CSC 2515: Structured Prediction

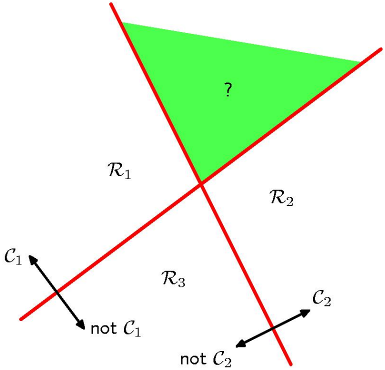
Raquel Urtasun & Rich Zemel

University of Toronto

March 16, 2015

Discriminant Functions for $K > 2$ classes

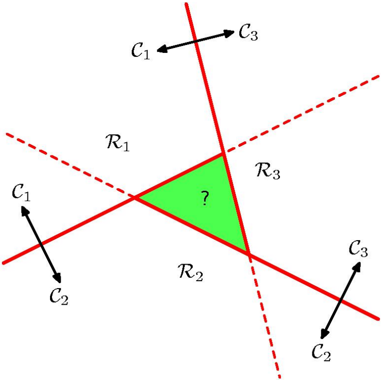
- Use $K - 1$ classifiers, each solving a two class problem of separating point in a class C_k from points not in the class.
- Known as **1 vs all** or **1 vs the rest** classifier



- PROBLEM: More than one good answer!

Discriminant Functions for $K > 2$ classes

- Introduce $K(K-1)/2$ two-way classifiers, one for each possible pair of classes
- Each point is classified according to majority vote amongst the disc. func.
- Known as the **1 vs 1 classifier**



- PROBLEM: Two-way preferences need not be transitive

K-Class Discriminant

- We can avoid these problems by considering a single K-class discriminant comprising K functions of the form

$$y_k(\mathbf{x}) = \mathbf{w}_k^T \mathbf{x} + w_{k,0}$$

K-Class Discriminant

- We can avoid these problems by considering a single K-class discriminant comprising K functions of the form

$$y_k(\mathbf{x}) = \mathbf{w}_k^T \mathbf{x} + w_{k,0}$$

and then assigning a point $\mathbf{x}$ to class C_k if

$$\forall j \neq k \qquad y_k(\mathbf{x}) > y_j(\mathbf{x})$$

K-Class Discriminant

- We can avoid these problems by considering a single K-class discriminant comprising K functions of the form

$$y_k(\mathbf{x}) = \mathbf{w}_k^T \mathbf{x} + w_{k,0}$$

  and then assigning a point $\mathbf{x}$ to class C_k if

$$\forall j \neq k \qquad y_k(\mathbf{x}) > y_j(\mathbf{x})$$

- Note that $\mathbf{w}_k^T$ is now a vector, not the k-th coordinate

K-Class Discriminant

- We can avoid these problems by considering a single K-class discriminant comprising K functions of the form

$$y_k(\mathbf{x}) = \mathbf{w}_k^T \mathbf{x} + w_{k,0}$$

and then assigning a point $\mathbf{x}$ to class C_k if

$$\forall j \neq k \qquad y_k(\mathbf{x}) > y_j(\mathbf{x})$$

- Note that $\mathbf{w}_k^T$ is now a vector, not the k-th coordinate
- In this lecture we will look at generalizations of this idea.

Contents

- Introduction to Structured prediction
- Inference
- Learning

What is structured prediction?

Structured Prediction

- In "typical" machine learning

$$f : \mathcal{X} \rightarrow \Re$$

the input $\mathcal{X}$ can be anything, and the output is a real number (e.g., classification, regression)

Structured Prediction

- In "typical" machine learning

$$f : \mathcal{X} \rightarrow \Re$$

the input $\mathcal{X}$ can be anything, and the output is a real number (e.g., classification, regression)

- In **Structured Prediction**

$$f : \mathcal{X} \rightarrow \mathcal{Y}$$

the input $\mathcal{X}$ can be anything, and the output is a **complex** object (e.g., image segmentation, parse tree)

Structured Prediction

- In "typical" machine learning

$$f : \mathcal{X} \rightarrow \Re$$

  the input $\mathcal{X}$ can be anything, and the output is a real number (e.g., classification, regression)

- In **Structured Prediction**

$$f : \mathcal{X} \rightarrow \mathcal{Y}$$

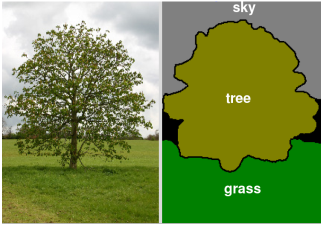
  the input $\mathcal{X}$ can be anything, and the output is a **complex** object (e.g., image segmentation, parse tree)

- In this lecture $\mathcal{Y}$ is a discrete space, ask me later if you are interested in continuous variables.

Structured Prediction and its Applications

We want to predict multiple random variables which are related

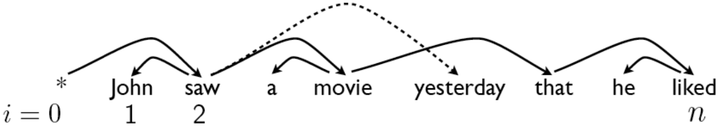
- Computer Vision:
  - Semantic Segmentation (output: pixel-wise labeling)
  - Object detection (output: 2D or 3D bounding boxes)
  - Stereo Reconstruction (output: 3D map)
  - Scene Understanding (output: 3D bounding box reprinting the layout)



Structured Prediction and its Applications

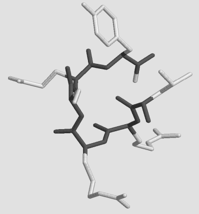
We want to predict multiple random variables which are related

- Natural Language processing
  - Machine Translation (output: sentence in another language)
  - Parsing (output: parse tree)



- Computational Biology
  - Protein Folding (output: 3D protein)

MRLLILALLGICSLTAYIVEGVGSEVSDKR
TCVSLTTQRLPVSRIKTYTITEGSLRAVIF
ITKRGLKVCADPQATWVRDVVRSMDRKSNT
RNNMIQTKPTGTQQSTNTAVTLTG



Why structured?

- Independent prediction is good but...

Why structured?

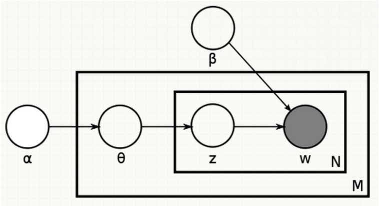
- Independent prediction is good but...

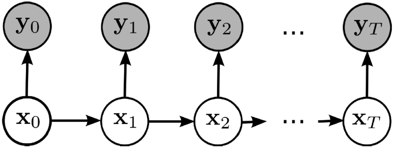
- Neighboring pixels should have same labels (if they look similar).

Graphical Model

A graphical model defines

- A family of probability distributions over a set of random variables
- This is expressed via a graph, which encodes the conditional independences of the distribution





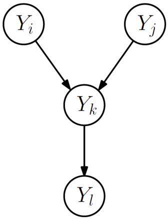
- Two types of graphical models: Directed and undirected

Bayesian Networks (Directed Graphical Models)

- The graph $G = (V, \mathcal{E})$ is acyclic and directed
- Factorization over distributions by conditioning on parent nodes

$$p(\mathbf{y}) = \prod_{i \in V} p(y_i | y_{pa}(i))$$

- Example



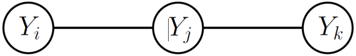
$$p(\mathbf{y}) = p(y_l | y_k) p(y_k | y_i, y_j) p(y_i) p(y_j)$$

Undirected Graphical Model

- Also called Markov Random Field, or Markov Network
- Graph $G = (V, \mathcal{E})$ is undirected and has no self-edges
- Factorization over cliques

$$p(\mathbf{y}) = \frac{1}{Z} \prod_{r \in \mathcal{R}} \psi_r(\mathbf{y}_r)$$

  with $Z = \sum_{\mathbf{y} \in \mathcal{Y}} \prod_{r \in \mathcal{R}} \psi_r(\mathbf{y}_r)$ the partition function
- Example



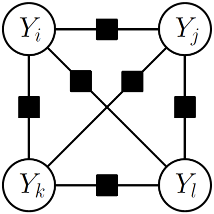
$$p(\mathbf{y}) = \frac{1}{Z} \psi(y_i, y_j)\psi(y_j, y_k)\psi(y_i)\psi(y_j)\psi(y_k)$$

- **Difficulty**: Exponentially many configurations
- Undirected models will be the focus of this lecture

Factor Graph Representation

- Graph $G = (V, \mathcal{F}, \mathcal{E})$, with variable nodes $\mathcal{V}$, factor nodes $\mathcal{F}$ and edges $\mathcal{E}$
- **Scope** of a factor $N(F) = \{i \in V : (i, F) \in \mathcal{E}\}$
- Factorization over factors

$$p(\mathbf{y}) = \frac{1}{Z} \prod_{F \in \mathcal{F}} \psi_F(\mathbf{y}_{N(F)})$$



Factor Graph vs Graphical Model

- Factor graphs are explicit about the factorization

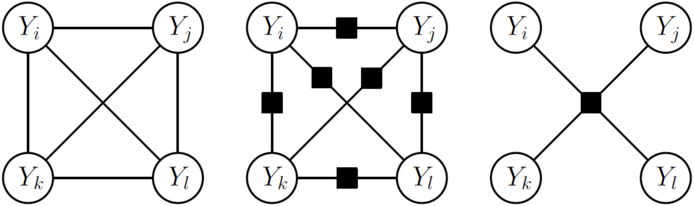


Figure : from [Nowozin et al]

Capacity

- They define the family of distributions and thus the *capacity*

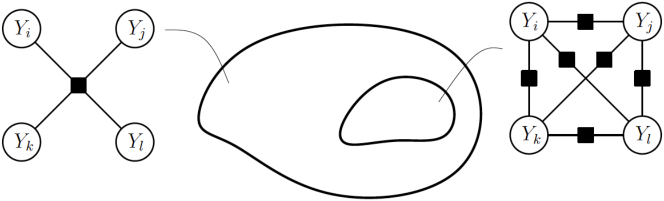


Figure : from [Nowozin et al]

Markov Random Fields vs Conditional Random Fields

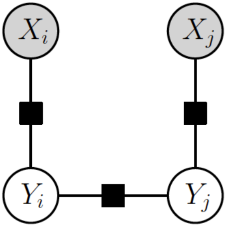
- Markov Random Fields (MRFs) define

$$p(\mathbf{y}) = \frac{1}{Z} \prod_{F \in \mathcal{F}} \psi_F(\mathbf{y}_{N(F)})$$

- Conditional Random Fields (CRFs) define

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{F \in \mathcal{F}} \psi_F(\mathbf{y}_{N(F)}; \mathbf{x})$$

- $\mathbf{x}$ is not a random variable (i.e., not part of the probability distribution)



Energy vs Probabilities

- The probability is completely determined by the energy

$$
\begin{aligned}
p(\mathbf{y}) &= \frac{1}{Z} \prod_{F \in \mathcal{F}} \psi_F(\mathbf{y}_{N(F)}) \\
&= \frac{1}{Z} \exp\left(\log(\psi_F(\mathbf{y}_{N(F)}))\right) \\
&= \frac{1}{Z} \exp\left(-\sum_{F \in \mathcal{F}} E_F(y_F)\right)
\end{aligned}
$$

where $E_F(y_F) = -\log(\psi_F(\mathbf{y}_{N(F)}))$

Parameterization: log linear model

- Factor graphs define a family of distributions
- We are interestested in identifying individual members by parameters

$$E_F(\mathbf{y}_F) = -\mathbf{w}^T \phi_F(\mathbf{y}_F)$$

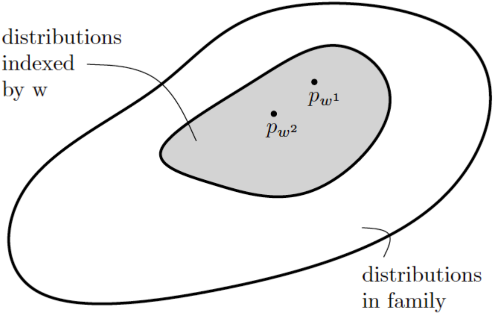


Figure : from [Nowozin et al]

Learning Tasks

- Estimation of the parameters $\mathbf{w}$

$$E_F(\mathbf{y}_F) = -\mathbf{w}^T \phi_F(\mathbf{y}_F)$$

- Learn the structure of the model
- Learn with hidden variables

Inference Tasks

Given an input $x \in \mathcal{X}$ we want to compute

- **MAP estimate** or minimum energy configuration

$$
\begin{aligned}
\operatorname*{argmax}_{y \in \mathcal{Y}} p(\mathbf{y}|\mathbf{x}) &= \operatorname*{argmax}_{y \in \mathcal{Y}} \frac{1}{Z} \prod_{F \in \mathcal{F}} \psi_F(\mathbf{y}_{N(F)}; \mathbf{x}, \mathbf{w}) \\
&= \operatorname*{argmax}_{y \in \mathcal{Y}} \exp(-\sum_{F \in \mathcal{F}} E_F(\mathbf{y}_F, \mathbf{x}, \mathbf{w})) \\
&= \operatorname*{argmin}_{y \in \mathcal{Y}} \sum_{F \in \mathcal{F}} E_F(\mathbf{y}_F, \mathbf{x}, \mathbf{w})
\end{aligned}
$$

Inference Tasks

Given an input $x \in \mathcal{X}$ we want to compute

- **MAP estimate** or minimum energy configuration

$$
\begin{aligned}
\operatorname*{argmax}_{y \in \mathcal{Y}} p(\mathbf{y}|\mathbf{x}) &= \operatorname*{argmax}_{y \in \mathcal{Y}} \frac{1}{Z} \prod_{F \in \mathcal{F}} \psi_F(\mathbf{y}_{N(F)}; \mathbf{x}, \mathbf{w}) \\
&= \operatorname*{argmax}_{y \in \mathcal{Y}} \exp(-\sum_{F \in \mathcal{F}} E_F(\mathbf{y}_F, \mathbf{x}, \mathbf{w})) \\
&= \operatorname*{argmin}_{y \in \mathcal{Y}} \sum_{F \in \mathcal{F}} E_F(\mathbf{y}_F, \mathbf{x}, \mathbf{w})
\end{aligned}
$$

- **Marginals** $p(y_i)$ or max marginals $\max_{y_i \in \mathcal{Y}_i} p(y_i)$, which requires computing the partition function Z, i.e.,

$$
\begin{aligned}
\log(Z(\mathbf{x}, \mathbf{w})) &= \log \sum_{\mathbf{y} \in \mathcal{Y}} \exp(-E(\mathbf{y}; \mathbf{x}, \mathbf{w})) \\
\mu_F(\mathbf{y}_F) &= p(\mathbf{y}_F | \mathbf{x}, \mathbf{w})
\end{aligned}
$$

Inference in Markov Random Fields

MAP Inference

Compute the MAP estimate is typically NP-hard

$$\max_{y \in \mathcal{Y}} p(\mathbf{y}|x) = \max_{y \in \mathcal{Y}} \sum_{r \in \mathcal{R}} \mathbf{w}^T \phi_r(\mathbf{y}_r)$$

MAP Inference

Compute the MAP estimate is typically NP-hard

$$\max_{y \in \mathcal{Y}} p(\mathbf{y}|x) \quad = \quad \max_{y \in \mathcal{Y}} \sum_{r \in \mathcal{R}} \mathbf{w}^T \phi_r(\mathbf{y}_r)$$

Notable exceptions are:

- Belief propagation for tree-structure models

MAP Inference

Compute the MAP estimate is typically NP-hard

$$\max_{y \in \mathcal{Y}} p(\mathbf{y}|x) \quad = \quad \max_{y \in \mathcal{Y}} \sum_{r \in \mathcal{R}} \mathbf{w}^T \phi_r(\mathbf{y}_r)$$

Notable exceptions are:

- Belief propagation for tree-structure models
- Graph cuts for binary energies with sub modular potentials

MAP Inference

Compute the MAP estimate is typically NP-hard

$$\max_{y \in \mathcal{Y}} p(\mathbf{y}|x) \quad = \quad \max_{y \in \mathcal{Y}} \sum_{r \in \mathcal{R}} \mathbf{w}^T \phi_r(\mathbf{y}_r)$$

Notable exceptions are:

- Belief propagation for tree-structure models
- Graph cuts for binary energies with sub modular potentials
- Branch and bound: exponential in worst case, but works much faster in practice

MAP Inference

Compute the MAP estimate is typically NP-hard

$$\max_{y \in \mathcal{Y}} p(\mathbf{y}|x) = \max_{y \in \mathcal{Y}} \sum_{r \in \mathcal{R}} \mathbf{w}^T \phi_r(\mathbf{y}_r)$$

Notable exceptions are:

- Belief propagation for tree-structure models
- Graph cuts for binary energies with sub modular potentials
- Branch and bound: exponential in worst case, but works much faster in practice

Difficulties

- Deal with the exponentially many states in **y**

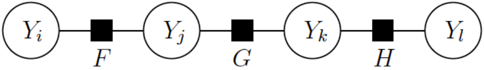
Belief Propagation

- Compact notation

$$\theta_r(\mathbf{y}_r) = \mathbf{w}^T \phi_r(\mathbf{y}_r)$$

- Inference can be written as

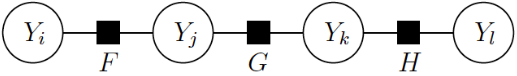
$$\max_{\mathbf{y} \in \mathcal{Y}} \sum_{r \in \mathcal{R}} \theta_r(\mathbf{y}_r)$$



- For the example

$$\max_{y_i, y_j, y_k, y_l} \quad \{\theta_F(y_i, y_j) + \theta_G(y_j, y_k) + \theta_G(y_k, y_l)\}$$

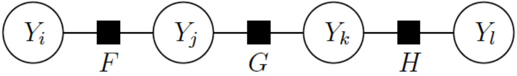
Belief Propagation



$$\theta^*(\mathbf{y}) = \max_{y_i, y_j, y_k, y_l} \{\theta_F(y_i, y_j) + \theta_G(y_j, y_k) + \theta_H(y_k, y_l)\}$$

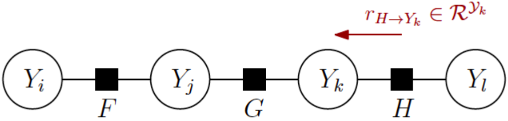
$$=$$

Belief Propagation



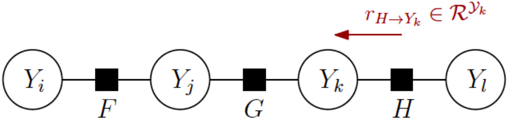
$$\begin{aligned}\theta^*(\mathbf{y}) &= \max_{y_i,y_j,y_k,y_l} \{\theta_F(y_i,y_j) + \theta_G(y_j,y_k) + \theta_H(y_k,y_l)\} \\ &= \max_{y_i,y_j} \theta_F(y_i,y_j) + \max_{y_k} \theta_G(y_j,y_k) + \max_{y_l} \theta_H(y_k,y_l)\end{aligned}$$

Belief Propagation



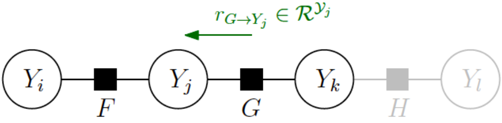
$$\theta^*(\mathbf{y}) = \max_{y_i, y_j} \theta_F(y_i, y_j) + \max_{y_k} \theta_G(y_j, y_k) + \underbrace{\max_{y_l} \theta_H(y_k, y_l)}_{r_{H \to y_k}(y_k)}$$

Belief Propagation



$$
\begin{aligned}
\theta^*(\mathbf{y}) &= \max_{y_i,y_j} \theta_F(y_i,y_j) + \max_{y_k} \theta_G(y_j,y_k) + \underbrace{\max_{y_l} \theta_H(y_k,y_l)}_{r_{H\to y_k}(y_k)} \\
&= \max_{y_i,y_j} \theta_F(y_i,y_j) + \max_{y_k} \theta_G(y_j,y_k) + r_{H\to y_k}(y_k)
\end{aligned}
$$

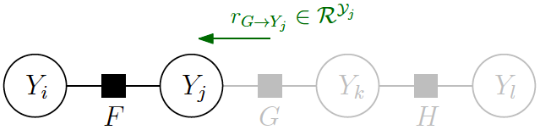
Belief Propagation



$$\theta^*(\mathbf{y}) = \max_{y_i, y_j} \theta_F(y_i, y_j) + \underbrace{\max_{y_k} \theta_G(y_j, y_k) + r_{H \to y_k}(y_k)}_{r_{G \to y_j}(y_j)}$$

$$= \max_{y_i, y_j} \theta_F(y_i, y_j) +$$

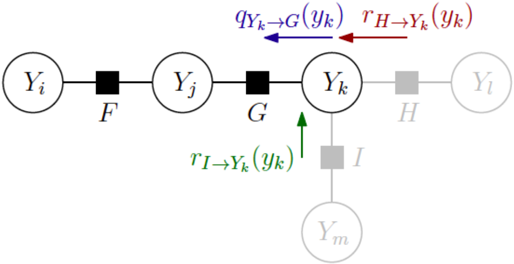
Belief Propagation



$$\theta^*(\mathbf{y}) = \max_{y_i, y_j} \theta_F(y_i, y_j) + \underbrace{\max_{y_k} \theta_G(y_j, y_k) + r_{H \to y_k}(y_k)}_{r_{G \to y_j}(y_j)}$$

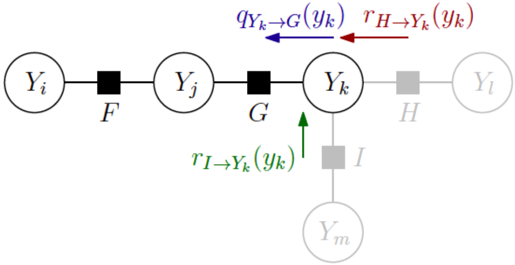
$$= \max_{y_i, y_j} \theta_F(y_i, y_j) + r_{G \to y_j}(y_j)$$

Tree Generalization



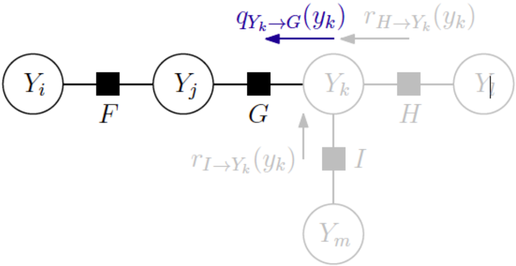
$$\theta^*(\mathbf{y}) \quad = \quad \max_{y_i, y_k, y_k, y_l, y_m} \theta_F(y_i, y_j) + \theta_G(y_j, y_k) + \theta_I(y_m, y_k) + \theta_H(y_l, y_k)$$

Tree Generalization



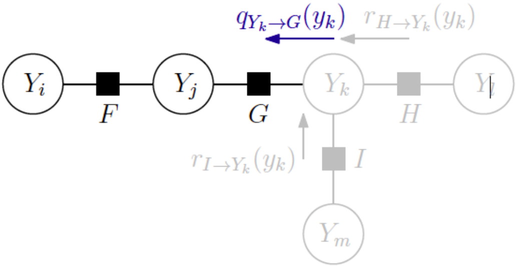
$$
\begin{aligned}
\theta^*(\mathbf{y}) &= \max_{y_i, y_k, y_k, y_l, y_m} \theta_F(y_i, y_j) + \theta_G(y_j, y_k) + \theta_I(y_m, y_k) + \theta_H(y_l, y_k) \\
&= \max_{y_i, y_j} \theta_F(y_i, y_j) + \max_{y_k} \theta_G(y_j, y_k) + \max_{y_m} \theta_I(y_m, y_k) + \max_{y_l} \theta_H(y_l, y_k)
\end{aligned}
$$

Tree Generalization



$$
\begin{aligned}
\theta^*(\mathbf{y}) &= \max_{y_i, y_k, y_k, y_l, y_m} \theta_F(y_i, y_j) + \theta_G(y_j, y_k) + \theta_I(y_m, y_k) + \theta_H(y_l, y_k) \\
&= \max_{y_i, y_j} \theta_F(y_i, y_j) + \max_{y_k} \theta_G(y_j, y_k) + \max_{y_m} \theta_I(y_m, y_k) + \max_{y_l} \theta_H(y_l, y_k) \\
&= \max_{y_i, y_j} \theta_F(y_i, y_j) + \max_{y_k} \theta_G(y_j, y_k) + r_{H \to y_k}(y_k) + r_{I \to y_k}(y_k)
\end{aligned}
$$

Tree Generalization



$$
\begin{aligned}
\theta^*(\mathbf{y}) &= \max_{y_i, y_k, y_k, y_l, y_m} \theta_F(y_i, y_j) + \theta_G(y_j, y_k) + \theta_I(y_m, y_k) + \theta_H(y_l, y_k) \\
&= \max_{y_i, y_j} \theta_F(y_i, y_j) + \max_{y_k} \theta_G(y_j, y_k) + \max_{y_m} \theta_I(y_m, y_k) + \max_{y_l} \theta_H(y_l, y_k) \\
&= \max_{y_i, y_j} \theta_F(y_i, y_j) + \max_{y_k} \theta_G(y_j, y_k) + r_{H \to y_k}(y_k) + r_{I \to y_k}(y_k) \\
&= \max_{y_i, y_j} \theta_F(y_i, y_j) + \max_{y_k} \theta_G(y_j, y_k) + q_{y_k \to G}(y_k)
\end{aligned}
$$

Factor Graph Max Product

Iteratively updates and passes messages:

- $r_{F \to y_i} \in \Re^{\mathcal{Y}_i}$: factor to variable message
- $q_{y_i \to F} \in \Re^{\mathcal{Y}_i}$: variable to factor message

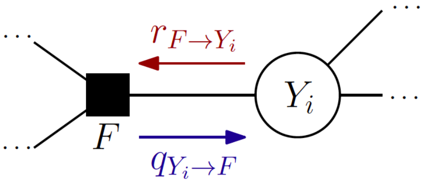


Figure : from [Nowozin et al]

Variable to factor

- Let $M(i)$ be the factors adjacent to variable i, $M(i) = \{F \in \mathcal{F} : (i, F) \in \mathcal{E}\}$
- Variable-to-factor message

$$q_{y_i \to F}(y_i) = \sum_{F' \in M(i) \setminus \{F\}} r_{F' \to y_i}(y_i)$$

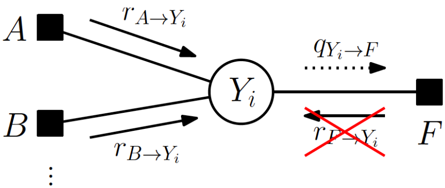


Figure : from [Nowozin et al]

Factor to variable

- Factor-to-variable message

$$r_{F\to y_i}(y_i) = \max_{y'_F \in \mathcal{Y}_F, y'_i = y_i} \left(\theta(y'_F) + \sum_{j \in N(F)\setminus\{i\}} q_{y_j \to F}(y'_j) \right)$$

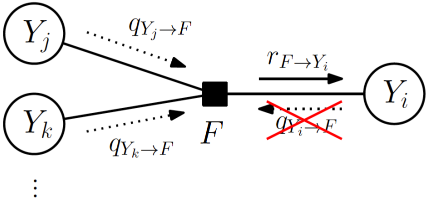


Figure : from [Nowozin et al]

Message Scheduling

1. Select one variable as tree root
2. Compute leaf-to-root messages
3. Compute root-to-leaf messages

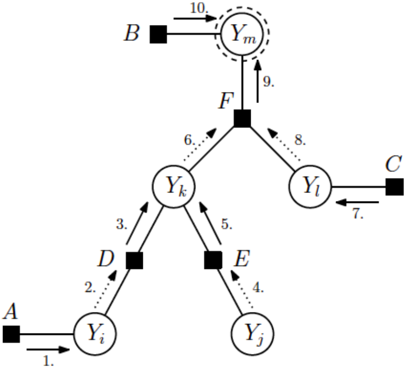


Figure : from [Nowozin et al]

Max Product v Sum Product

Max sum version of max-product

1. Compute leaf-to-root messages

$$q_{y_i \to F}(y_i) = \sum_{F' \in M(i) \setminus \{F\}} r_{F' \to y_i}(y_i)$$

2. Compute root-to-leaf messages

$$r_{F \to y_i}(y_i) = \max_{y'_F \in \mathcal{Y}_F, y'_i = y_i} \left(\theta(y'_F) + \sum_{j \in N(F) \setminus \{i\}} q_{y_j \to F(y'_j)} \right)$$

Max Product v Sum Product

Max sum version of max-product

1. Compute leaf-to-root messages

$$q_{y_i \to F}(y_i) = \sum_{F' \in M(i) \setminus \{F\}} r_{F' \to y_i}(y_i)$$

2. Compute root-to-leaf messages

$$r_{F \to y_i}(y_i) = \max_{y'_F \in \mathcal{Y}_F, y'_i = y_i} \left(\theta(y'_F) + \sum_{j \in N(F) \setminus \{i\}} q_{y_j \to F(y'_j)} \right)$$

Sum-product

1. Compute leaf-to-root messages

$$q_{y_i \to F}(y_i) = \sum_{F' \in M(i) \setminus \{F\}} r_{F' \to y_i}(y_i)$$

2. Compute root-to-leaf messages

$$r_{F \to y_i}(y_i) = \log \sum_{y'_F \in \mathcal{Y}_F, y'_i = y_i} \exp \left(\theta(y'_F) + \sum_{j \in N(F) \setminus \{i\}} q_{y'_j \to F}(y'_j) \right)$$

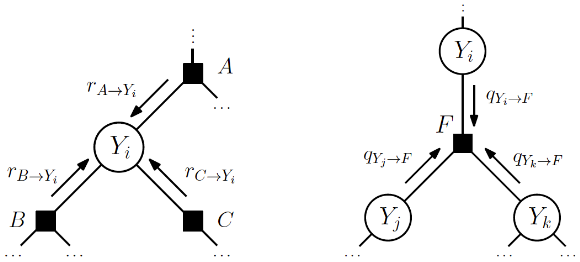
Computing marginals

- Partition function can be evaluated at the root

$$\log Z = \log \sum_{y_r} \exp \left(\sum_{F \in M(r)} r_{F \to y_r}(y_r) \right)$$

- Marginal distributions, for each factor

$$\mu_F(y_F) = p(y_F) = \frac{1}{Z} \exp \left(\theta_F(y_F) + \sum_{i \in N(F)} q_{y_i \to F}(y_i) \right)$$



Computing marginals

- Partition function can be evaluated at the root

$$\log Z = \log \sum_{y_r} \exp \left(\sum_{F \in M(r)} r_{F \to y_r}(y_r) \right)$$

- Marginal distributions, for each factor

$$\mu_F(y_F) = p(y_F) = \frac{1}{Z} \exp \left(\theta_F(y_F) + \sum_{i \in N(F)} q_{y_i \to F}(y_i) \right)$$

- Marginals at every node

$$\mu_{y_i}(y_i) = p(y_i) = \frac{1}{Z} \exp \left(\sum_{F \in M(i)} r_{F \to y_i}(y_i) \right)$$

Generalizations to loops

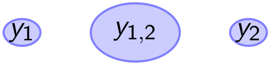
- It is call **loopy belief propagation** (Perl, 1988)
- No schedule that removes dependencies
- Different messaging schedules (synchronous/asynchronous, static/dynamic)
- Slight changes in the algorithm

MAP LP Relaxation Task

Integer Linear Program (LP) equivalence [Werner 2007]:

- Inference task:

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y}} \sum_r \theta_r(\mathbf{y}_r)$$



- Variables $b_r(\mathbf{y}_r)$:

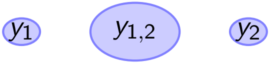
$$\max_{b_1,b_2,b_{12}} \begin{bmatrix} b_1(0) \\ b_1(1) \\ b_2(0) \\ b_2(1) \\ b_{12}(0,0) \\ b_{12}(1,0) \\ b_{12}(0,1) \\ b_{12}(1,1) \end{bmatrix}^\top \begin{bmatrix} \theta_1(0) \\ \theta_1(1) \\ \theta_2(0) \\ \theta_2(1) \\ \theta_{12}(0,0) \\ \theta_{12}(1,0) \\ \theta_{12}(0,1) \\ \theta_{12}(1,1) \end{bmatrix}$$

MAP LP Relaxation Task

Integer Linear Program (LP) equivalence [Werner 2007]:

- Inference task:

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y}} \sum_r \theta_r(\mathbf{y}_r)$$



- Variables $b_r(\mathbf{y}_r)$:

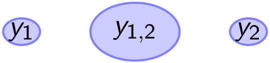
$$\max_{b_1,b_2,b_{12}} \begin{bmatrix} b_1(0) \\ b_1(1) \\ b_2(0) \\ b_2(1) \\ b_{12}(0,0) \\ b_{12}(1,0) \\ b_{12}(0,1) \\ b_{12}(1,1) \end{bmatrix}^\top \begin{bmatrix} \theta_1(0) \\ \theta_1(1) \\ \theta_2(0) \\ \theta_2(1) \\ \theta_{12}(0,0) \\ \theta_{12}(1,0) \\ \theta_{12}(0,1) \\ \theta_{12}(1,1) \end{bmatrix} \quad \text{s.t.} \quad b_r(\mathbf{y}_r) \in \{0,1\}$$

MAP LP Relaxation Task

Integer Linear Program (LP) equivalence [Werner 2007]:

- Inference task:

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y}} \sum_r \theta_r(\mathbf{y}_r)$$



- Variables $b_r(\mathbf{y}_r)$:

$$\max_{b_1,b_2,b_{12}} \begin{bmatrix} b_1(0) \\ b_1(1) \\ b_2(0) \\ b_2(1) \\ b_{12}(0,0) \\ b_{12}(1,0) \\ b_{12}(0,1) \\ b_{12}(1,1) \end{bmatrix}^\top \begin{bmatrix} \theta_1(0) \\ \theta_1(1) \\ \theta_2(0) \\ \theta_2(1) \\ \theta_{12}(0,0) \\ \theta_{12}(1,0) \\ \theta_{12}(0,1) \\ \theta_{12}(1,1) \end{bmatrix} \quad \text{s.t.} \quad \begin{array}{c} b_r(\mathbf{y}_r) \in \{0,1\} \\ \sum_{\mathbf{y}_r} b_r(\mathbf{y}_r) = 1 \end{array}$$

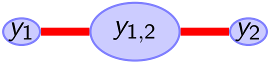
MAP LP Relaxation Task

Integer Linear Program (LP) equivalence [Werner 2007]:

- Inference task:

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y}} \sum_r \theta_r(\mathbf{y}_r)$$

- Variables $b_r(\mathbf{y}_r)$:



$$\max_{b_1,b_2,b_{12}} \begin{bmatrix} b_1(0) \\ b_1(1) \\ b_2(0) \\ b_2(1) \\ b_{12}(0,0) \\ b_{12}(1,0) \\ b_{12}(0,1) \\ b_{12}(1,1) \end{bmatrix}^\top \begin{bmatrix} \theta_1(0) \\ \theta_1(1) \\ \theta_2(0) \\ \theta_2(1) \\ \theta_{12}(0,0) \\ \theta_{12}(1,0) \\ \theta_{12}(0,1) \\ \theta_{12}(1,1) \end{bmatrix} \quad \text{s.t.} \quad \begin{array}{l} b_r(\mathbf{y}_r) \in \{0,1\} \\ \sum_{\mathbf{y}_r} b_r(\mathbf{y}_r) = 1 \\ \sum_{\mathbf{y}_p \setminus \mathbf{y}_r} b_p(\mathbf{y}_p) = b_r(\mathbf{y}_r) \end{array}$$

MAP LP Relaxation Task

$$\max_{b_1, b_2, b_{12}} \begin{bmatrix} b_1(1) \\ b_1(2) \\ b_2(1) \\ b_2(2) \\ b_{12}(1,1) \\ b_{12}(2,1) \\ b_{12}(1,2) \\ b_{12}(2,2) \end{bmatrix}^\top \begin{bmatrix} \theta_1(1) \\ \theta_1(2) \\ \theta_2(1) \\ \theta_2(2) \\ \theta_{12}(1,1) \\ \theta_{12}(2,1) \\ \theta_{12}(1,2) \\ \theta_{12}(2,2) \end{bmatrix} \quad \text{s.t.} \quad \begin{array}{l} b_r(\mathbf{y}_r) \in \{0,1\} \\ \sum_{y_r} b_r(\mathbf{y}_r) = 1 \\ \sum_{\mathbf{y}_p \setminus \mathbf{y}_r} b_p(\mathbf{y}_p) = b_r(\mathbf{y}_r) \end{array}$$

MAP LP Relaxation Task

$$\max_{b_r} \quad \sum_{r,\mathbf{y}_r} b_r(\mathbf{y}_r)\theta_r(\mathbf{y}_r) \quad \text{s.t.} \quad \begin{aligned} & b_r(\mathbf{y}_r) \in \{0,1\} \\ & \sum_{y_r} b_r(\mathbf{y}_r) = 1 \\ & \sum_{\mathbf{y}_p \setminus \mathbf{y}_r} b_p(\mathbf{y}_p) = b_r(\mathbf{y}_r) \end{aligned}$$

MAP LP Relaxation Task

$$\max_{b_r} \quad \sum_{r,\mathbf{y}_r} b_r(\mathbf{y}_r)\theta_r(\mathbf{y}_r) \qquad \text{s.t.} \quad \begin{array}{l} b_r(\mathbf{y}_r) \in \{0,1\} \\ \sum_{y_r} b_r(\mathbf{y}_r) = 1 \\ \text{Marginalization} \end{array}$$

MAP LP Relaxation Task

$$\max_{b_r} \quad \sum_{r,\mathbf{y}_r} b_r(\mathbf{y}_r)\theta_r(\mathbf{y}_r) \quad \text{s.t.} \quad \begin{array}{l} b_r(\mathbf{y}_r) \in \{0,1\} \\ \text{Local probability } b_r \\ \\ \text{Marginalization} \end{array}$$

MAP LP Relaxation Task

LP relaxation:

$$\max_{b_r} \sum_{r,\mathbf{y}_r} b_r(\mathbf{y}_r)\theta_r(\mathbf{y}_r) \quad \text{s.t.} \quad \begin{array}{l} \cancel{b_r(\mathbf{y}_r) \in \{0,1\}} \\ \text{Local probability } b_r \\ \\ \text{Marginalization} \end{array}$$

MAP LP Relaxation Task

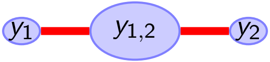
LP relaxation:

$$\max_{b_r} \quad \sum_{r,\mathbf{y}_r} b_r(\mathbf{y}_r)\theta_r(\mathbf{y}_r) \quad \text{s.t.} \quad \begin{array}{l} \cancel{b_r(\mathbf{y}_r) \in \{0,1\}} \\ \text{Local probability } b_r \\ \\ \text{Marginalization} \end{array}$$

Can be solved by any standard LP solver but **slow** because of typically many variables and constraints. Can we do better?

MAP LP Relaxation Task

Observation: Graph structure in marginalization constraints.



Use dual to take advantage of structure in constraint set

- Set of parents of region r: $P(r)$
- Set of children of region r: $C(r)$

$$\forall r, \mathbf{y}_r, p \in P(r) \qquad \sum_{\mathbf{y}_p \setminus \mathbf{y}_r} b_p(\mathbf{y}_p) = b_r(\mathbf{y}_r)$$

- Lagrange multipliers for every constraint:

$$\forall r, \mathbf{y}_r, p \in P(r) \qquad \lambda_{r \to p}(\mathbf{y}_r)$$

MAP LP Relaxation Task

Re-parameterization of score $\theta_r(\mathbf{y}_r)$:

$$\hat{\theta}_r(\mathbf{y}_r) = \theta_r(\mathbf{y}_r) + \sum_{p \in P(r)} \lambda_{r \to p}(\mathbf{y}_r) - \sum_{c \in C(r)} \lambda_{c \to r}(\mathbf{y}_c)$$

Properties of dual program:

$$\min_\lambda q(\lambda) = \min_\lambda \sum_r \max_{\mathbf{y}_r} \hat{\theta}_r(\mathbf{y}_r)$$

- **Dual upper-bounds primal** $\forall \lambda$
- Convex problem
- Unconstrained task
- Doing block coordinate descent in the dual results on message passing (Lagrange multipliers are your messages)

MAP LP Relaxation Task

Block-coordinate descent solvers iterate the following steps:

- Take a block of Lagrange multipliers
- Optimize sub-problem of dual function w.r.t. this block while keeping all other variables fixed

Advantage: fast due to analytically computable sub-problems

Same type of algorithms also exist to compute approximate marginals

Graph-Cuts for MRF Inference

Theorem [Kolmogorov and Zabih, 2004]: If the energy function is a function of binary variables containing only unary and pairwise factors, the discrete energy minimization problem

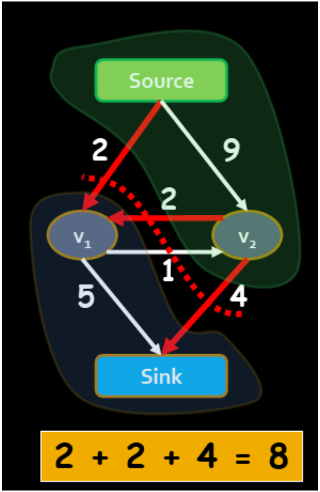
$$\min_{\mathbf{y}} \sum_{r \in \mathcal{R}} E(\mathbf{y}_r, x)$$

can be formulated as a graph cut problem if an only off all pairwise energies are sub modular

$$E_{i,j}(0,0) + E_{i,j}(1,1) \leq E_{i,j}(0,1) + E_{i,j}(1,0)$$

The ST-mincut problem

- The st-mincut is the st-cut with the minimum cost

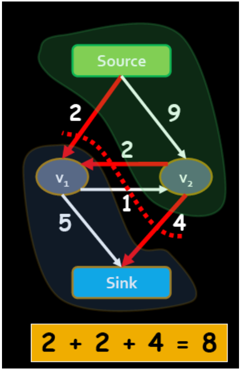


[Source: P. Kohli]

Back to our energy minimization

Construct a graph such that

1. Any st-cut corresponds to an assignment of x
2. The cost of the cut is equal to the energy of x : E(x)



[Source: P. Kohli]

$$E(x) = \sum_i \theta_i (x_i) + \sum_{i,j} \theta_{ij} (x_i, x_j)$$

For all ij $\theta_{ij}(0,1) + \theta_{ij}(1,0) \geq \theta_{ij}(0,0) + \theta_{ij}(1,1)$

Equivalent (transformable)

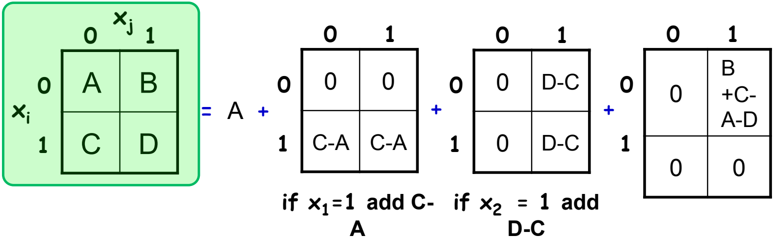
$$E(x) = \sum_i c_i x_i + \sum_{i,j} c_{ij} x_i(1 - x_j)$$

$c_{ij} \geq 0$

[Source: P. Kohli]

How are they equivalent?

$A = \theta_{ij}(0,0)$ $B = \theta_{ij}(0,1)$ $C = \theta_{ij}(1,0)$ $D = \theta_{ij}(1,1)$

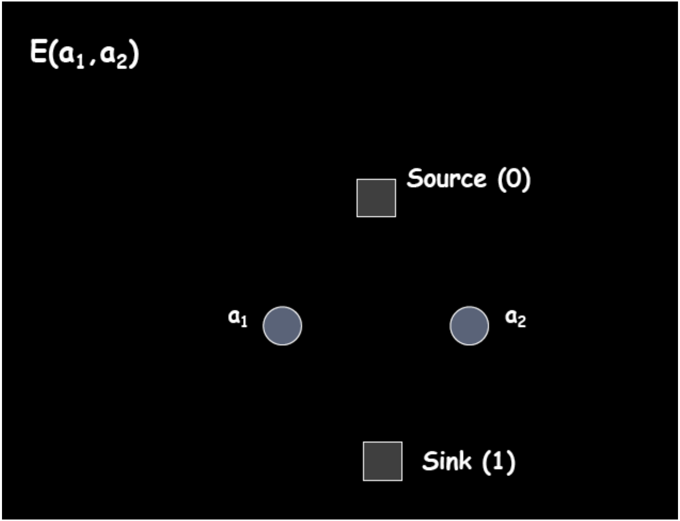


$$\theta_{ij}(x_i,x_j) = \theta_{ij}(0,0) + (\theta_{ij}(1,0)-\theta_{ij}(0,0))\, x_i + (\theta_{ij}(1,0)-\theta_{ij}(0,0))\, x_j + (\theta_{ij}(1,0) + \theta_{ij}(0,1) - \theta_{ij}(0,0) - \theta_{ij}(1,1))\,(1-x_i)\, x_j$$

$B+C-A-D \geq 0$ is true from the submodularity of θ_{ij}

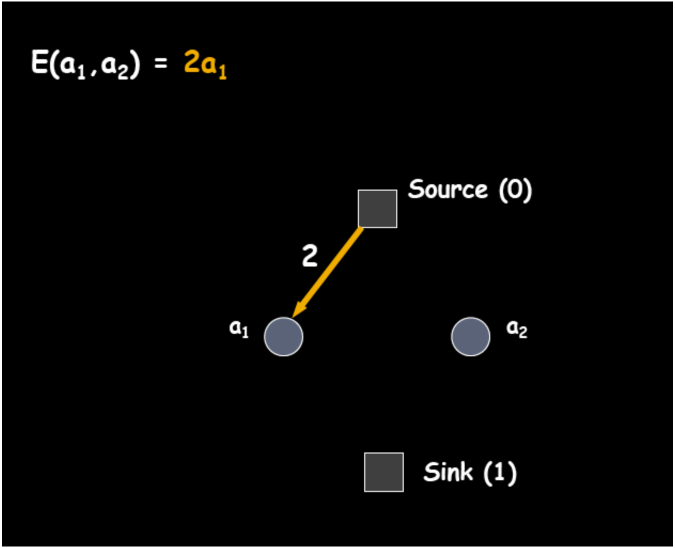
[Source: P. Kohli]

Graph Construction



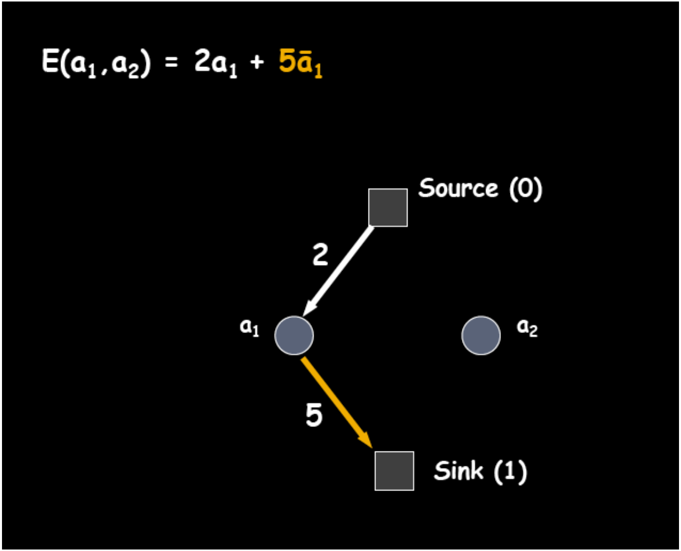
[Source: P. Kohli]

Graph Construction



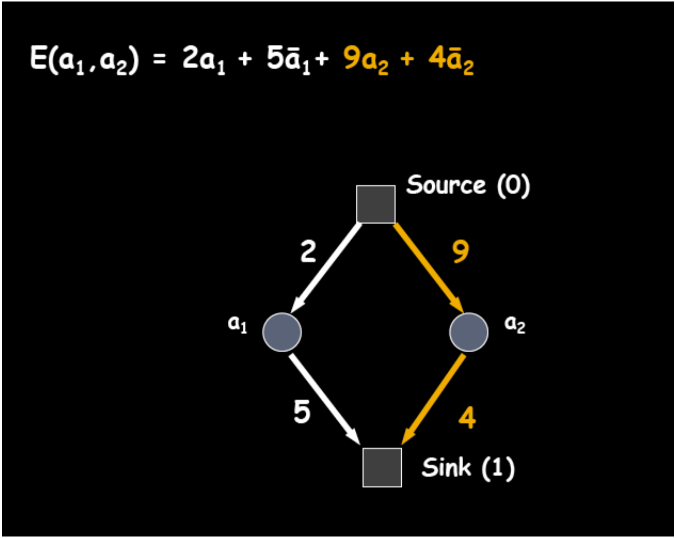
[Source: P. Kohli]

Graph Construction



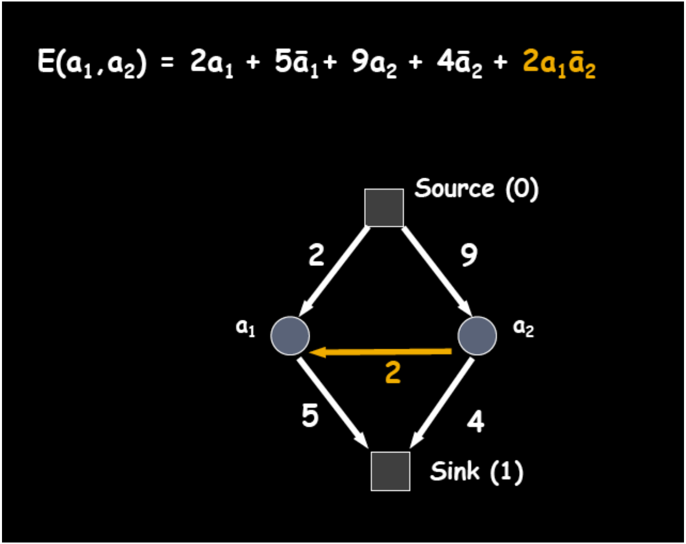
[Source: P. Kohli]

Graph Construction



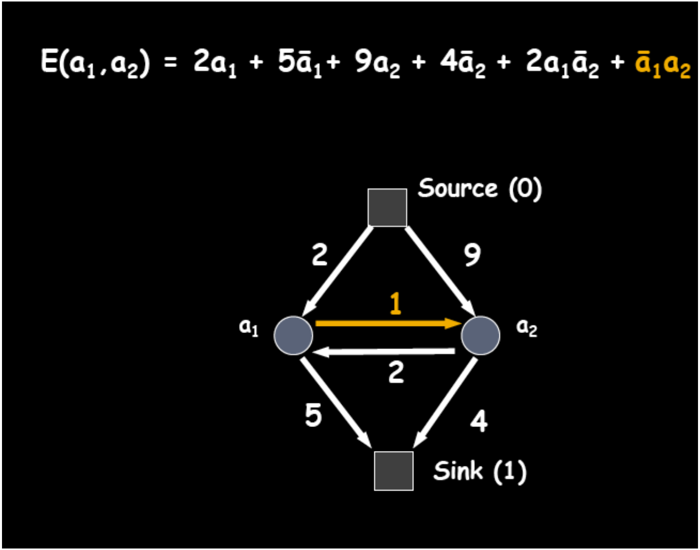
[Source: P. Kohli]

Graph Construction



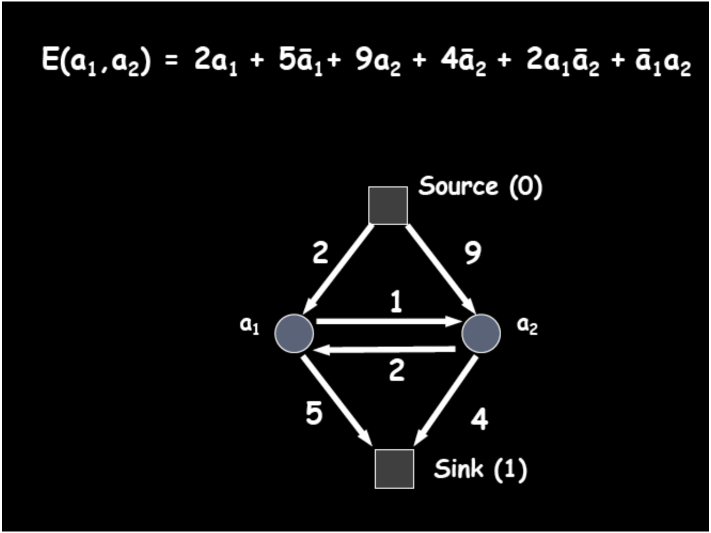
[Source: P. Kohli]

Graph Construction



[Source: P. Kohli]

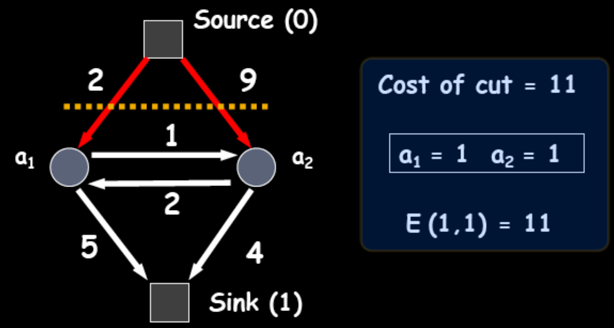
Graph Construction



[Source: P. Kohli]

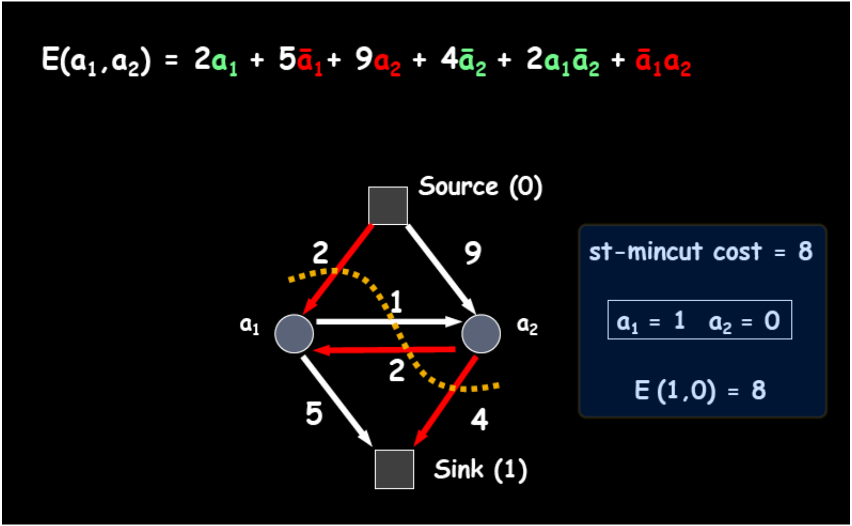
Graph Construction

$$E(a_1, a_2) = 2a_1 + 5\bar{a}_1 + 9a_2 + 4\bar{a}_2 + 2a_1\bar{a}_2 + \bar{a}_1 a_2$$



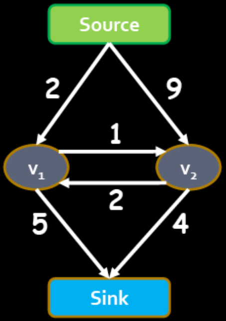
[Source: P. Kohli]

Graph Construction



[Source: P. Kohli]

How to compute the St-mincut?



Solve the dual **maximum flow** problem

Compute the maximum flow between Source and Sink s.t.

Edges: Flow < Capacity

Nodes: Flow in = Flow out

Min-cut\Max-flow Theorem

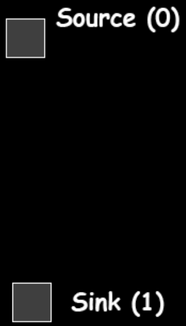
In every network, the maximum flow equals the cost of the st-mincut

Assuming non-negative capacity

[Source: P. Kohli]

How does the code look like

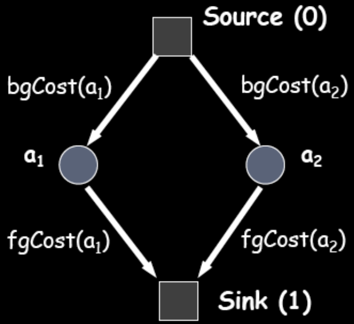
```
Graph *g;

For all pixels p

    /* Add a node to the graph */
    nodeID(p) = g->add_node();

    /* Set cost of terminal edges */
    set_weights(nodeID(p), fgCost(p), bgCost(p));

end

for all adjacent pixels p,q
    add_weights(nodeID(p), nodeID(q),  cost(p,q));
end

g->compute_maxflow();

label_p = g->is_connected_to_source(nodeID(p));
// is the label of pixel p (0 or 1)
```



[Source: P. Kohli]

How does the code look like

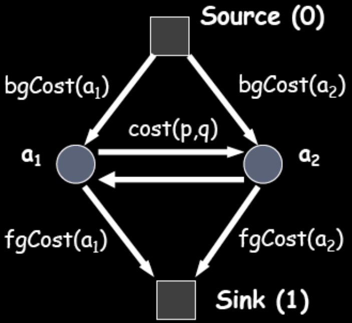
```
Graph *g;

For all pixels p

    /* Add a node to the graph */
    nodeID(p) = g->add_node();

    /* Set cost of terminal edges */
    set_weights(nodeID(p), fgCost(p), bgCost(p));

end

for all adjacent pixels p,q
    add_weights(nodeID(p), nodeID(q),  cost(p,q));
end

g->compute_maxflow();

label_p = g->is_connected_to_source(nodeID(p));
// is the label of pixel p (0 or 1)
```



[Source: P. Kohli]

How does the code look like

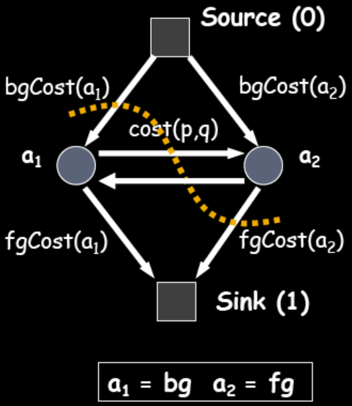
```
Graph *g;

For all pixels p

        /* Add a node to the graph */
        nodeID(p) = g->add_node();

        /* Set cost of terminal edges */
        set_weights(nodeID(p), fgCost(p), bgCost(p));

end

for all adjacent pixels p,q
        add_weights(nodeID(p), nodeID(q),  cost(p,q));
end

g->compute_maxflow();

label_p = g->is_connected_to_source(nodeID(p));
// is the label of pixel p (0 or 1)
```



[Source: P. Kohli]

How does the code look like

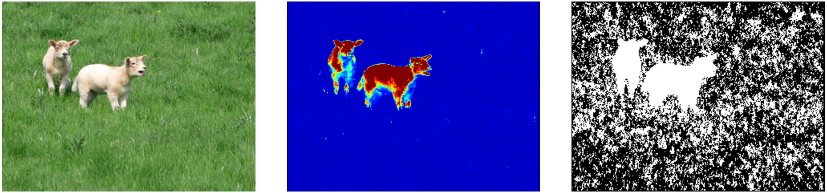
```
Graph *g;

For all pixels p

        /* Add a node to the graph */
        nodeID(p) = g->add_node();

        /* Set cost of terminal edges */
        set_weights(nodeID(p), fgCost(p), bgCost(p));

end

for all adjacent pixels p,q
        add_weights(nodeID(p), nodeID(q), cost(p,q));
end

g->compute_maxflow();

label_p = g->is_connected_to_source(nodeID(p));
// is the label of pixel p (0 or 1)
```



[Source: P. Kohli]

Example: Figure-Ground Segmentation

Binary labeling problem



(Original) (Color model) (Indep. Prediction)

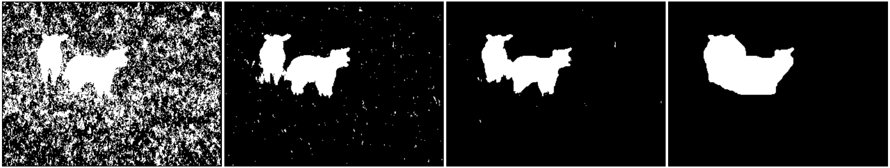
Figure : from [Nowozin et al]

Example: Figure-Ground Segmentation

- Markov Random Field

$$E(\mathbf{y}, \mathbf{x}, \mathbf{w}) = \sum_i \log p(y_i|x_i) + w \sum_{(i,j)\in\mathcal{E}} C(x_i, x_j) I(y_i \neq y_j)$$

with $C(x_i, x_j) = \exp(\gamma||x_i - x_j||^2)$, and $w \geq 0$.



(w=0) (w small) (w medium) (large w)

Figure : from [Nowozin et al]

- Why do we need the condition $w \geq 0$?

Generalization to Multi-label Problems

- Optimal solution is not possible anymore
- Solve to optimality subproblems that include current iterate
- This guarantees decrease in the objective

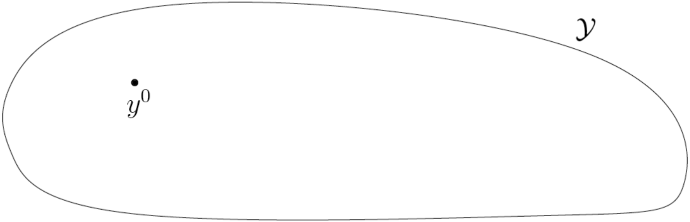


Figure : from [Nowozin et al]

Generalization to Multi-label Problems

- Optimal solution is not possible anymore
- Solve to optimality subproblems that include current iterate
- This guarantees decrease in the objective

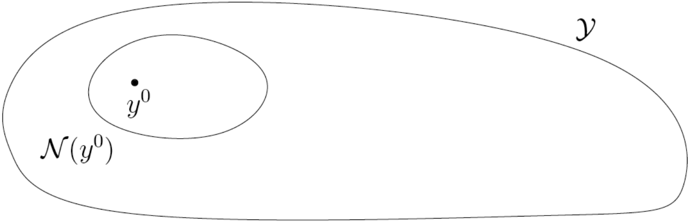


Figure : from [Nowozin et al]

Generalization to Multi-label Problems

- Optimal solution is not possible anymore
- Solve to optimality subproblems that include current iterate
- This guarantees decrease in the objective

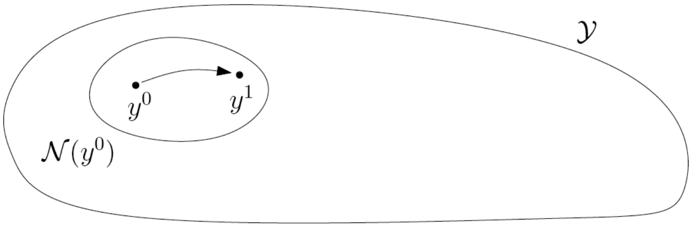


Figure : from [Nowozin et al]

Generalization to Multi-label Problems

- Optimal solution is not possible anymore
- Solve to optimality subproblems that include current iterate
- This guarantees decrease in the objective

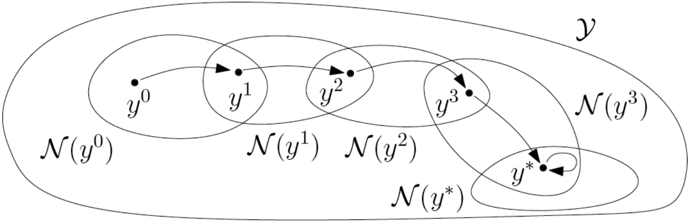


Figure : from [Nowozin et al]

Metric vs Semimetric

Two general classes of pairwise interactions

- **Metric** if it satisfies for any set of labels α, β, γ

$$\begin{aligned} V(\alpha,\beta) = 0 &\leftrightarrow \alpha = \beta \\ V(\alpha,\beta) &= V(\beta,\alpha) \geq 0 \\ V(\alpha,\beta) &\leq V(\alpha,\gamma) + V(\gamma,\beta) \end{aligned}$$

Metric vs Semimetric

Two general classes of pairwise interactions

- **Metric** if it satisfies for any set of labels α, β, γ

$$\begin{aligned} V(\alpha,\beta) = 0 \quad &\leftrightarrow \quad \alpha = \beta \\ V(\alpha,\beta) \quad &= \quad V(\beta,\alpha) \geq 0 \\ V(\alpha,\beta) \quad &\leq \quad V(\alpha,\gamma) + V(\gamma,\beta) \end{aligned}$$

- **Semi-metric** if it satisfies for any set of labels α, β, γ

$$\begin{aligned} V(\alpha,\beta) = 0 \quad &\leftrightarrow \quad \alpha = \beta \\ V(\alpha,\beta) \quad &= \quad V(\beta,\alpha) \geq 0 \end{aligned}$$

Examples for 1D label set

- Truncated quadratic is a semi-metric

$$V(\alpha, \beta) = \min(K, |\alpha - \beta|^2)$$

with K a constant.

Examples for 1D label set

- Truncated quadratic is a semi-metric

$$V(\alpha, \beta) = \min(K, |\alpha - \beta|^2)$$

with K a constant.

- Truncated absolute distance is a metric

$$V(\alpha, \beta) = \min(K, |\alpha - \beta|)$$

with K a constant.

Examples for 1D label set

- Truncated quadratic is a semi-metric

$$V(\alpha, \beta) = \min(K, |\alpha - \beta|^2)$$

with K a constant.

- Truncated absolute distance is a metric

$$V(\alpha, \beta) = \min(K, |\alpha - \beta|)$$

with K a constant.

- Potts model is a metric

$$V(\alpha, \beta) = K \cdot T(\alpha \neq \beta)$$

with $T(\cdot) = 1$ if the argument is true and 0 otherwise.

Move Making Algorithms

- **Alpha Expansion**: Checks if current nodes want to switch to label α
- **Alpha - Beta Swaps**: Checks if a node with class α wants to switch to β.
- Binary problems that can be solve exactly for certain type of potentials

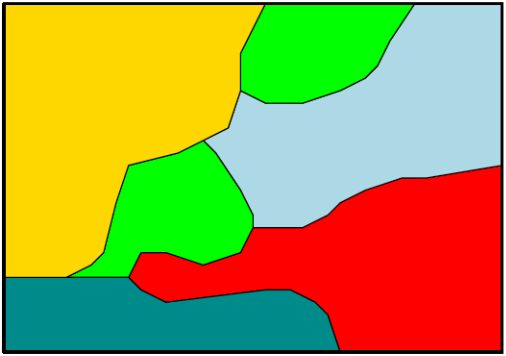


Figure : Alpha-beta Swaps. Figure from [Nowozin et al]

Move Making Algorithms

- **Alpha Expansion**: Checks if current nodes want to switch to label α
- **Alpha - Beta Swaps**: Checks if a node with class α wants to switch to β.
- Binary problems that can be solve exactly for certain type of potentials

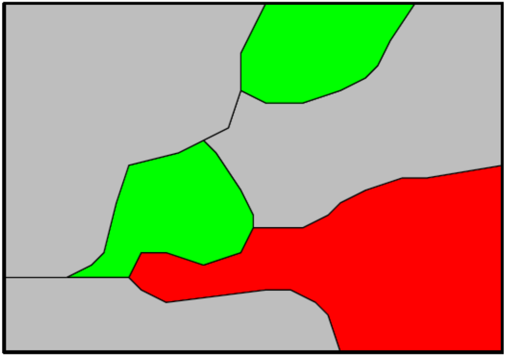


Figure : Alpha-beta Swaps. Figure from [Nowozin et al]

Move Making Algorithms

- **Alpha Expansion**: Checks if current nodes want to switch to label α
- **Alpha - Beta Swaps**: Checks if a node with class α wants to switch to β.
- Binary problems that can be solve exactly for certain type of potentials

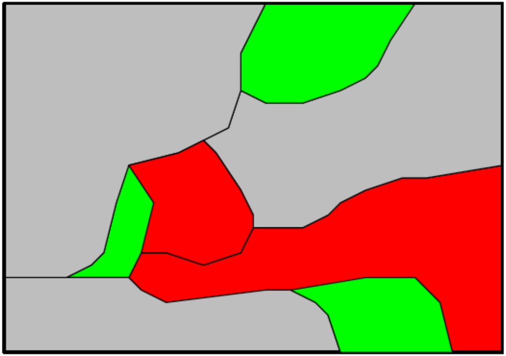


Figure : Alpha-beta Swaps. Figure from [Nowozin et al]

Move Making Algorithms

- **Alpha Expansion**: Checks if current nodes want to switch to label α
- **Alpha - Beta Swaps**: Checks if a node with class α wants to switch to β.
- Binary problems that can be solve exactly for certain type of potentials

Figure : Alpha-beta Swaps. Figure from [Nowozin et al]

Move Making Algorithms

- **Alpha Expansion**: Checks if current nodes want to switch to label α
- **Alpha - Beta Swaps**: Checks if a node with class α wants to switch to β.
- Binary problems that can be solve exactly for certain type of potentials

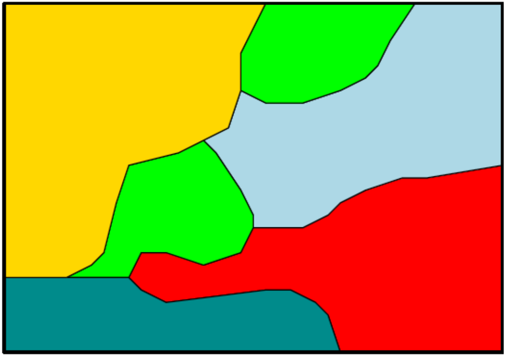


Figure : Alpha-beta Swaps. Figure from [Nowozin et al]

Binary Moves

- $\alpha - \beta$ moves works for semi-metrics
- α expansion works for V being a metric

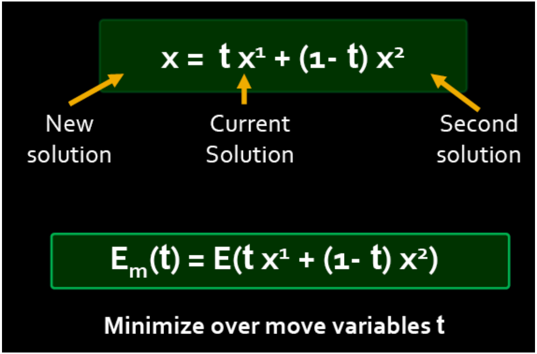
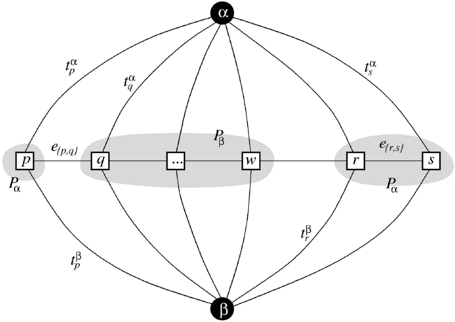


Figure : from P. Kohli tutorial on graph-cuts

- For certain x^1 and x^2, the move energy is sub-modular

Graph Construction

- The set of vertices includes the two terminals α and β, as well as image pixels p in the sets $\mathcal{P}_\alpha$ and $\mathcal{P}_\beta$ (i.e., $f_p \in \{\alpha, \beta\}$).
- Each pixel $p \in \mathcal{P}_{\alpha\beta}$ is connected to the terminals α and β, called t-links.
- Each set of pixels $p, q \in \mathcal{P}_{\alpha\beta}$ which are neighbors is connected by an edge $e_{p,q}$



edge	weight	for
t_p^α	$D_p(\alpha) + \sum_{q \in \mathcal{N}_p, q \notin \mathcal{P}_{\alpha\beta}} V(\alpha, f_q)$	$p \in \mathcal{P}_{\alpha\beta}$
t_p^β	$D_p(\beta) + \sum_{q \in \mathcal{N}_p, q \notin \mathcal{P}_{\alpha\beta}} V(\beta, f_q)$	$p \in \mathcal{P}_{\alpha\beta}$
$e_{\{p,q\}}$	$V(\alpha, \beta)$	$\{p,q\} \in \mathcal{N}$, $p, q \in \mathcal{P}_{\alpha\beta}$