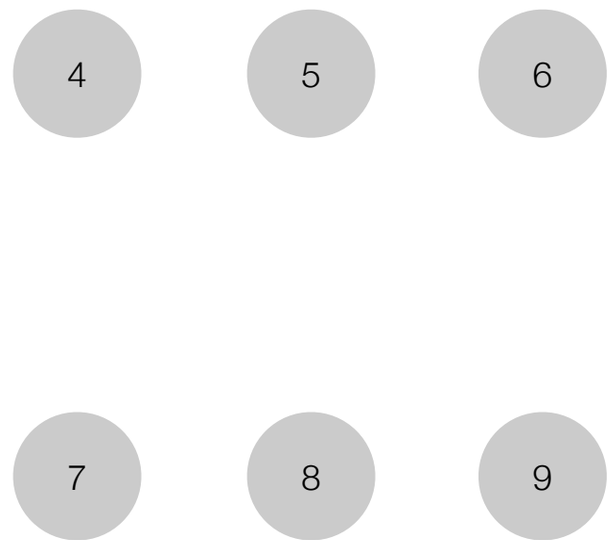
Generating modular networks

- For each node:
  - **Roll K-sided die** with bias π to determine $z_i=1,...,K$, the (unobserved) module assignment for i^{th} node
- For each pair of nodes (i,j):
  - If $z_i=z_j$, **flip "in community" coin** with bias θ_c to determine edge A_{ij}
  - If $z_i \neq z_j$, **flip "between communities" coin** with bias θ_d to determine edge A_{ij}

Stochastic block models (Holland, Laskey, Leinhardt 1983; Wang and Wong, 1987)

Generating modular networks

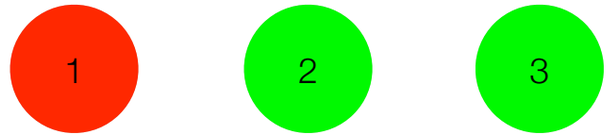
- For each node:
  - **Roll K-sided die** with bias π to determine $z_i=1,...,K$, the (unobserved) module assignment for i^{th} node
- For each pair of nodes (i,j):
  - If $z_i=z_j$, **flip "in community" coin** with bias θ_c to determine edge A_{ij}
  - If $z_i \neq z_j$, **flip "between communities" coin** with bias θ_d to determine edge A_{ij}



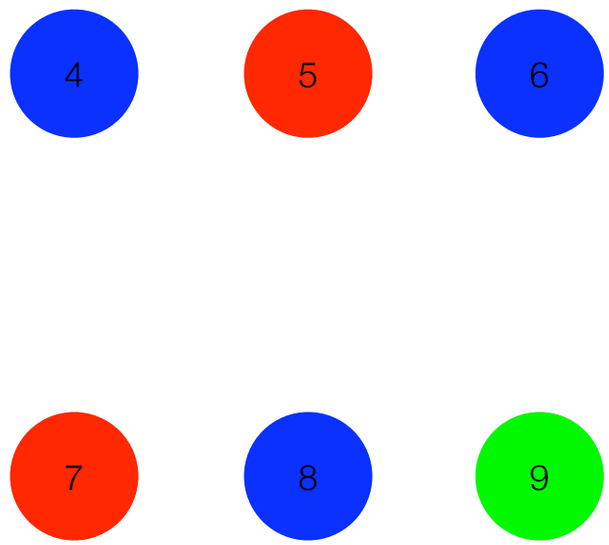
Stochastic block models (Holland, Laskey, Leinhardt 1983; Wang and Wong, 1987)

Generating modular networks

- For each node:
  - **Roll K-sided die** with bias π to determine z_i=1,...,K, the (unobserved) module assignment for i^{th} node



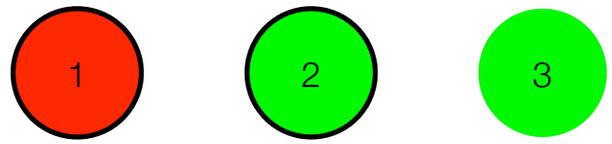
- For each pair of nodes (i,j):
  - If z_i=z_j, **flip "in community" coin** with bias θ_c to determine edge A_{ij}
  - If z_i≠z_j, **flip "between communities" coin** with bias θ_d to determine edge A_{ij}



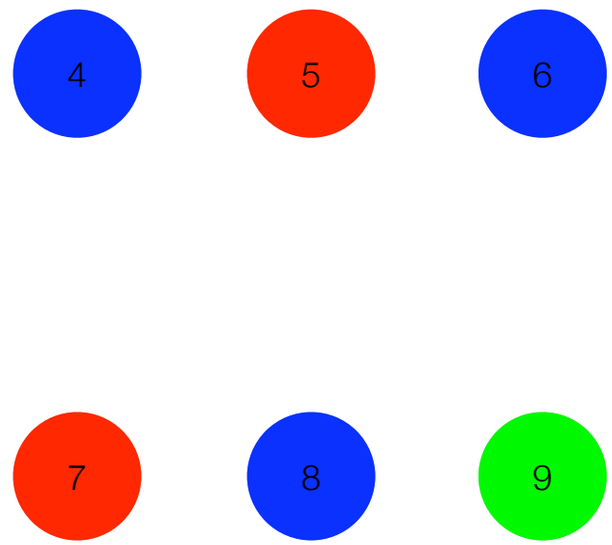
Stochastic block models (Holland, Laskey, Leinhardt 1983; Wang and Wong, 1987)

Generating modular networks

- For each node:
  - **Roll K-sided die** with bias π to determine z_i=1,...,K, the (unobserved) module assignment for i^{th} node



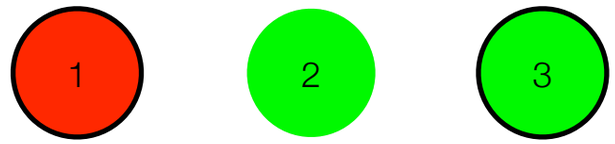
- For each pair of nodes (i,j):
  - If z_i=z_j, **flip "in community" coin** with bias θ_c to determine edge A_{ij}
  - If z_i≠z_j, **flip "between communities" coin** with bias θ_d to determine edge A_{ij}



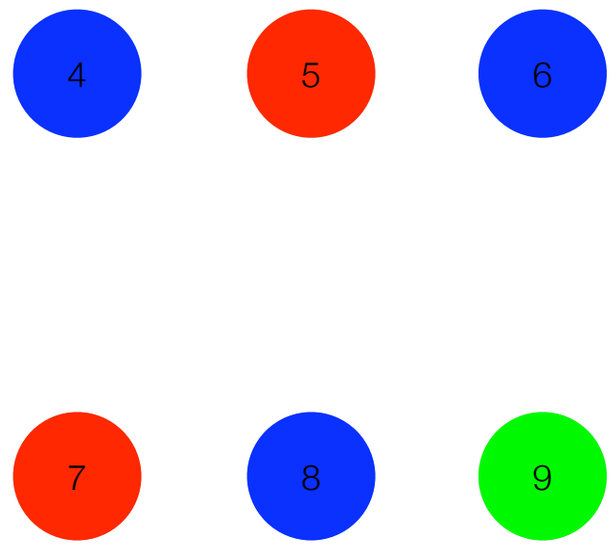
Stochastic block models (Holland, Laskey, Leinhardt 1983; Wang and Wong, 1987)

Generating modular networks

- For each node:
  - **Roll K-sided die** with bias π to determine z_i=1,...,K, the (unobserved) module assignment for i^{th} node



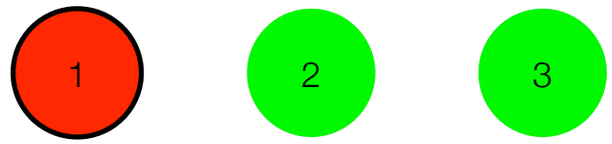
- For each pair of nodes (i,j):
  - If z_i=z_j, **flip "in community" coin** with bias θ_c to determine edge A_{ij}
  - If z_i≠z_j, **flip "between communities" coin** with bias θ_d to determine edge A_{ij}



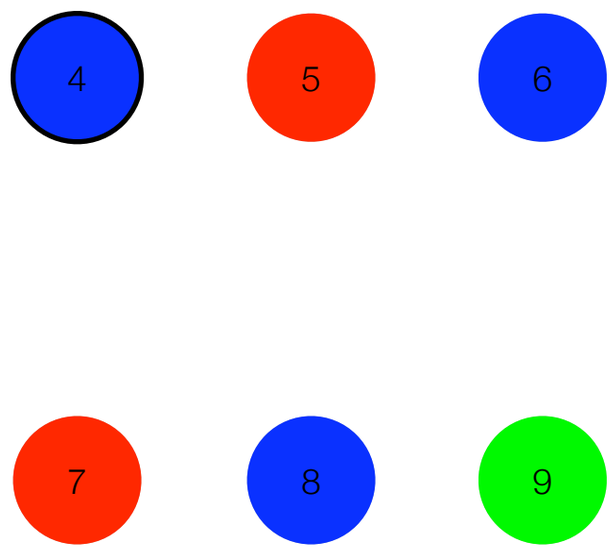
Stochastic block models (Holland, Laskey, Leinhardt 1983; Wang and Wong, 1987)

Generating modular networks

- For each node:
  - **Roll K-sided die** with bias π to determine z_i=1,...,K, the (unobserved) module assignment for i^{th} node



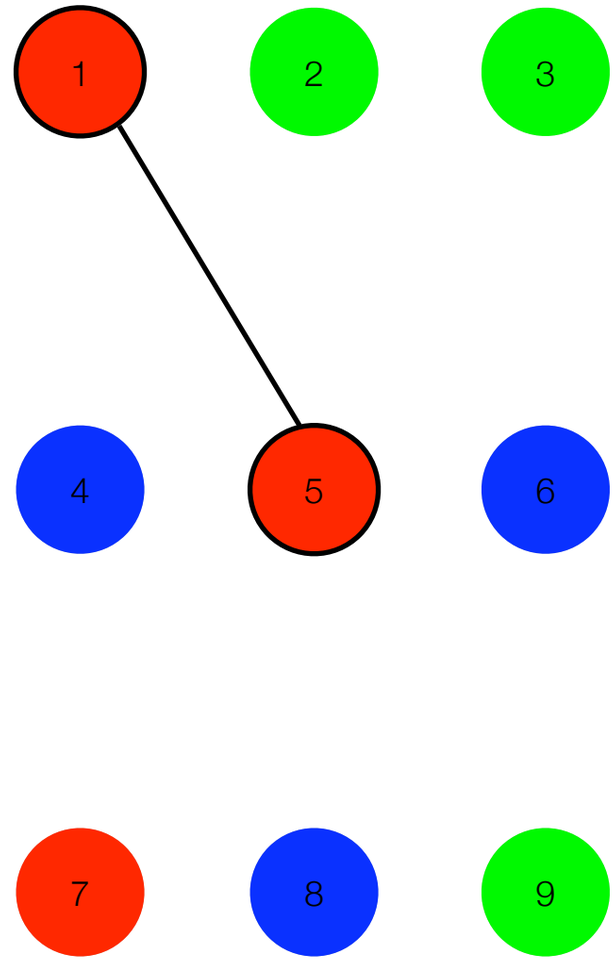
- For each pair of nodes (i,j):
  - If z_i=z_j, **flip "in community" coin** with bias θ_c to determine edge A_{ij}
  - If z_i≠z_j, **flip "between communities" coin** with bias θ_d to determine edge A_{ij}



Stochastic block models (Holland, Laskey, Leinhardt 1983; Wang and Wong, 1987)

Generating modular networks

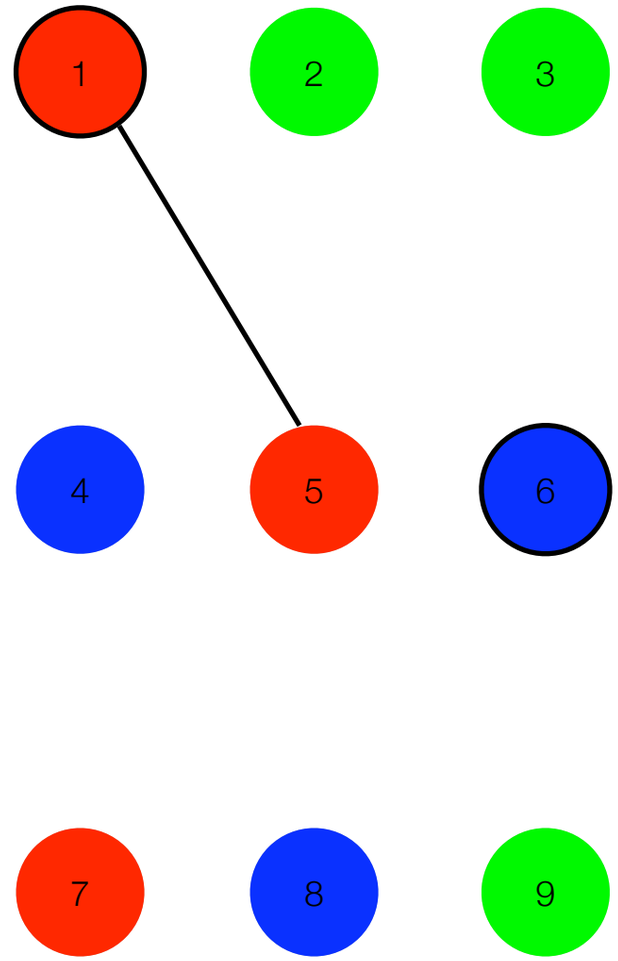
- For each node:
  - **Roll K-sided die** with bias π to determine z_i=1,...,K, the (unobserved) module assignment for i^{th} node
- For each pair of nodes (i,j):
  - If z_i=z_j, **flip "in community" coin** with bias θ_c to determine edge A_{ij}
  - If z_i≠z_j, **flip "between communities" coin** with bias θ_d to determine edge A_{ij}



Stochastic block models (Holland, Laskey, Leinhardt 1983; Wang and Wong, 1987)

Generating modular networks

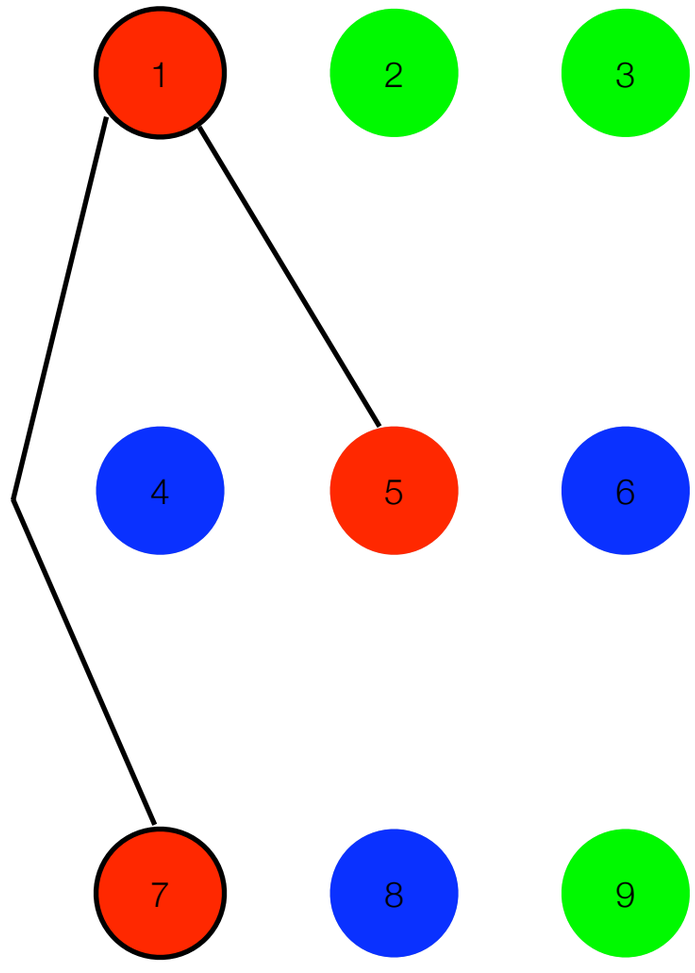
- For each node:
  - **Roll K-sided die** with bias π to determine z_i=1,...,K, the (unobserved) module assignment for i^{th} node
- For each pair of nodes (i,j):
  - If z_i=z_j, **flip "in community" coin** with bias θ_c to determine edge A_{ij}
  - If z_i≠z_j, **flip "between communities" coin** with bias θ_d to determine edge A_{ij}



Stochastic block models (Holland, Laskey, Leinhardt 1983; Wang and Wong, 1987)

Generating modular networks

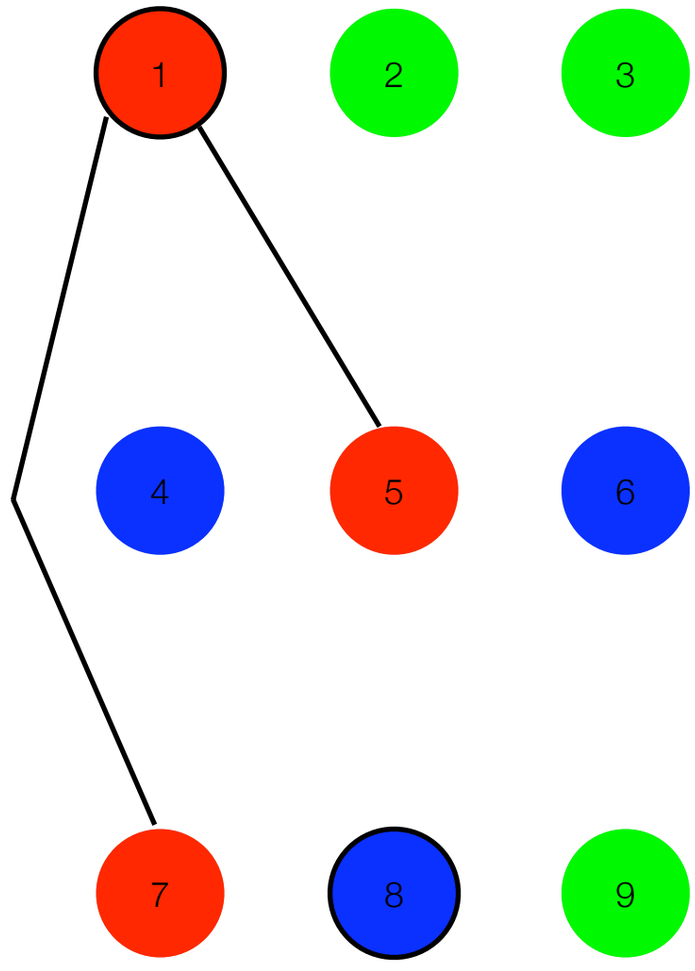
- For each node:
  - **Roll K-sided die** with bias π to determine z_i=1,...,K, the (unobserved) module assignment for i^{th} node
- For each pair of nodes (i,j):
  - If z_i=z_j, **flip "in community" coin** with bias θ_c to determine edge A_{ij}
  - If z_i≠z_j, **flip "between communities" coin** with bias θ_d to determine edge A_{ij}



Stochastic block models (Holland, Laskey, Leinhardt 1983; Wang and Wong, 1987)

Generating modular networks

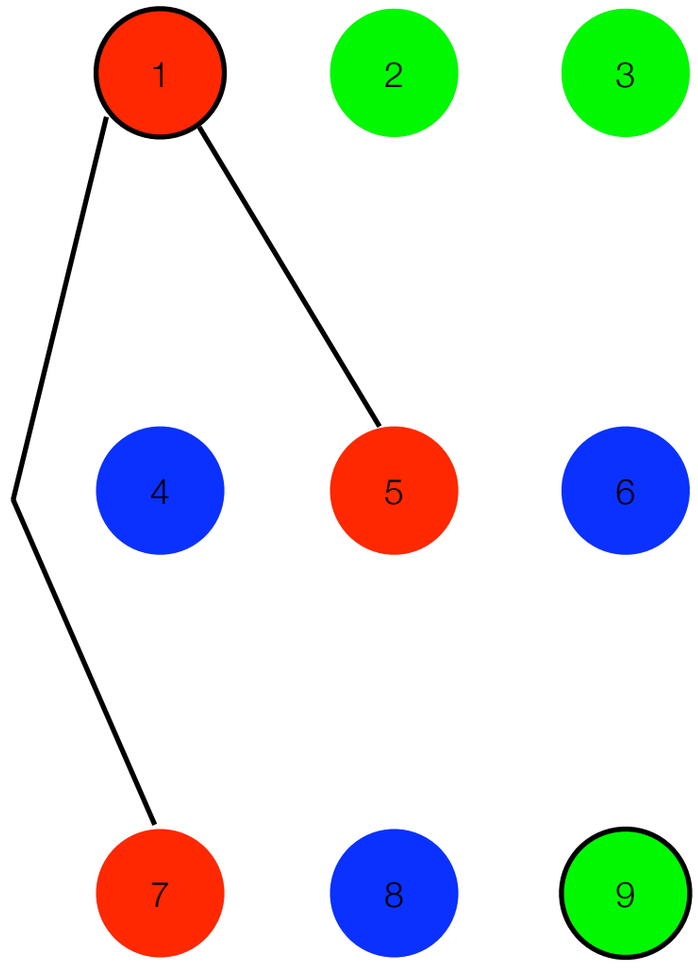
- For each node:
  - **Roll K-sided die** with bias π to determine z_i=1,...,K, the (unobserved) module assignment for i^{th} node
- For each pair of nodes (i,j):
  - If z_i=z_j, **flip "in community" coin** with bias θ_c to determine edge A_{ij}
  - If z_i≠z_j, **flip "between communities" coin** with bias θ_d to determine edge A_{ij}



Stochastic block models (Holland, Laskey, Leinhardt 1983; Wang and Wong, 1987)

Generating modular networks

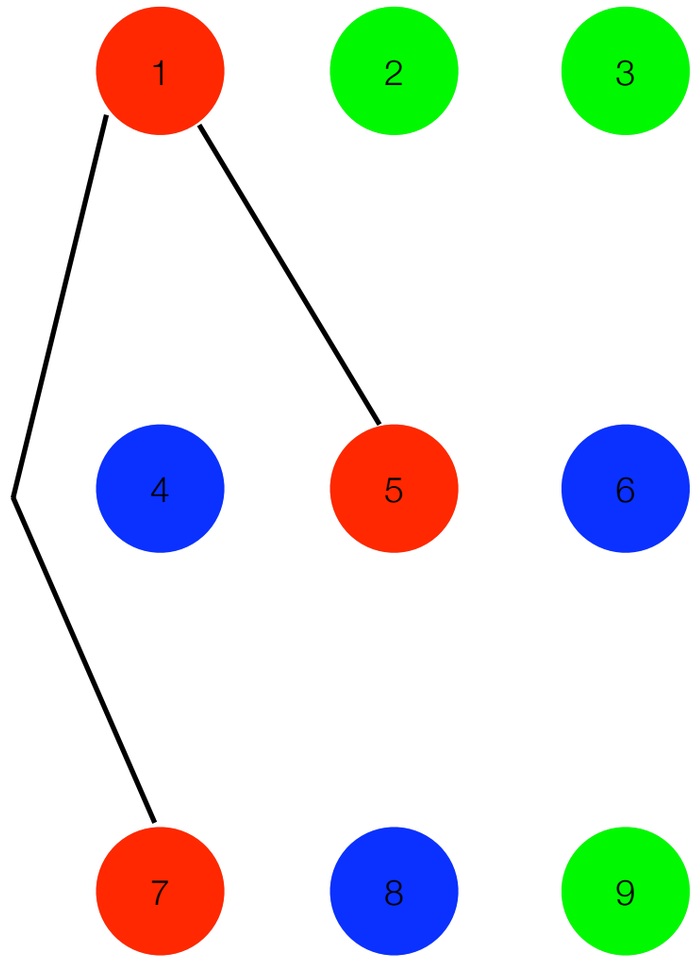
- For each node:
  - **Roll K-sided die** with bias π to determine z_i=1,...,K, the (unobserved) module assignment for i^{th} node
- For each pair of nodes (i,j):
  - If z_i=z_j, **flip "in community" coin** with bias θ_c to determine edge A_{ij}
  - If z_i≠z_j, **flip "between communities" coin** with bias θ_d to determine edge A_{ij}



Stochastic block models (Holland, Laskey, Leinhardt 1983; Wang and Wong, 1987)

Generating modular networks

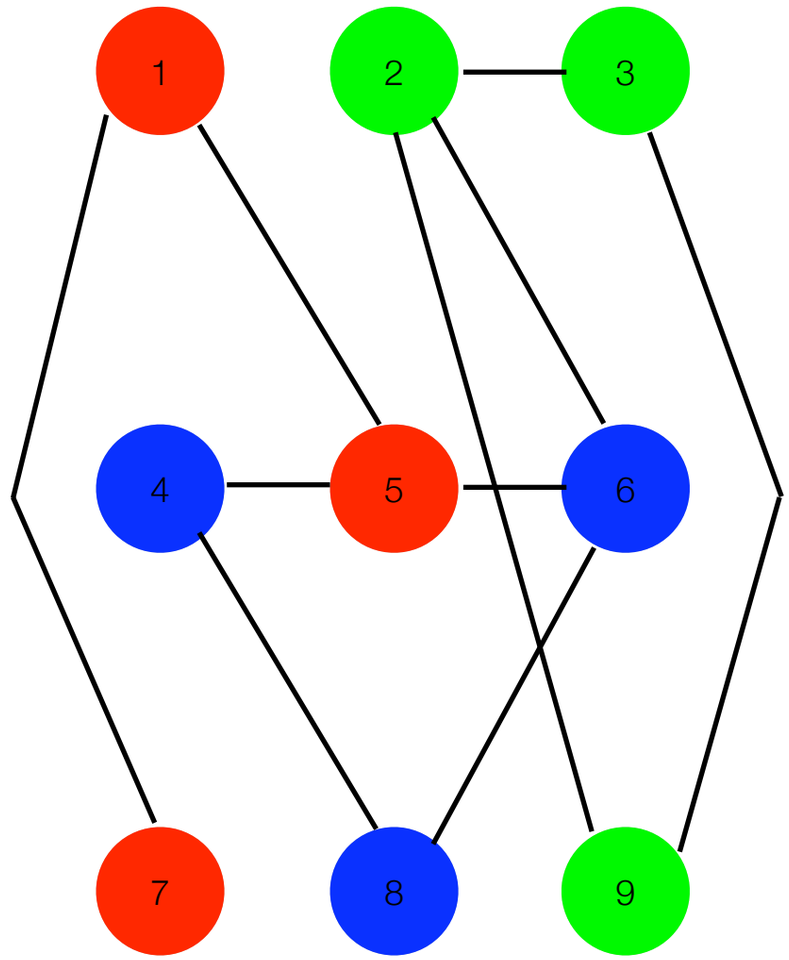
- For each node:
  - **Roll K-sided die** with bias π to determine z_i=1,...,K, the (unobserved) module assignment for i^{th} node
- For each pair of nodes (i,j):
  - If z_i=z_j, **flip "in community" coin** with bias θ_c to determine edge A_{ij}
  - If z_i≠z_j, **flip "between communities" coin** with bias θ_d to determine edge A_{ij}



Stochastic block models (Holland, Laskey, Leinhardt 1983; Wang and Wong, 1987)

Generating modular networks

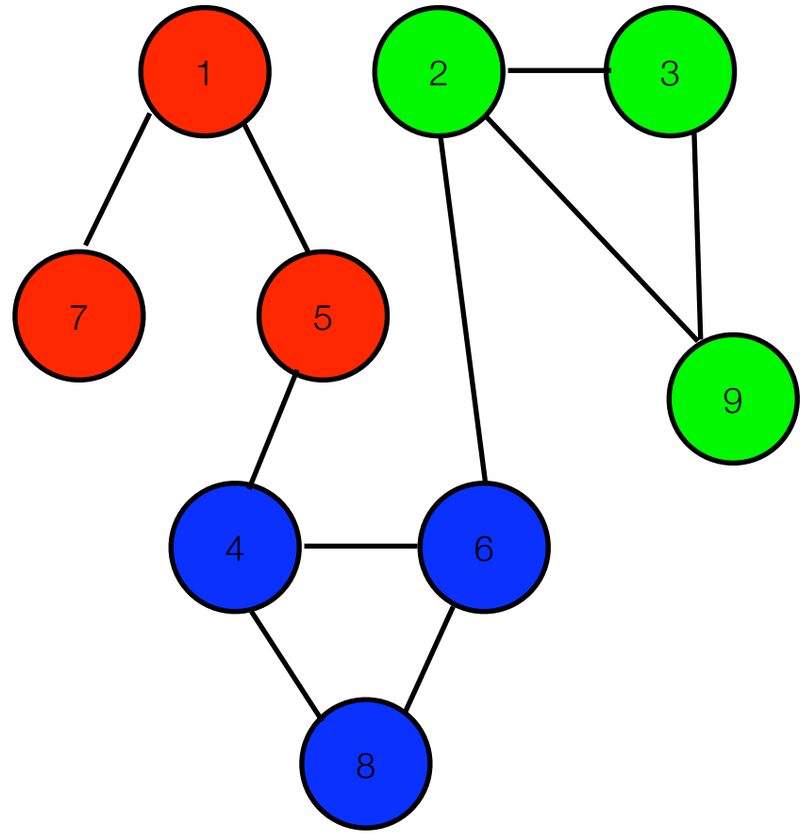
- For each node:
  - **Roll K-sided die** with bias π to determine z_i=1,...,K, the (unobserved) module assignment for i^{th} node
- For each pair of nodes (i,j):
  - If z_i=z_j, **flip "in community" coin** with bias θ_c to determine edge A_{ij}
  - If z_i≠z_j, **flip "between communities" coin** with bias θ_d to determine edge A_{ij}



Stochastic block models (Holland, Laskey, Leinhardt 1983; Wang and Wong, 1987)

Generating modular networks

- For each node:
  - **Roll K-sided die** with bias π to determine z_i=1,...,K, the (unobserved) module assignment for i^{th} node
- For each pair of nodes (i,j):
  - If z_i=z_j, **flip "in community" coin** with bias θ_c to determine edge A_{ij}
  - If z_i≠z_j, **flip "between communities" coin** with bias θ_d to determine edge A_{ij}



Stochastic block models (Holland, Laskey, Leinhardt 1983; Wang and Wong, 1987)

Generating modular networks

- Die rolling, coin flipping <-> infinite-range spin-glass Potts model:

$$\mathcal{H} \equiv -\ln p(\mathbf{A}, \vec{z}|\vec{\pi}, \vec{\theta}) = -\sum_{i,j}(J_L A_{ij} - J_G)\delta_{z_i,z_j} + \sum_{\mu=1}^{K} h_\mu \sum_{i=1}^{N} \delta_{z_i,\mu}$$

$$\begin{aligned} J_G &\equiv \ln \vartheta_c/\vartheta_d \\ J_L &\equiv \ln(1-\vartheta_d)/(1-\vartheta_c) + J_G \\ h_\mu &\equiv -\ln \pi_\mu \end{aligned}$$

- Infer *distributions* over spin assignments, coupling constants, and chemical potentials and find number of occupied spin states

$$p(A|K) = \sum_{\vec{z}} \int d\vec{\theta} \int d\vec{\pi}\ p(\mathbf{A}, \vec{z}, \vec{\pi}, \vec{\theta}) = \sum_{\vec{z}} \int d\vec{\theta} \int d\vec{\pi}\ e^{-\mathcal{H}} p(\vec{\theta}) p(\vec{\pi})$$

Extends Newman (2004, 2006), Hastings (2006), Bornholdt & Reichardt (2006)

Generating modular networks

- Die rolling, coin flipping <-> infinite-range spin-glass Potts model:

$$\mathcal{H} \equiv -\ln p(\mathbf{A}, \vec{z}|\vec{\pi}, \vec{\theta}) = -\sum_{i,j}(J_L A_{ij} - J_G)\delta_{z_i,z_j} + \sum_{\mu=1}^{K} h_\mu \sum_{i=1}^{N} \delta_{z_i,\mu}$$

$$\begin{aligned} J_G &\equiv \ln \vartheta_c/\vartheta_d \\ J_L &\equiv \ln(1-\vartheta_d)/(1-\vartheta_c) + J_G \\ h_\mu &\equiv -\ln \pi_\mu \end{aligned}$$

- Infer *distributions* over spin assignments, coupling constants, and chemical potentials and find number of occupied spin states

$$p(A|K) = \sum_{\vec{z}} \int d\vec{\theta} \int d\vec{\pi}\; p(\mathbf{A}, \vec{z}, \vec{\pi}, \vec{\theta}) = \sum_{\vec{z}} \int d\vec{\theta} \int d\vec{\pi}\; e^{-\mathcal{H}} p(\vec{\theta}) p(\vec{\pi}) \quad \longleftarrow$$

Can do integrals, but sum is intractable, $O(K^N)$; use mean-field variational technique

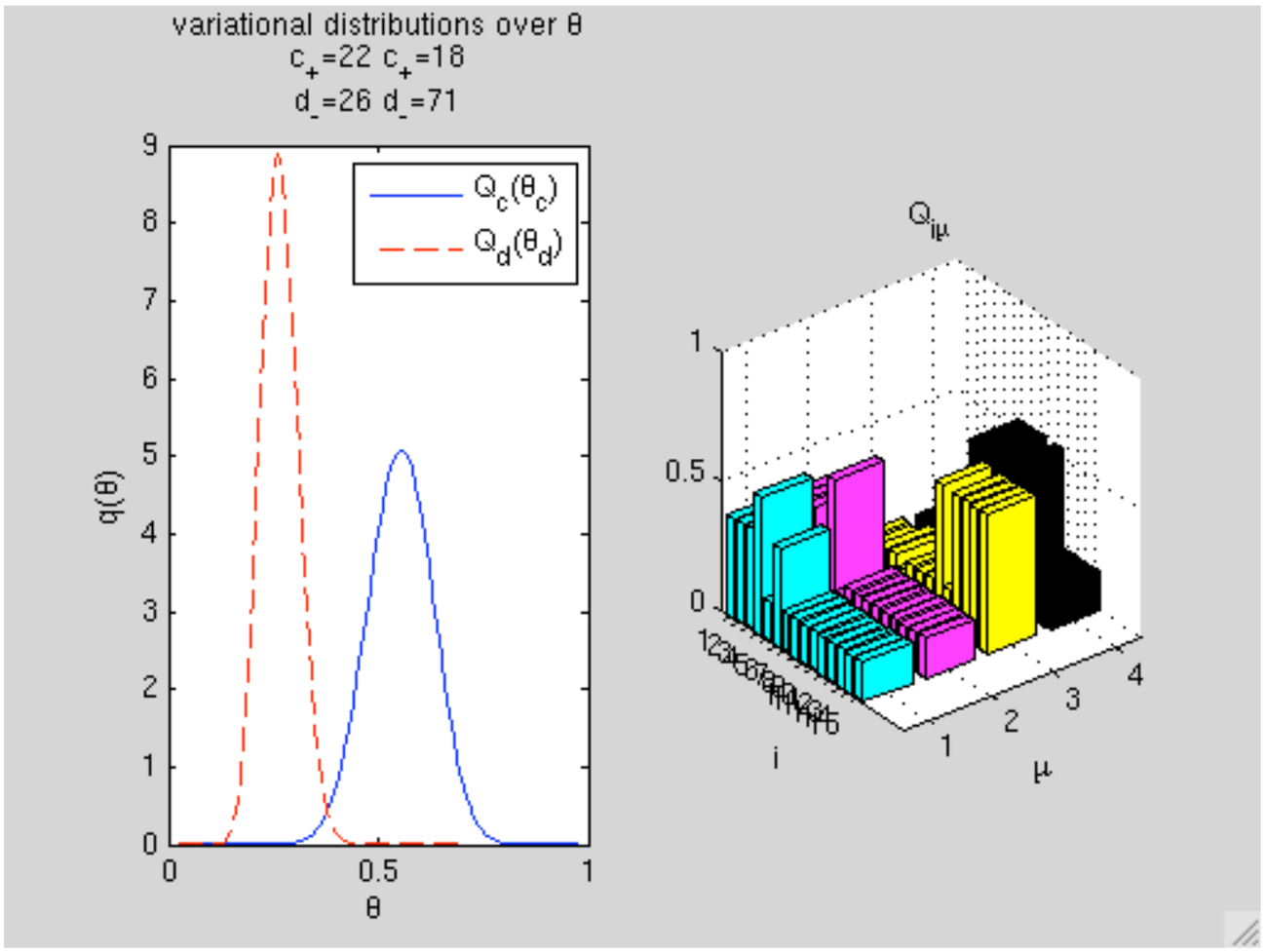
Extends Newman (2004, 2006), Hastings (2006), Bornholdt & Reichardt (2006)

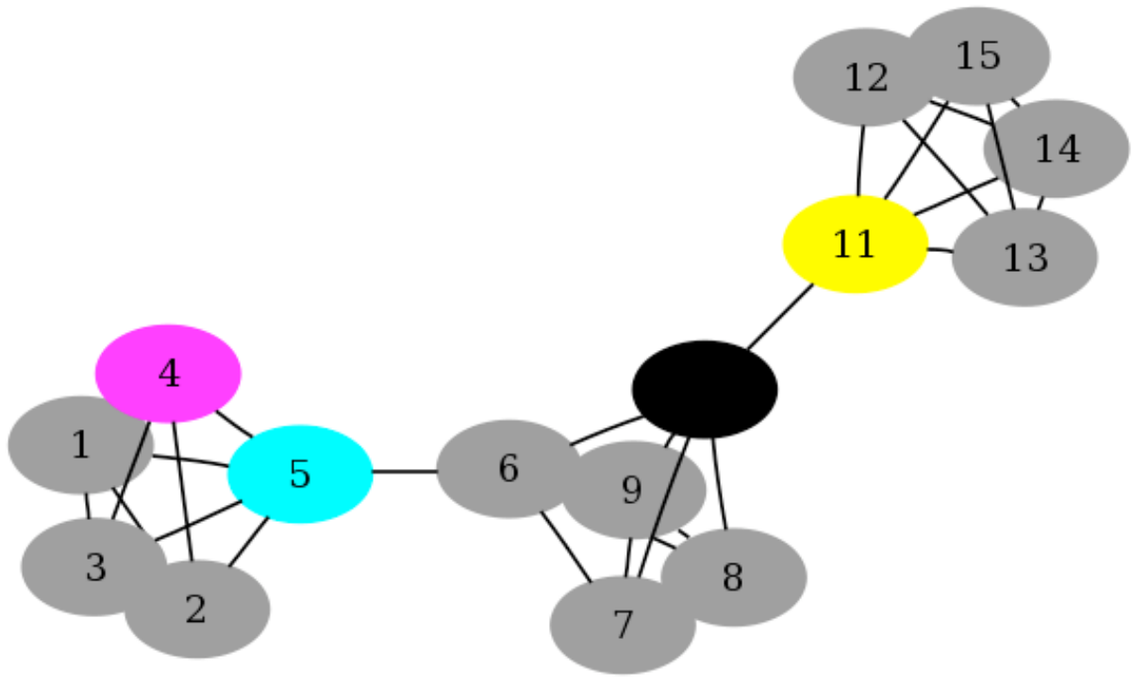
Validation: Toy graph

- Automatic complexity control: probability of occupation for extraneous modules goes to zero

Validation: Toy graph

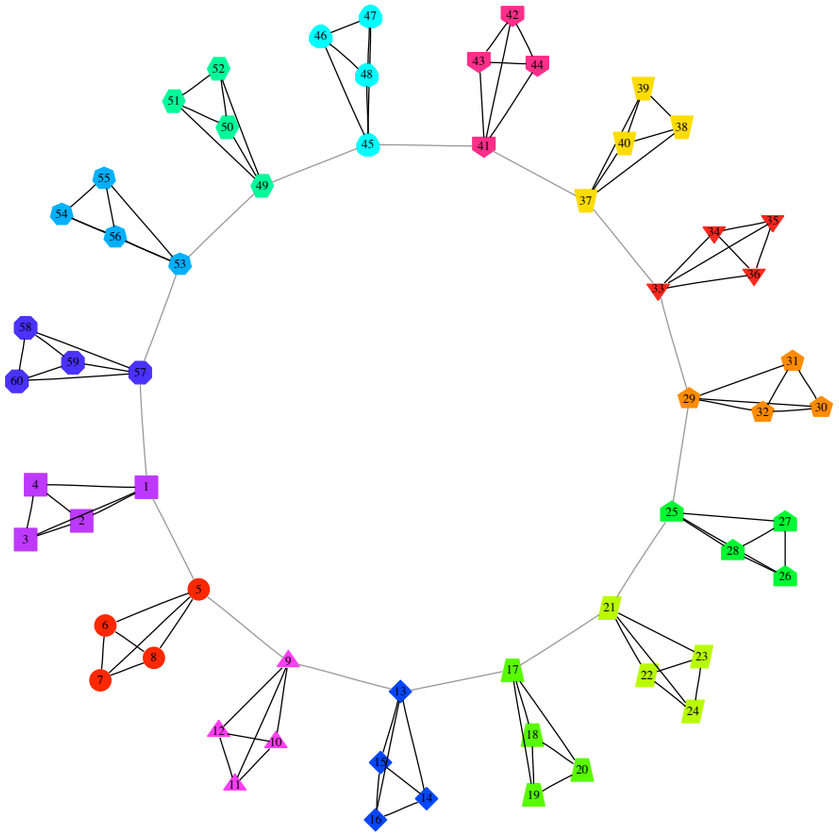
- Automatic complexity control: probability of occupation for extraneous modules goes to zero



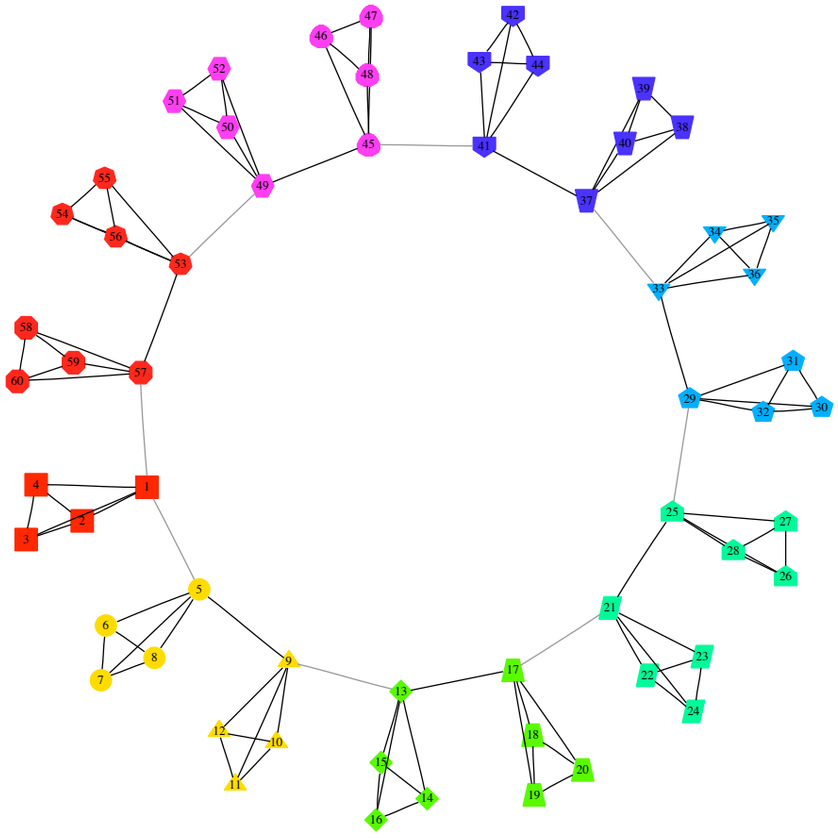


The “resolution limit” problem

Variational Bayesian approach correctly infers complexity

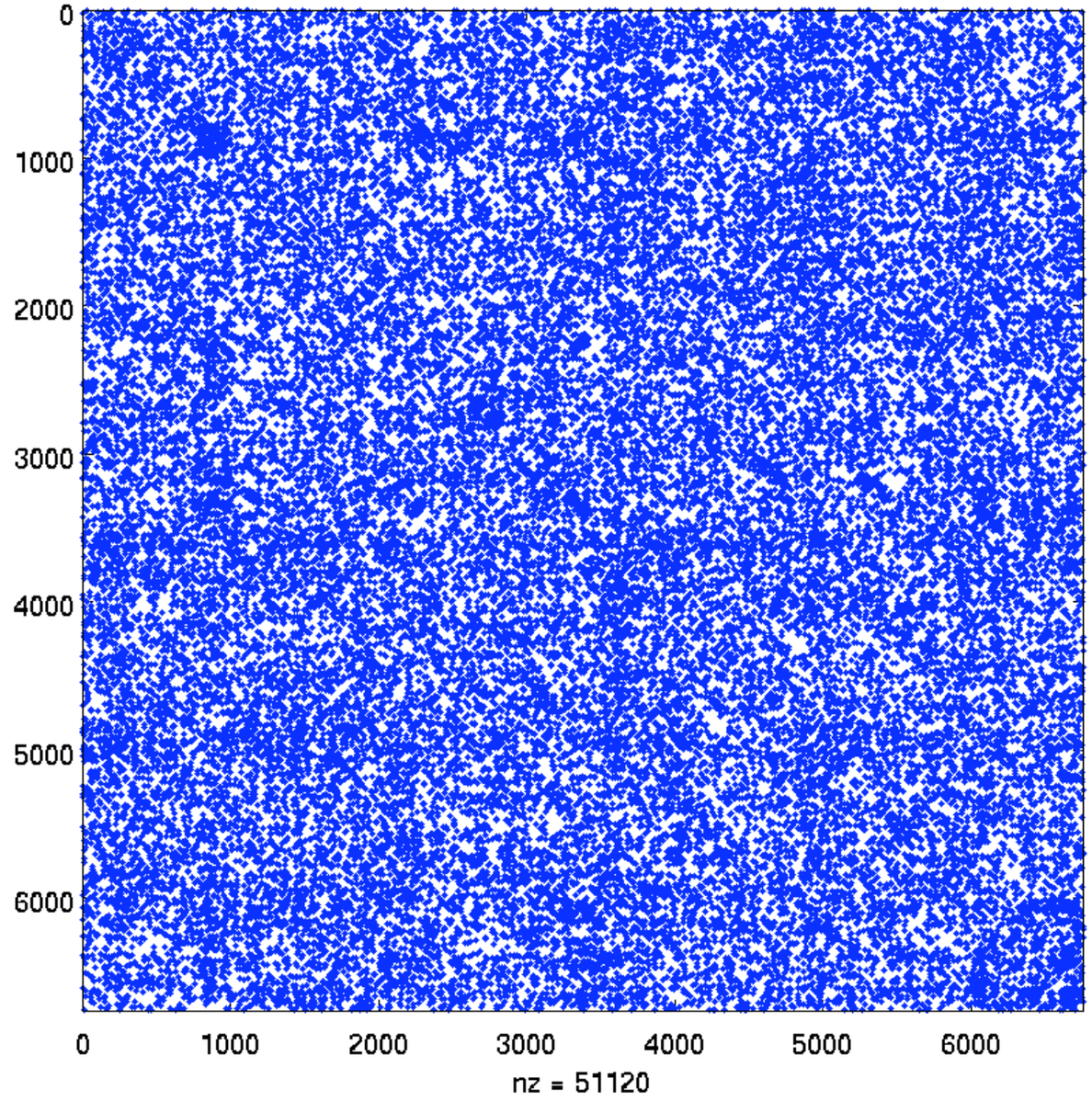


Variational Bayes

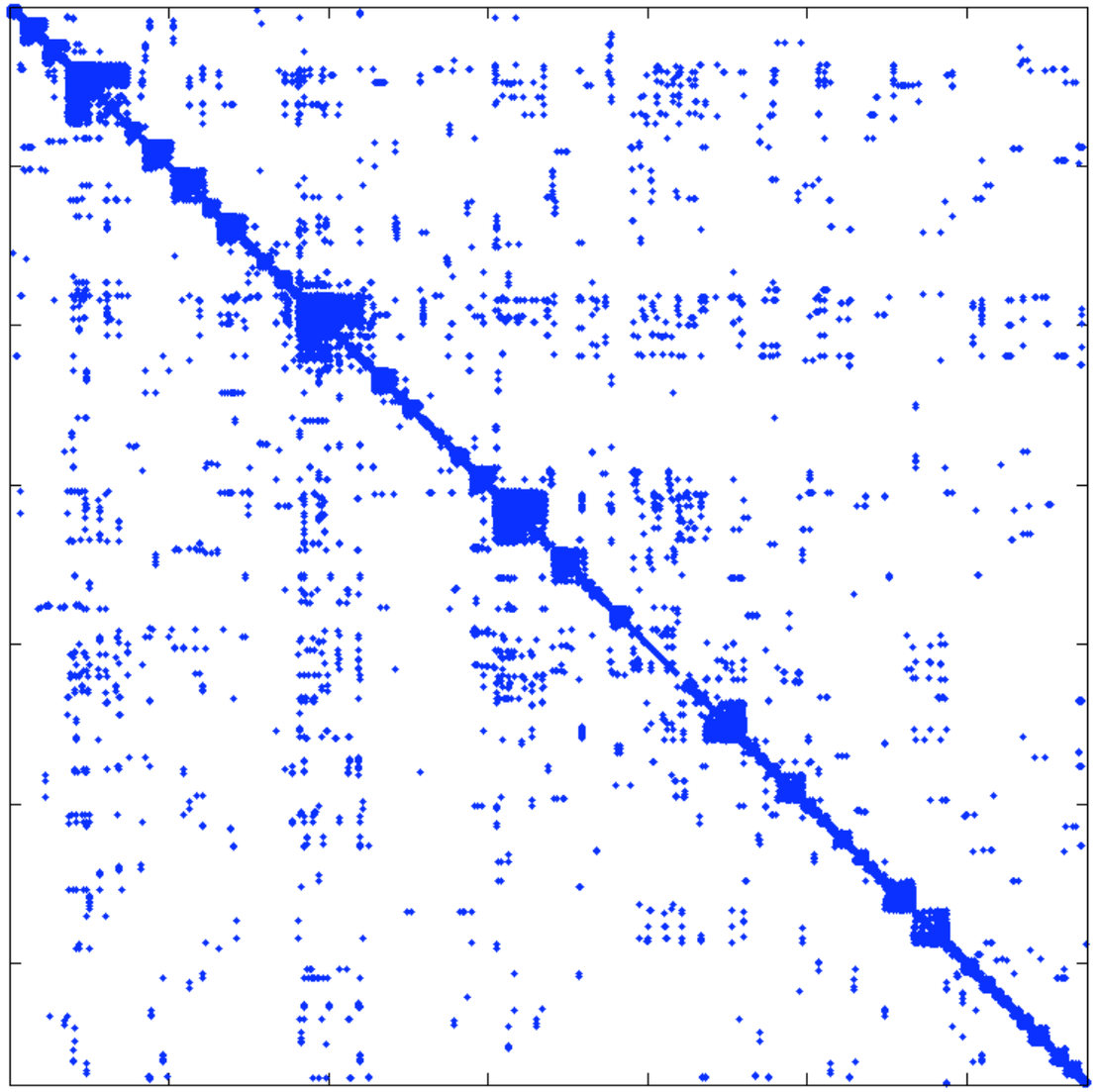


Girvan-Newman
modularity

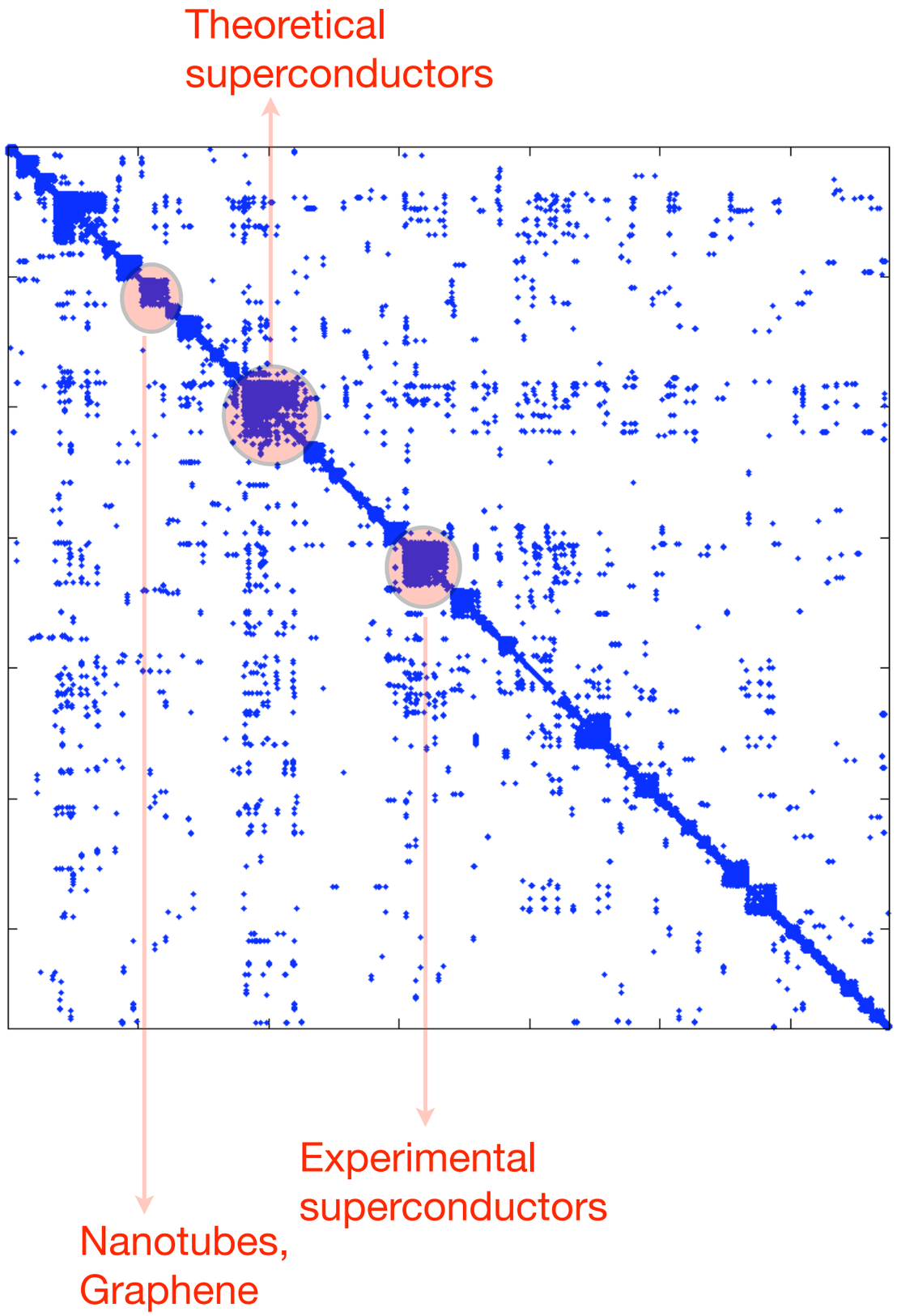
APS March Meeting 2008 co-authorship network



APS March Meeting 2008 co-authorship network



APS March Meeting 2008 co-authorship network



Conclusions

- Used variational *Bayesian inference to infer distributions* over module assignments, model parameters, and model complexity

- Resulted in a principled, accurate, and scalable algorithm which *addresses the resolution limit problem*

- Validated technique on synthetic and real networks

- Future: extend model to handle alternative network structure, using same *framework*

- preprint: http://arxiv.org/abs/0709.3512

Acknowledgements

- Useful discussions
  - Jonathan Goodman (NYU)
  - Joel Bader (Hopkins)
  - Matt Hastings (LANL)
  - Aaron Clauset (SFI)
  - Edo Airoldi (Princeton)