Introduction to PGM

Vladisav Jelisavcic
Mathematical Institute of the Serbian Academy of Sciences and Arts

vladisav@mi.sanu.ac.rs

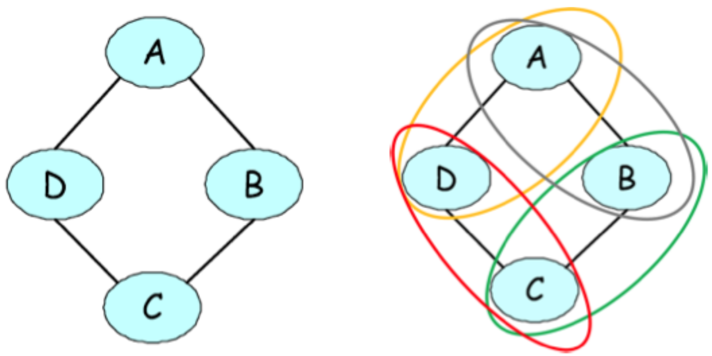
Probabilistic Graphical Models

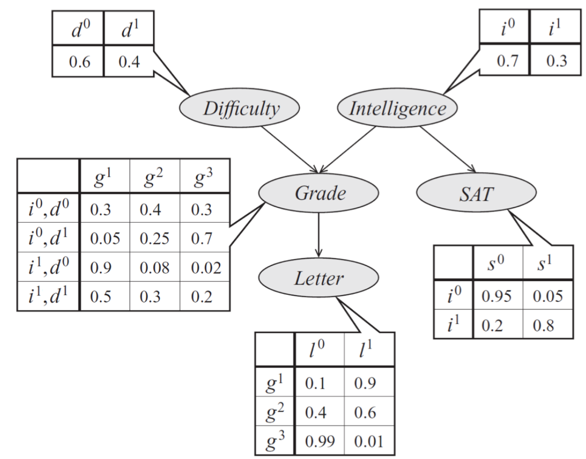
- Predict multiple variables that depend on each other
- Represent dependency as a graph
- Model uncertainty

Probabilistic Graphical Models

- Representation:
  - Markov Networks vs Bayesian Networks
- Inference:
  - Marginal inference
  - Maximum a posterior (MAP) inference
- Learning
  - Known structure
  - Structure learning

Markov Networks vs Bayesian Networks

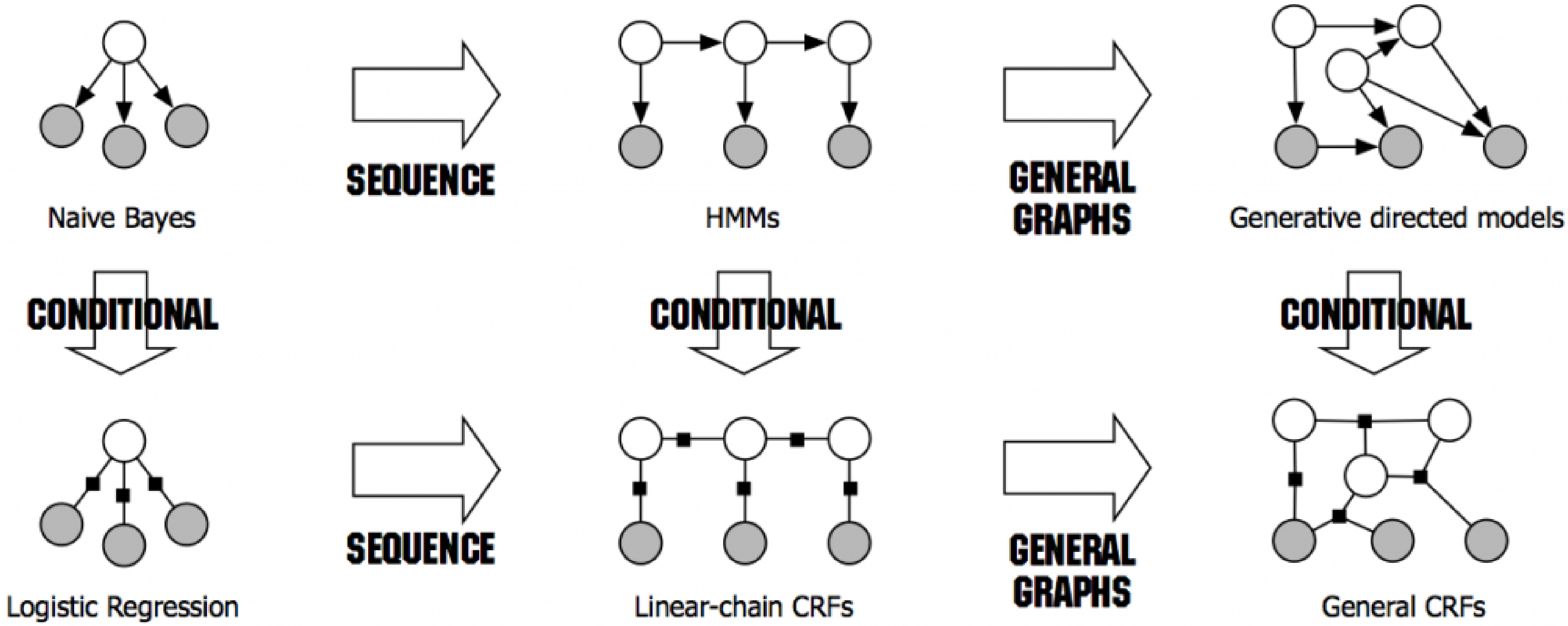




$\tilde{p}(A, B, C, D) = \phi(A, B)\phi(B, C)\phi(C, D)\phi(D, A),$

$p(l, g, i, d, s) = p(l \mid g)p(g \mid i, d)p(i)p(d)p(s \mid i).$

Markov Networks vs Bayesian Networks



Inference

- Marginal inference:

$$p(x_1) = \sum_{x_2} \sum_{x_3} \cdots \sum_{x_n} p(x_1, x_2, \ldots, x_n).$$

- Maximum a posterior (MAP) inference:

$$\max_{x_1, \ldots, x_n} p(x_1, \ldots, x_n, y = 1)$$

Applications

- Vision:
  - denoising, segmentation, generation, in-painting,...
- NLP:
  - POS tagging, generation, translation,...
- Audio
  - Super-resolution, speech synthesis, speech recognition, ...
- Bioinformatics:
  - Gene expression prediction, brain connectome modeling, ...
- ....

Markov Random Fields

- Hammersley-Clifford theorem:

  Probability distribution that has a strictly positive mass or density satisfies one of the Markov properties with respect to an ***undirected*** *graph G if and only if its density can be factorized over the cliques (or complete subgraphs) of the graph (it is a Gibbs random field)*

$$p(x_1, \dots, x_n) = \frac{1}{Z} \prod_{c \in C} \phi_c(x_c),$$

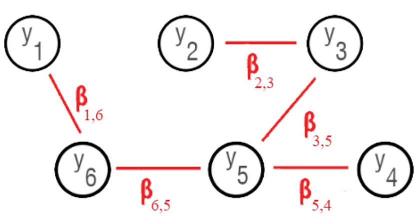
$$Z = \sum_{x_1, \dots, x_n} \prod_{c \in C} \phi_c(x_c)$$

- *MRF's factorize if at least one of the following conditions is fulfilled:*
  - *the density is positive*
  - *the graph is chordal (by equivalence to a Bayesian network)*

Pairwise Markov Networks

- Association potential for each variable
- Interaction potential for each edge in the graph
- Partition function Z
- Generative model

$$p(Y) = \frac{1}{Z}\prod f_i(y)\prod g_{ij}(y_i, y_j)$$

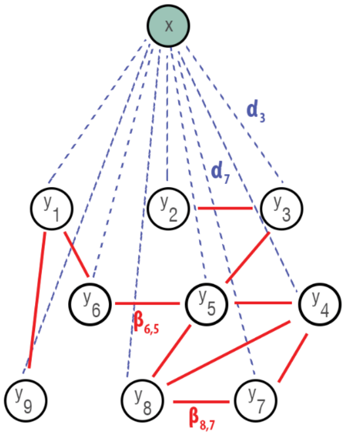


$$\begin{bmatrix} \alpha_1 & 0 & 0 & 0 & 0 & \beta_{16} \\ 0 & \alpha_2 & \beta_{23} & 0 & 0 & 0 \\ 0 & \beta_{23} & \alpha_3 & 0 & \beta_{35} & 0 \\ 0 & 0 & 0 & \alpha_4 & \beta_{45} & 0 \\ 0 & 0 & \beta_{35} & \beta_{45} & \alpha_5 & \beta_{56} \\ \beta_{16} & 0 & 0 & 0 & \beta_{56} & \alpha_6 \end{bmatrix}$$

Conditional Random Fields

- Association potential for each variable $f(y_i|x)$
- Interaction potential for each edge in the graph $g(y_i, y_j|x)$
- Partition function Z
- Discriminative model

$$p(Y|X) = \frac{1}{Z} \prod f(y_i|X) \prod g_{ij}(y_i, y_j|X)$$



Discrete vs Continuous

- Discrete:
  - structured classification
  - partition function makes inference challenging

- Continuous:
  - structured regression
  - not tractable in general case
  - closed form expression for Z in special cases

Inference

- NP-hard in many cases (both marginal and MAP)
- Tractability depends on the structure of the graph that describes that probability
- Useful answers still possible
  via approximate inference methods

Exact inference: variable elimination

- Intuition (example BN – linear chain):

$$p(x_1, \ldots, x_n) = p(x_1) \prod_{i=2}^{n} p(x_i \mid x_{i-1}).$$

  - naive: $O(d^n)$

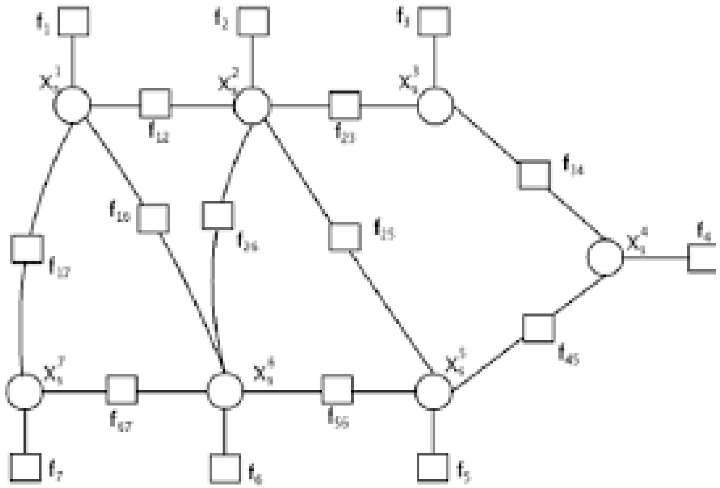
$$p(x_n) = \sum_{x_1} \cdots \sum_{x_{n-1}} p(x_1, \ldots, x_n)$$

  - push variables: $O(nd^2)$

$$\begin{aligned} p(x_n) &= \textstyle\sum_{x_1} \cdots \sum_{x_{n-1}} p(x_1) \prod_{i=2}^{n} p(x_i \mid x_{i-1}) \\ &= \textstyle\sum_{x_{n-1}} p(x_n \mid x_{n-1}) \sum_{x_{n-2}} p(x_{n-1} \mid x_{n-2}) \cdots \sum_{x_1} p(x_2 \mid x_1) p(x_1). \end{aligned}$$

Exact inference: variable elimination

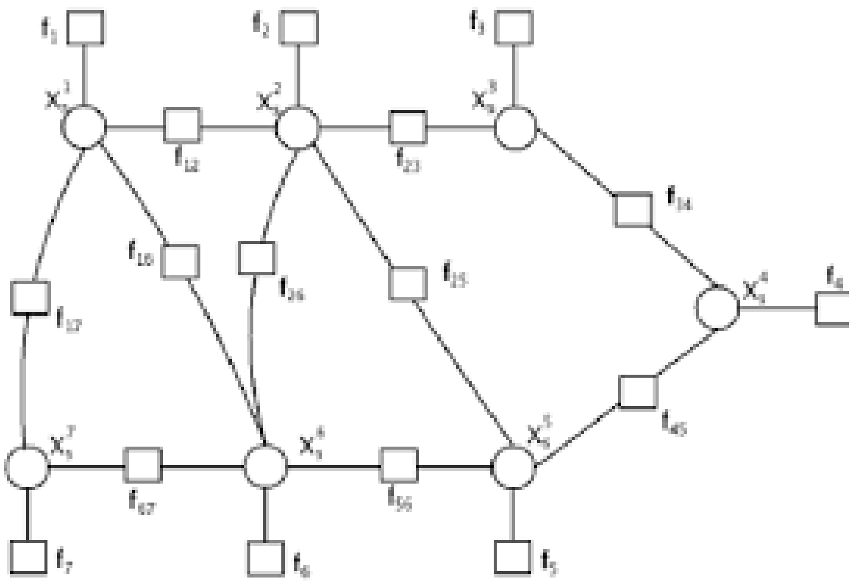
- Factor graph:

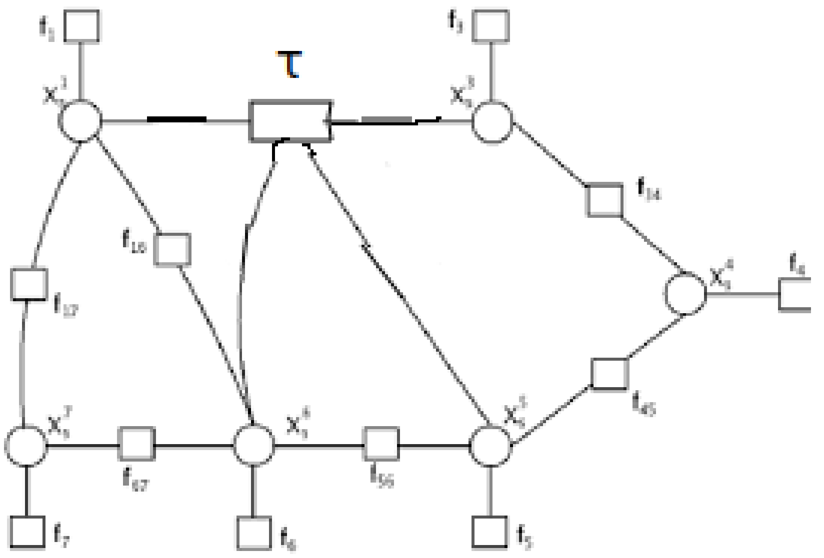


$$p(x_1, .., x_n) = \prod_{c \in C} \phi_c(x_c).$$

- For each variable Xi (ordered according to O):
  - Multiply all factors Φi containing Xi
  - Marginalize out Xi to obtain new factor τ
  - Replace the factors in Φi by τ

Exact inference: variable elimination





Exact inference: variable elimination

- Running time: $O(md^M)$
  - M: maximum size of any factor during the elimination process
  - m: number of variables

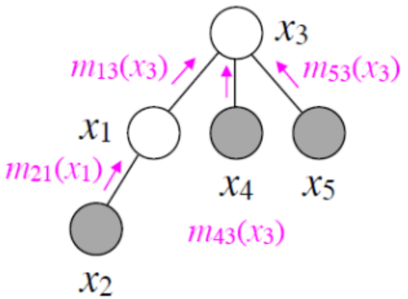
- Different ordering dramatically alter
  the running time of the variable elimination algorithm
- It is NP-hard to find the best ordering
- Heuristics:
  - Min-neighbors: Choose a variable with the fewest dependent variables
  - Min-weight: Choose variables to minimize
    the product of the cardinalities of its dependent variables
  - Min-fill: Choose vertices to minimize
    the size of the factor that will be added to the graph

Belief propagation

- Variable elimination (VE) can answer marginal queries of the form $P(Y|E=e)$
- If we want to ask the model for another query, e.g. $P(Y2|E2=e2)$, we need to restart the algorithm from scratch (wasteful)
- VE produces many intermediate factors τ as a side-product of the main computation
  - same factors that we needed for other marginal queries
  - after first run of VE, we can easily answer new marginal queries at essentially no additional cost

Belief propagation

- Intuition:
  - if we apply VE algorithm on a tree
    in order to compute a marginal p(Xi),
    we can easily find an optimal ordering for this problem
    by rooting the tree at Xi
    and iterating through the nodes in post-order



  - when xk is marginalized out, it receives all the signal
    from variables underneath it from the tree
  - this signal can be completely summarized in a factor τ(xj)
  - τ(xj) as a message that xj sends to xk
    to summarize all it knows about its children variables

Belief propagation

- If we apply VE algorithm on a tree
  in order to compute a marginal p(Xk)
  - root the tree at Xk
  - factors can be reused

- Message passing algorithm:
  - Node xi sends a message to a neighbor xj
    whenever it has received messages from all nodes besides xj
  - There will always be a node with a message to send,
    unless all the messages have been sent out
  - Terminates after precisely 2|E| steps,
    since each edge can receive messages only twice:
    once from xi→xj, and once more in the opposite direction

Sum-product message passing

- While there is a node xi ready to transmit to xj, send the message:

$$m_{i \to j}(x_j) = \sum_{x_i} \phi(x_i)\phi(x_i, x_j) \prod_{\ell \in N(i) \setminus j} m_{\ell \to i}(x_i).$$

- This message is precisely the factor τ that xi would transmit to xj during a round of variable elimination with the goal of computing p(xj)

- Any marginal query over xi can be answered in constant time using the equation:

$$p(x_i) = \prod_{\ell \in N(i)} m_{\ell \to i}(x_i)$$

Max-product message passing

- Answers MAP inference queries: $\max_{x_1,\ldots,x_n} p(x_1,\ldots,x_n)$
- While there is a node xi ready to transmit to xj, send the message:

$$m_{i\to j}(x_j) = \max_{x_i} \phi(x_i)\phi(x_i, x_j) \prod_{\ell\in N(i)\setminus j} m_{\ell\to i}(x_i)$$

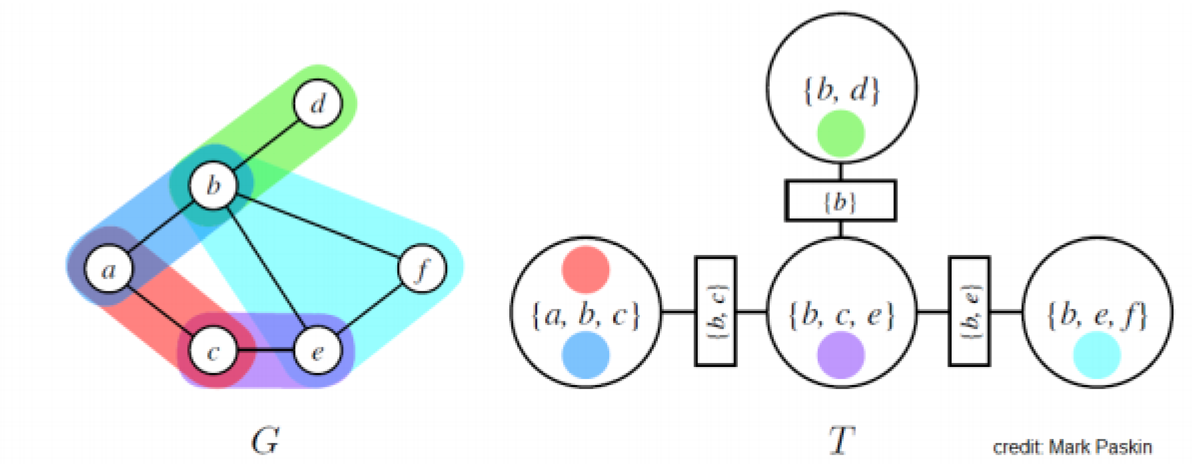
- Any MAP query over xi can be answered in constant time using the equation:

$$p(x_i) = \prod_{\ell\in N(i)} m_{\ell\to i}(x_i)$$

- Property that makes this work is the distributivity of both the sum and the max operator over products
- Most probable assignment by keeping back-pointers during the optimization procedure

Junction tree

- Turn a graph into a tree of clusters that are amenable to the variable elimination, then perform message-passing on this tree

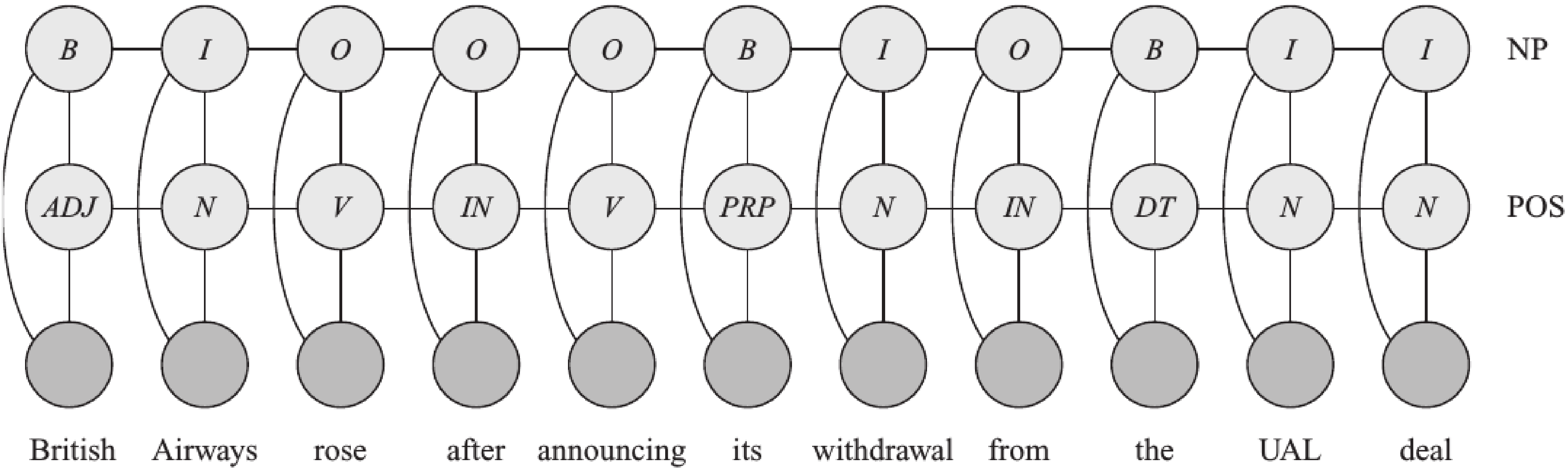


Loopy belief propagation

- Approximate
- Disregard loops in the graph and perform message passing anyway
- Fixed number of steps or until convergence
- Messages are typically initialized uniformly
- Given an ordering on the edges,
  at each time t we iterate over a pair of adjacent variables xi,xj
  in that order and simply perform the update:

$$m_{i \to j}^{t+1}(x_j) = \sum_{x_i} \phi(x_i)\phi(x_i, x_j) \prod_{\ell \in N(i) \setminus j} m_{\ell \to i}^{t}(x_i).$$

Application: POS tagging with CRF



KEY

B	Begin noun phrase	*V*	Verb
I	Within noun phrase	*IN*	Preposition
O	Not a noun phrase	*PRP*	Possesive pronoun
N	Noun	*DT*	Determiner (e.g., a, an, the)
ADJ	Adjective		

Application: POS tagging with CRF

$$p(\mathbf{y}|\mathbf{x}) = \underbrace{\frac{1}{Z(\mathbf{x})}}_{\text{Normalization}} \prod_{t=1}^{T} \exp\left\{\sum_{k=1}^{K} \underbrace{\theta_k}_{\text{Weight}} \underbrace{f_k(y_t, y_{t-1}, \mathbf{x}_t)}_{\text{Feature}}\right\}$$

Prediction

- Structured prediction = MAP inference

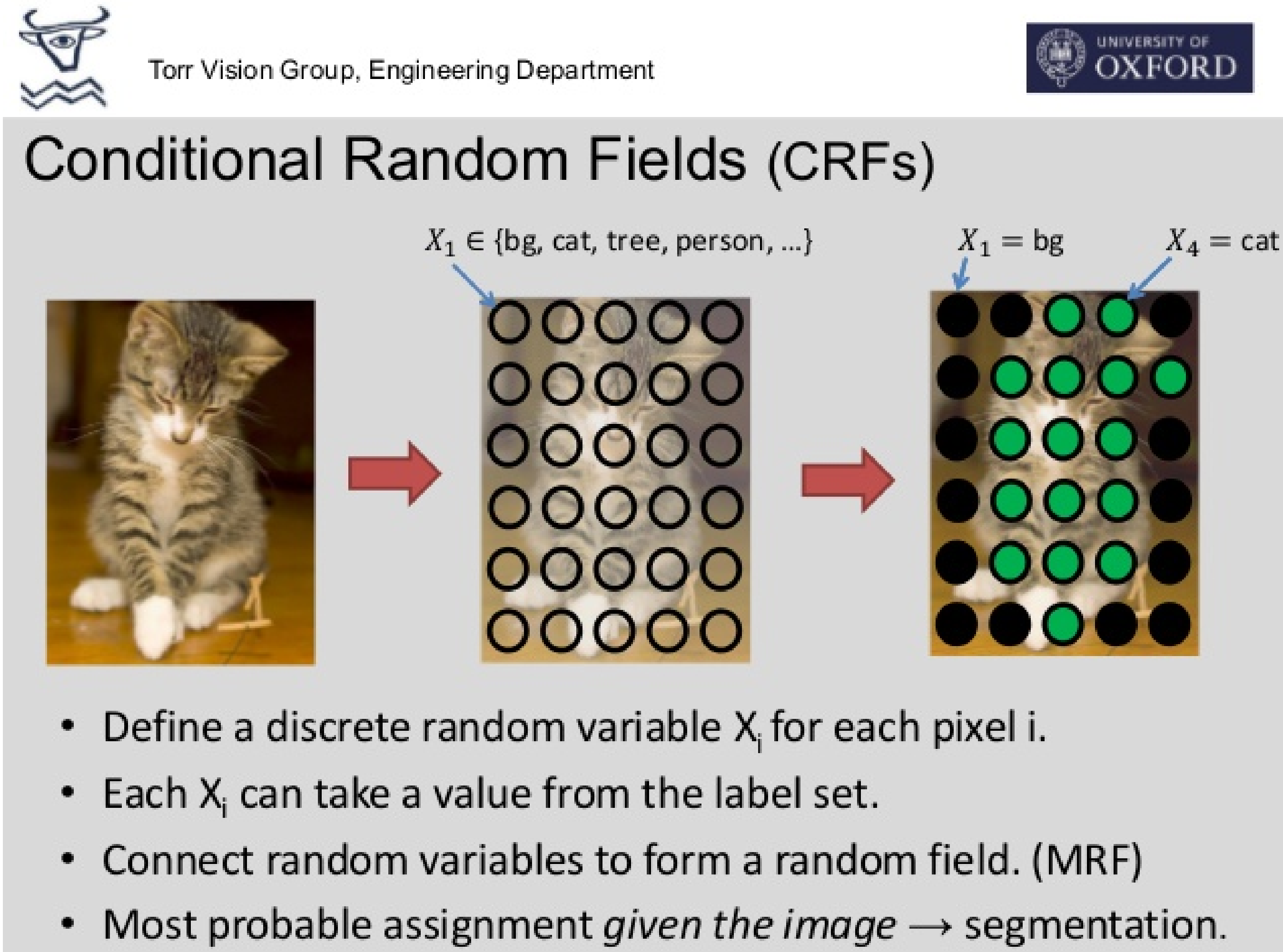
$$\arg\max_{y} \log p(y|x) = \arg\max_{y} \sum_{c} \theta_c(\mathbf{f}y_c, \mathbf{f}x_c).$$

- More efficient algorithms:
  - Graphcuts
  - ILP
  - Dual decomposition
  - ...

Learning

- Log-likelihood: $$\frac{1}{|D|}\log p(D;\theta) = \frac{1}{|D|}\sum_{x,y\in D}\theta^T f(x,y) - \frac{1}{|D|}\sum_{x\in D}\log Z(x,\theta).$$
- Gradient: $$\frac{1}{|D|}\sum_{x,y\in D} f(x,y) - \frac{1}{|D|}\sum_{x\in D}\mathbb{E}_{y\sim p(y|x)}[f(x,y)]$$
- Hessian: $$\mathrm{cov}_{y\sim p(y|x)}[f(x,y)]$$
- Calculating the log-partition part of the gradient requires one full inference for each data point

Application:
Image segmentation with CRF



Continuous Markov Fields

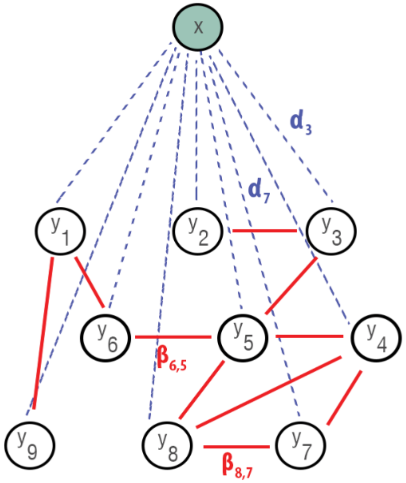
- Structured regression
- Partition function has no closed form solution (in general case)
- Efficient in special cases:
  - quadratic potentials/Gaussian model

Continuous CRF

$$P(Y|X) = \frac{1}{Z}\exp(-\sum_k \alpha^k \sum_i (y_i - R_i^k(X))^2 - \sum_l \beta^l \sum_{(i,j)} S_{ij}^l (y_i - y_j)^2)$$

Y ~ Multivariate Gaussian distribution

$$P(Y|X) = (2\pi)^{-p/2}\det(\Sigma)^{-1/2}\exp\left(-\frac{1}{2}(Y-\mu(X)))^T\Sigma^{-1}(Y-\mu(X)))\right)$$



Continuous CRF

- MAP inference:

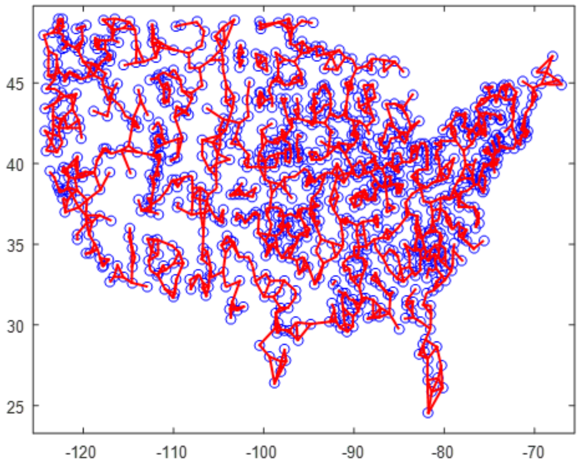
$$b_i = 2 \sum_k \alpha_i^k R_i^k(X)$$

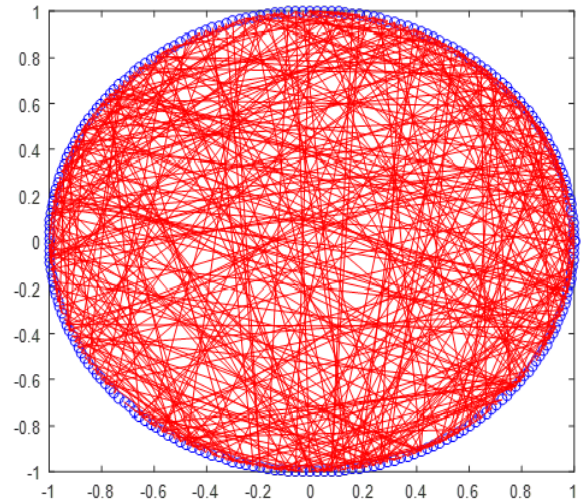
- Learning: maximum likelihood
- Convex

GCRF: applications

- Bioinformatics,
  healthcare,
  energy forecasting,
  sensor networks,
  weather forecasting,

  ...





Structure learning

- Sparsity inducing regularization
- Chow-Liu
- BIC/AIC

Structured prediction models vs neural networks

- In fully observed models, the likelihood is convex;
  in latent variable models it is not
- If you add connections among the nodes in the output layer,
  and if you have a good set of features,
  then sometimes you don't need a hidden layer
- If you can afford to leave out the hidden,
  then in practice you always want to do so,
  because this avoids all of the problems with local minima

*[Sutton, Charles, and Andrew McCallum. "An introduction to conditional random fields." Foundations and Trends® in Machine Learning 4.4 (2012): 267-373.]

Structured prediction models vs neural networks

- For harder problems one might expect that even after modeling output structure, incorporating hidden state will still yield additional benefit

- Once hidden state is introduced into the model, whether it be a neural network or a structured model, it seems to be inevitable that convexity will be lost (at least given our current understanding of machine learning)

*[Sutton, Charles, and Andrew McCallum. "An introduction to conditional random fields." Foundations and Trends® in Machine Learning 4.4 (2012): 267-373.]

Literature

- Sutton, Charles, and Andrew McCallum. "An introduction to conditional random fields." Foundations and Trends® in Machine Learning 4.4 (2012): 267-373.
- Koller, Daphne, Nir Friedman, and Francis Bach. Probabilistic graphical models: principles and techniques. MIT press, 2009.
- Wainwright, Martin J., and Michael I. Jordan. "Graphical models, exponential families, and variational inference." Foundations and Trends® in Machine Learning 1.1–2 (2008): 1-305.
- Elkan, Charles. "Log-linear models and conditional random fields." Tutorial notes at CIKM 8 (2008): 1-12.
- Zheng, Shuai, et al. "Conditional random fields as recurrent neural networks." Proceedings of the IEEE international conference on computer vision. 2015.
- Radosavljevic, Vladan, Slobodan Vucetic, and Zoran Obradovic. "Continuous conditional random fields for regression in remote sensing." ECAI. 2010.
- Stojkovic, I., Jelisavcic, V., Milutinovic, V., & Obradovic, Z. Distance Based Modeling of Interactions in Structured Regression.
- https://ermongroup.github.io/cs228-notes/