Machine Learning – Lecture 16

Inference & Applications of MRFs

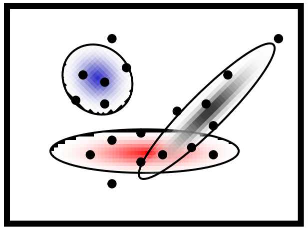
30.06.2015

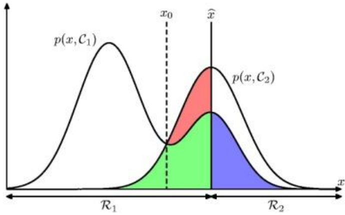
Bastian Leibe
RWTH Aachen
http://www.vision.rwth-aachen.de

leibe@vision.rwth-aachen.de

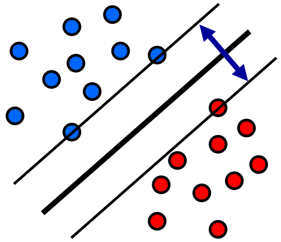
Course Outline

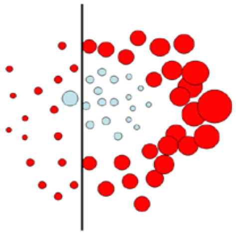
- **Fundamentals (2 weeks)**
  - Bayes Decision Theory
  - Probability Density Estimation



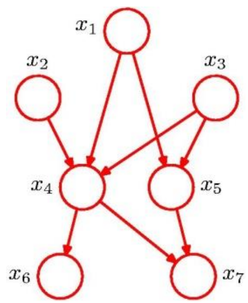


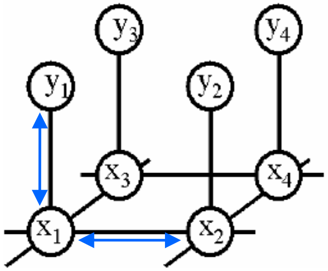
- **Discriminative Approaches (5 weeks)**
  - Linear Discriminant Functions
  - Statistical Learning Theory & SVMs
  - Ensemble Methods & Boosting
  - Decision Trees & Randomized Trees





- **Generative Models (4 weeks)**
  - Bayesian Networks
  - Markov Random Fields
  - Exact Inference
  - Applications

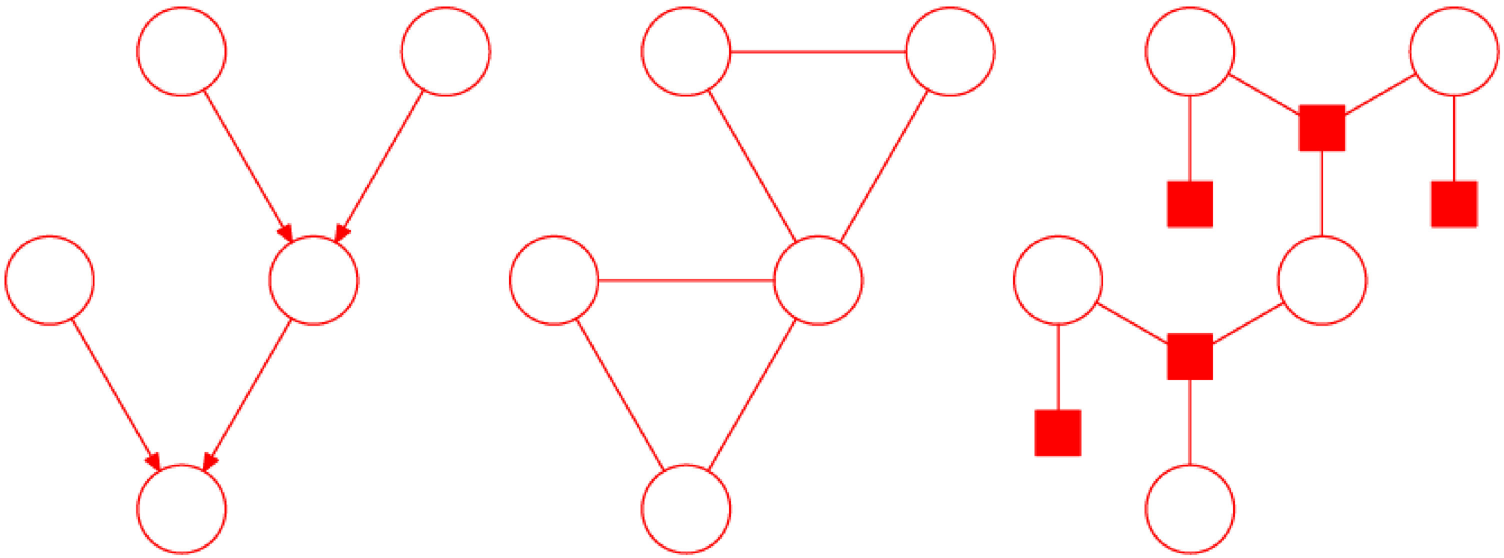




Topics of This Lecture

- Recap: Exact inference
  - Sum-Product algorithm
  - Max-Sum algorithm
  - Junction Tree algorithm
- Applications of Markov Random Fields
  - Application examples from computer vision
  - Interpretation of clique potentials
  - Unary potentials
  - Pairwise potentials
- Solving MRFs with Graph Cuts
  - Graph cuts for image segmentation
  - s-t mincut algorithm
  - Extension to non-binary case
  - Applications

Recap: Factor Graphs



- **Joint probability**
  - Can be expressed as product of factors: $p(\mathbf{x}) = \frac{1}{Z} \prod_s f_s(\mathbf{x}_s)$
  - Factor graphs make this explicit through separate factor nodes.
- **Converting a directed polytree**
  - Conversion to undirected tree creates loops due to moralization!
  - Conversion to a factor graph again results in a tree!

Image source: C. Bishop, 2006

Recap: Sum-Product Algorithm

- Objectives
  - Efficient, exact inference algorithm for finding marginals.
- Procedure:
  - Pick an arbitrary node as root.
  - Compute and propagate messages from the leaf nodes to the root, storing received messages at every node.
  - Compute and propagate messages from the root to the leaf nodes, storing received messages at every node.
  - Compute the product of received messages at each node for which the marginal is required, and normalize if necessary.

$$p(x) \propto \prod_{s \in \mathrm{ne}(x)} \mu_{f_s \to x}(x)$$

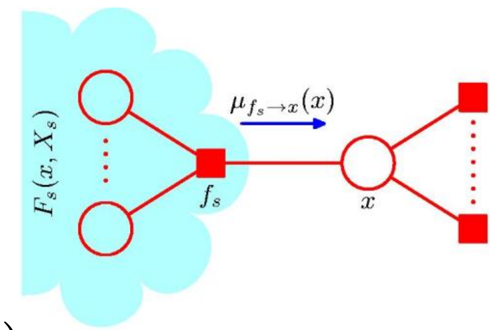
- Computational effort
  - Total number of messages = 2 · number of graph edges.

Recap: Sum-Product Algorithm

- Two kinds of messages
  - Message from factor node to variable nodes:
    - **Sum** of factor contributions

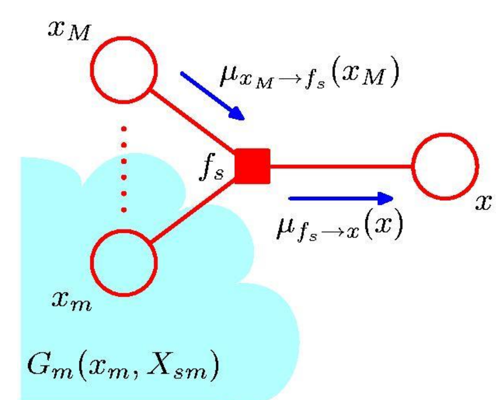
$$\mu_{f_s \to x}(x) \equiv \sum_{X_s} F_s(x, X_s)$$

$$= \sum_{X_s} f_s(\mathbf{x}_s) \prod_{m \in \mathrm{ne}(f_s) \setminus x} \mu_{x_m \to f_s}(x_m)$$



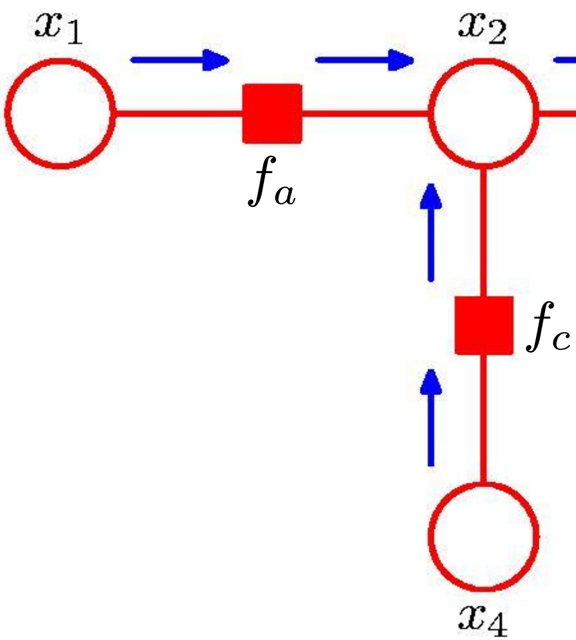
  - Message from variable node to factor node:
    - **Product** of incoming messages

$$\mu_{x_m \to f_s}(x_m) \equiv \prod_{l \in \mathrm{ne}(x_m) \setminus f_s} \mu_{f_l \to x_m}(x_m)$$



$\Rightarrow$ Simple propagation scheme.

Recap: Sum-Product from Leaves to Root

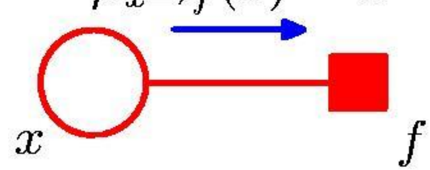


Message definitions:

$$\mu_{f_s \to x}(x) \equiv \sum_{X_s} f_s(\mathbf{x}_s) \prod_{m \in \mathrm{ne}(f_s) \setminus x} \mu_{x_m \to f_s}(x_m)$$

$$\mu_{x_m \to f_s}(x_m) \equiv \prod_{l \in \mathrm{ne}(x_m) \setminus f_s} \mu_{f_l \to x_m}(x_m)$$

$$\mu_{x \to f}(x) = 1$$



$$\mu_{f \to x}(x) = f(x)$$

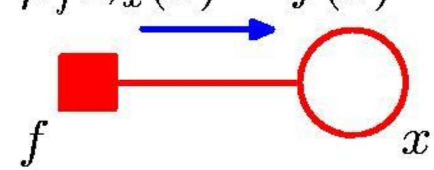
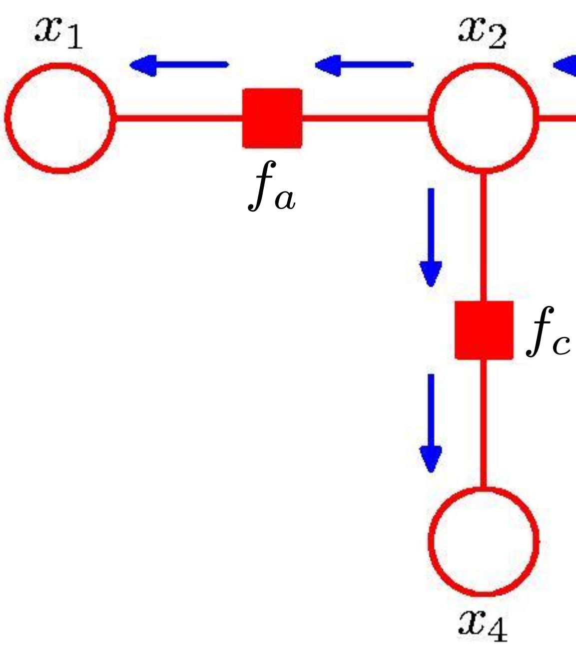


Image source: C. Bishop, 2006

Recap: Sum-Product from Root to Leaves

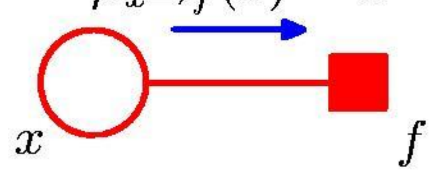


Message definitions:

$$\mu_{f_s \to x}(x) \equiv \sum_{X_s} f_s(\mathbf{x}_s) \prod_{m \in \mathrm{ne}(f_s) \setminus x} \mu_{x_m \to f_s}(x_m)$$

$$\mu_{x_m \to f_s}(x_m) \equiv \prod_{l \in \mathrm{ne}(x_m) \setminus f_s} \mu_{f_l \to x_m}(x_m)$$

$$\mu_{x \to f}(x) = 1$$



$$\mu_{f \to x}(x) = f(x)$$

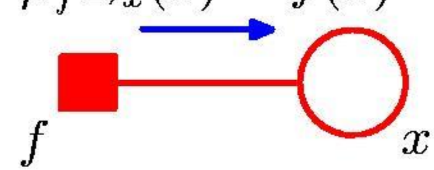


Image source: C. Bishop, 2006

Recap: Max-Sum Algorithm

- Objective: an efficient algorithm for finding
  - Value $\mathbf{x}^{\max}$ that maximises $p(\mathbf{x})$;
  - Value of $p(\mathbf{x}^{\max})$.

  $\Rightarrow$ Application of dynamic programming in graphical models.

- Key ideas
  - We are interested in the maximum value of the joint distribution

$$p(\mathbf{x}^{\max}) = \max_{\mathbf{x}} p(\mathbf{x})$$

  $\Rightarrow$ Maximize the product $p(\mathbf{x})$.

  - For numerical reasons, use the logarithm.

$$\ln\left(\max_{\mathbf{x}} p(\mathbf{x})\right) = \max_{\mathbf{x}} \ln p(\mathbf{x}).$$

  $\Rightarrow$ Maximize the sum (of log-probabilities).

Recap: Max-Sum Algorithm

- **Initialization (leaf nodes)**

$$\mu_{x \to f}(x) = 0 \qquad\qquad \mu_{f \to x}(x) = \ln f(x)$$

- **Recursion**
  - **Messages**

$$\mu_{f \to x}(x) = \max_{x_1,\ldots,x_M} \left[\ln f(x, x_1, \ldots, x_M) + \sum_{m \in \mathrm{ne}(f_s) \setminus x} \mu_{x_m \to f}(x_m) \right]$$

$$\mu_{x \to f}(x) = \sum_{l \in \mathrm{ne}(x) \setminus f} \mu_{f_l \to x}(x)$$

  - **For each node, keep a record of which values of the variables gave rise to the maximum state:**

$$\phi(x) = \arg\max_{x_1,\ldots,x_M} \left[\ln f(x, x_1, \ldots, x_M) + \sum_{m \in \mathrm{ne}(f_s) \setminus x} \mu_{x_m \to f}(x_m) \right]$$

Recap: Max-Sum Algorithm

- **Termination (root node)**
  - **Score of maximal configuration**

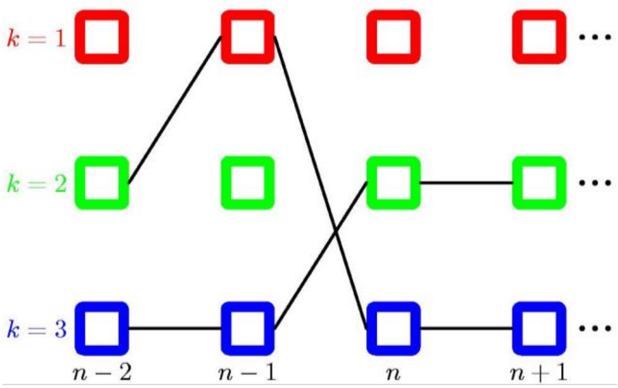
$$p^{\max} = \max_x \left[\sum_{s \in \mathrm{ne}(x)} \mu_{f_s \to x}(x) \right]$$

  - **Value of root node variable giving rise to that maximum**

$$x^{\max} = \arg\max_x \left[\sum_{s \in \mathrm{ne}(x)} \mu_{f_s \to x}(x) \right]$$

  - **Back-track to get the remaining variable values**

$$x_{n-1}^{\max} = \phi(x_n^{\max})$$



Topics of This Lecture

- Factor graphs
  - Construction
  - Properties
- Sum-Product Algorithm for computing marginals
  - Key ideas
  - Derivation
  - Example
- Max-Sum Algorithm for finding most probable value
  - Key ideas
  - Derivation
  - Example
- Algorithms for loopy graphs
  - Junction Tree algorithm
  - Loopy Belief Propagation

Junction Tree Algorithm

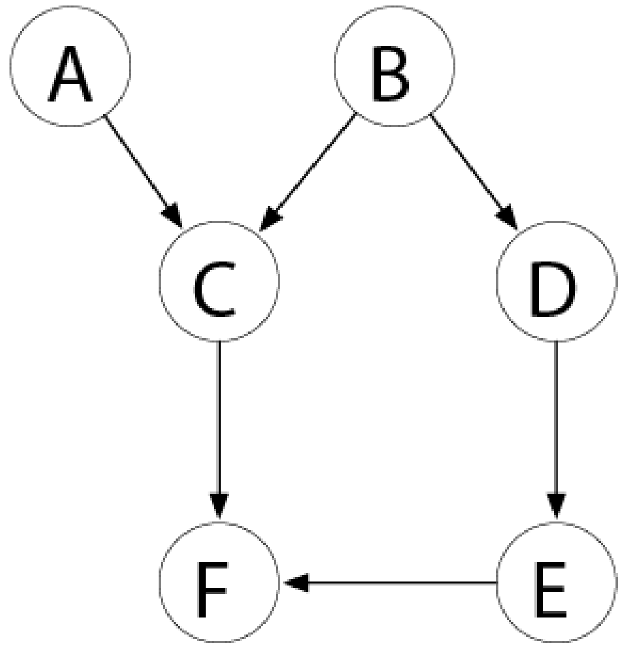
- Motivation
  - Exact inference on general graphs.
  - Works by turning the initial graph into a junction tree and then running a sum-product-like algorithm.
  - Intractable on graphs with large cliques.

- Main steps
  1. If starting from directed graph, first convert it to an undirected graph by moralization.
  2. Introduce additional links by triangulation in order to reduce the size of cycles.
  3. Find cliques of the moralized, triangulated graph.
  4. Construct a new graph from the maximal cliques.
  5. Remove minimal links to break cycles and get a junction tree.

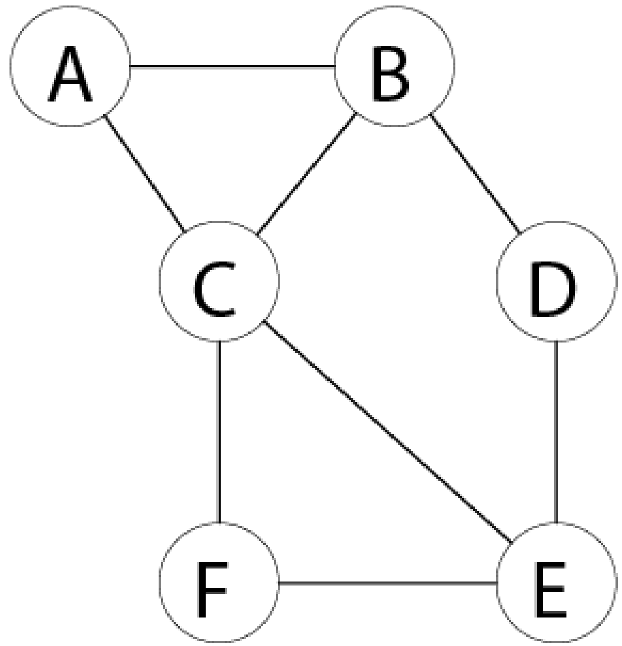
  $\Rightarrow$ Apply regular message passing to perform inference.

Junction Tree Algorithm

- **Starting from an directed graph…**



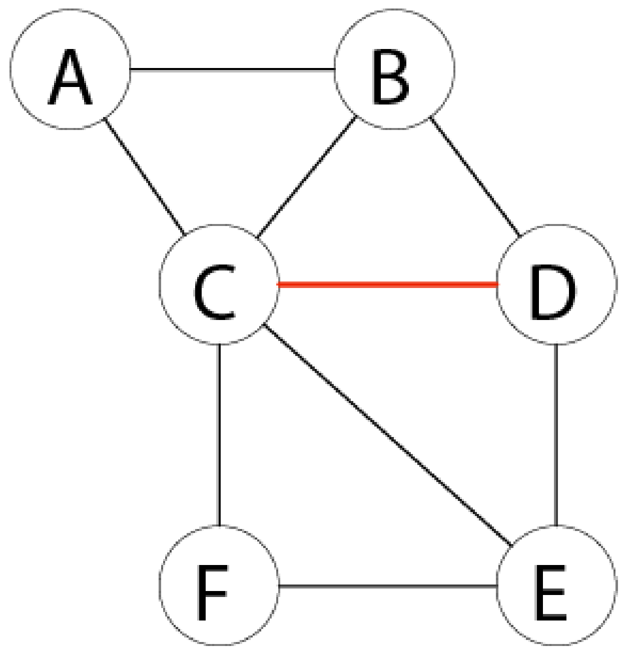
Junction Tree Algorithm



1. **Convert to an undirected graph through moralization.**
   - **Marry the parents of each node.**
   - **Remove edge directions.**

RWTH AACHEN UNIVERSITY

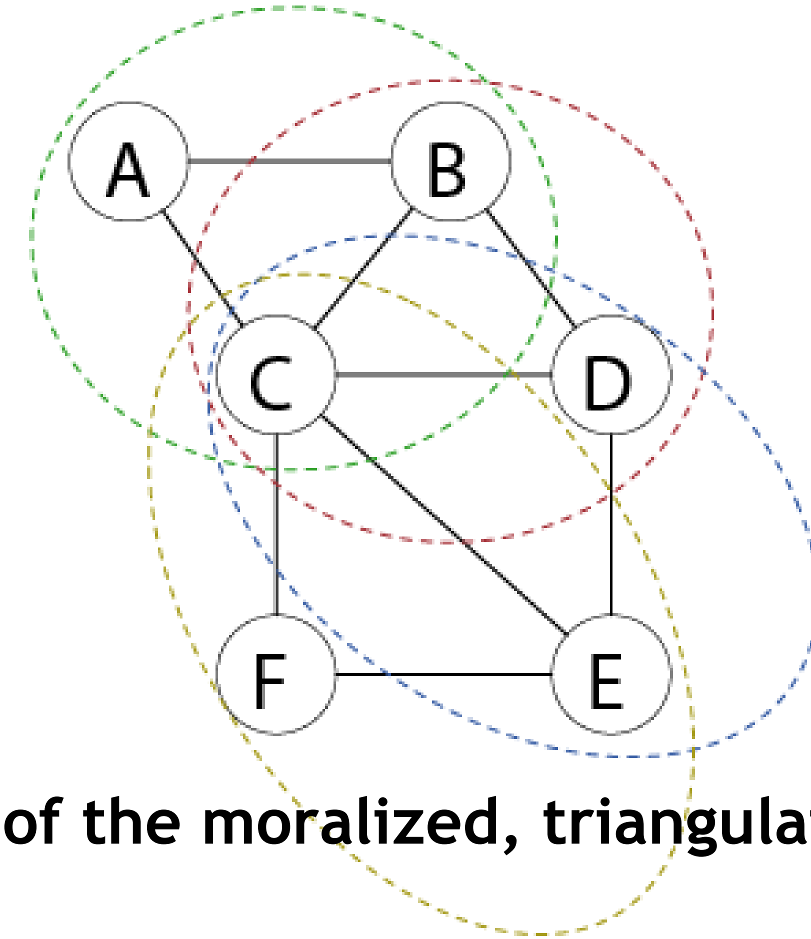
Junction Tree Algorithm



2. Triangulate

- Such that there is no loop of length > 3 without a chord.
- This is necessary so that the final junction tree satisfies the "running intersection" property (explained later).

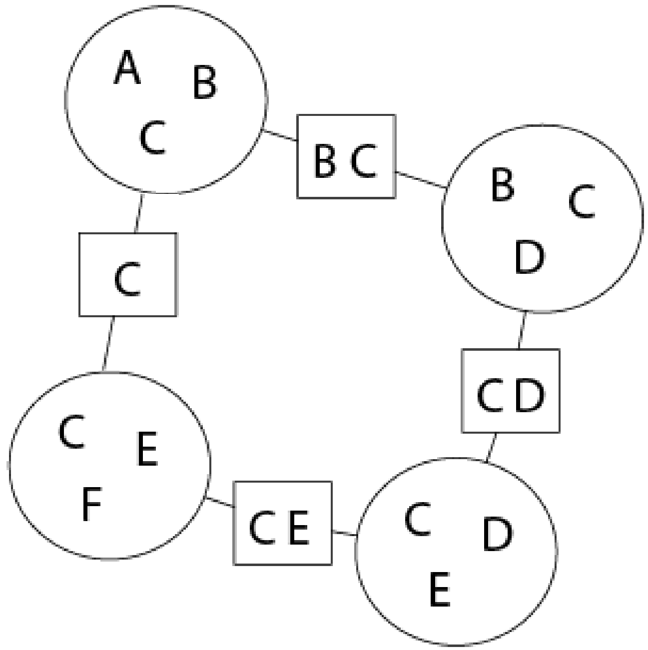
Junction Tree Algorithm



3. **Find cliques** of the moralized, triangulated graph.

Image source: Z. Gharahmani

Junction Tree Algorithm

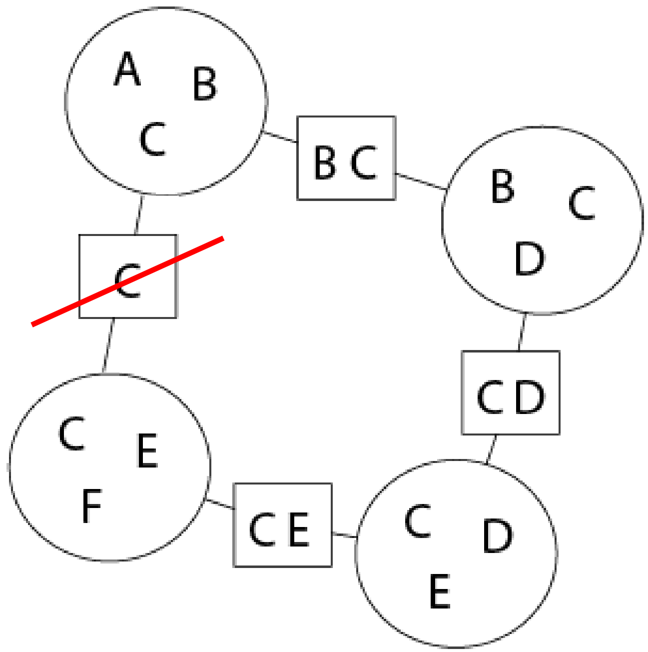


4. Construct a new junction graph from maximal cliques.

- Create a node from each clique.
- Each link carries a list of all variables in the intersection.
  - Drawn in a "separator" box.

Image source: Z. Gharahmani

Junction Tree Algorithm



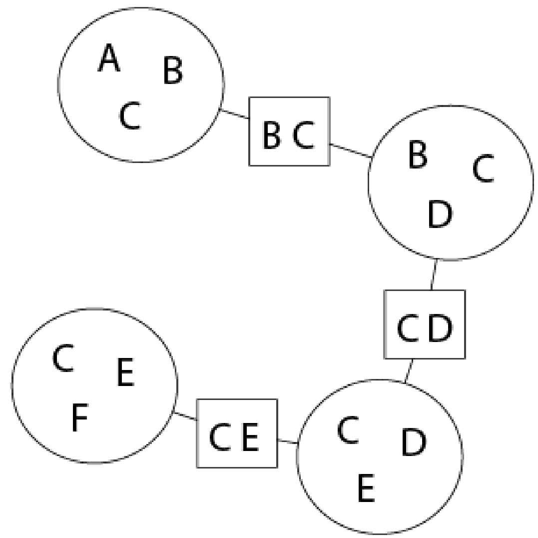
5. Remove links to break cycles ⇒ junction tree.

- For each cycle, remove a link with the minimal number of shared nodes until all cycles are broken.
- Result is a maximal spanning tree, the junction tree.

Image source: Z. Gharahmani

RWTH AACHEN UNIVERSITY

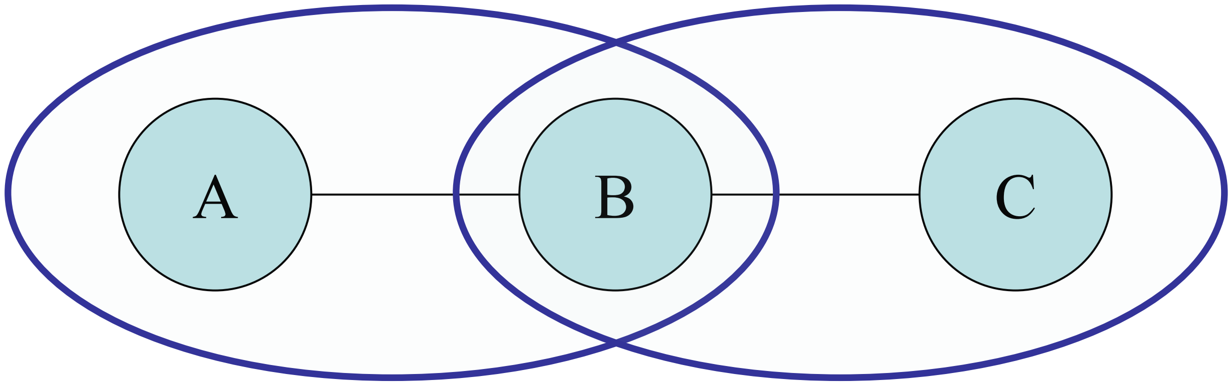
Junction Tree - Properties



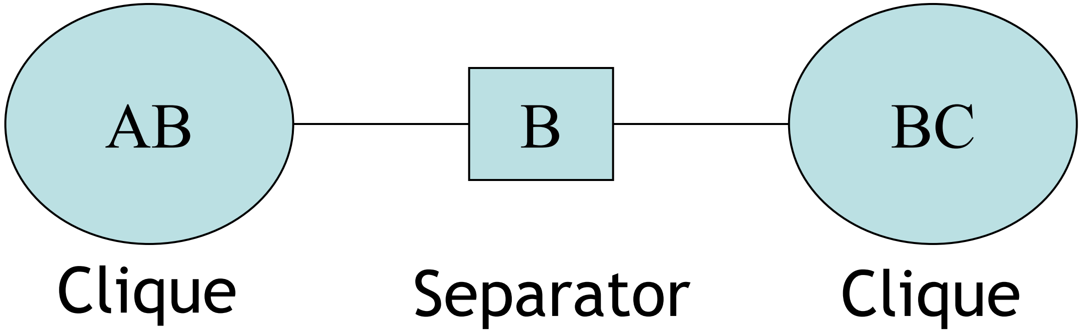
- **Running intersection property**
  - ***"If a variable appears in more than one clique, it also appears in all intermediate cliques in the tree".***
  - **This ensures that neighboring cliques have consistent probability distributions.**
  - **Local consistency → global consistency**

Interpretation of the Junction Tree

- **Undirected graphical model**



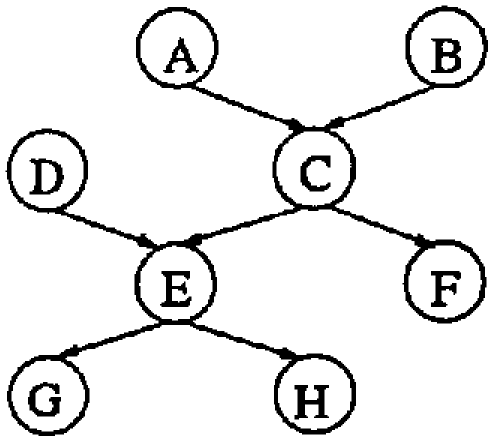
- **Junction tree**



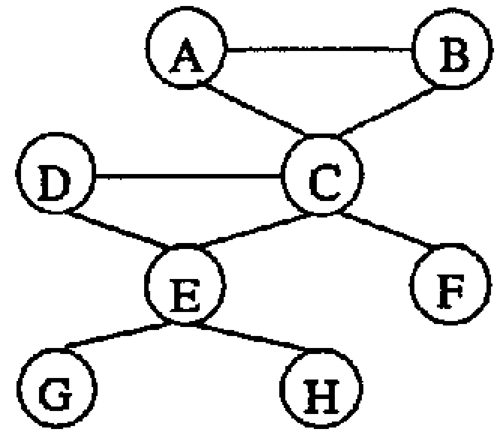
P(U) = ∏ P(Clique) / ∏ P(Separator)

$$P(A,B,C) = P(A,B)\,P(B,C) \,/\, P(B)$$

Junction Tree: Example 1



(a) DAG

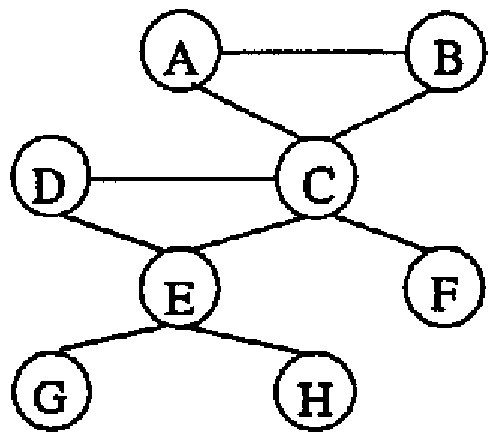


(b) Moral graph

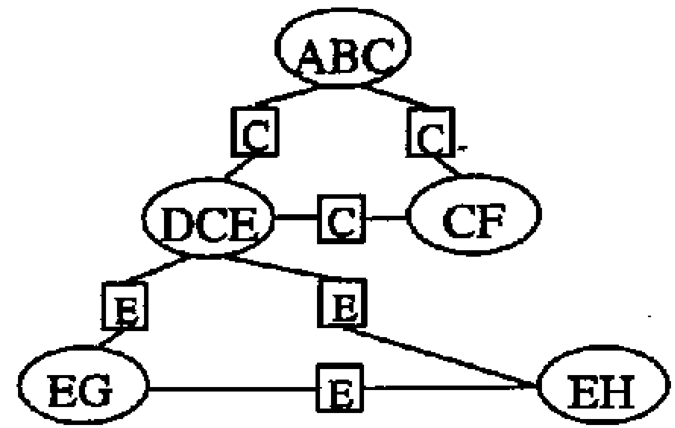
- Algorithm
  1. Moralization
  2. Triangulation (not necessary here)

Image source: J. Pearl, 1988

Junction Tree: Example 1



(b) Moral graph

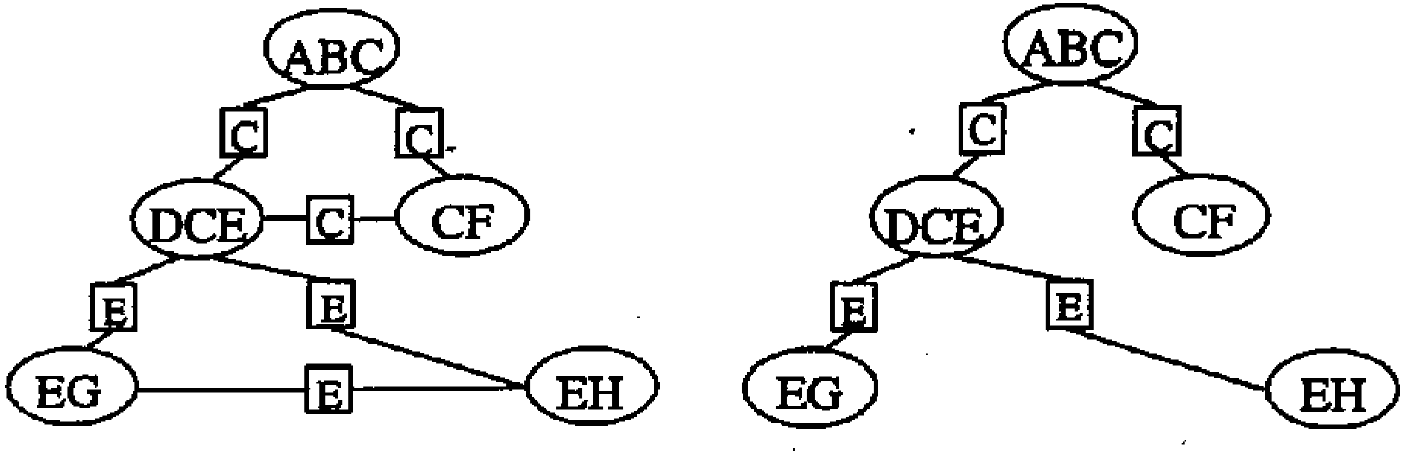


(c) Junction graph

- Algorithm
  1. Moralization
  2. Triangulation (not necessary here)
  3. Find cliques
  4. Construct junction graph

Image source: J. Pearl, 1988

Junction Tree: Example 1

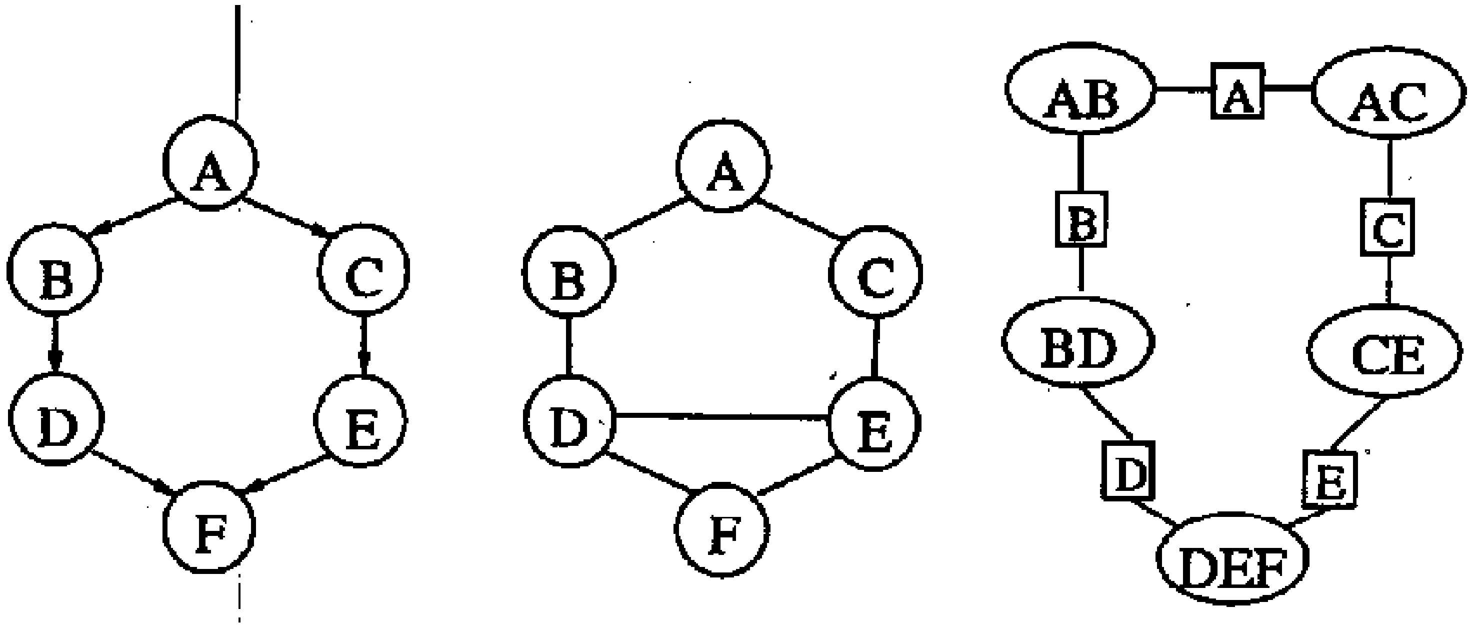


(c) Junction graph (d) Junction tree

- Algorithm
  1. Moralization
  2. Triangulation (not necessary here)
  3. Find cliques
  4. Construct junction graph
  5. Break links to get junction tree

Image source: J. Pearl, 1988

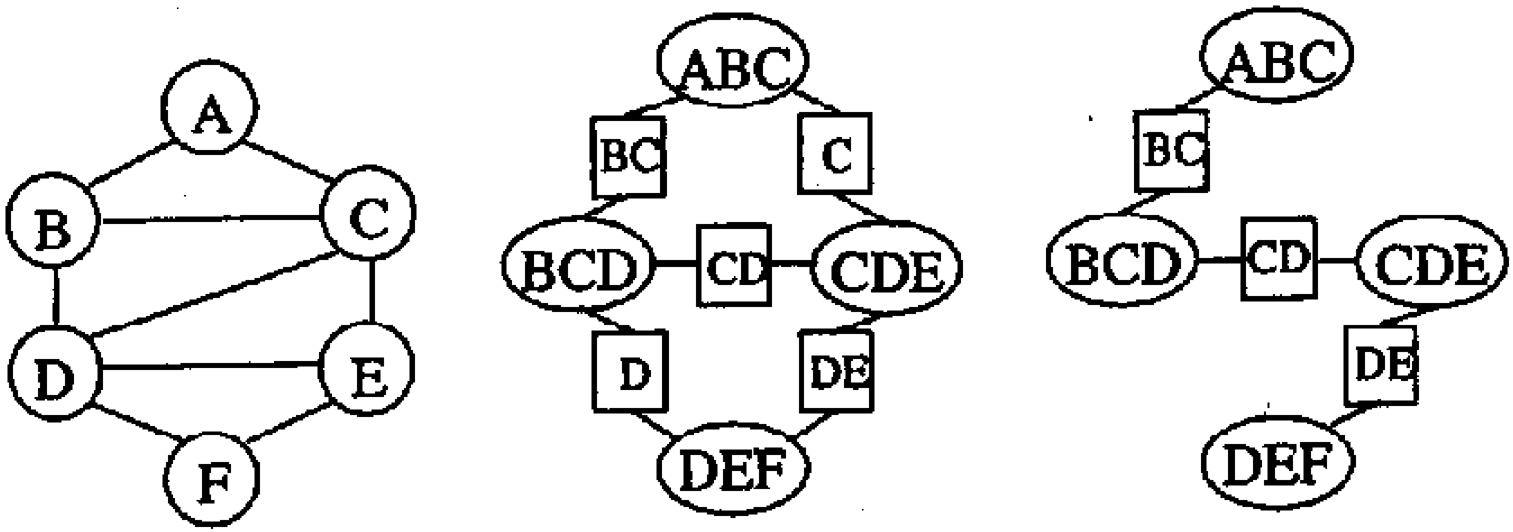
Junction Tree: Example 2



- **Without triangulation step**
  - The final graph will contain cycles that we cannot break without losing the running intersection property!

Image source: J. Pearl, 1988

Junction Tree: Example 2



- **When applying the triangulation**
  - Only small cycles remain that are easy to break.
  - Running intersection property is maintained.

Image source: J. Pearl, 1988

Junction Tree Algorithm

- Good news
  - The junction tree algorithm is efficient in the sense that for a given graph there does not exist a computationally cheaper approach.

- Bad news
  - This may still be too costly.
  - Effort determined by number of variables in the largest clique.
  - Grows exponentially with this number (for discrete variables).

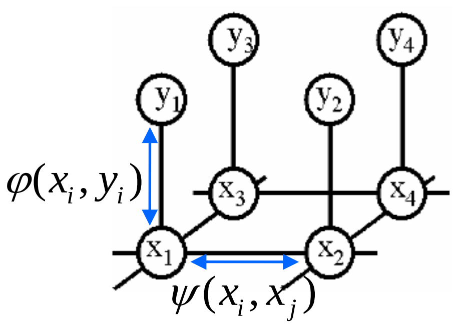
  $\Rightarrow$ Algorithm becomes impractical if the graph contains large cliques!

Loopy Belief Propagation

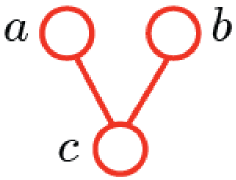
- Alternative algorithm for loopy graphs
  - Sum-Product on general graphs.
  - Strategy: simply ignore the problem.
  - Initial unit messages passed across all links, after which messages are passed around until convergence
    - Convergence is not guaranteed!
    - Typically break off after fixed number of iterations.
  - Approximate but tractable for large graphs.
  - Sometime works well, sometimes not at all.

Topics of This Lecture

- Recap: Exact inference
  - Sum-Product algorithm
  - Max-Sum algorithm
  - Junction Tree algorithm
- **Applications of Markov Random Fields**
  - **Application examples from computer vision**
  - **Interpretation of clique potentials**
  - **Unary potentials**
  - **Pairwise potentials**
- Solving MRFs with Graph Cuts
  - Graph cuts for image segmentation
  - s-t mincut algorithm
  - Extension to non-binary case
  - Applications



Markov Random Fields (MRFs)



- What we've learned so far…
  - We know they are undirected graphical models.
  - Their joint probability factorizes into clique potentials,

$$p(\mathbf{x}) = \frac{1}{Z} \prod_C \psi_C(\mathbf{x}_C)$$

    which are conveniently expressed as energy functions.

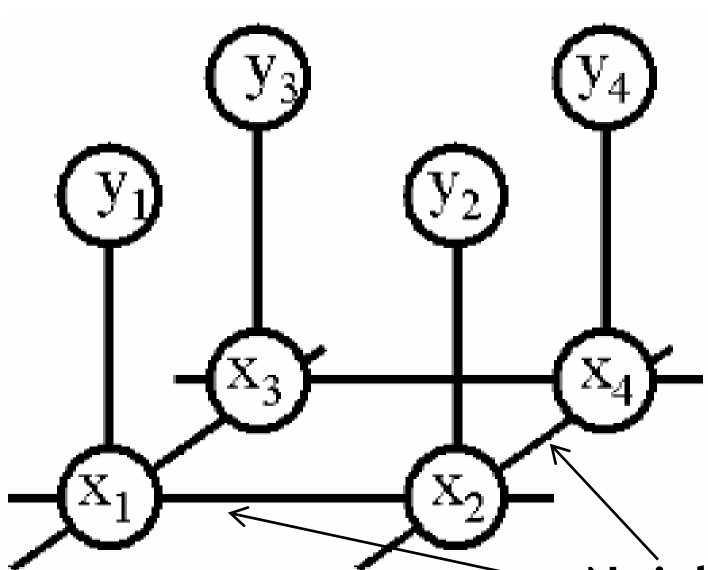
$$\psi_C(\mathbf{x}_C) = \exp\{-E(\mathbf{x}_C)\}$$

  - We know how to perform inference for them.
    - Sum/Max-Product BP for exact inference in tree-shaped MRFs.
    - Loopy BP for approximate inference in arbitrary MRFs.
    - Junction Tree algorithm for converting arbitrary MRFs into trees.
- But what are they actually good for?
  - And how do we apply them in practice?

Image source: C. Bishop, 2006

Markov Random Fields

- Allow rich probabilistic models.
  - But built in a local, modular way.
  - Learn local effects, get global effects out.
- Very powerful when applied to regular structures.
  - Such as images…



Observed evidence

Hidden “true states”

Neighborhood relations

Slide adapted from William Freeman

Applications of MRFs

- **Movie “No Way Out” (1987)**

Applications of MRFs

- Many applications for low-level vision tasks
  - Image denoising

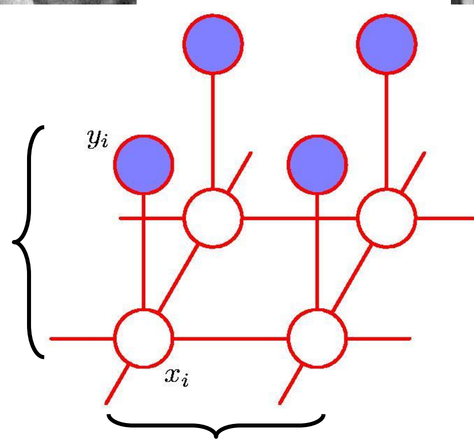
Results by [Roth & Black, CVPR'05]

Applications of MRFs

- **Many applications for low-level vision tasks**
  - **Image denoising**

Noisy observations

Observation process



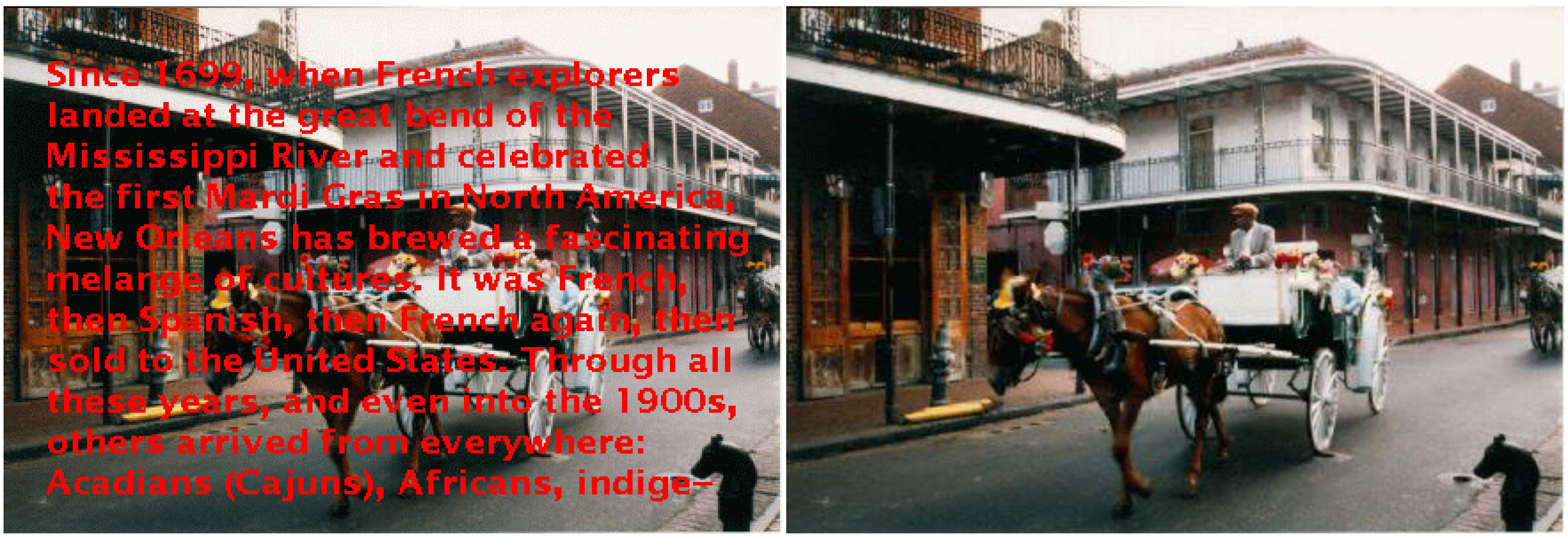
"True" image content

"Smoothness constraints"

Results by [Roth & Black, CVPR'05]

Applications of MRFs

- Many applications for low-level vision tasks
  - Image denoising
  - Inpainting



Results by [Roth & Black, CVPR'05]

Applications of MRFs

- Many applications for low-level vision tasks
  - Image denoising
  - Inpainting
  - Image restoration

Results by [Roth & Black, CVPR'05]

Applications of MRFs

- Many applications for low-level vision tasks
  - Image denoising
  - Inpainting
  - Image restoration
  - Image segmentation

Image source: Pawan M. Kumar

Applications of MRFs

- Many applications for low-level vision tasks
  - Image denoising
  - Inpainting
  - Image restoration
  - Image segmentation
  - Super-resolution

Convert a low-res image into a high-res image!

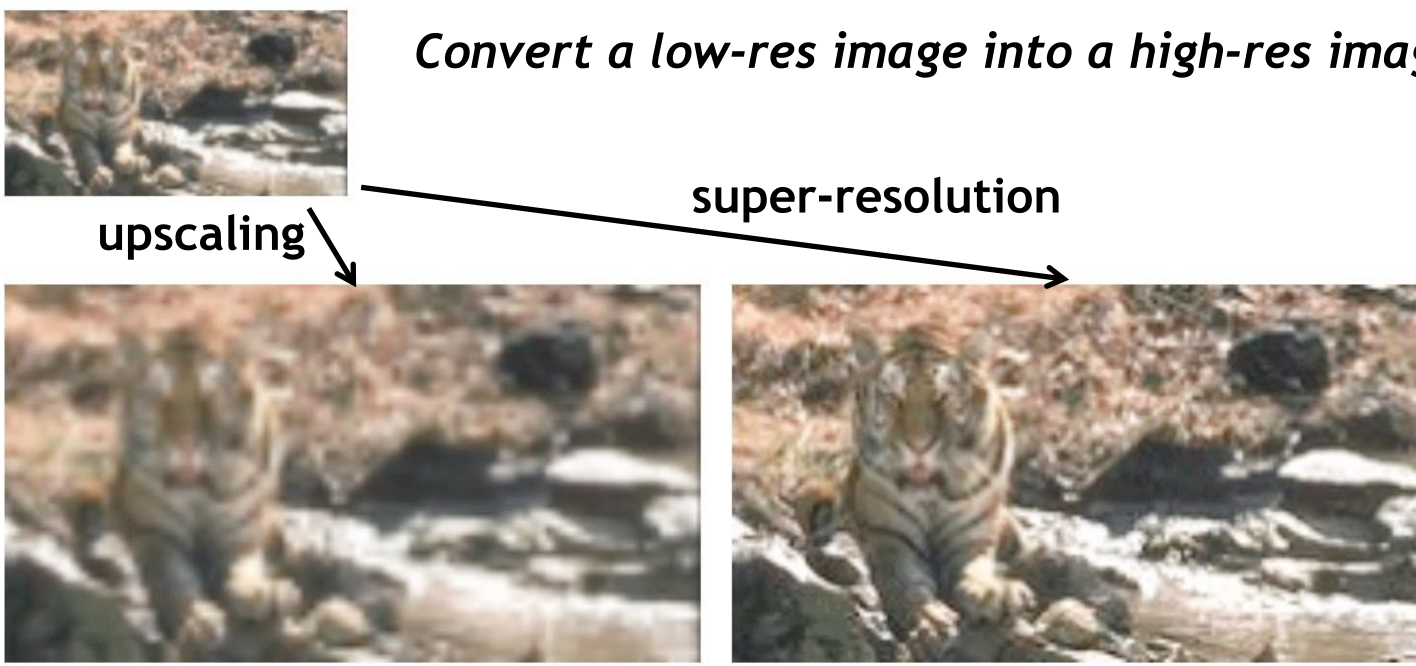


Image source: [Freeman et al., CG&A'03]

Applications of MRFs

- Many applications for low-level vision tasks
  - Image denoising
  - Inpainting
  - Image restoration
  - Image segmentation
  - Super-resolution

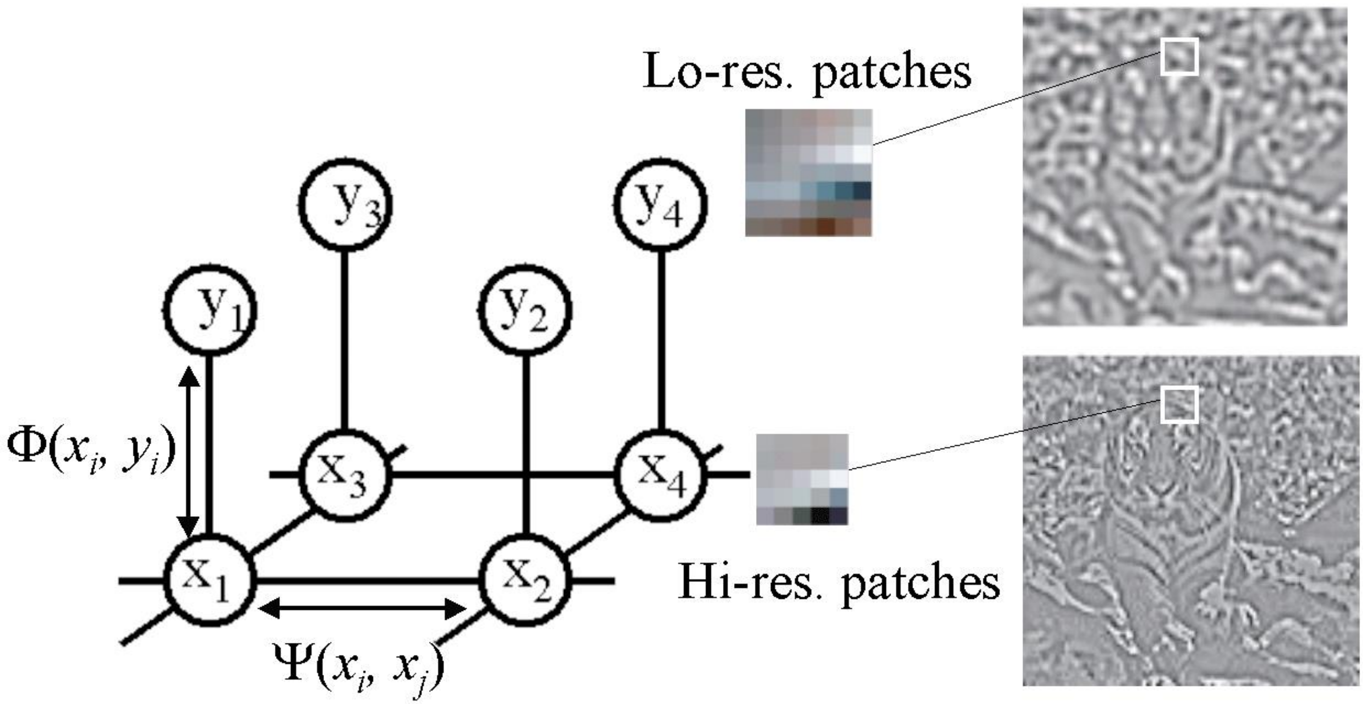
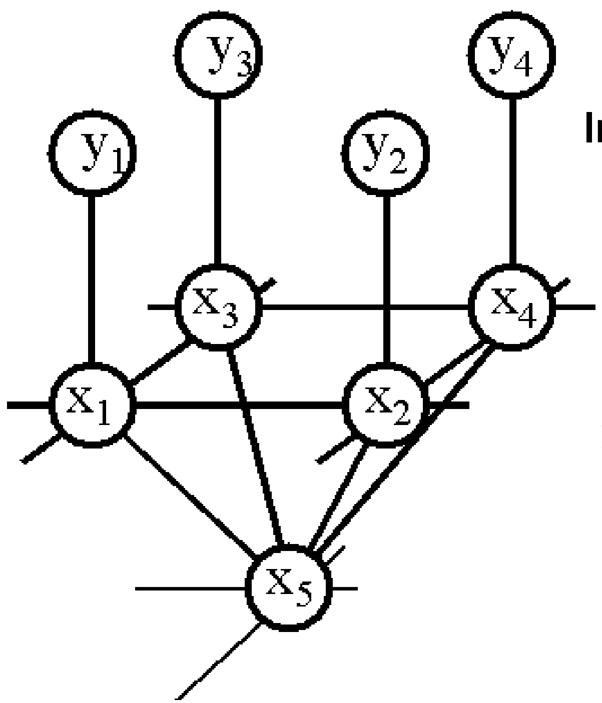
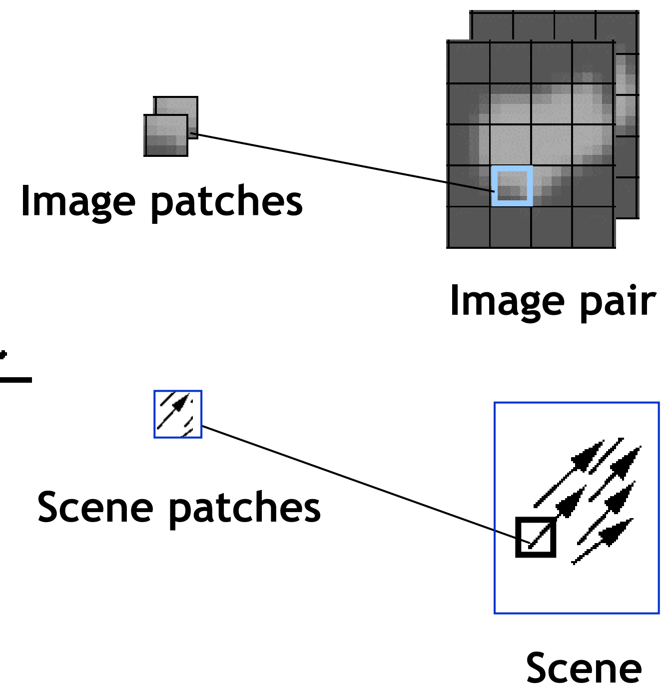


Image source: [Freeman et al., CG&A'03]

Applications of MRFs

- Many applications for low-level vision tasks
  - Image denoising
  - Inpainting
  - Image restoration
  - Image segmentation
  - Super-resolution
  - Optical flow



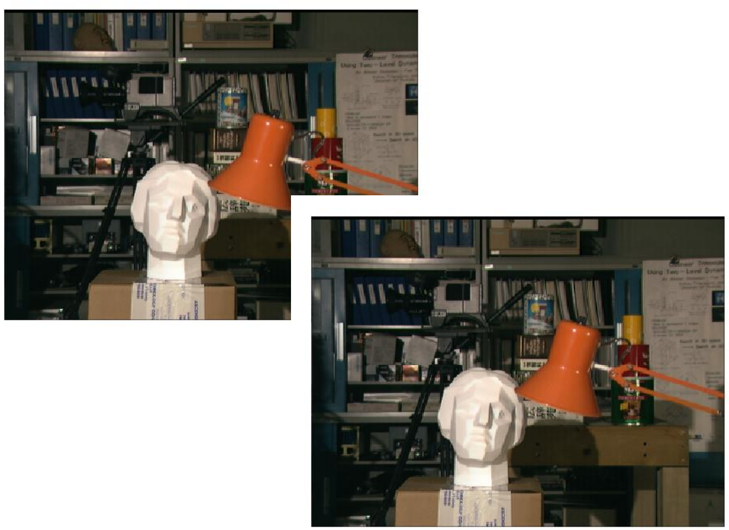


Machine Learning, WS 13/14

Image source: William Freeman

Applications of MRFs

- Many applications for low-level vision tasks
  - Image denoising
  - Inpainting
  - Image restoration
  - Image segmentation
  - Super-resolution
  - Optical flow
  - Stereo depth estimation



Stereo image pair

Disparity map

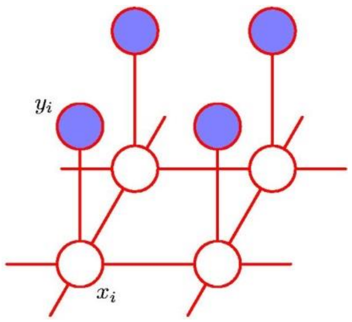
Applications of MRFs

- Many applications for low-level vision tasks
  - Image denoising
  - Inpainting
  - Image restoration
  - Image segmentation
  - Super-resolution
  - Optical flow
  - Stereo depth estimation

- MRFs have become a standard tool for such tasks.
  - Let's look at how they are applied in detail…

MRF Structure for Images

- **Basic structure**

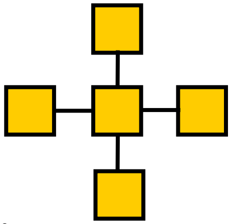


Noisy observations

"True" image content

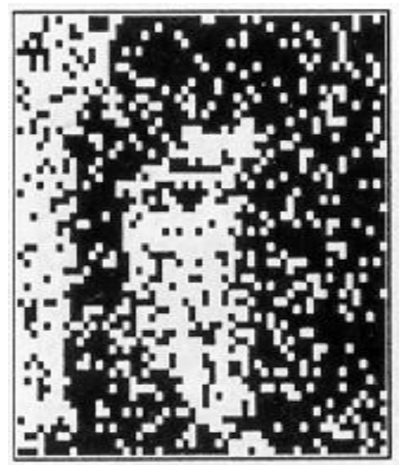
- **Two components**
  - Observation model
    - How likely is it that node x_i has label L_i given observation y_i?
    - This relationship is usually learned from training data.
  - Neighborhood relations
    - Simplest case: 4-neighborhood
    - Serve as smoothing terms.

    $\Rightarrow$ Discourage neighboring pixels to have different labels.
    - This can either be learned or be set to fixed "penalties".

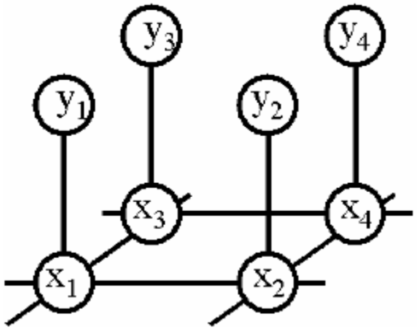


MRF Nodes as Pixels


Original image

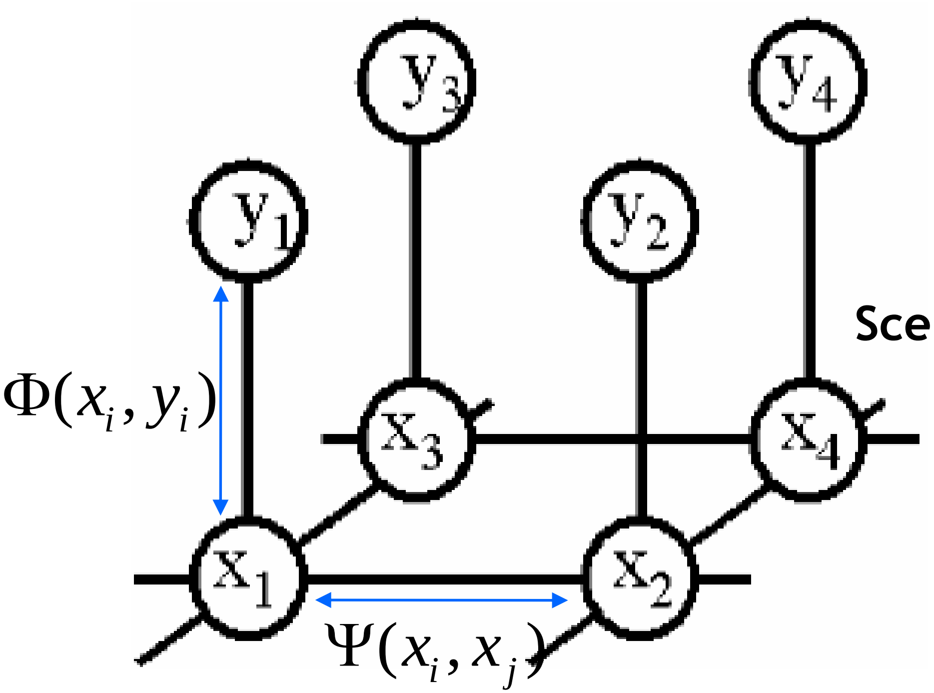

Degraded image

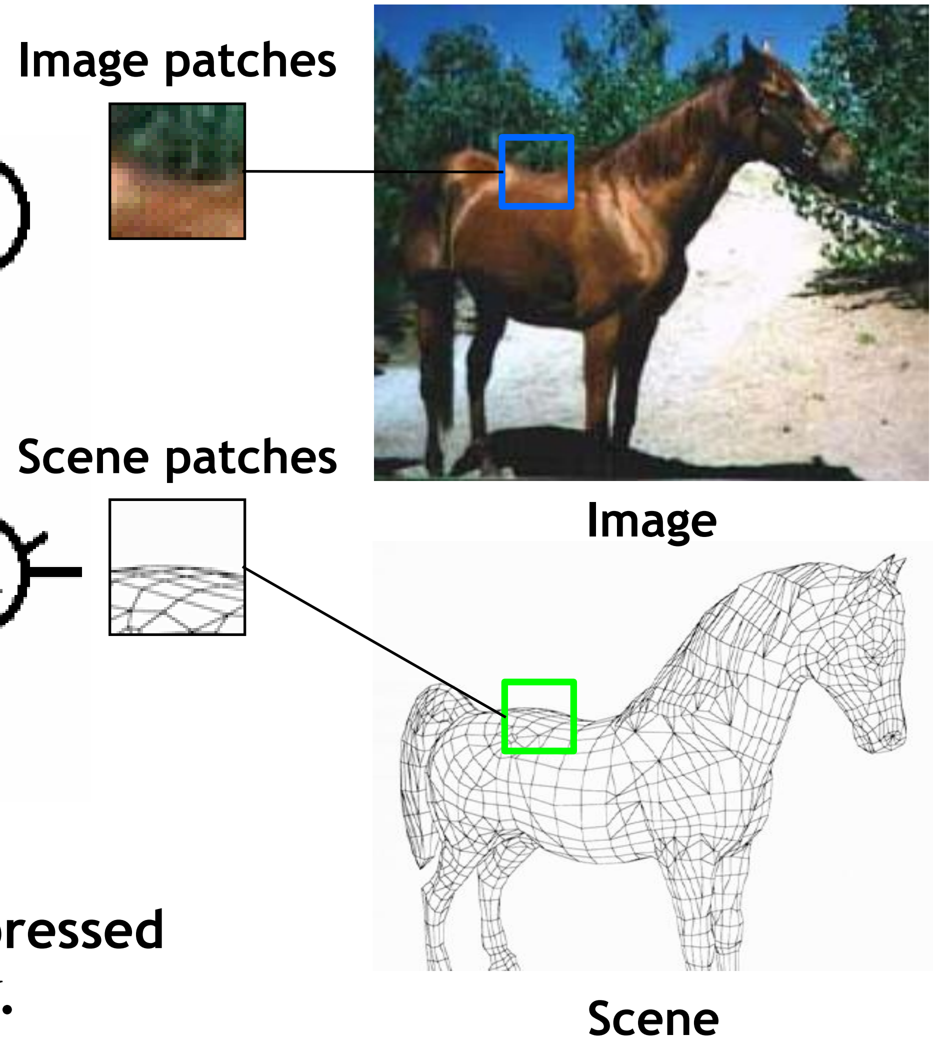

Reconstruction from MRF modeling pixel neighborhood statistics



These neighborhood statistics can be learned from training data!

MRF Nodes as Patches

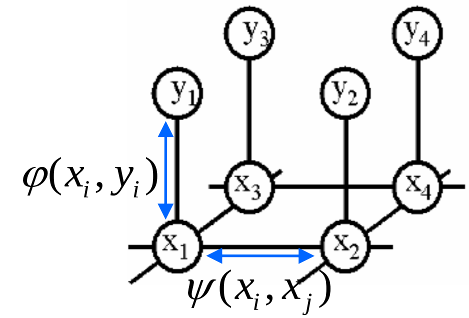




More general relationships expressed by potential functions Φ and Ψ.

Network Joint Probability

- **Interpretation of the factorized joint probability**



$$P(x, y) = \prod_i \Phi(x_i, y_i) \prod_{i,j} \Psi(\underbrace{x_i, x_j}_{\text{Neighboring scene nodes}})$$

Scene

Image

Image-scene compatibility function

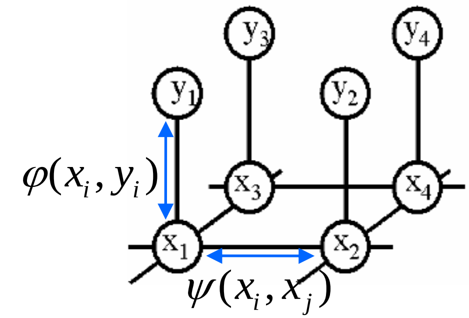
Local observations

Scene-scene compatibility function

Neighboring scene nodes

Energy Formulation

- **Energy function**



$$E(x,y) = \underbrace{\sum_i \varphi(x_i, y_i)}_{\text{Single-node potentials}} + \underbrace{\sum_{i,j} \psi(x_i, x_j)}_{\text{Pairwise potentials}}$$

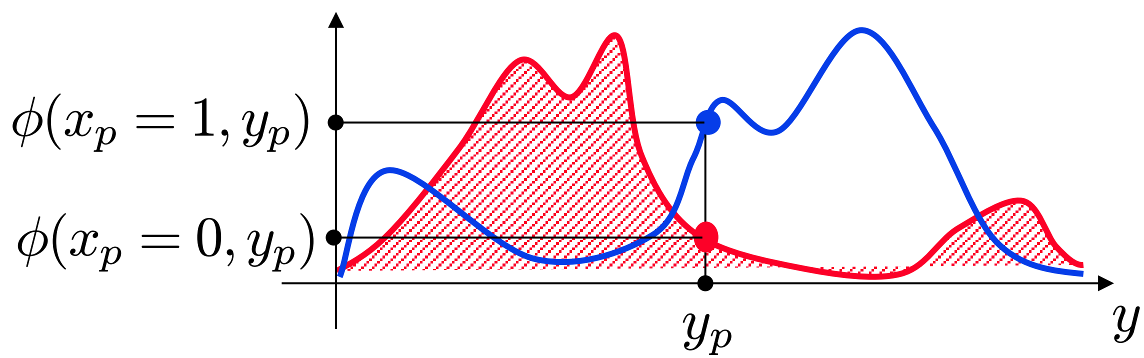
- **Single-node (unary) potentials** φ
  - Encode local information about the given pixel/patch.
  - How likely is a pixel/patch to belong to a certain class (e.g. foreground/background)?
- **Pairwise potentials** ψ
  - Encode neighborhood information.
  - How different is a pixel/patch's label from that of its neighbor? (e.g. based on intensity/color/texture difference, edges)

How to Set the Potentials? Some Examples

- **Unary potentials**
  - **E.g., color model, modeled with a Mixture of Gaussians**

$$\phi(x_i, y_i; \theta_\phi) = \log \sum_k \theta_\phi(x_i, k) p(k|x_i) \mathcal{N}(y_i; \bar{y}_k, \Sigma_k)$$

$\Rightarrow$ **Learn color distributions for each label**



How to Set the Potentials? Some Examples

- **Pairwise potentials**
  - Potts Model

$$\psi(x_i, x_j; \theta_\psi) = \theta_\psi \delta(x_i \neq x_j)$$

    - Simplest discontinuity preserving model.
    - Discontinuities between any pair of labels are penalized equally.
    - Useful when labels are unordered or number of labels is small.

  - Extension: "contrast sensitive Potts model"

$$\psi(x_i, x_j, g_{ij}(y); \theta_\psi) = \theta_\psi g_{ij}(y) \delta(x_i \neq x_j)$$

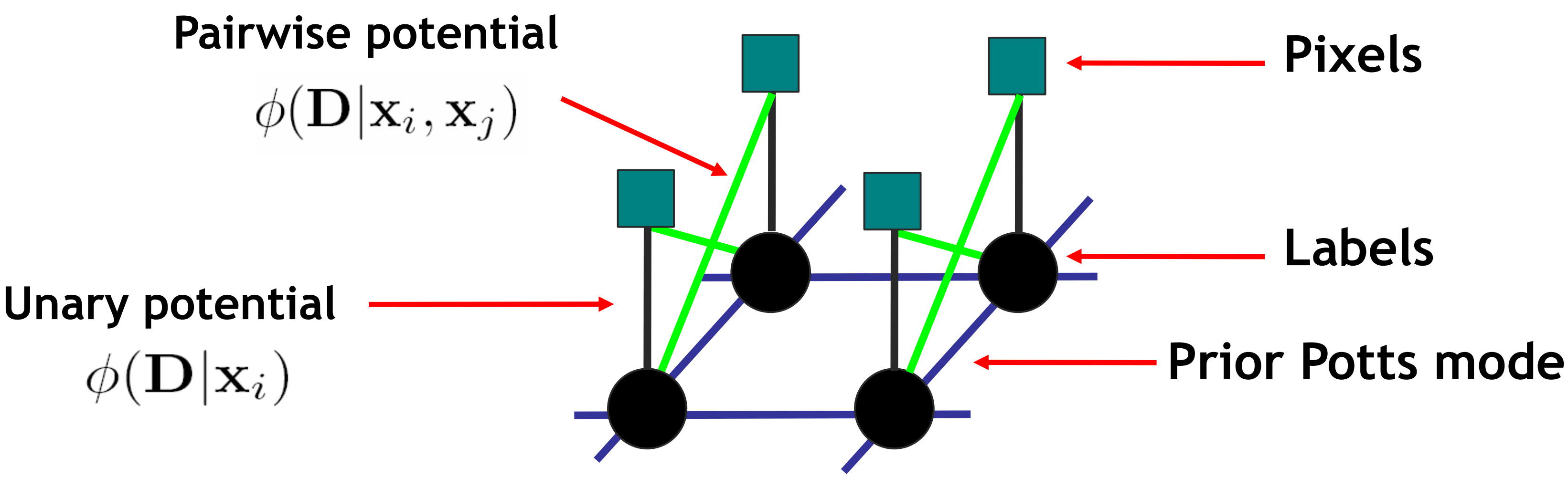
where

$$g_{ij}(y) = e^{-\beta \left\| y_i - y_j \right\|^2} \qquad \beta = 2 \cdot avg\left(\left\| y_i - y_j \right\|^2 \right)$$

    - Discourages label changes except in places where there is also a large change in the observations.

Extension: Conditional Random Fields (CRF)

- **Idea: Model conditional instead of joint probability**



- **Energy formulation**

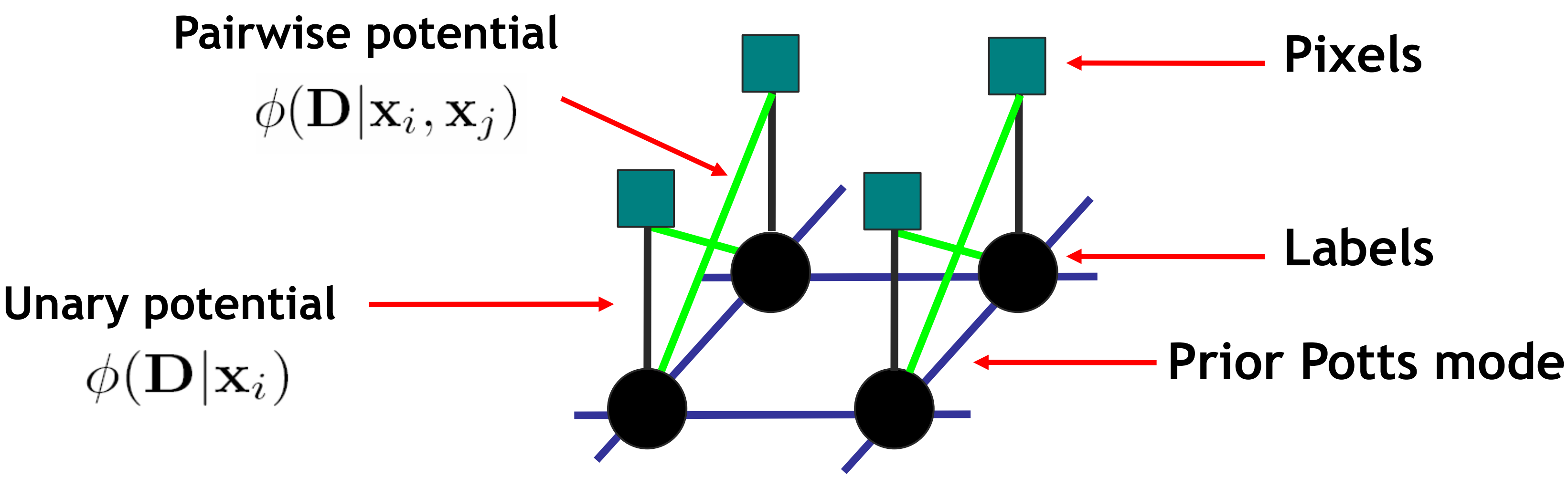
$$E(\mathbf{x}) = \sum_{i \in S} \left(\phi(\mathbf{D}|\mathbf{x}_i) + \sum_{j \in N_i} \left(\phi(\mathbf{D}|\mathbf{x}_i, \mathbf{x}_j) + \psi(\mathbf{x}_i, \mathbf{x}_j) \right) \right) + \text{const}$$

Unary likelihood — Contrast Term — Uniform Prior (Potts Model)

Slide credit: Phil Torr

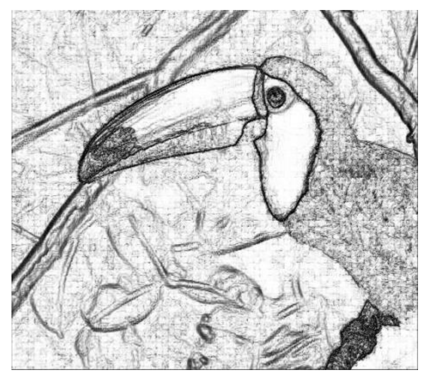
Example: MRF for Image Segmentation

- **MRF structure**



Data (D)

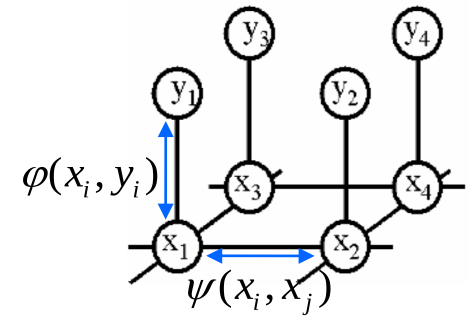
Unary likelihood



Pair-wise Terms

MAP Solution

Energy Minimization



- **Goal:**
  - Infer the optimal labeling of the MRF.
- **Many inference algorithms are available, e.g.**
  - Simulated annealing ← *What you saw in the movie.*
  - Iterated conditional modes (ICM) ← *Too simple.*
  - Belief propagation ← *Last lecture*
  - Graph cuts ← *Next Lecture!*
  - Variational methods
  - Monte Carlo sampling

  } ← *For more complex problems*
- **Recently, Graph Cuts have become a popular tool**
  - Only suitable for a certain class of energy functions.
  - But the solution can be obtained very fast for typical vision problems (~1MPixel/sec).

References and Further Reading

- A gentle introduction to Graph Cuts can be found in the following paper:
  - Y. Boykov, O. Veksler, Graph Cuts in Vision and Graphics: Theories and Applications. In *Handbook of Mathematical Models in Computer Vision*, edited by N. Paragios, Y. Chen and O. Faugeras, Springer, 2006.

- Try the GraphCut implementation at
  http://www.cs.ucl.ac.uk/staff/V.Kolmogorov/software.html