A Bayesian approach to network modularity

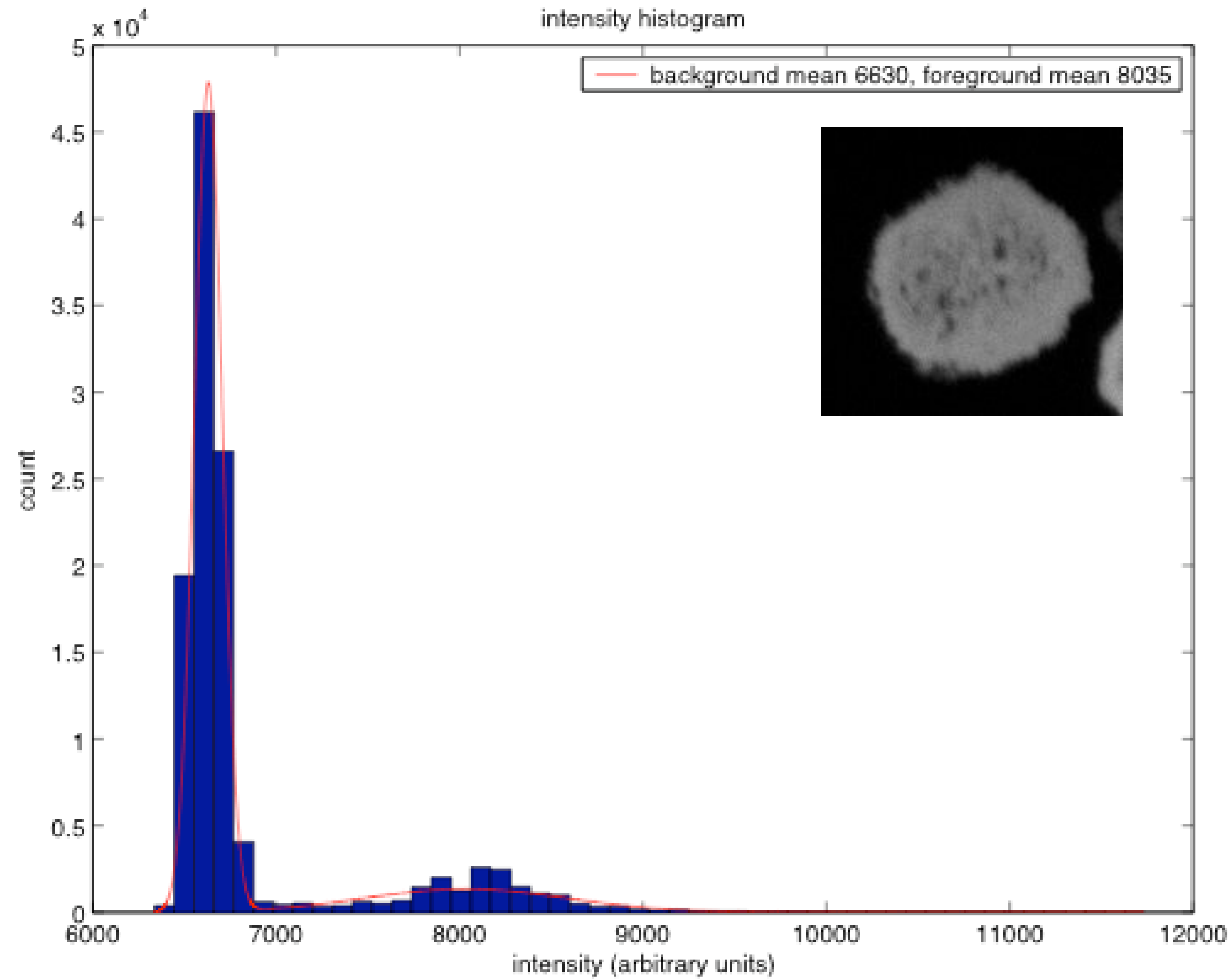
Jake Hofman
Wiggins Lab
Columbia University
2007.10.19

Outline

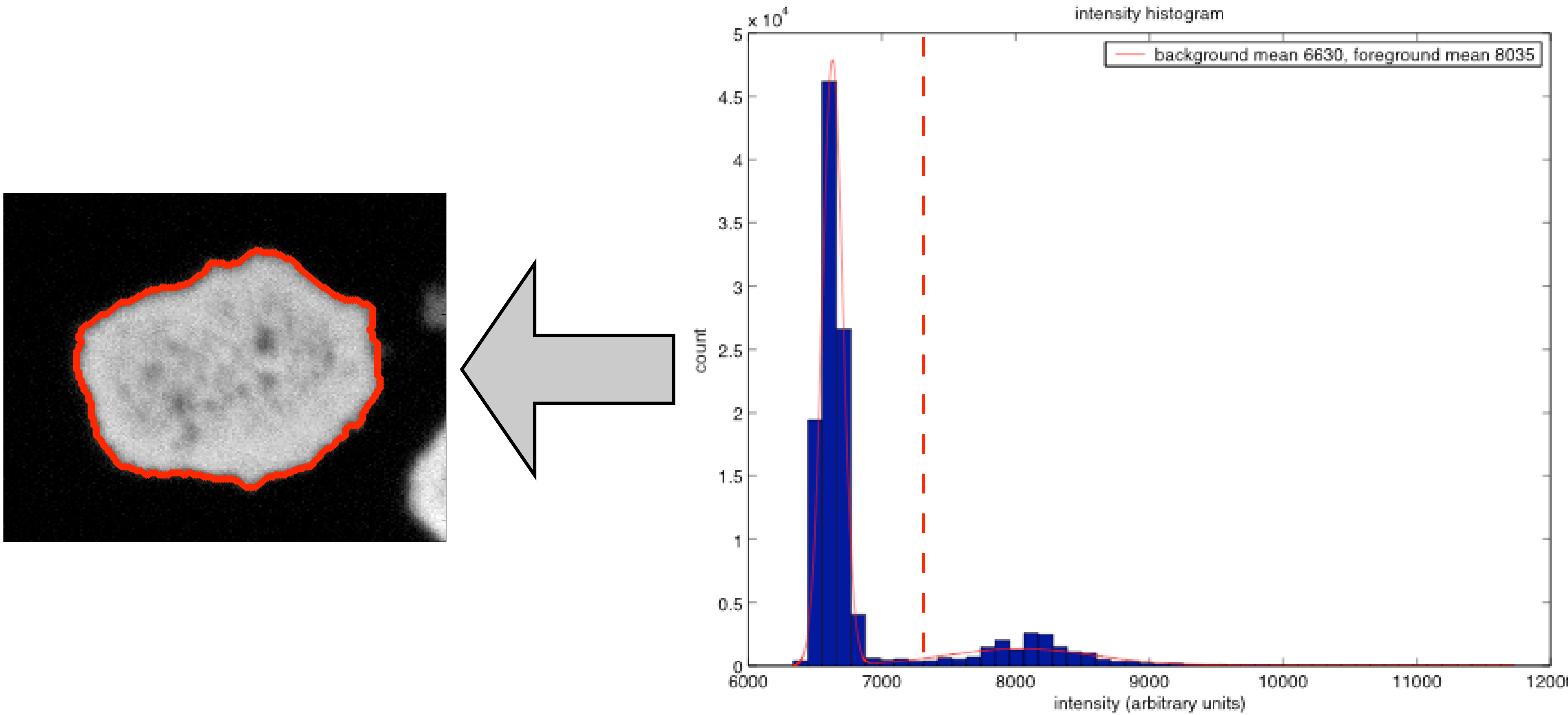
- Past work: image data
- Extension to modular networks
- Bayesian inference and complexity control
- Generating and inferring modular networks
- Validation and preliminary applications

Overview: modeling image data

- Given an image:
    - Assign pixels to clusters (foreground/background)?



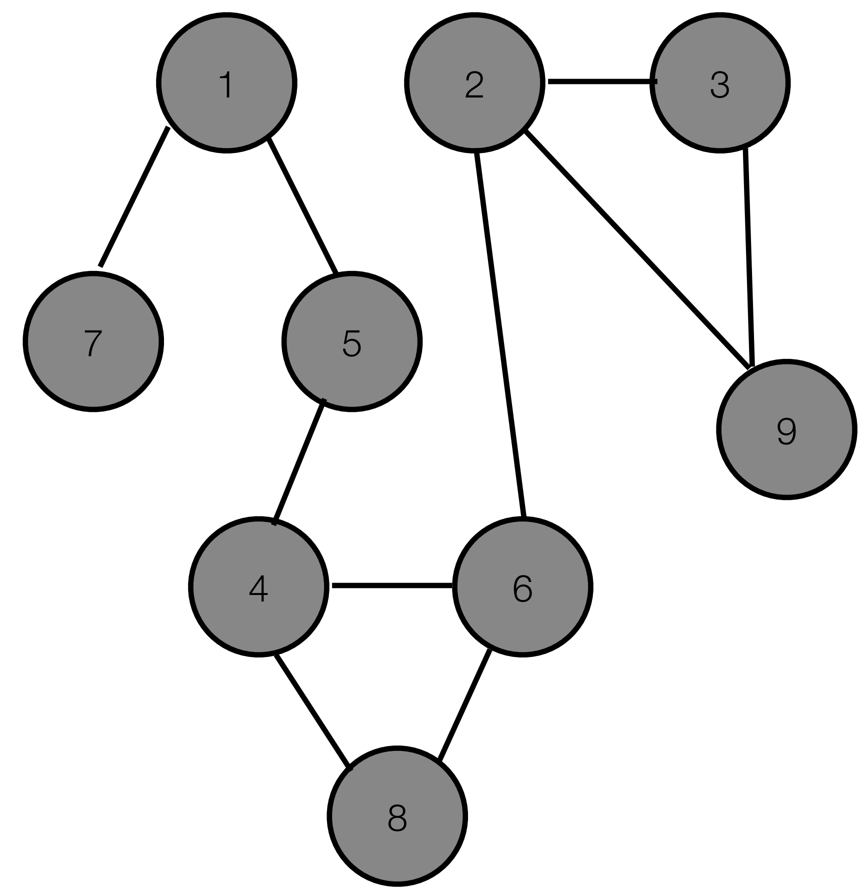
Overview: modeling image data



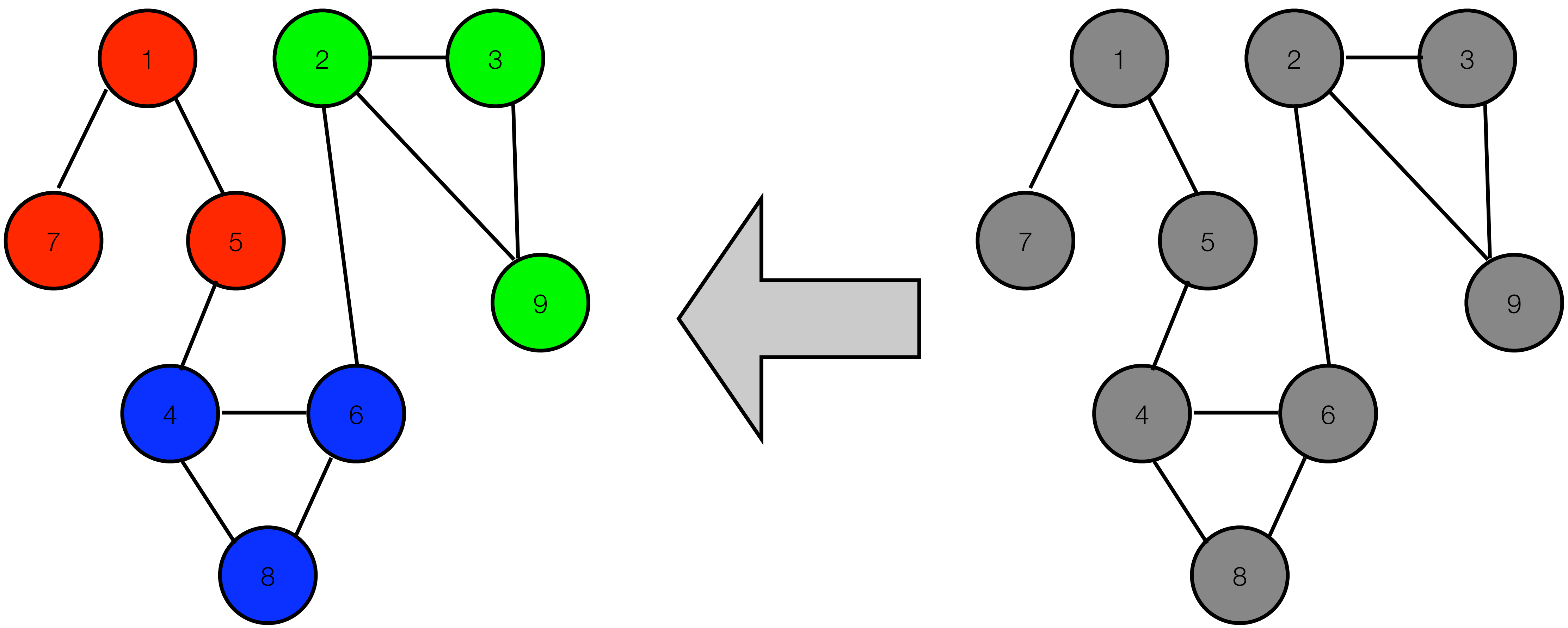
With a generative model of intensity histograms, rules of probability tell us how to calculate model parameters (e.g. threshold)

Overview: modular networks

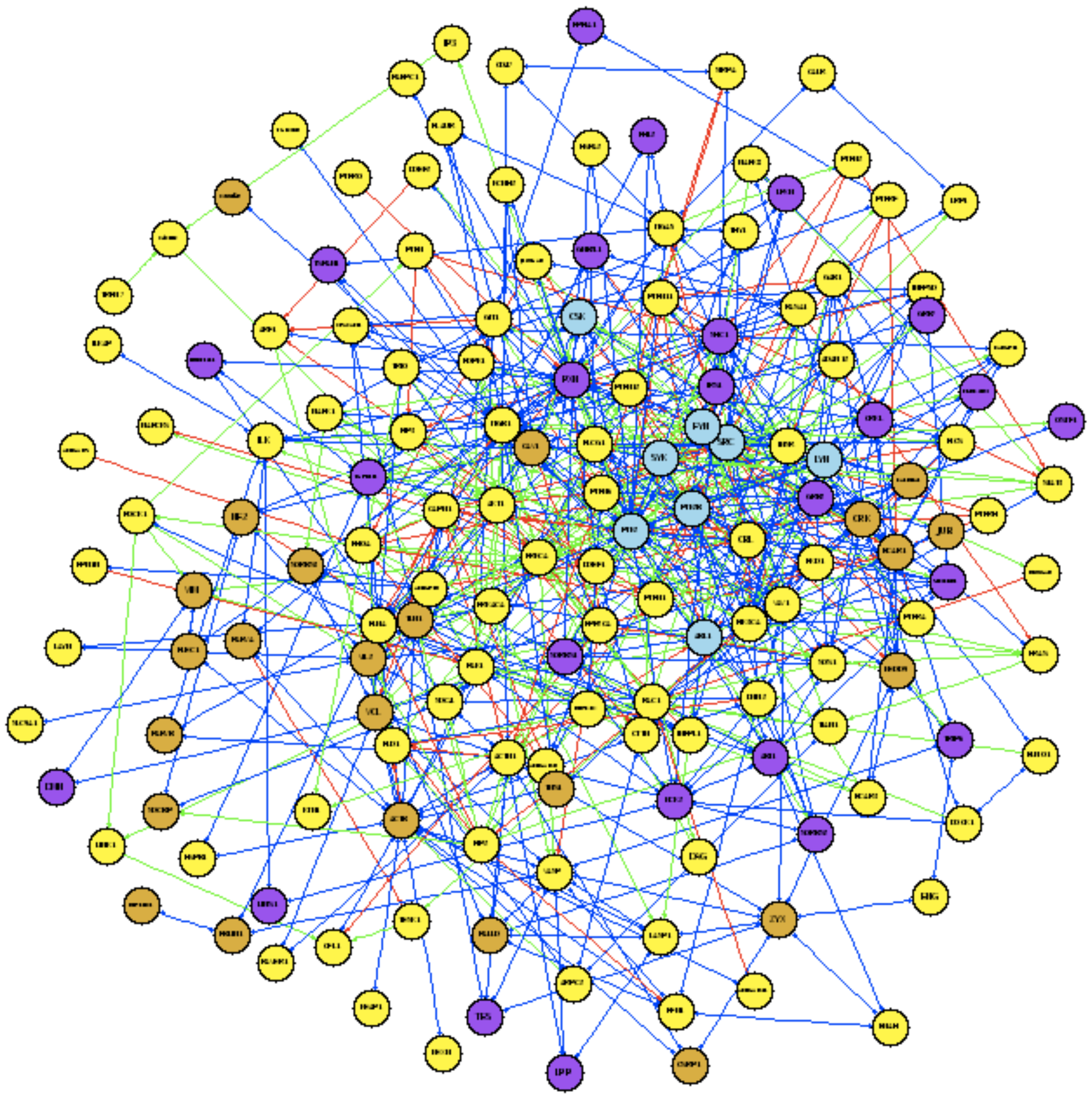
- Given a network:
  - Assign nodes to modules?
  - Determine number of modules (scale/complexity)?



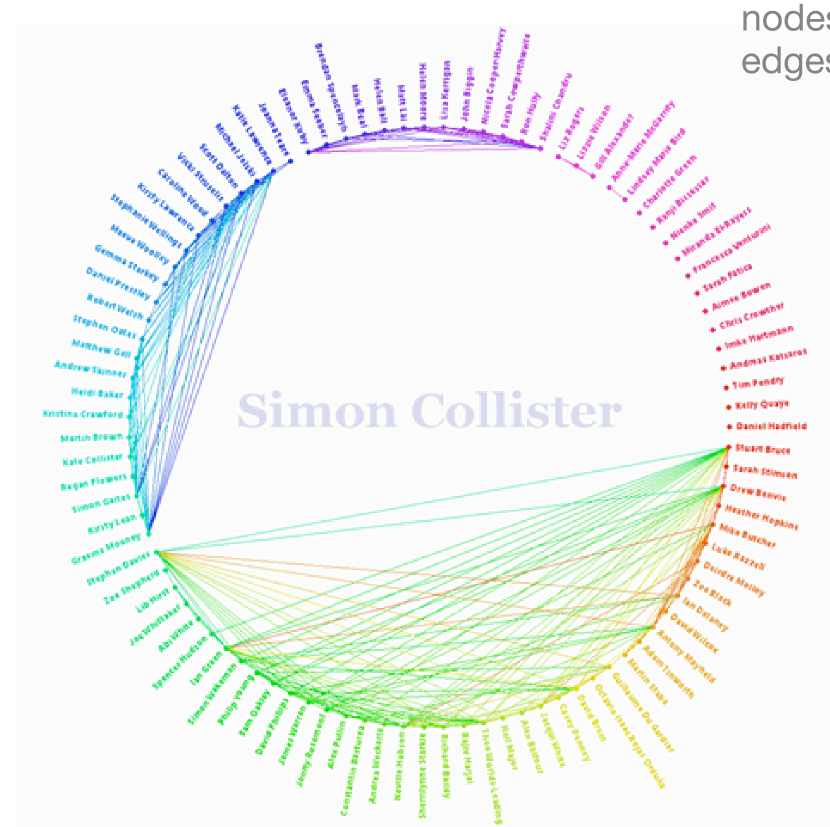
Overview: modular networks



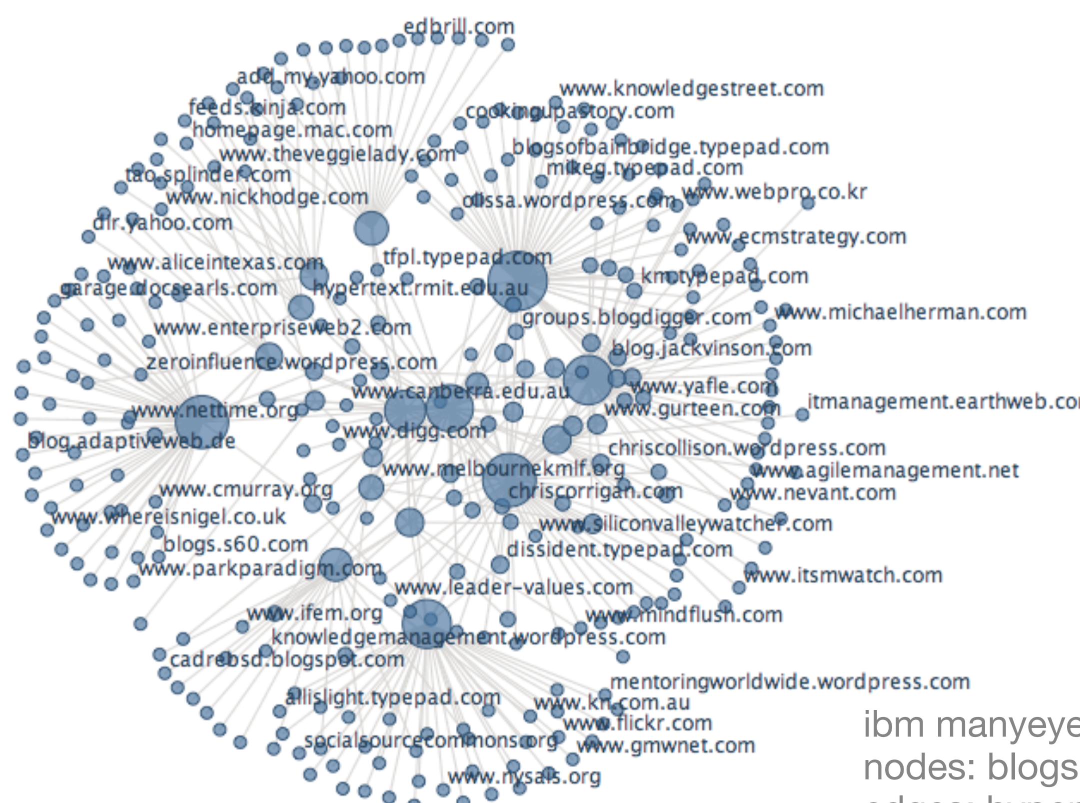
With a generative model of modular networks, rules of probability tell us how to calculate model parameters (e.g. number of modules & assignments)



http://adhesome.org
nodes: proteins
edges: interactions (e..g binding, regulation)



http://facebook.com
nodes: people
edges: friendship



ibm manyeyes
nodes: blogs
edges: hyperlinks

Networks describe objects (**nodes**) and interactions (**edges**) between the objects.

Generative models

Know model (parameters, assignment variables, complexity)

Generate synthetic data

Infer model (parameters, latent variables, complexity)

Observe real data

Bayesian inference

- Given prior belief and data D, infer posterior distribution over model parameters θ and complexity K

- Use Bayes' rule to "invert" probabilities

likelihood prior

$$\underbrace{p(\theta|\mathcal{D}, K)}_{\text{posterior}} = \frac{\overbrace{p(\mathcal{D}|\theta, K)}^{\text{likelihood}}\,\overbrace{p(\theta|K)}^{\text{prior}}}{\underbrace{p(\mathcal{D}|K)}_{\text{evidence}}}$$

posterior

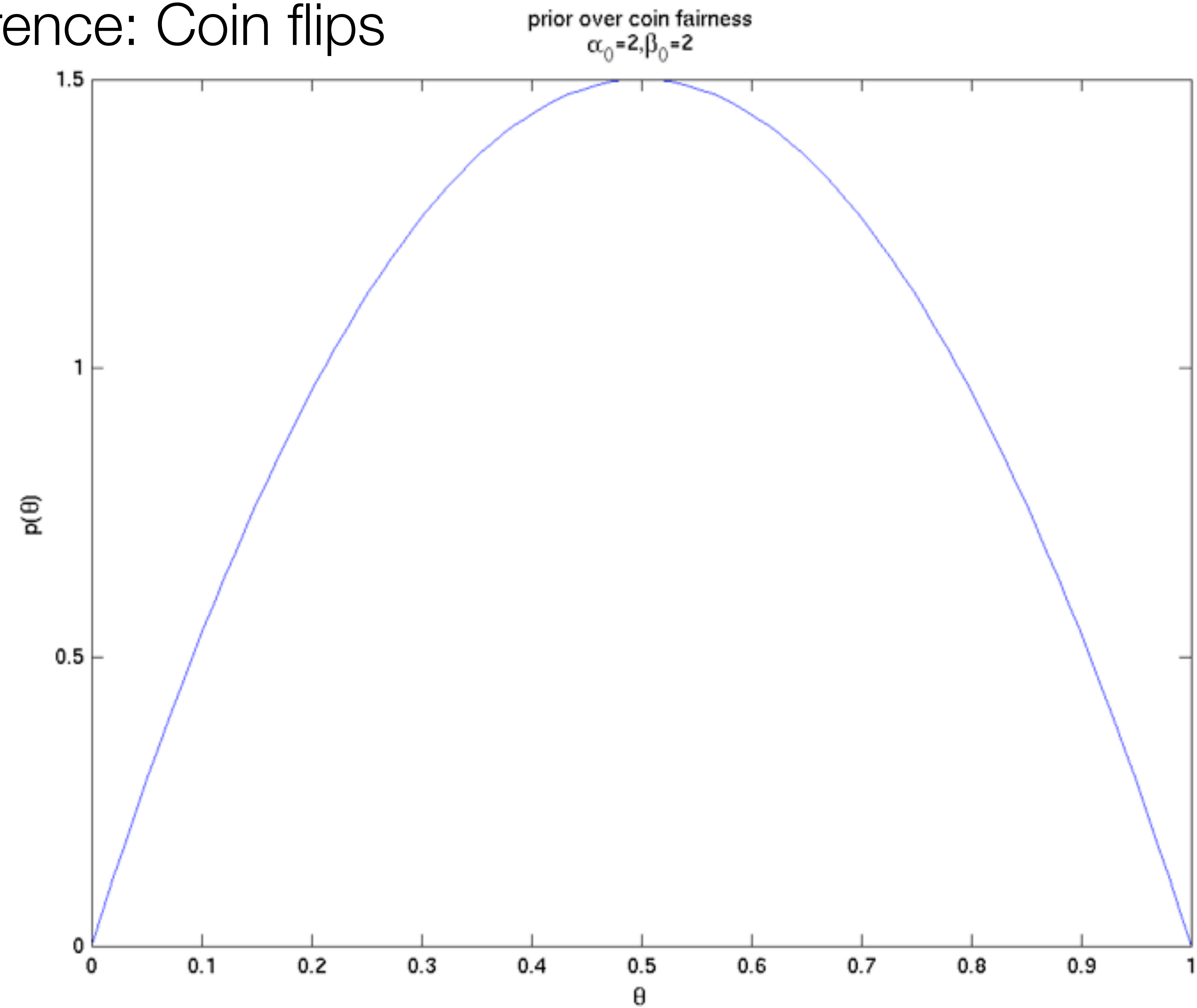
evidence

where

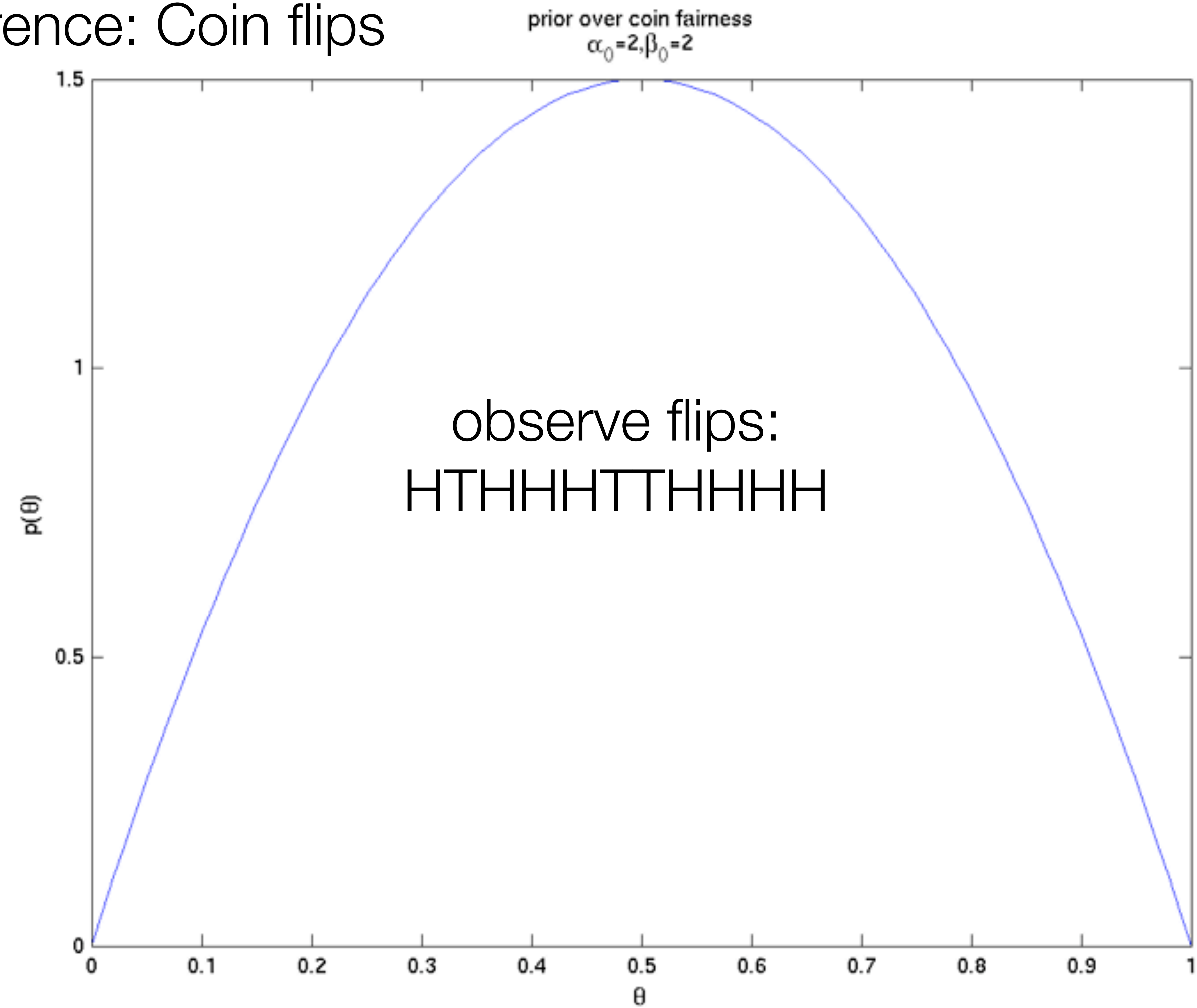
$$p(\mathcal{D}|K) = \int d\theta\ p(\mathcal{D}|\theta, K)\,p(\theta|K)$$

Inference: Coin flips

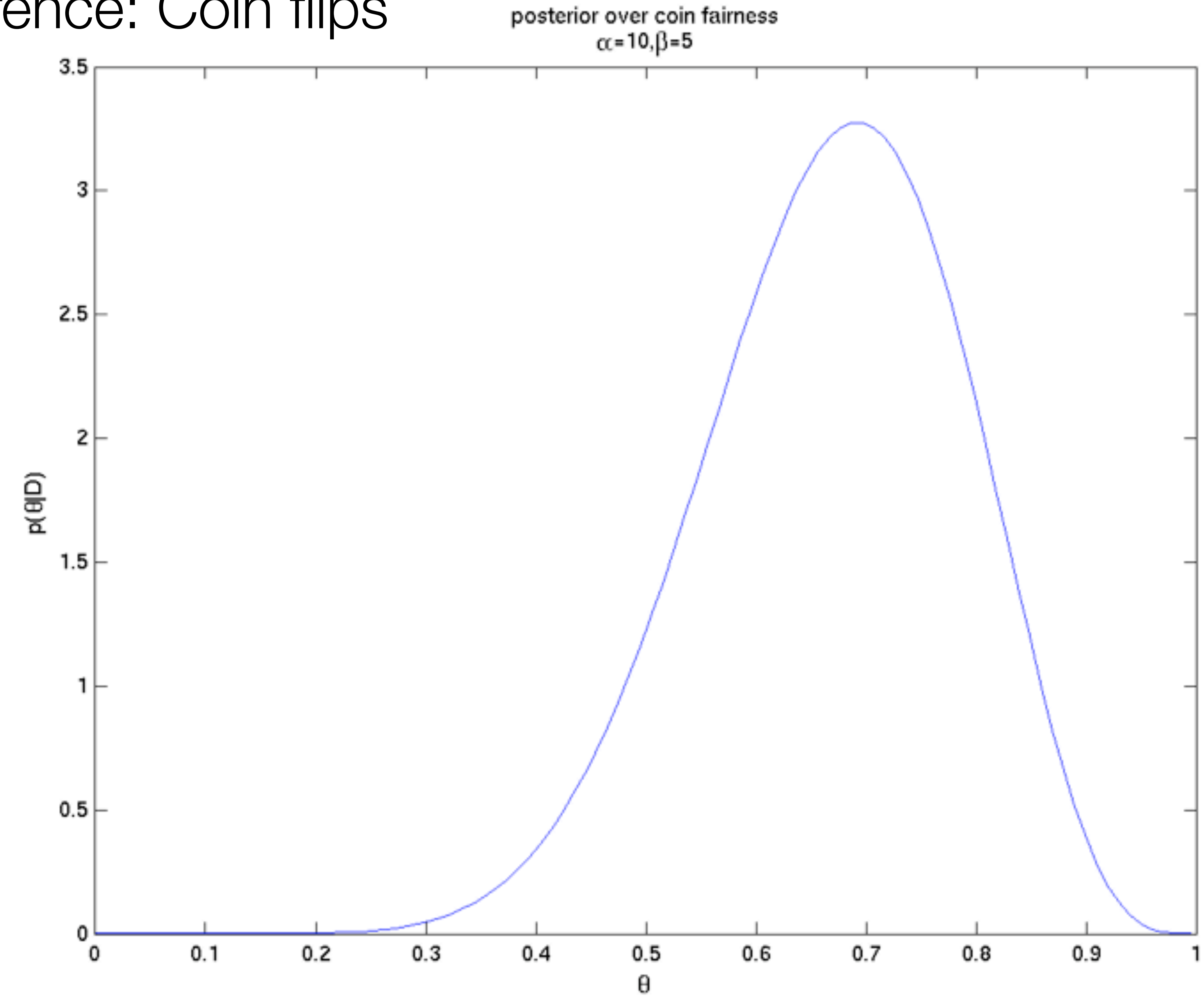
Inference: Coin flips



Inference: Coin flips

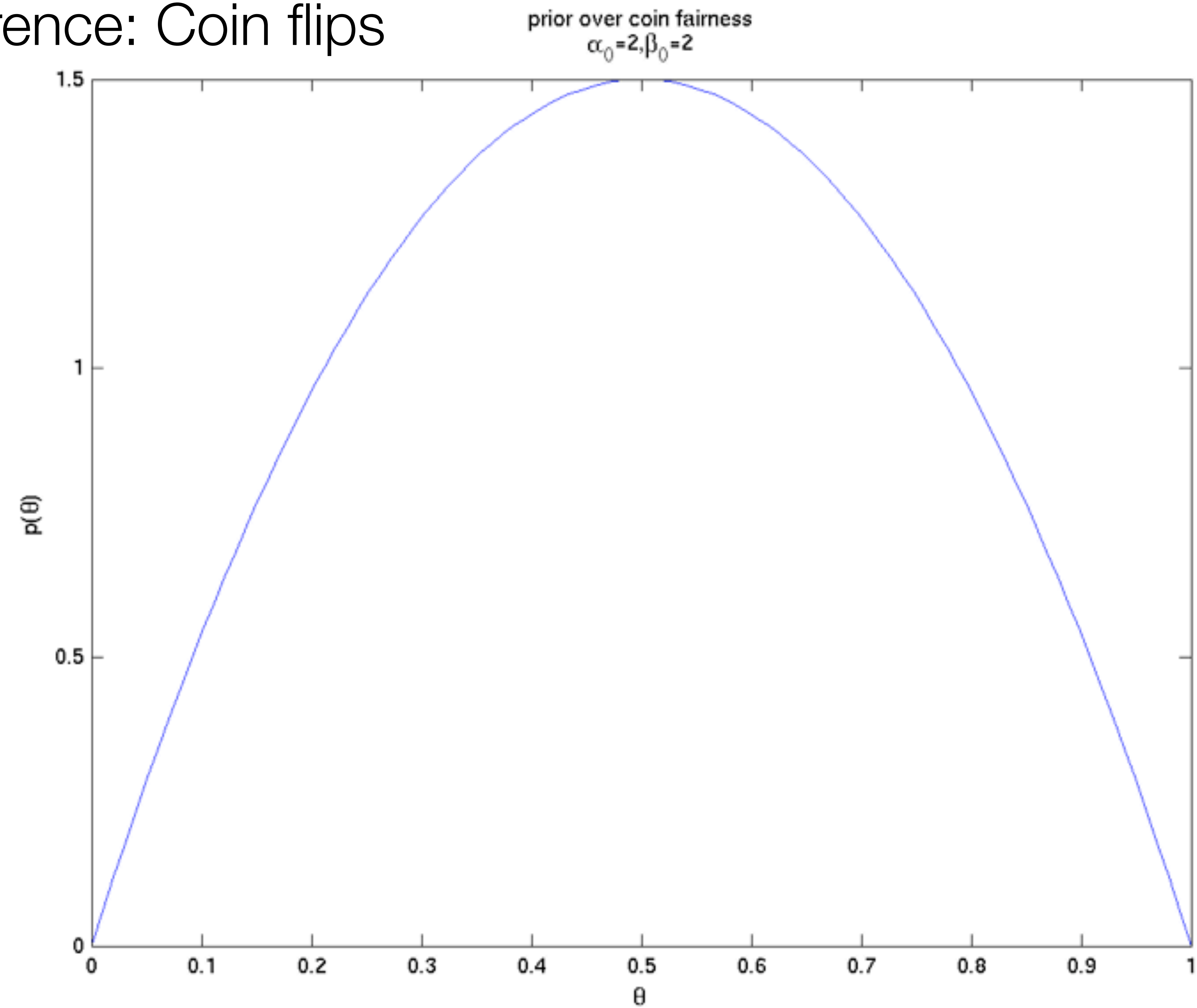


Inference: Coin flips

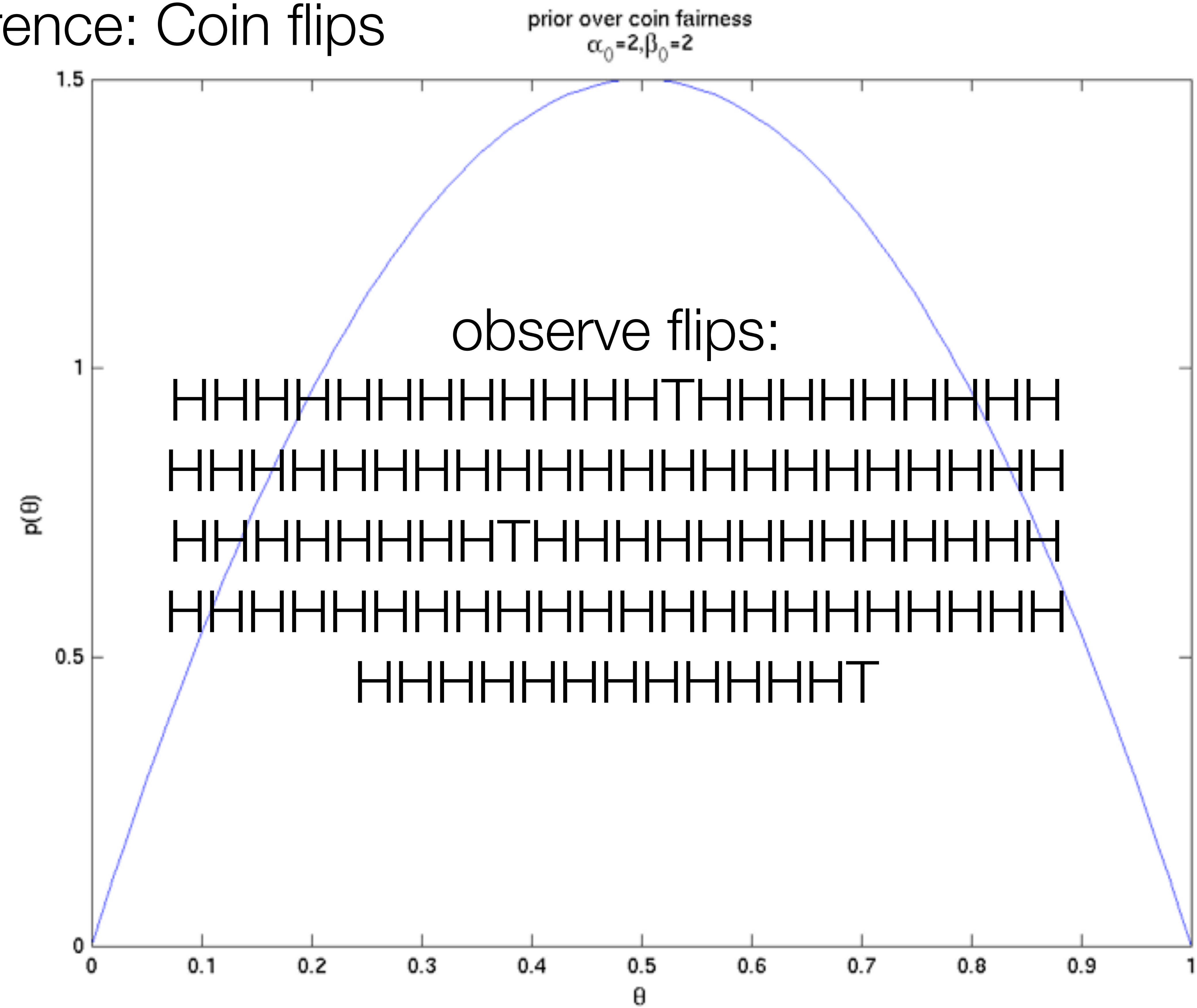


Inference: Coin flips

Inference: Coin flips



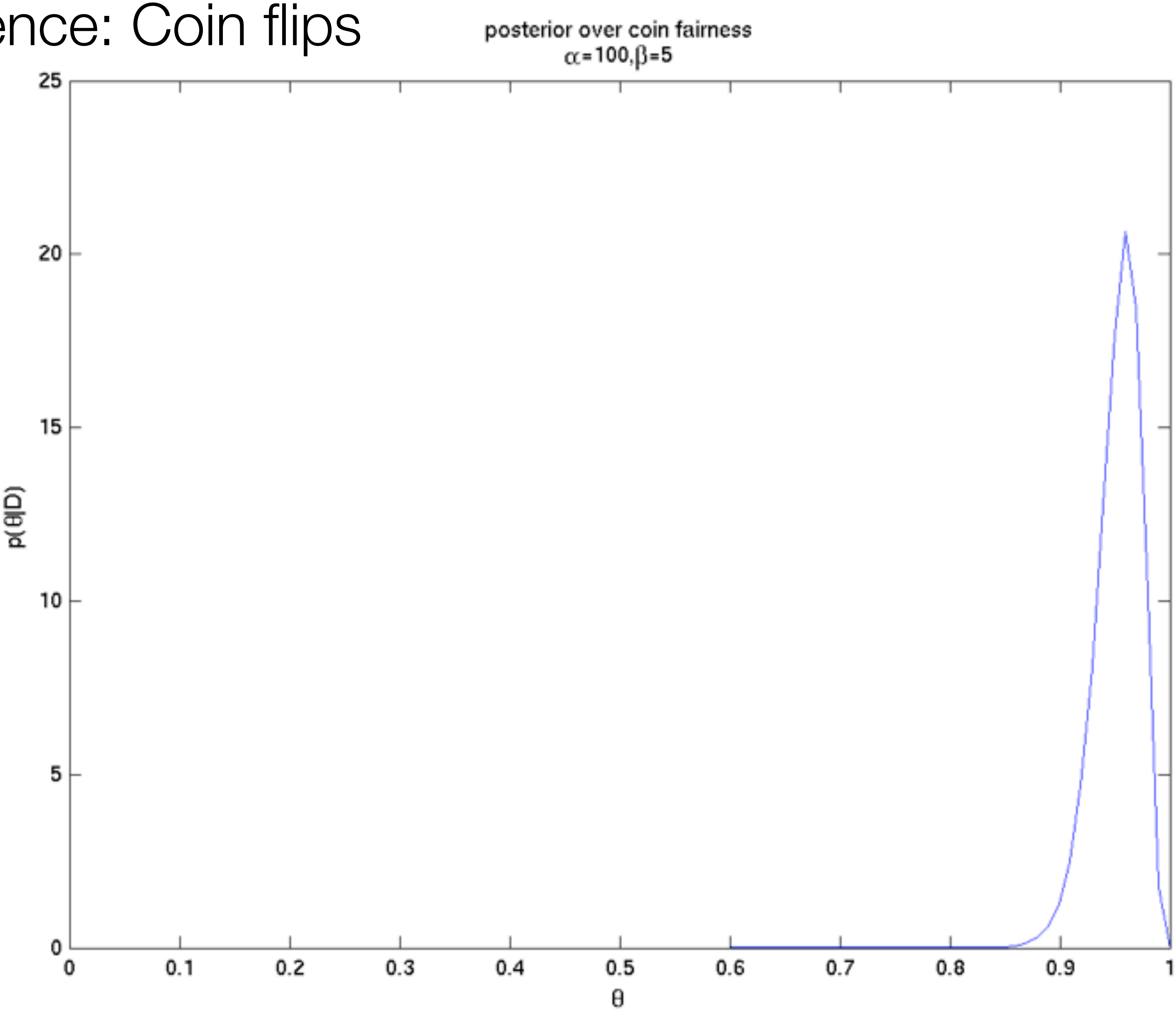
Inference: Coin flips



observe flips:

HHHHHHHHHHHHTHHHHHHHHH
HHHHHHHHHHHHHHHHHHHHHH
HHHHHHHHTHHHHHHHHHHHHH
HHHHHHHHHHHHHHHHHHHHHH
HHHHHHHHHHHT

Inference: Coin flips

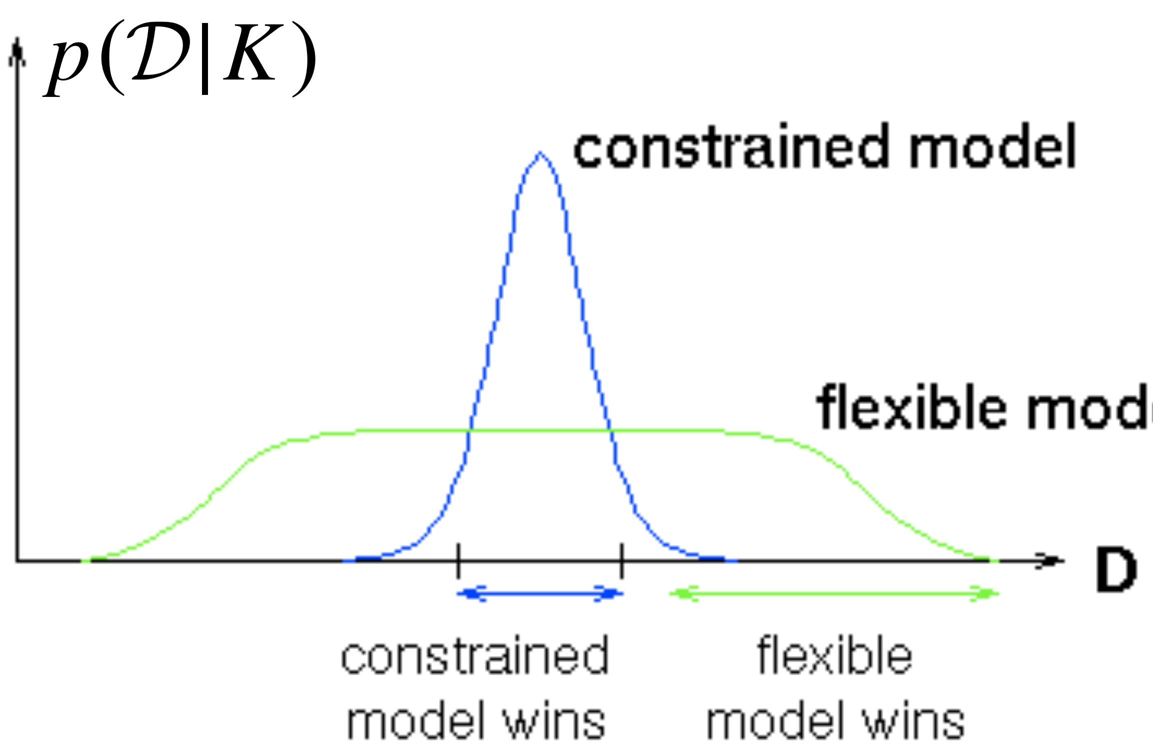


Bayesian complexity control

- Given prior belief and data D, infer posterior distribution over model complexity K

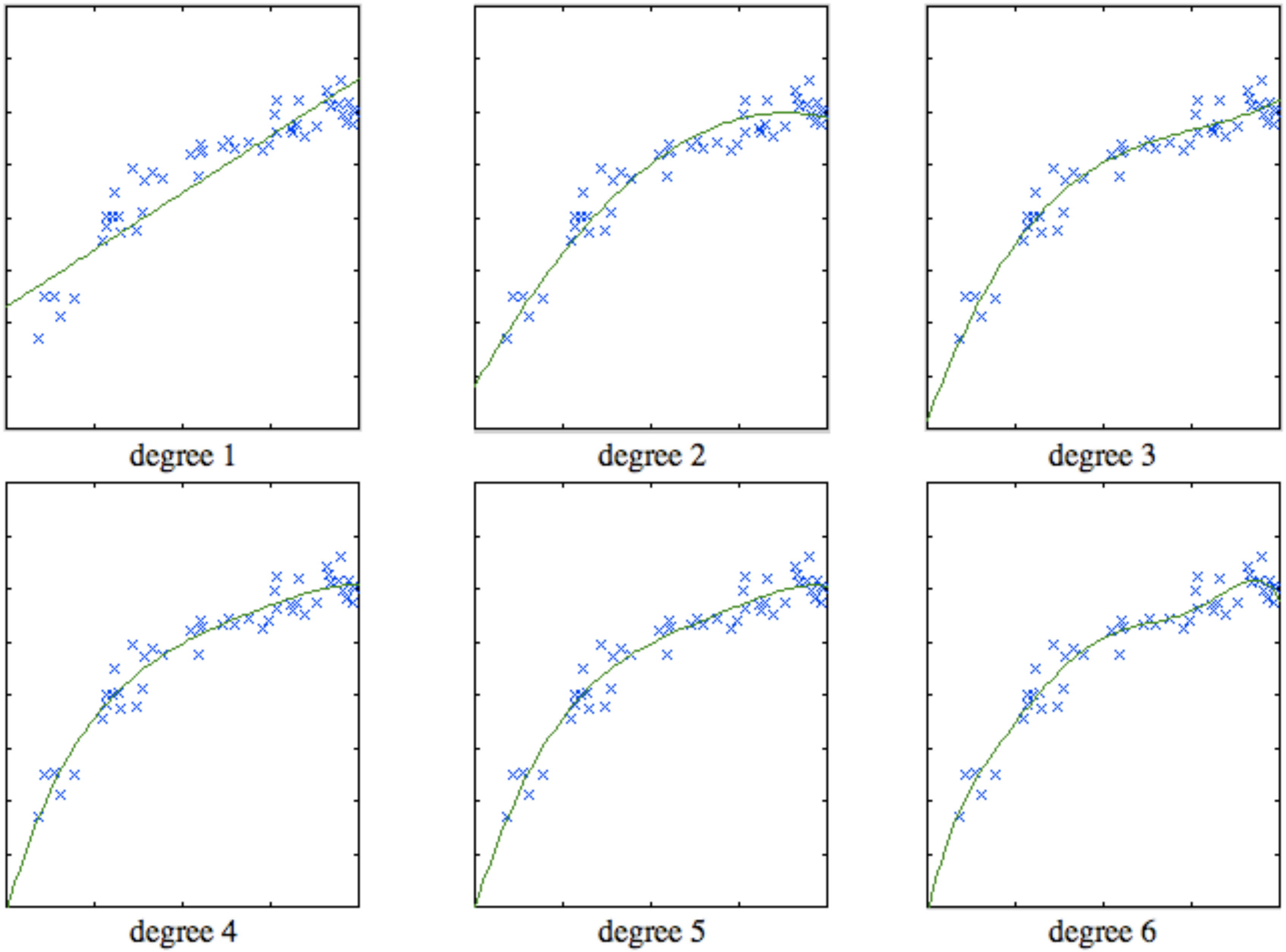
$$p(K|\mathcal{D}) = \frac{p(\mathcal{D}|K)\,p(K)}{p(\mathcal{D})}$$

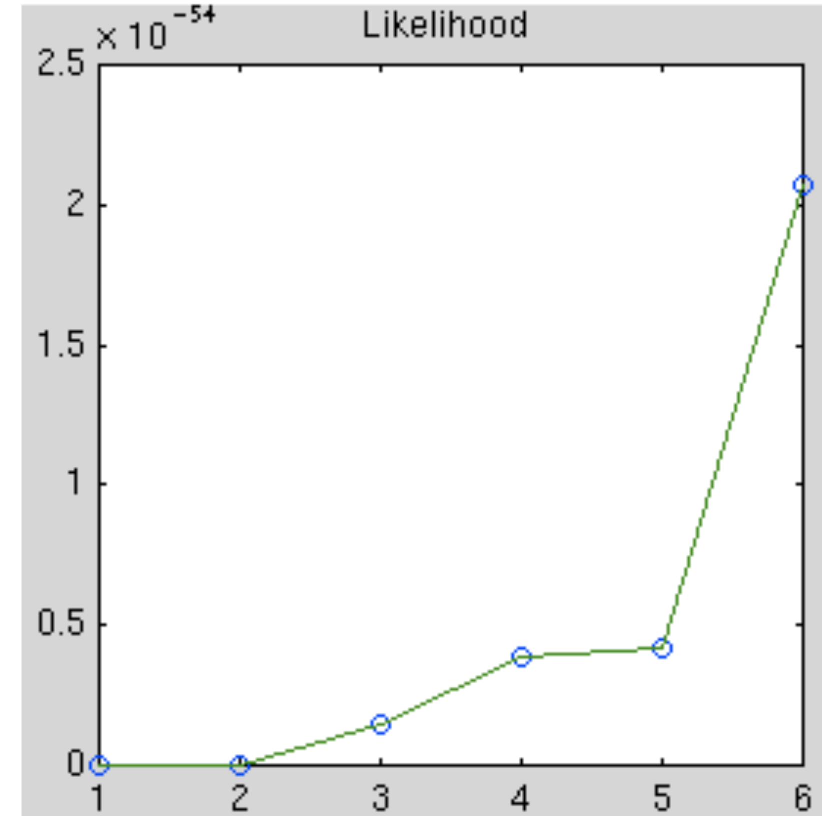
- If p(K) sufficiently weak, maximize evidence to find optimal complexity

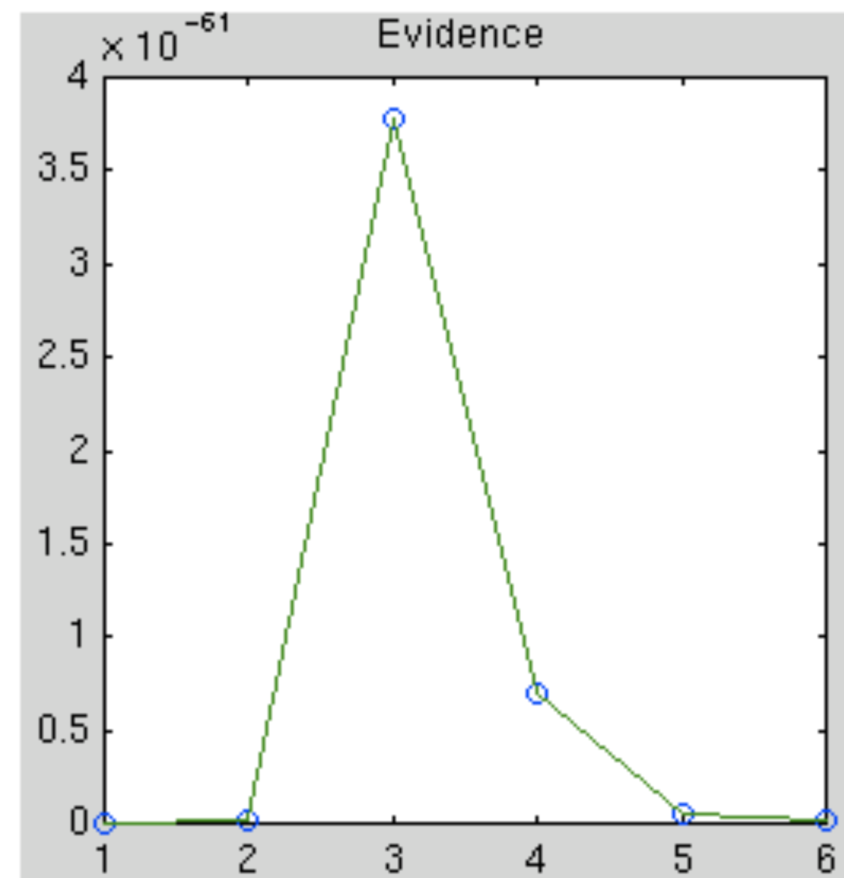


$$\underbrace{p(\mathcal{D}|K)}_{\text{evidence}} = \int d\theta \; p(\mathcal{D}|\theta, K)\,p(\theta|K)$$

Complexity control:
Optimal degree for regression

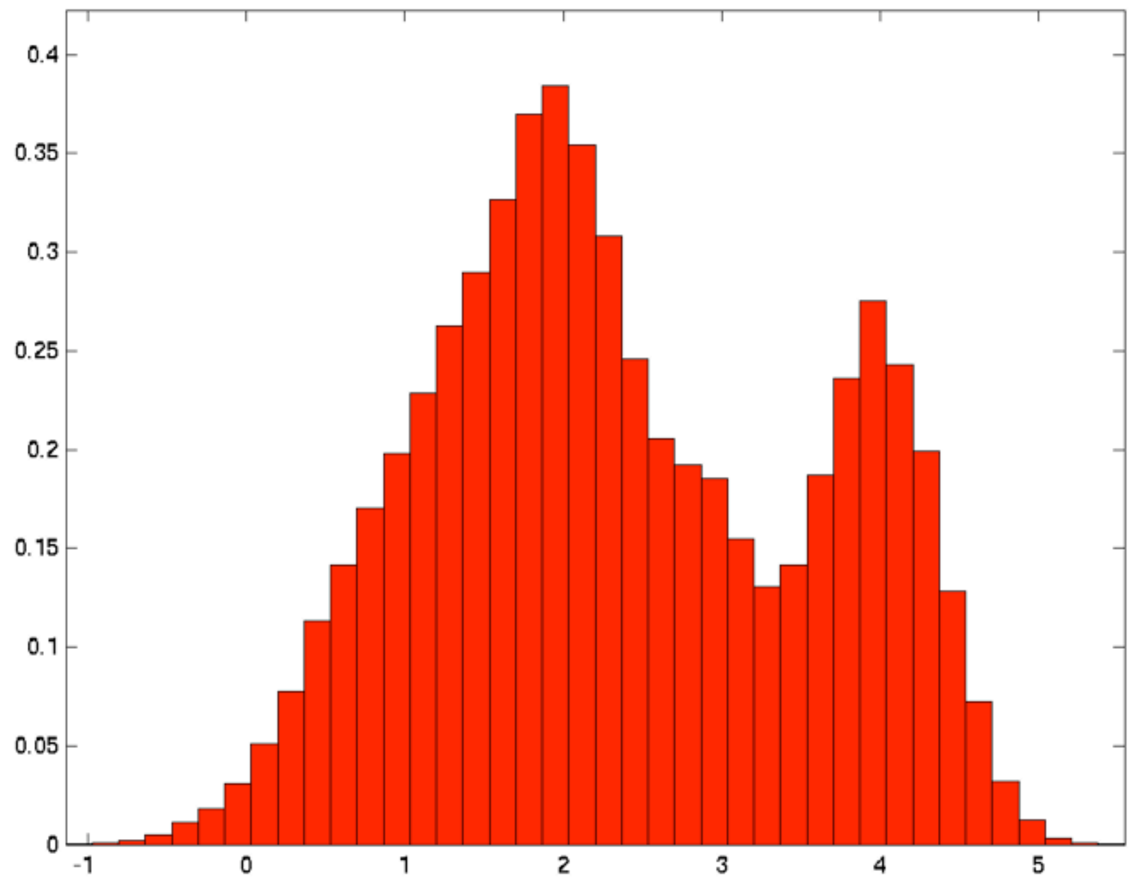


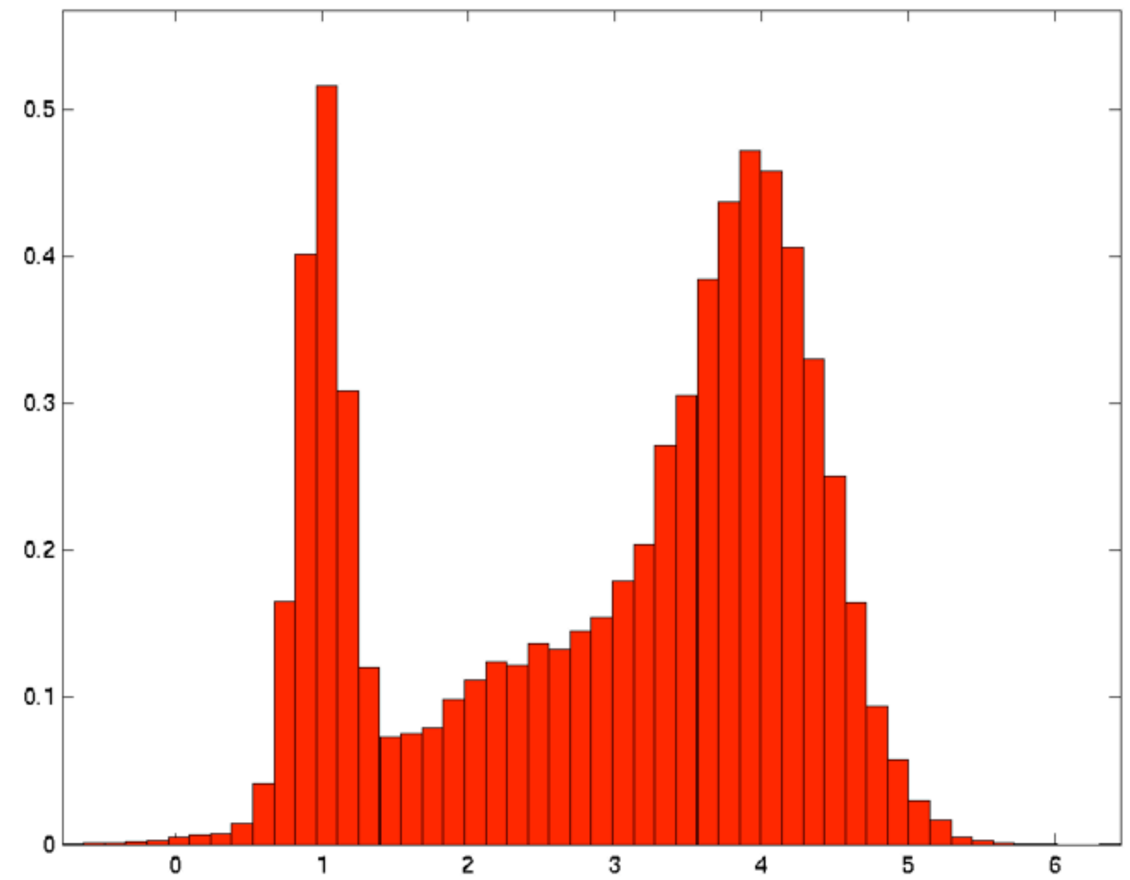




Likelihood always increases with complexity
Evidence incorporates complexity control
http://research.microsoft.com/~minka/statlearn/demo/

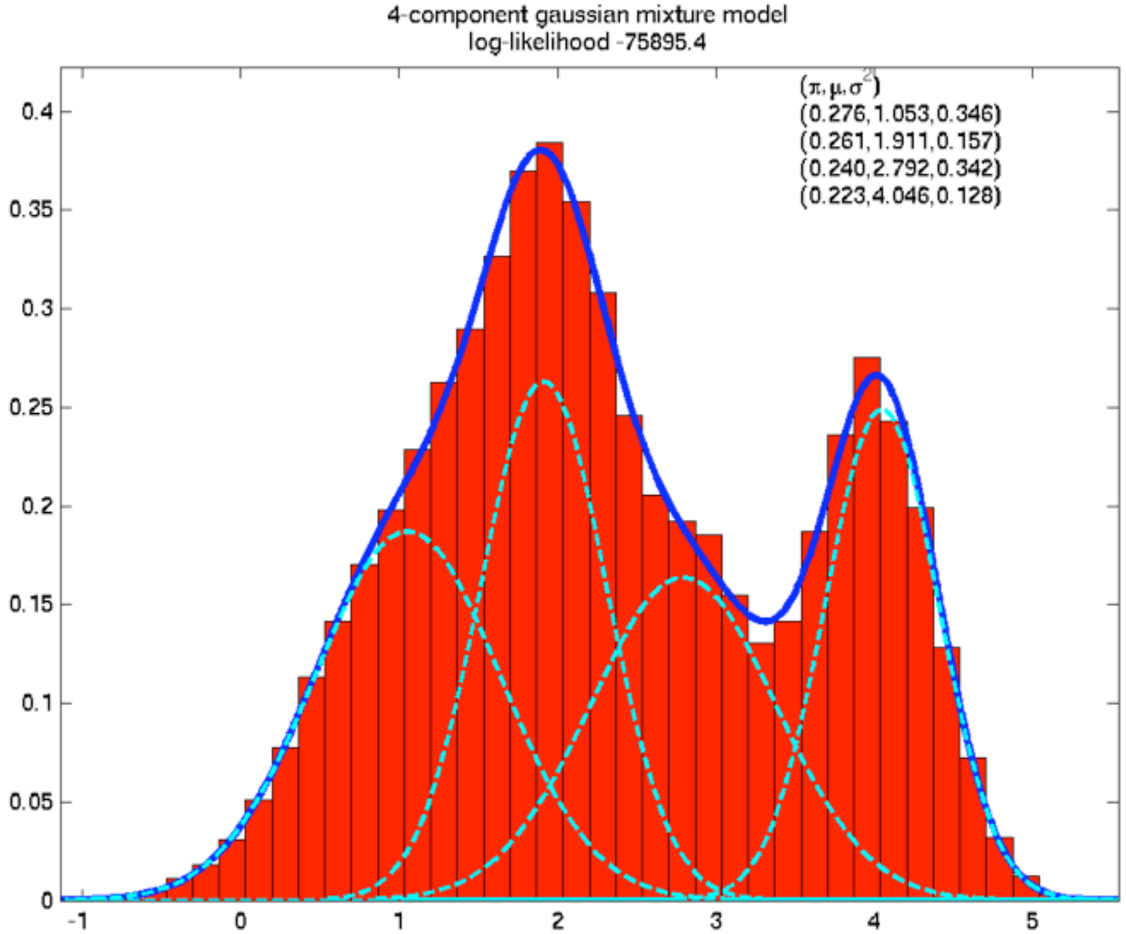
Complexity control:
Optimal number of clusters

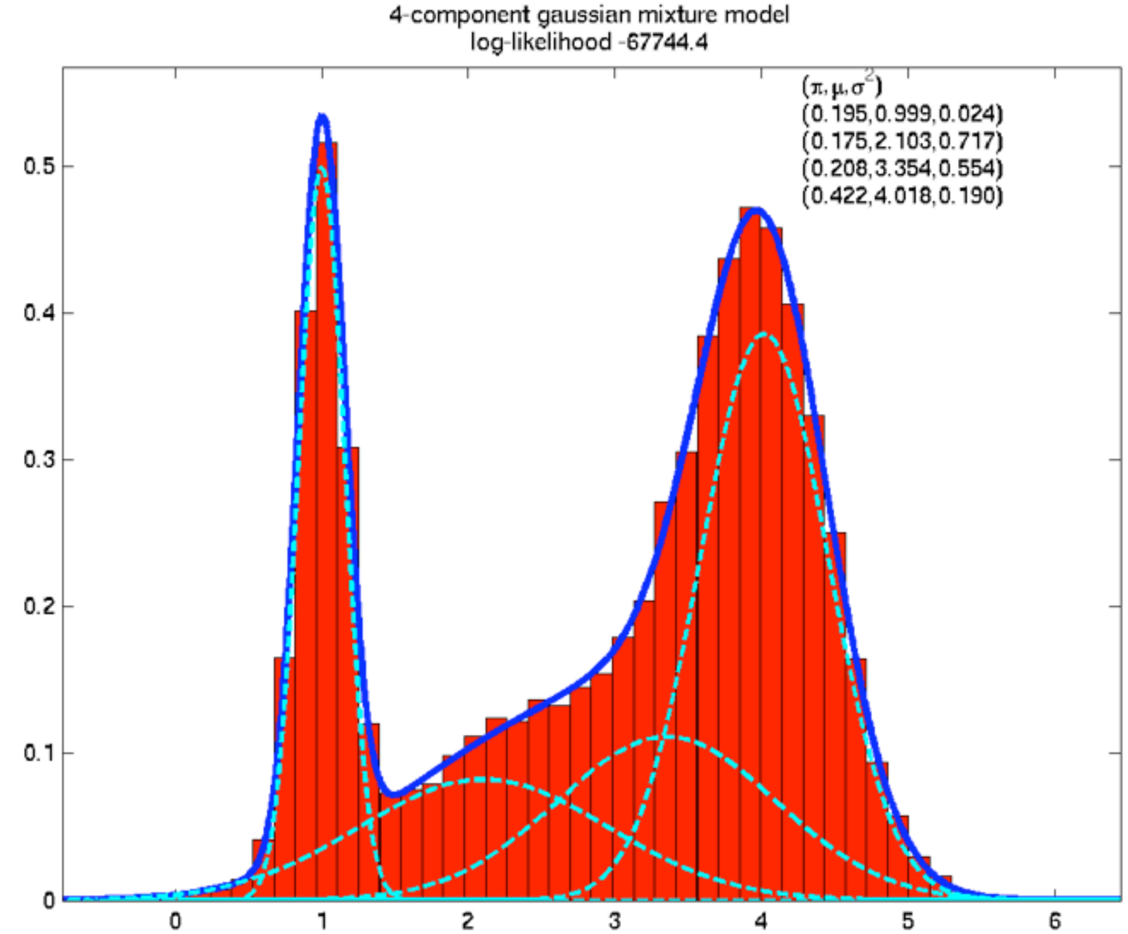




Likelihood always increases with complexity
Evidence incorporates complexity control

Complexity control:
Optimal number of clusters

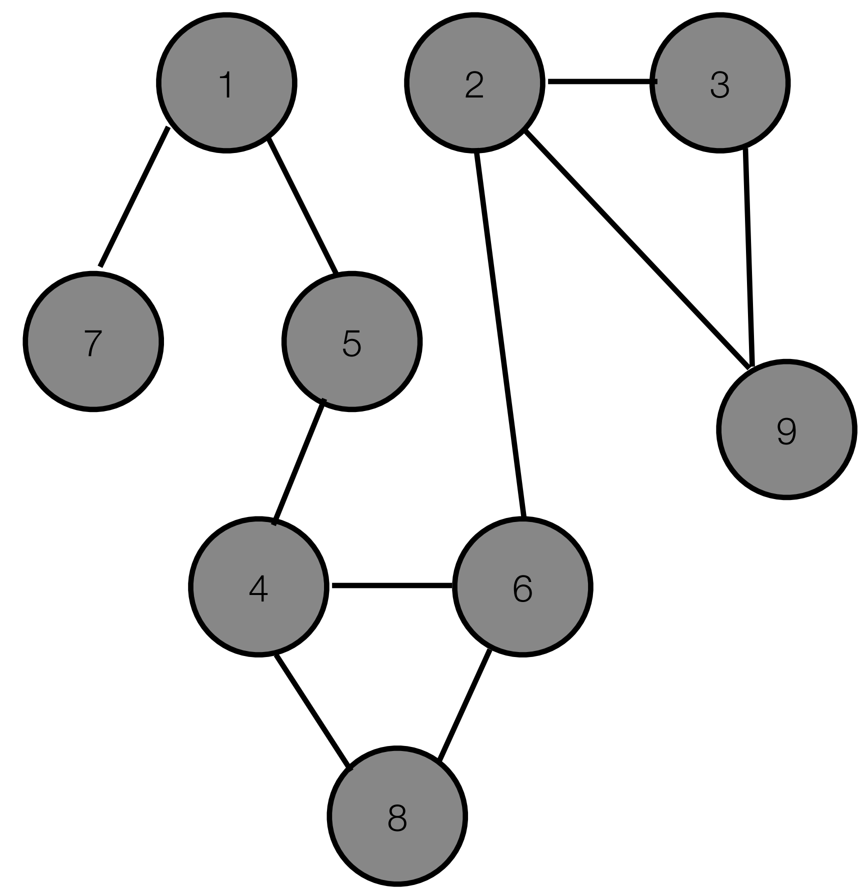




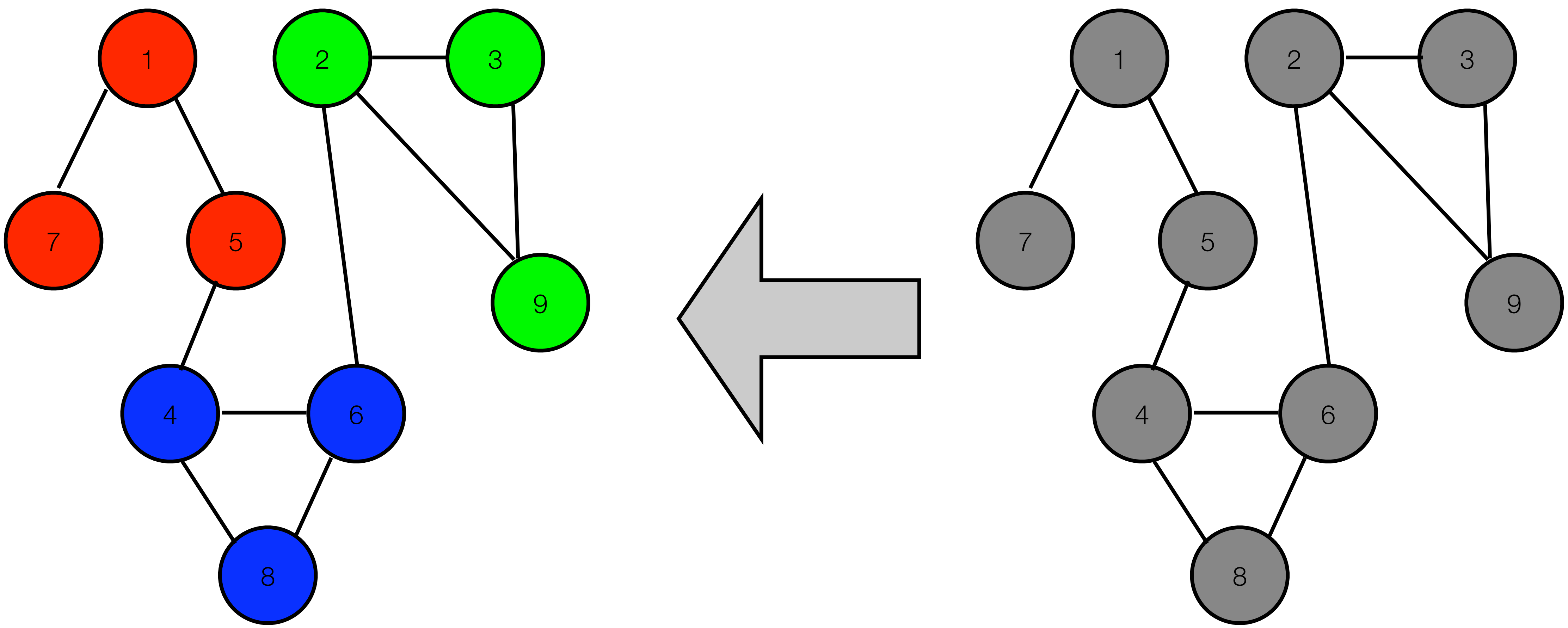
Likelihood always increases with complexity
Evidence incorporates complexity control

Overview: modular networks

- Given a network:
    - Assign nodes to modules?
    - Determine number of modules (scale/complexity)?



Overview: modular networks



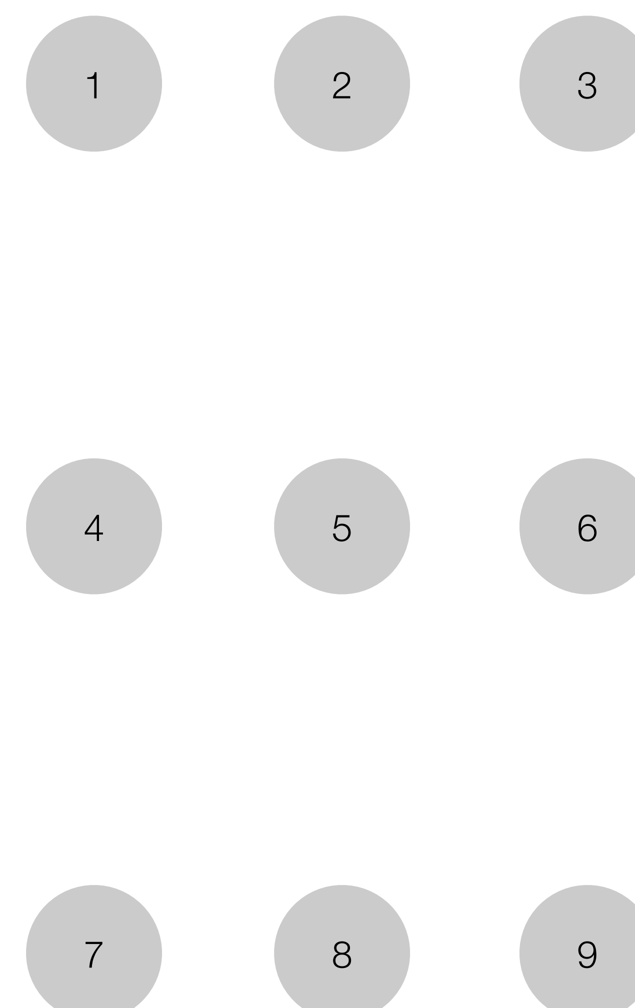
With a generative model of modular networks, rules of probability tell us how to calculate model parameters (e.g. number of modules & assignments)

Generating modular networks

- For each node:
  - **Roll K-sided die** 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
  - If $z_i \neq z_j$, **flip "between communities" coin** with bias θ_d to determine edge

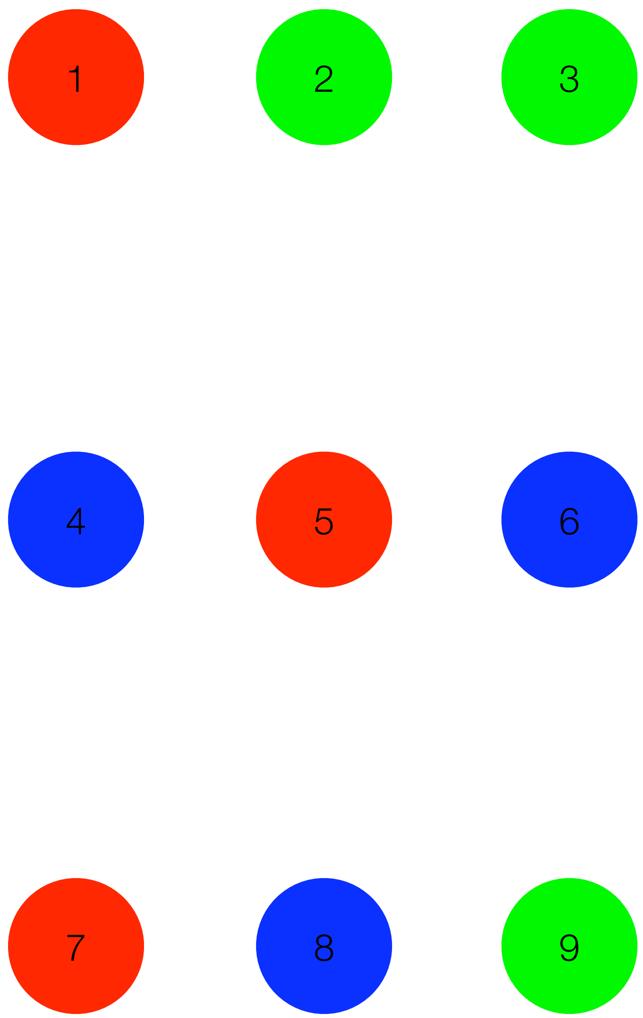
Generating modular networks

- For each node:
  - **Roll K-sided die** to determine $z_i = 1, \ldots, K$, 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
  - If $z_i \neq z_j$, **flip “between communities” coin** with bias θ_d to determine edge



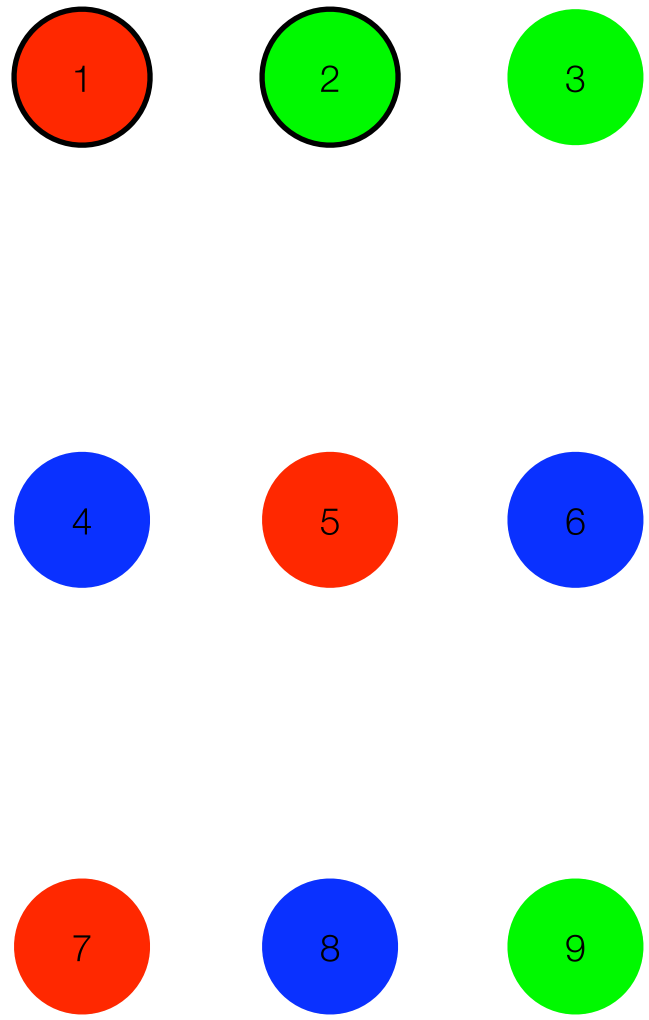
Generating modular networks

- For each node:
  - **Roll K-sided die** to determine $z_i=1,\ldots,K$, 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
  - If $z_i \neq z_j$, **flip "between communities" coin** with bias θ_d to determine edge



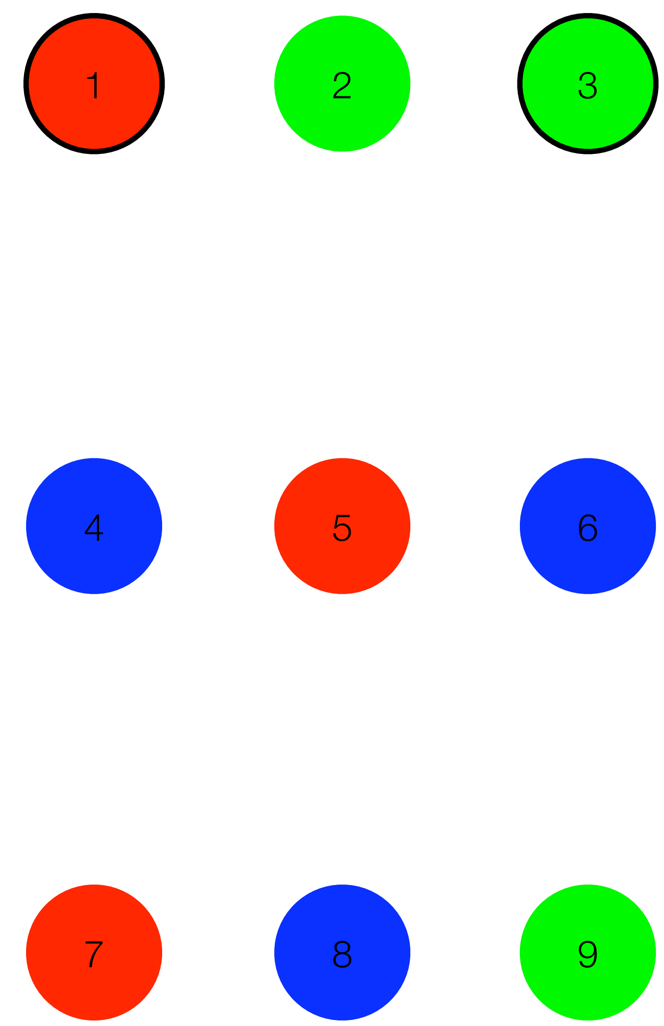
Generating modular networks

- For each node:
    - **Roll K-sided die** to determine $z_i=1,\ldots,K$, 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
    - If $z_i \neq z_j$, **flip "between communities" coin** with bias θ_d to determine edge



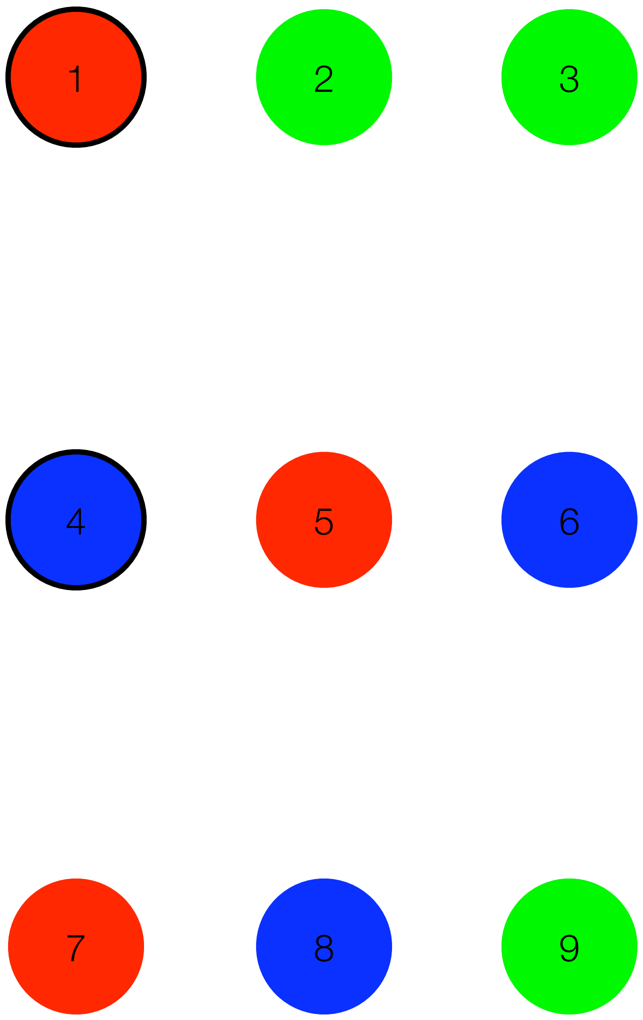
Generating modular networks

- For each node:
  - **Roll K-sided die** to determine $z_i=1,\ldots,K$, 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
  - If $z_i \neq z_j$, **flip "between communities" coin** with bias θ_d to determine edge



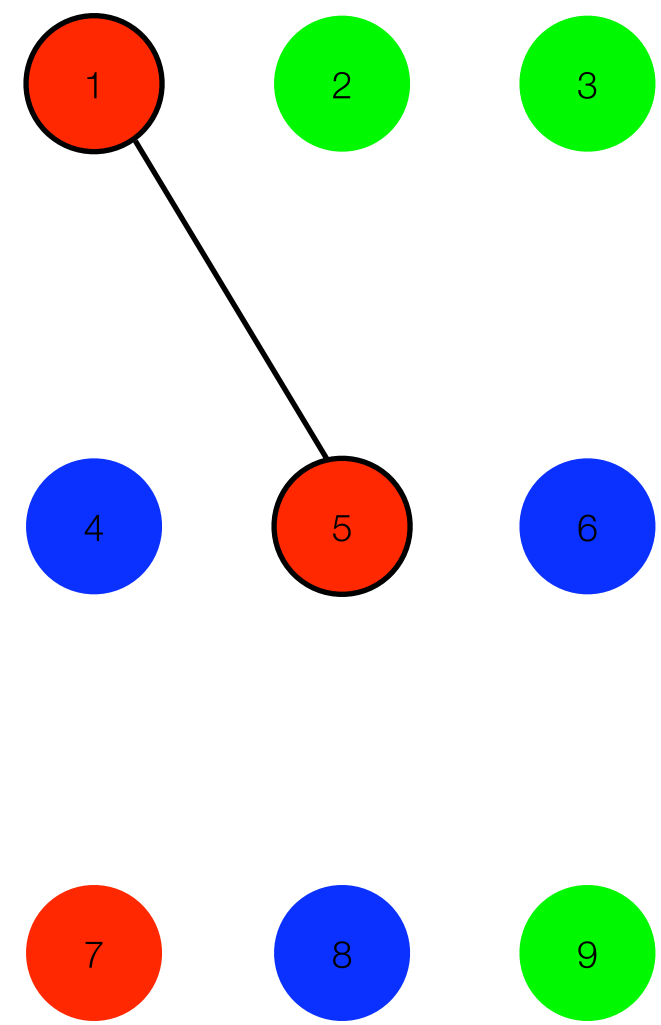
Generating modular networks

- For each node:
  - **Roll K-sided die** to determine $z_i=1,\ldots,K$, 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
  - If $z_i \neq z_j$, **flip "between communities" coin** with bias θ_d to determine edge



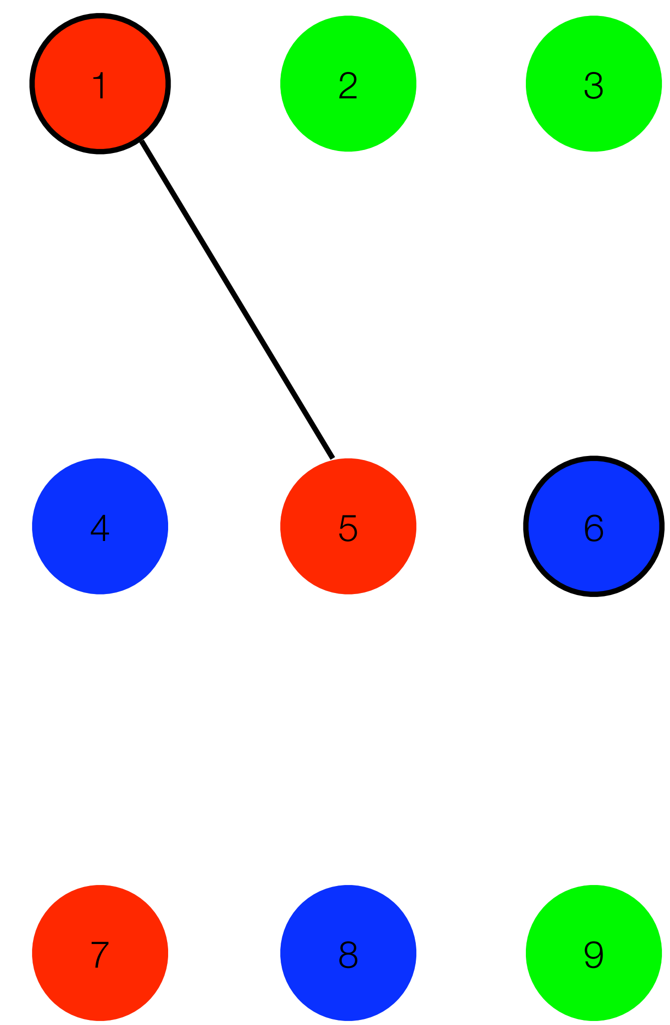
Generating modular networks

- For each node:
  - **Roll K-sided die** to determine $z_i=1,\ldots,K$, 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
  - If $z_i \neq z_j$, **flip "between communities" coin** with bias θ_d to determine edge



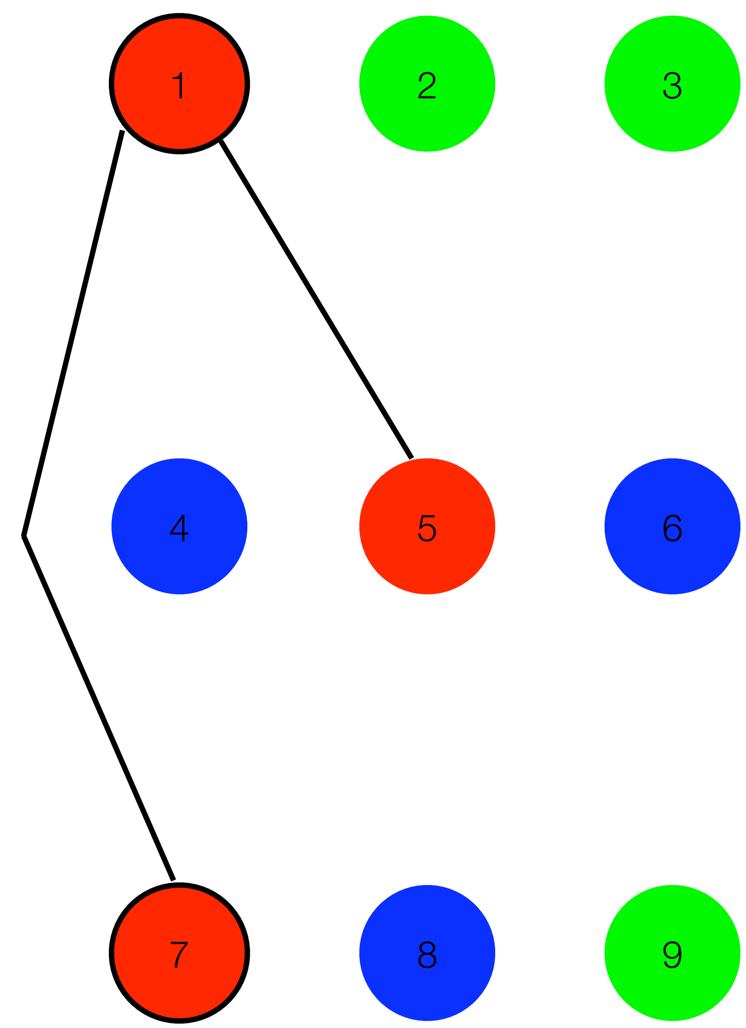
Generating modular networks

- For each node:
  - **Roll K-sided die** to determine $z_i=1,\ldots,K$, 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
  - If $z_i \neq z_j$, **flip "between communities" coin** with bias θ_d to determine edge



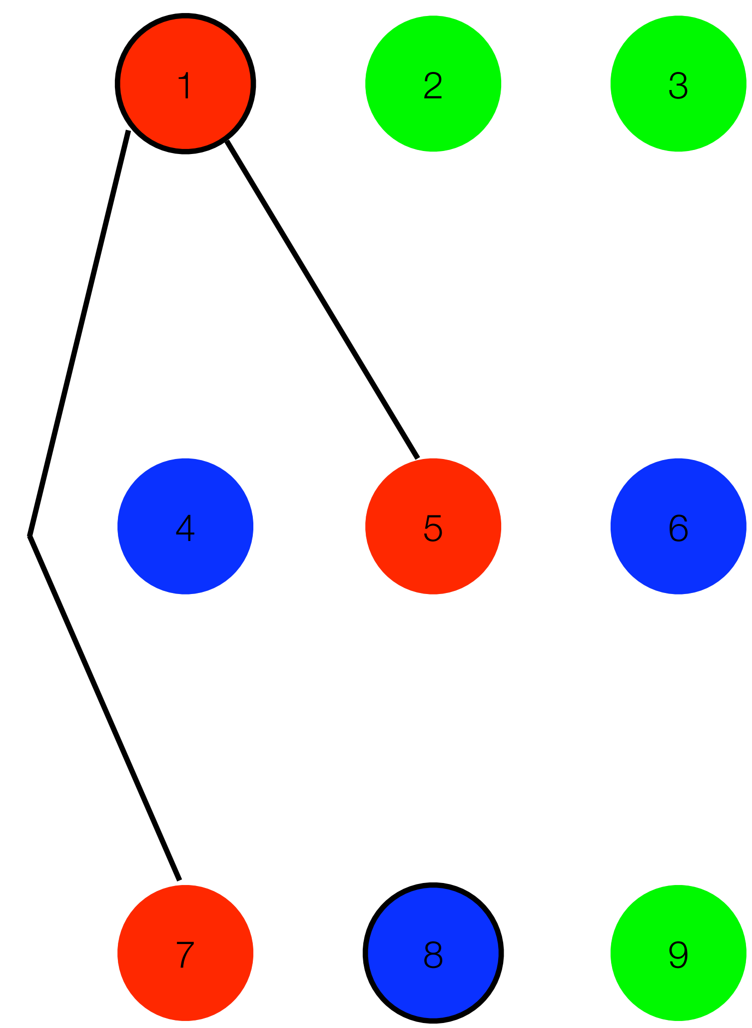
Generating modular networks

- For each node:
  - **Roll K-sided die** 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
  - If $z_i \neq z_j$, **flip "between communities" coin** with bias θ_d to determine edge



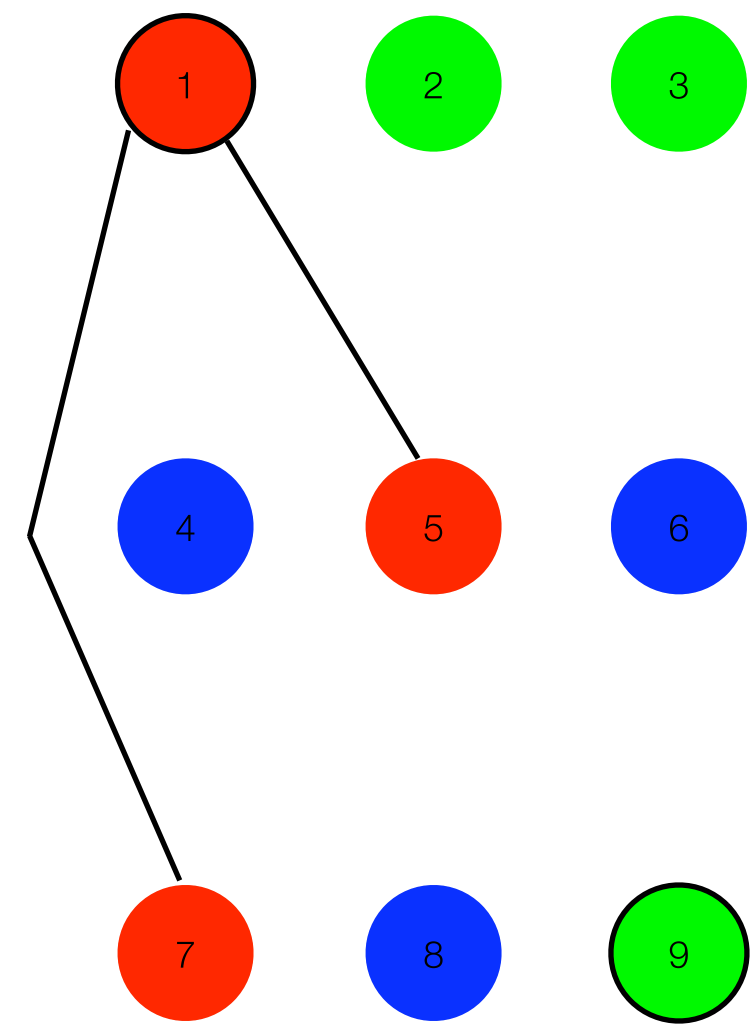
Generating modular networks

- For each node:
  - **Roll K-sided die** 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
  - If $z_i \neq z_j$, **flip "between communities" coin** with bias θ_d to determine edge



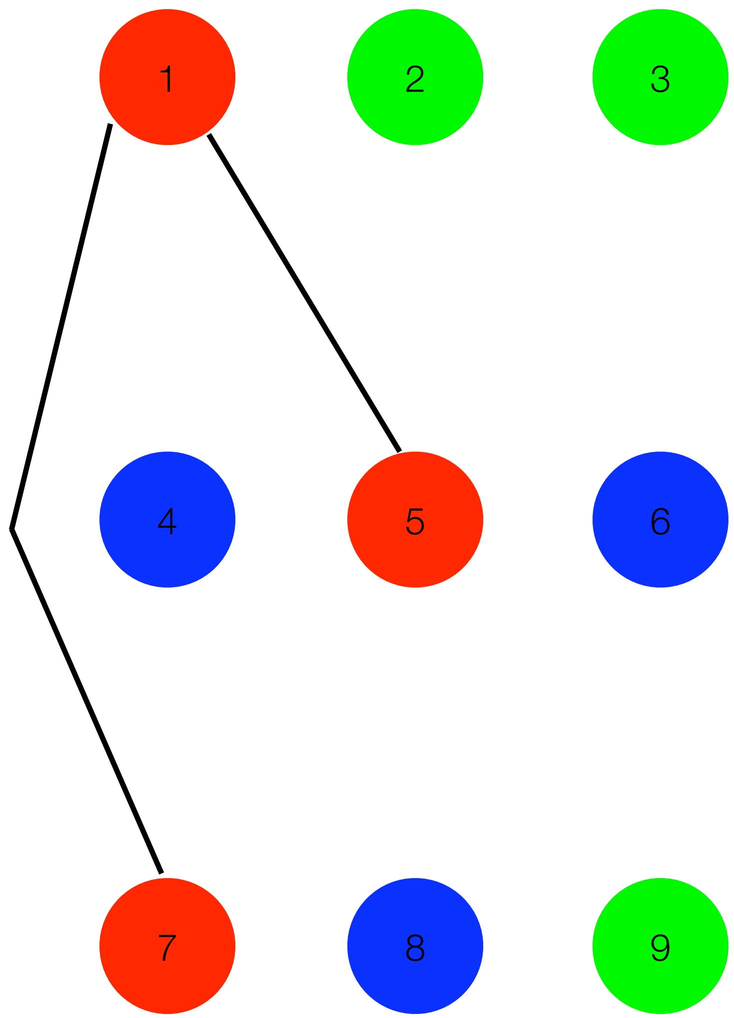
Generating modular networks

- For each node:
  - **Roll K-sided die** to determine $z_i=1,\ldots,K$, 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
  - If $z_i \neq z_j$, **flip "between communities" coin** with bias θ_d to determine edge



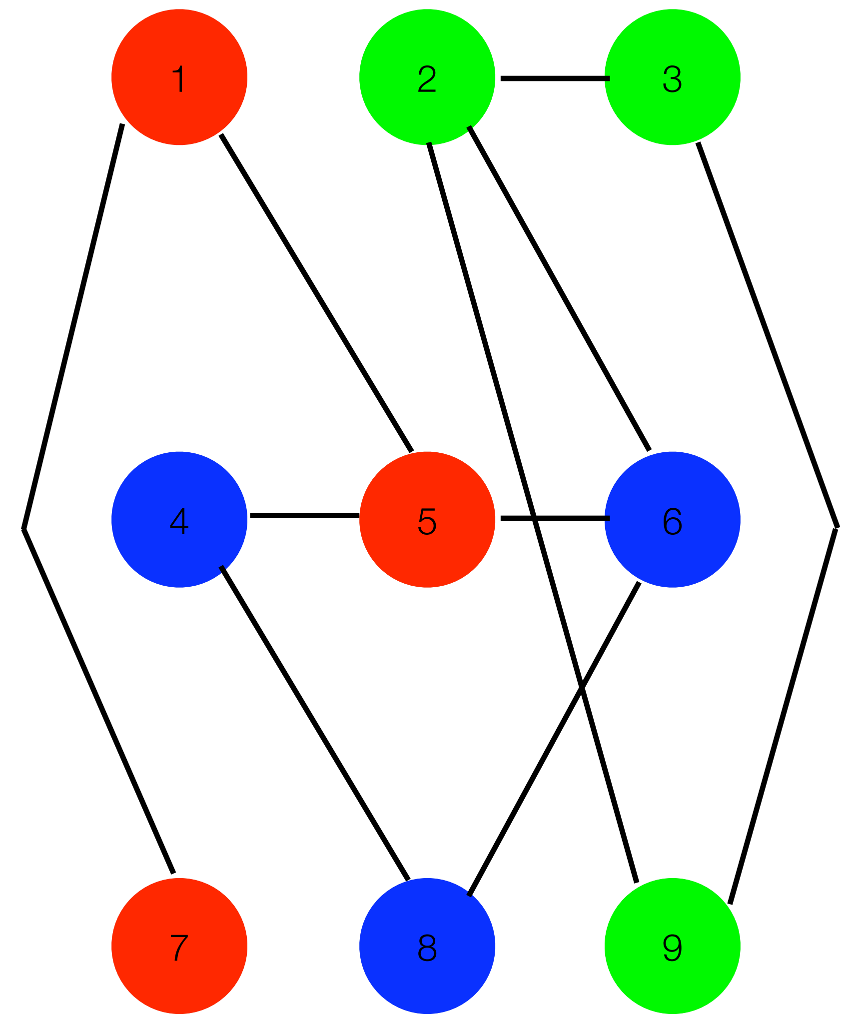
Generating modular networks

- For each node:
  - **Roll K-sided die** to determine $z_i=1,\ldots,K$, 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
  - If $z_i \neq z_j$, **flip "between communities" coin** with bias θ_d to determine edge



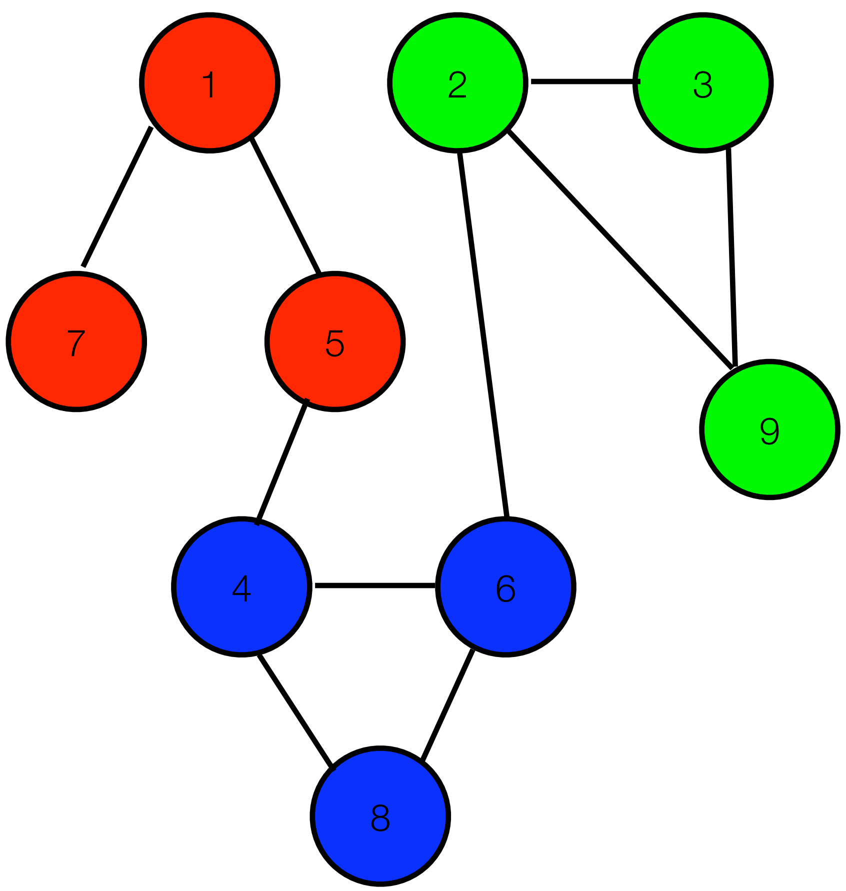
Generating modular networks

- For each node:
  - **Roll K-sided die** to determine $z_i=1,\dots,K$, 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
  - If $z_i \neq z_j$, **flip "between communities" coin** with bias θ_d to determine edge



Generating modular networks

- For each node:
  - **Roll K-sided die** to determine $z_i=1,\ldots,K$, 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
  - If $z_i \neq z_j$, **flip "between communities" coin** with bias θ_d to determine edge



Generating modular networks

Die rolling, coin flipping, and priors:

$$
\begin{aligned}
p(\vec{z}|\vec{\pi}) &\equiv \prod_{\mu=1}^{K} \pi_\mu^{n_\mu} \\
p(\mathbf{A}|\vec{z},\vec{\pi},\vec{\theta}) &\equiv \theta_c^{c_+}(1-\theta_c)^{c_-}\theta_d^{d_+}(1-\theta_d)^{d_-} \\
p(\vec{\theta}) &\equiv \mathcal{B}(\theta_c;\tilde{c}_{+_0},\tilde{c}_{-_0})\mathcal{B}(\theta_d;\tilde{d}_{+_0},\tilde{d}_{-_0}) \\
p(\vec{\pi}) &\equiv \mathcal{D}(\vec{\pi};\tilde{\vec{n}})
\end{aligned}
$$

where counts are:

edges within modules	c_+	$\equiv$	$\sum_{i,j} A_{ij}\delta_{z_i,z_j}$
non-edges within modules	c_-	$\equiv$	$\sum_{i,j}(1-A_{ij})\delta_{z_i,z_j}$
edges between modules	d_+	$\equiv$	$\sum_{i,j} A_{ij}(1-\delta_{z_i,z_j})$
non-edges between modules	d_-	$\equiv$	$\sum_{i,j}(1-A_{ij})(1-\delta_{z_i,z_j})$
nodes in each module	n_μ	$\equiv$	$\sum_{i=1}^{N}\delta_{z_i,\mu}$

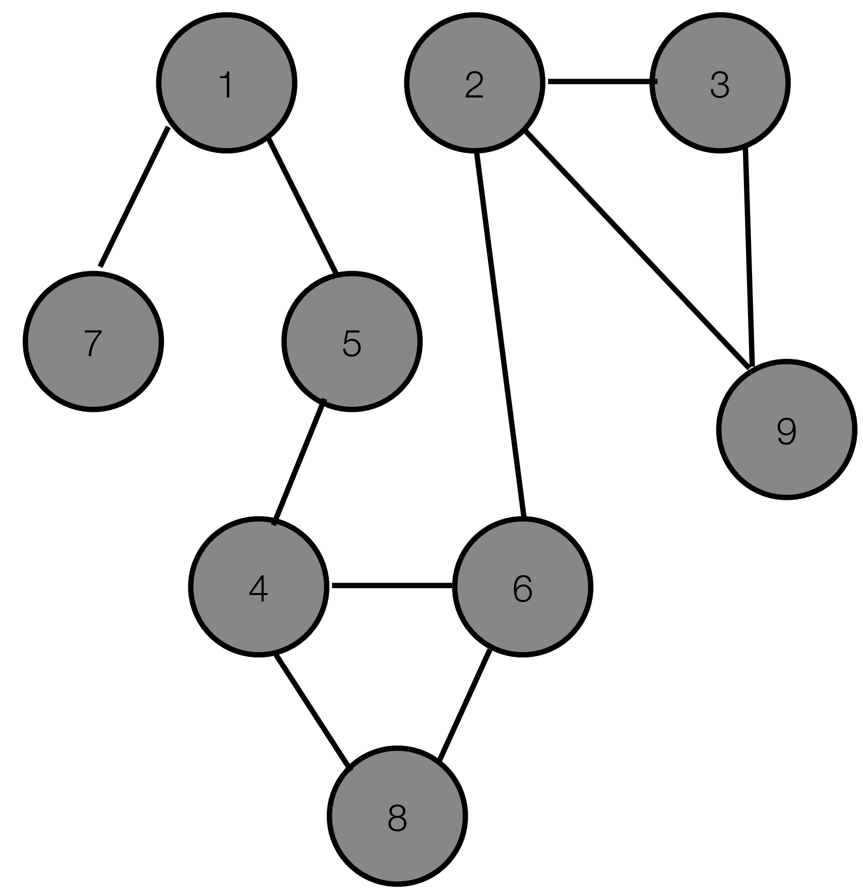
Stochastic block models (Holland, Laskey, Leinhardt 1983)

Inferring modular networks

- From observed graph structure, infer distributions over module assignments, model parameters, and model complexity

$$p(\vec{\pi}, \vec{\theta}|\mathbf{A}, K) = \frac{p(\mathbf{A}|\vec{\pi}, \vec{\theta}, K)p(\vec{\pi}, \vec{\theta}|K)}{p(\mathbf{A}|K)}$$

$$p(\vec{z}|\mathbf{A}, K) = \frac{p(\mathbf{A}|\vec{z}, K)p(\vec{z}|K)}{p(\mathbf{A}|K)}$$



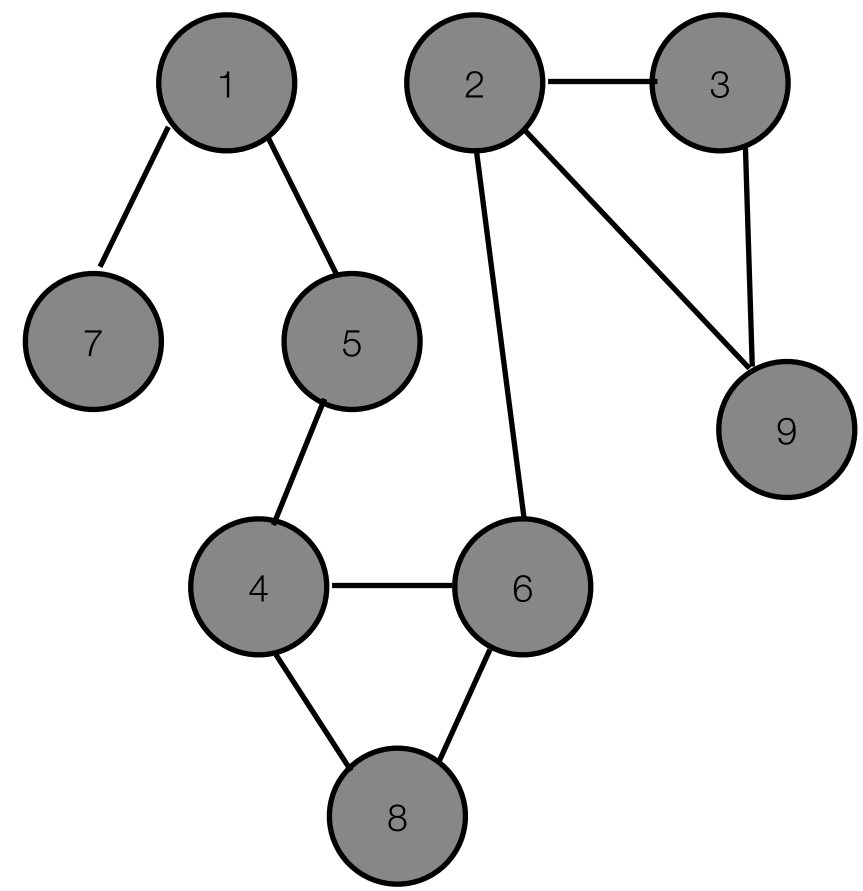
$$p(\mathbf{A}|K) \quad = \quad \sum_{\vec{z}} \int d\vec{\theta} \int d\vec{\pi} \; p(\mathbf{A}, \vec{z}, \vec{\pi}, \vec{\theta}|K)$$

Inferring modular networks

- From observed graph structure, infer distributions over module assignments, model parameters, and model complexity

$$p(\vec{\pi},\vec{\theta}|\mathbf{A},K) = \frac{p(\mathbf{A}|\vec{\pi},\vec{\theta},K)p(\vec{\pi},\vec{\theta}|K)}{p(\mathbf{A}|K)}$$

$$p(\vec{z}|\mathbf{A},K) = \frac{p(\mathbf{A}|\vec{z},K)p(\vec{z}|K)}{p(\mathbf{A}|K)}$$



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

Can do integrals, but sum is intractable, $O(K^N)$

Approximate inference for modular networks

- Jensen's inequality (log of expected value bounds expected value of log) for *any* distribution q

$$
\begin{aligned}
-\ln p(\mathbf{A}|K) &= -\ln \sum_{\vec{z}} \int d\vec{\theta} \int d\vec{\pi}\ p(\mathbf{A}, \vec{z}, \vec{\pi}, \vec{\theta}|K) \\
&= -\ln \sum_{\vec{z}} \int d\vec{\theta} \int d\vec{\pi}\ q(\vec{z}, \vec{\pi}, \vec{\theta}) \frac{p(\mathbf{A}, \vec{z}, \vec{\pi}, \vec{\theta}|K)}{q(\vec{z}, \vec{\pi}, \vec{\theta})} \\
&\leq \underbrace{-\sum_{\vec{z}} \int d\vec{\theta} \int d\vec{\pi}\ q(\vec{z}, \vec{\pi}, \vec{\theta}) \ln \frac{p(\mathbf{A}, \vec{z}, \vec{\pi}, \vec{\theta}|K)}{q(\vec{z}, \vec{\pi}, \vec{\theta})}}_{F\{q;A\}}
\end{aligned}
$$

Variational Bayes (Feynman 1972; MacKay, Jordan, Ghahramani, Jaakola, Saul 1999)

Approximate inference for modular networks

- F is a functional of q; find approximation to posterior by optimizing approximation to evidence

- Take q(z, π, θ)=q(z)q(π)q(θ); $Q_{i\mu}$ is probability node i in module μ

$$
\begin{aligned}
F\{q;\mathbf{A}\} \quad = \quad & -\ln\frac{\mathcal{Z}_{\vec{\pi}}\mathcal{Z}_c\mathcal{Z}_d}{\tilde{\mathcal{Z}}_{\vec{\pi}}\tilde{\mathcal{Z}}_c\tilde{\mathcal{Z}}_d} + \sum_{\mu=1}^{K}\sum_{i=1}^{N} Q_{i\mu}\ln Q_{i\mu} \\
& -(\tilde{c}_+ - (\langle c_+\rangle + \tilde{c}_{+_0}))\langle\ln\theta_c\rangle \\
& -(\tilde{c}_- - (\langle c_-\rangle + \tilde{c}_{+_0}))\langle\ln(1-\theta_c)\rangle \\
& -(\tilde{d}_+ - (\langle d_+\rangle + \tilde{d}_{+_0}))\langle\ln\theta_d\rangle \\
& -(\tilde{d}_- - (\langle d_-\rangle + \tilde{d}_{-_0}))\langle\ln(1-\theta_d)\rangle \\
& -\sum_{\mu=1}^{K}(\tilde{n}_\mu - (\langle n_\mu\rangle + \tilde{n}_{\mu_0}))\langle\ln\pi_\mu\rangle
\end{aligned}
$$

where expected counts are:

$$
\begin{aligned}
\langle c_+\rangle &= \frac{1}{2}Tr(\mathbf{Q}^T\mathbf{A}\mathbf{Q}) \\
\langle c_-\rangle &= \frac{1}{2}Tr(\mathbf{Q}^T\bar{\mathbf{A}}\mathbf{Q}) \\
\langle d_+\rangle &= M - \langle c_+\rangle \\
\langle d_-\rangle &= C - M - \langle c_-\rangle \\
\langle n_\mu\rangle &= \sum_{j=1}^{N} Q_{j\mu}
\end{aligned}
$$

Approximate inference for modular networks

Initialization: Initialize the N-by-K matrix $\mathbf{Q} = \mathbf{Q}_0$ and set the pseudocounts $\tilde{c}_{+_0}, \tilde{c}_{-_0}, \tilde{d}_{+_0}, \tilde{d}_{-_0}$, and $\tilde{n}_{\mu_0}$.

Main loop: Until convergence in $F\{q; \mathbf{A}\}$,

1. update $\tilde{c}_\pm$'s, $\tilde{d}_\pm$'s and $\tilde{n}_\mu$'s from expected counts and pseudocounts

$$\tilde{c}_+ = \langle c_+ \rangle + \tilde{c}_{+_0} = \frac{1}{2} Tr(\mathbf{Q}^T \mathbf{A} \mathbf{Q}) + \tilde{c}_{+_0} \tag{14}$$

$$\tilde{c}_- = \langle c_- \rangle + \tilde{c}_{-_0} = \frac{1}{2} Tr(\mathbf{Q}^T \bar{\mathbf{A}} \mathbf{Q}) + \tilde{c}_{-_0} \tag{15}$$

$$\tilde{d}_+ = \langle d_+ \rangle + \tilde{d}_{+_0} = M - \langle c_+ \rangle + \tilde{d}_{+_0} \tag{16}$$

$$\tilde{d}_- = \langle d_- \rangle + \tilde{d}_{-_0} = C - M - \langle c_- \rangle + \tilde{d}_{-_0} \tag{17}$$

$$\tilde{n}_\mu = \langle n_\mu \rangle + \tilde{n}_{\mu_0} = \sum_{j=1}^{N} Q_{j\mu} + \tilde{n}_{\mu_0}, \tag{18}$$

where $\bar{\mathbf{A}}$ is the logical negation of $\mathbf{A}$, $C = N(N-1)/2$, and $M = \frac{1}{2}\sum_{i,j} A_{ij}$;

2. update expected value of coupling constants

$$\langle J_L \rangle = \langle \ln \frac{\vartheta_c(1-\vartheta_d)}{\vartheta_d(1-\vartheta_c)} \rangle \tag{19}$$

$$= \psi(\tilde{c}_+) - \psi(\tilde{c}_-) - \psi(\tilde{d}_+) + \psi(\tilde{d}_-) \tag{20}$$

$$\langle J_G \rangle = \langle \ln \frac{(1-\vartheta_d)}{(1-\vartheta_c)} \rangle \tag{21}$$

$$= \psi(\tilde{d}_-) - \psi(\tilde{d}_+ + \tilde{d}_-) \tag{22}$$

$$-\psi(\tilde{c}_-) + \psi(\tilde{c}_+ + \tilde{c}_-),$$

where $\psi(x)$ is the digamma function;

3. update $\mathbf{Q}$ as

$$\mathbf{Q} \leftarrow \frac{1}{\mathcal{Z}} e^{\langle J_L \rangle \mathbf{A}\mathbf{Q} - \langle J_G \rangle \langle \vec{n} \rangle + \langle \ln \vec{\pi} \rangle} \tag{23}$$

where $\langle \ln \pi_\mu \rangle = \psi(\tilde{n}_\mu) - \psi(\sum_\mu \tilde{n}_\mu)$ and $\mathcal{Z}$ is set by the normalization $\sum_\mu Q_{i\mu} = 1$;

4. calculate the optimal free energy under the updated parameter distributions

$$F\{q; \mathbf{A}\} = -\ln \frac{\mathcal{Z}_c \mathcal{Z}_d \mathcal{Z}_{\vec{\pi}}}{\tilde{\mathcal{Z}}_c \tilde{\mathcal{Z}}_d \tilde{\mathcal{Z}}_{\vec{\pi}}} + \sum_{\mu=1}^{K} \sum_{i=1}^{N} Q_{i\mu} \ln Q_{i\mu}. \tag{24}$$

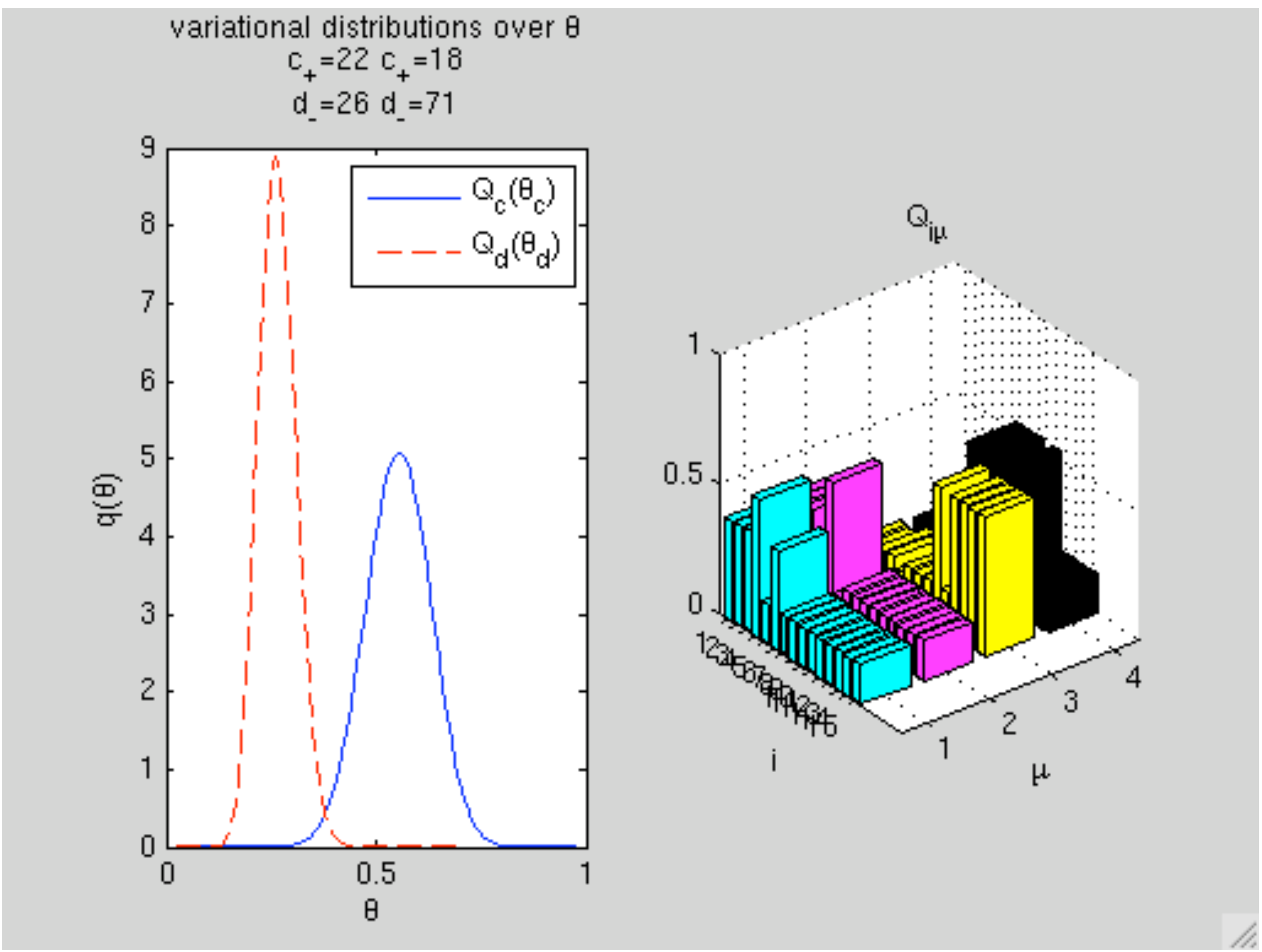
- Iteratively optimize F{q;A} by updating distributions over parameters {π, θ} and latent variables {z}

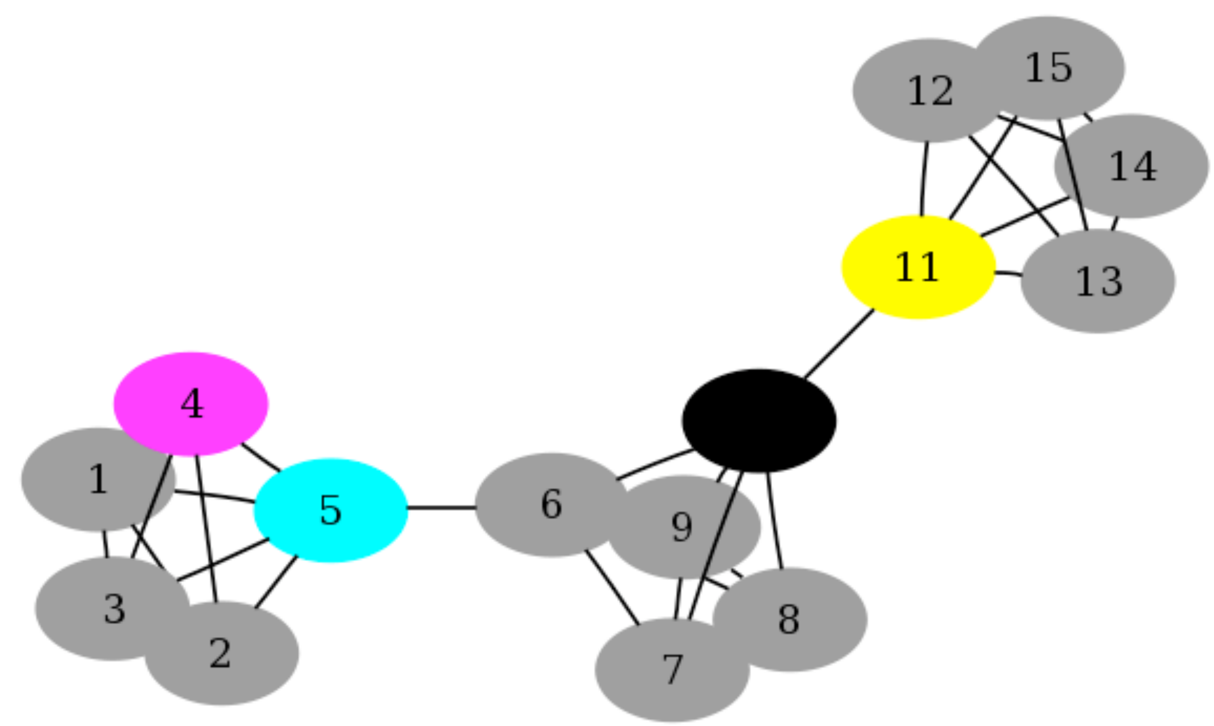
Validation: Toy graph

- 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





Validation: Complexity control

- Comparison of our method (VB) to alternative method (ICL, based on BIC) for synthetic N=60 node networks and K_{True}=3,4,5 modules

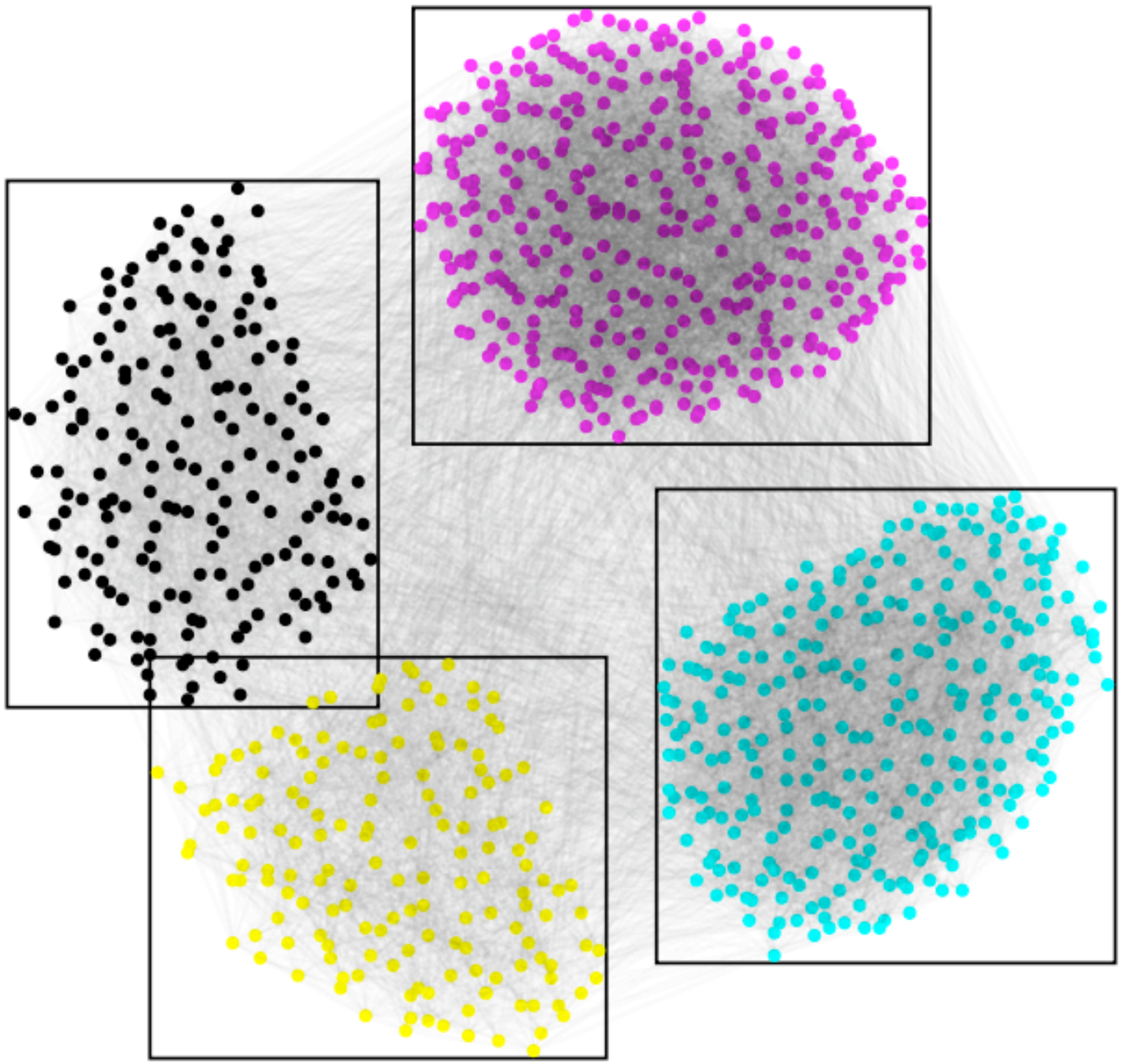
$K_{\mathrm{True}}/K_{\mathrm{VB}}$

	2	3	4	5	6
3	0	**99**	1	0	0
4	0	0	**90**	10	0
5	0	1	5	**91**	3

$K_{\mathrm{True}}/K_{\mathrm{ICL}}$

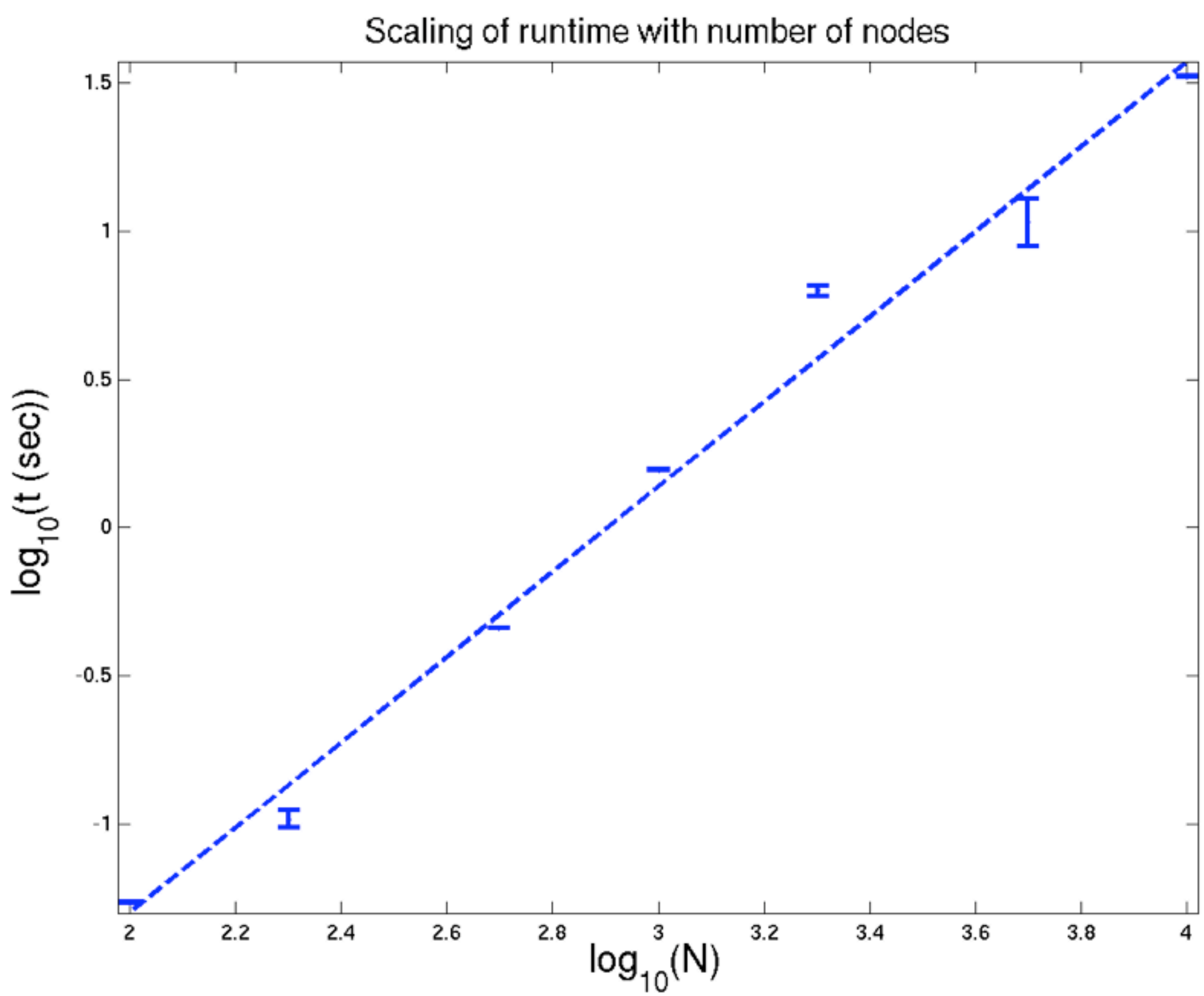
	2	3	4	5	6
3	0	**100**	0	0	0
4	4	25	**71**	0	0
5	26	55	17	**2**	0

Validation: Large-scale network



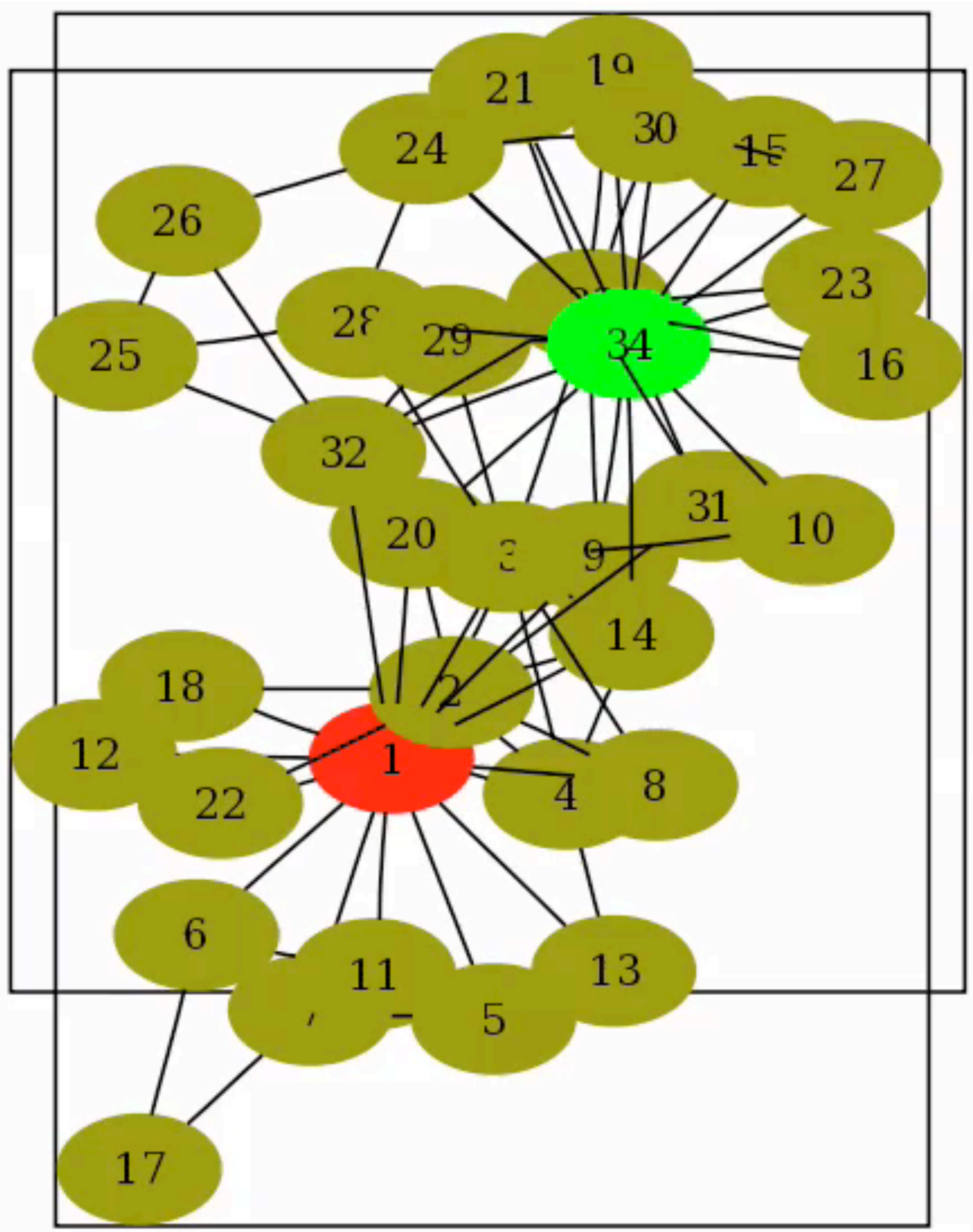
Validation: Runtime

- Main loop runtime for 10^4 nodes in MATLAB ~30 seconds

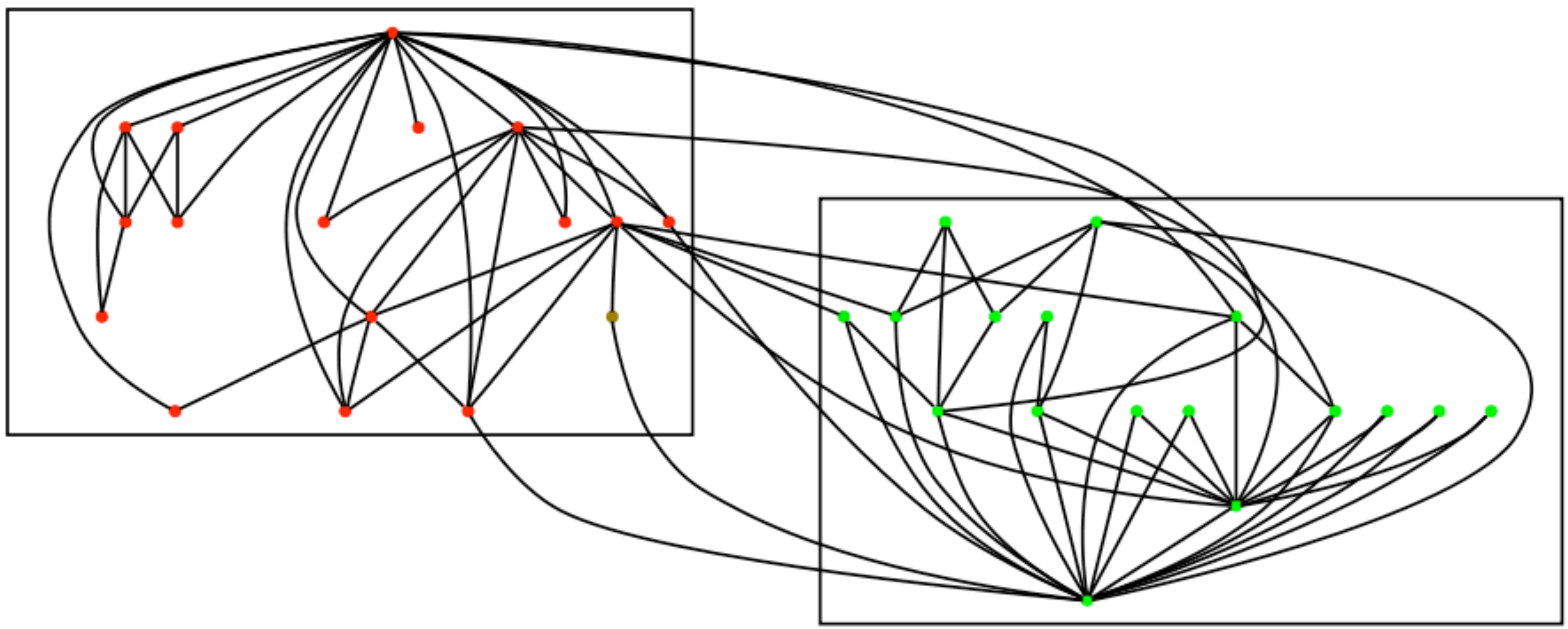


Preliminary Applications: Zachary’s karate network

Preliminary Applications: Zachary’s karate network



Preliminary Applications: Zachary’s karate network



Conclusions

- Extended work on image data to phrase modularity as a modeling problem
- Used laws of probability to infer distributions over module assignments, model parameters, and model complexity
- Resulted in a principled, accurate, and scalable algorithm
- Validated technique on synthetic and real networks
- Future: apply to protein-protein network to determine functional modules; extend model to handle alternative network structure
- preprint: http://arxiv.org/abs/0709.3512

Acknowledgments

- Chris Wiggins
- Useful discussions:
  - Joel Bader (Johns Hopkins NDC)
  - Matt Hastings (LANL)
  - Aaron Clauset (SFI)