Combinatorial Optimization in Computer Vision (IN2245)

Frank R. Schmidt
Csaba Domokos

Winter Semester 2015/2016

9. Belief Propagation

Inference revisited

Inference means the procedure to estimate the probability distribution, encoded by a graphical model, for a given data.

Assume we are given a factor graph and the observation x.

- **Maximum A Posteriori (MAP) inference**: find the *state* $y^* \in \mathcal{Y}$ *of maximum probability*,

$$y^* \in \operatorname*{argmax}_{y \in \mathcal{Y}} p(Y = y \mid x) = \operatorname*{argmin}_{y \in \mathcal{Y}} E(y; x) .$$

- **Probabilistic inference**: find the value of the *log partition function* and the *marginal distributions* for each factor,

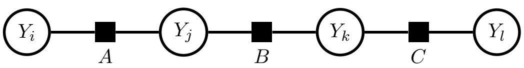
$$\log Z(x) = \log \sum_{y \in \mathcal{Y}} \exp(-E(y; x)) ,$$

$$\mu_F(y_F) = p(Y_F = y_F \mid x) \quad \forall F \in \mathcal{F}, \forall y_F \in \mathcal{Y}_F .$$

Sum-product algorithm

Inference on chains

Assume that we are given the following factor graph and a corresponding energy function $E(y)$, where $\mathcal{Y} = \mathcal{Y}_i \times \mathcal{Y}_j \times \mathcal{Y}_k \times \mathcal{Y}_l$.



We want to compute $p(y)$ for any $y \in \mathcal{Y}$ by making use of the factorization

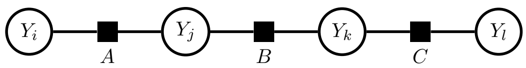
$$p(y) = \frac{1}{Z} \exp(-E(y)) .$$

Problem: we also need to calculate the *partition function*

$$Z = \sum_{y \in \mathcal{Y}} \exp(-E(y)) = \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \sum_{y_k \in \mathcal{Y}_k} \sum_{y_l \in \mathcal{Y}_l} \exp(-E(y_i, y_j, y_k, y_l)) ,$$

which looks expensive (the sum has $|\mathcal{Y}_i| \cdot |\mathcal{Y}_j| \cdot |\mathcal{Y}_k| \cdot |\mathcal{Y}_l|$ terms).

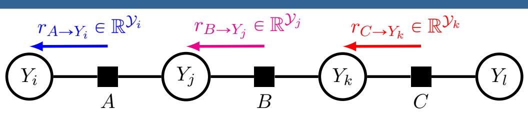
Inference on chains (cont.)



We can expand the *partition function* as

$$\begin{aligned} Z &= \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \sum_{y_k \in \mathcal{Y}_k} \sum_{y_l \in \mathcal{Y}_l} \exp(-E(y_i, y_j, y_k, y_l)) \\ &= \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \sum_{y_k \in \mathcal{Y}_k} \sum_{y_l \in \mathcal{Y}_l} \exp\Big(-\big(E_A(y_i, y_j) + E_B(y_j, y_k) + E_C(y_k, y_l)\big)\Big) \\ &= \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \sum_{y_k \in \mathcal{Y}_k} \sum_{y_l \in \mathcal{Y}_l} \exp(-E_A(y_i, y_j)) \exp(-E_B(y_j, y_k)) \exp(-E_C(y_k, y_l)) \\ &= \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \exp(-E_A(y_i, y_j)) \sum_{y_k \in \mathcal{Y}_k} \exp(-E_B(y_j, y_k)) \sum_{y_l \in \mathcal{Y}_l} \exp(-E_C(y_k, y_l)) . \end{aligned}$$

Inference on chains (cont.)

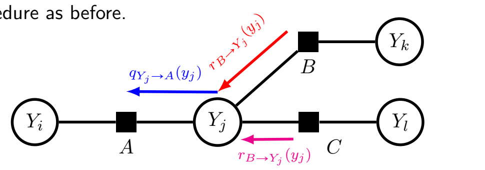


Note that we can successively *eliminate* variables, that is

$$\begin{aligned} Z &= \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \exp(-E_A(y_i, y_j)) \sum_{y_k \in \mathcal{Y}_k} \exp(-E_B(y_j, y_k)) \underbrace{\sum_{y_l \in \mathcal{Y}_l} \exp(-E_C(y_k, y_l))}_{r_{C \to Y_k}(y_k)} \\ &= \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \exp(-E_A(y_i, y_j)) \underbrace{\sum_{y_k \in \mathcal{Y}_k} \exp(-E_B(y_j, y_k)) r_{C \to Y_k}(y_k)}_{r_{B \to Y_j}(y_j)} \\ &= \sum_{y_i \in \mathcal{Y}_i} \underbrace{\sum_{y_j \in \mathcal{Y}_j} \exp(-E_A(y_i, y_j)) r_{B \to Y_j}(y_j)}_{r_{A \to Y_i}(y_i)} = \sum_{y_i \in \mathcal{Y}_i} r_{A \to Y_i}(y_i) . \end{aligned}$$

Inference on trees

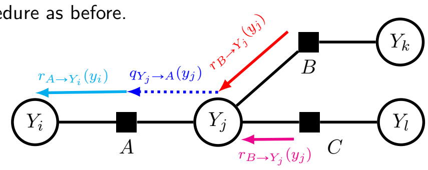
Now we are assuming a tree-structured factor graph and applying the same elimination procedure as before.



$$\begin{aligned} Z &= \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \exp(-E_A(y_i, y_j)) \underbrace{\sum_{y_k \in \mathcal{Y}_k} \exp(-E_B(y_j, y_k))}_{r_{B \to Y_j}(y_j)} \underbrace{\sum_{y_l \in \mathcal{Y}_l} \exp(-E_C(y_j, y_l))}_{r_{C \to Y_j}(y_j)} \\ &= \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \exp(-E_A(y_i, y_j)) \underbrace{r_{B \to Y_j}(y_j) r_{C \to Y_j}(y_j)}_{q_{Y_j \to A}(y_j)} \\ &= \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \exp(-E_A(y_i, y_j)) q_{Y_j \to A}(y_j) \end{aligned}$$

Inference on trees (cont.)

Now we are assuming a tree-structured factor graph and applying the same elimination procedure as before.

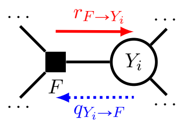


$$Z = \underbrace{\sum_{y_i\in\mathcal{Y}_i}\sum_{y_j\in\mathcal{Y}_j}\exp(-E_A(y_i,y_j))q_{Y_j\to A}(y_j)}_{r_{A\to Y_i}(y_i)}$$

$$= \sum_{y_i\in\mathcal{Y}_i} r_{A\to Y_i}(y_i)\ .$$

Messages

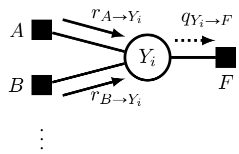
Message: pair of vectors at each factor graph edge $(i,F)\in\mathcal{E}$.



1. **Variable-to-factor** message $q_{Y_i\to F}\in\mathbb{R}^{\mathcal{Y}_i}$ is given by

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

where $M(i)=\{F\in\mathcal{F} : (i,F)\in\mathcal{E}\}$ denotes the set of factors adjacent to Y_i.



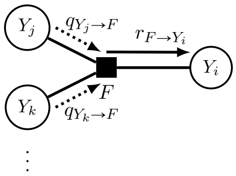
2. **Factor-to-variable** message: $r_{F\to Y_i}\in\mathbb{R}^{\mathcal{Y}_i}$.

Messages (cont.)

2. **Factor-to-variable** message $r_{F\to Y_i}\in\mathbb{R}^{\mathcal{Y}_i}$ is given by

$$r_{F\to Y_i}(y_i) = \sum_{\substack{y'_F\in\mathcal{Y}_F,\\ y'_i=y_i}}\left(\exp(-E_F(y'_F))\prod_{l\in N(F)\setminus\{i\}} q_{Y_l\to F}(y'_l)\right)\ ,$$

where $N(F)=\{i\in V : (i,F)\in\mathcal{E}\}$ denotes the set of variables adjacent to F.



Message scheduling

One can remark that the message updates depend on each other.

$$r_{F\to Y_i}(y_i) = \sum_{\substack{y'_F\in\mathcal{Y}_F,\\ y'_i=y_i}}\left(\exp(-E_F(y'_F))\prod_{l\in N(F)\setminus\{i\}} q_{Y_l\to F}(y'_l)\right) \qquad (1)$$

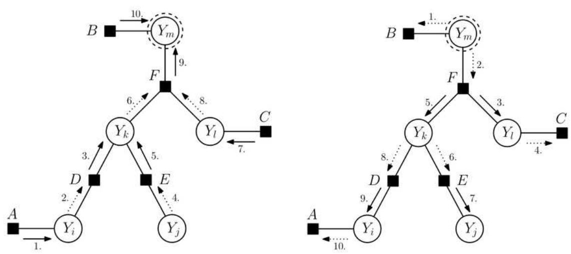
$$q_{Y_i\to F}(y_i) = \prod_{F'\in M(i)\setminus\{F\}} r_{F'\to Y_i}(y_i) \qquad (2)$$

The only messages that do not depend on previous computation are the following.

- The factor-to-variable messages in which no other variable is adjacent to the factor; then the product in (1) will be empty.
- The variable-to-factor messages in which no other factor is adjacent to the variable; then the product in (2) is empty and the message will be one.

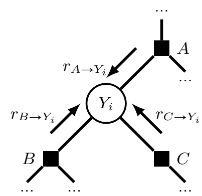
Message scheduling on trees

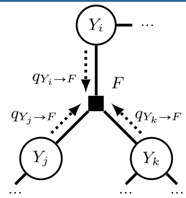
For tree-structured factor graphs there always exist at least one such message that can be computed initially, hence all the dependencies can be resolved.



1. Select one variable node as root of the tree (e.g., Y_m)
2. Compute leaf-to-root messages (e.g., by applying depth-first-search)
3. Compute root-to-leaf messages (reverse order as before)

Inference result: Z and the marginals





Partition function is evaluated at the root node (i)

$$Z = \sum_{y_i\in\mathcal{Y}_i}\prod_{F\in M(i)} r_{F\to Y_i}(y_i)\ .$$

The marginal distribution for each factor can be computed as

$$\mu_F(y_F) = p(y_F) = \frac{1}{Z}\exp(-E_F(y_F))\prod_{i\in N(F)} q_{Y_i\to F}(y_i)\ .$$

Optimality and complexity *

The ordering $\preceq$ over the vertex set V of a directed acyclic graph is called **topological ordering**, if for each $s\in V$, we have $t\preceq s$ for all $t\in\pi(s)$, where $\pi(s)$ denotes the set of all parents of node s.

Assume a tree-structured factor graph. If the messages are computed in a topological order for the sum-product algorithm, then it converges after $2|V|$ iterations and provides the exact marginals.

If $|\mathcal{Y}_i|\leqslant m$ for all $i\in V$, then the complexity of the algorithm $\mathcal{O}(|V|\cdot m^K)$, where $K=\max_{F\in\mathcal{F}}|N(F)|$.

Reminder: Assuming $f,g:\mathbb{R}\to\mathbb{R}$, the notation $f(x)=\mathcal{O}(g(x))$ means that there exists $C>0$ and $x_0\in\mathbb{R}$ such that $|f(x)|\leqslant C|g(x)|$ for all $x>x_0$.

Max-sum algorithm

Message passing for MAP inference

$$y^* \in \operatorname*{argmax}_{y\in\mathcal{Y}} p(y) = \operatorname*{argmax}_{y\in\mathcal{Y}} \frac{1}{Z}\tilde{p}(y) = \operatorname*{argmax}_{y\in\mathcal{Y}} \tilde{p}(y)\ .$$

Similar to the *sum-product algorithm* one can obtain the so-called **max-sum algorithm** to solve the above maximization.

By applying the $\ln$ function, we have

$$\begin{aligned}\ln \max_{y\in\mathcal{Y}} \tilde{p}(y) &= \max_{y\in\mathcal{Y}} \ln \tilde{p}(y)\\ &= \max_{y\in\mathcal{Y}} \ln \prod_{F\in\mathcal{F}} \exp(-E_F(y_F))\\ &= \max_{y\in\mathcal{Y}} \sum_{F\in\mathcal{F}} -E_F(y_F)\ .\end{aligned}$$

Messages

The messages become as follows

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

$$r_{F\to Y_i}(y_i) = \max_{\substack{y'_F\in\mathcal{Y}_F,\\ y'_i=y_i}} \left(-E_F(y'_F) + \sum_{l\in N(F)\setminus\{i\}} q_{Y_l\to F}(y'_l)\right).$$

The max-sum algorithm provides exact MAP inference for tree-structured factor graphs. In general, for graphs with cycles there is no guarantee for convergence.

Choosing an optimal state

First we define the **singleton max-marginal** as

$$v_i(y_i) = \max_{y'\in\mathcal{Y}, y'_i=y_i} p(y')\ .$$

The following back-tracking algorithm is applied for choosing an optimal y^*.

1. Initialize the procedure at the root node (Y_i) by choosing any $y_i^* \in \operatorname{argmax}_{y_i\in\mathcal{Y}_i} v_i(y_i)$ and set $\mathcal{I} = \{i\}$.
2. In a topological order, for each $j \in V\setminus\{i\}$ choose a configuration y_j^* at the node Y_j such that

$$y_j^* \in \operatorname*{argmax}_{y_j\in\mathcal{Y}_j} \max_{\substack{y'\in\mathcal{Y},\\ y'_j=y_j, \forall i\in\mathcal{I}\ y'_i=y_i^*}} p(y')\ ,$$

and set $\mathcal{I} = \mathcal{I}\cup\{j\}$.

Sum-product and Max-sum comparison

- Sum-product algorithm

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

$$r_{F\to Y_i}(y_i) = \sum_{\substack{y'_F\in\mathcal{Y}_F,\\ y'_i=y_i}} \left(\exp(-E_F(y'_F)) \prod_{l\in N(F)\setminus\{i\}} q_{Y_l\to F}(y'_l)\right)$$

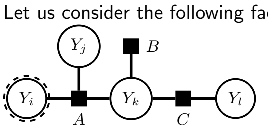
- Max-sum algorithm

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

$$r_{F\to Y_i}(y_i) = \max_{\substack{y'_F\in\mathcal{Y}_F,\\ y'_i=y_i}} \left(-E_F(y'_F) + \sum_{l\in N(F)\setminus\{i\}} q_{Y_l\to F}(y'_l)\right)$$

Example *

Let us consider the following factor graph with binary variables:



$E_A(0, y_j, y_k)$

	y_k = 0	y_k = 1
y_j = 0	1	0
y_j = 1	0	1

$E_A(1, y_j, y_k)$

	y_k = 0	y_k = 1
y_j = 0	0	-1
y_j = 1	0	0

$E_B(y_k)$

	y_k
0	1
1	0.5

$E_C(y_k, y_l)$

	y_l = 0	y_l = 1
y_k = 0	0	0.5
y_k = 1	0.5	0

Let us chose the node Y_i as root. We calculate the messages for the max-sum algorithm from leaf–to–root direction in a topological order as follows.

1. $q_{Y_l\to C}(0) = q_{Y_l\to C}(1) = 0$
2. $r_{C\to Y_k}(0) = \max_{y_l\in\{0,1\}} \{-E_c(0, y_l) + q_{Y_l\to C}(0)\} = \max_{y_l\in\{0,1\}} -E_c(0, y_l) = 0$
   $r_{C\to Y_k}(1) = \max_{y_l\in\{0,1\}} \{-E_c(1, y_l) + q_{Y_l\to C}(1)\} = \max_{y_l\in\{0,1\}} -E_c(1, y_l) = 0$
3. $r_{B\to Y_k}(0) = -1$
   $r_{B\to Y_k}(1) = -0.5$
4. $q_{Y_k\to A}(0) = r_{B\to Y_k}(0) + r_{C\to Y_k}(0) = -1 + 0 = -1$
   $q_{Y_k\to A}(1) = r_{B\to Y_k}(1) + r_{C\to Y_k}(1) = -0.5 + 0 = -0.5$

Example (cont.) *

5. $q_{Y_j\to A}(0) = q_{Y_j\to A}(1) = 0$
6. $r_{A\to Y_i}(0) = \max_{y_j, y_k\in\{0,1\}}\{-E_A(0, y_j, y_k) + q_{Y_j\to A}(y_j) + q_{Y_k\to A}(y_k)\} = -0.5$
   $r_{A\to Y_i}(1) = \max_{y_j, y_k\in\{0,1\}}\{-E_A(1, y_j, y_k) + q_{Y_j\to A}(y_j) + q_{Y_k\to A}(y_k)\} = 0.5$

In order to calculate the maximal state y^* we apply back-tracking

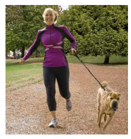
1. $y_i^* \in \operatorname{argmax}_{y_i\in\{0,1\}} r_{A\to Y_i}(y_i) = \{1\}$
2. $y_j^* \in \operatorname{argmax}_{y_j} \max_{y_j, y_k\in\{0,1\}}\{-E_A(1, y_j, y_k) + q_{Y_i\to A}(1) + q_{Y_k\to A}(y_k)\} = \{0\}$
3. $$\begin{aligned} y_k^* &\in \operatorname*{argmax}_{y_k\in\{0,1\}}\{r_{A\to Y_k}(1, 0, y_k) + r_{B\to Y_k}(y_k) + r_{C\to Y_k}(y_k)\}\\ &= \operatorname*{argmax}_{y_k\in\{0,1\}}\{-E_A(1, 0, y_k) + r_{B\to Y_k}(y_k)\} = \{1\}\end{aligned}$$
4. $y_l^* \in \operatorname{argmax}_{y_l\in\{0,1\}}\{-E_C(y_k, 1) + q_{Y_k\to C}(1)\} = \{0\}$

Therefore, the optimal state $y^* = (1, 0, 1, 0)$.

Human pose estimation

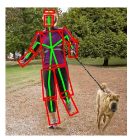
The model

The goal is to recognize an articulated object with joints connecting different parts, here it is a human body.



An object is composed of a number of rigid parts. Each part is modeled as a rectangle parameterized by (x, y, s, θ), where

- (x, y) means the **center of the rectangle**,
- $s \in [0, 1]$ is a **scaling factor**, and
- **the orientation** is given by θ.



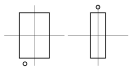
In overall, we have a four-dimensional pose space.

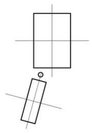
We denote the **locations** of two (connected) parts by $l_i = (x_i, y_i, s_i, \theta_i)$ and $l_j = (x_j, y_j, s_j, \theta_j)$, respectively.

The model (cont.)

An object (e.g., human body) is given by a configuration $L = (l_1, \ldots, l_n)$, where l_i specifies the location of **part** v_i. The connections encode generic relationships such as "close to", "to the left of", or more precise geometrical constraints such as ideal joint angles.

- The **location of a joint** between v_i and v_j is specified by two points (x_{ij}, y_{ij}) and (x_{ji}, y_{ji}).
- The **relative orientation** is given by θ_{ij}, which is the difference between the orientation of the two parts.



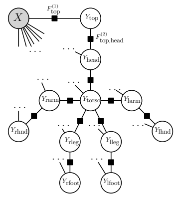


In principle, all parts depend on each other, however, tree structured model can be considered for an articulated pose.

Graphical representation

The structure is encoded by a graph $G = (V, E)$, where $V = \{v_1, \ldots, v_n\}$ corresponds to n parts, and there is an edge $(v_i, v_j) \in E$ for each pair of connected parts v_i and v_j.

We want to minimize the following energy function



$$L^* \in \operatorname*{argmin}_L \left(\sum_{i=1}^{n} m_i(l_i) + \sum_{(v_i,v_j)\in E} d_{ij}(l_i, l_j) \right) ,$$

where $m_i(l_i)$ measures the degree of mismatch when the part v_i is placed at location l_i and $d_{ij}(l_i, l_j)$ measures the degree of deformation of the model when part v_i is placed at location l_i and part v_j is placed at location l_j.

Note that MAP inference can be efficiently done by making use of *Max-sum algorithm*.

Image filters *

The **image filtering** is a technique for modifying or enhancing an image (e.g., smoothing, edge detection, sharpening). For example, the smoothing of an input signal means of removing (or filtering out) high-frequency components.

A **digital image** can be considered as a two dimensional (discretized) signal that is $f : \mathbb{Z}^2 \to \mathbb{Z}^D$. For example $D = 3$ for color images.

Here we consider **linear filtering** in which the value of an output pixel is a linear combination of the values of the pixels in the input pixel's neighborhood. In a spatially discrete setting, a linear filter is a weighted sum:

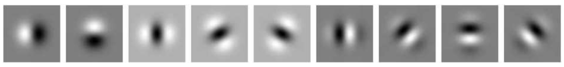
$$g(x_0, y_0) = [f * w](x_0, y_0) = \sum_{m,n} w(m, n) f(x_0 - m, y_0 - n)$$

which is also called **discrete convolution** of f and w. In practice this summation extends over a certain neighborhood. The matrix of weights $w(m, n)$ is called a **mask**.

(For more details please refer to the course of **Variational Methods for Computer Vision**.)

Unary energies *

An image patch centered at some position is represented by a vector that collects all the responses of a set of Gaussian derivative filters of different orders, orientations and scales at that point. This vector is normalized and called the **iconic index** at that position.



The unary energies are defined as

$$m_i(l_i) = -\ln \mathcal{N}(\alpha(l_i), \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) ,$$

where $\alpha(l_i)$ is the iconic index at location l_i in the image.

The parameters for each part (i.e. the mean vector $\boldsymbol{\mu}_i$ and the covariance matrix $\boldsymbol{\Sigma}_i$) can be obtained by maximum likelihood estimation for a given set of training samples.

Pairwise energies *

The pairwise energies have a special form as follows.

$$d_{ij}(l_i, l_j) = -\ln \mathcal{N}(T_{ji}(l_j) - T_{ij}(l_i), \mathbf{0}, \mathbf{D}_{ij}) ,$$

where where T_{ij}, T_{ji} and $\mathbf{D}_{ij}$ are the connection parameters

$$\begin{aligned} T_{ij}(l_i) &= (x_i', y_i', s_i, \cos(\theta_i + \theta_{ij}), \sin(\theta_i + \theta_{ij})), \\ T_{ji}(l_j) &= (x_j', y_j', s_j, \cos(\theta_j), \sin(\theta_j)), \\ \mathbf{D}_{ij} &= \mathrm{diag}(\sigma_x^2, \sigma_y^2, \sigma_s^2, 1/k, 1/k) . \end{aligned}$$

$T_{ij}(l_i)$ and $T_{ji}(l_j)$ are one-to-one mappings encoding the set of possible transformed locations.

This special form for the pairwise energies allows for matching algorithms that run in $\mathcal{O}(h')$, where h' is the number of grid locations in a discretization of the space. This results in the time complexity $\mathcal{O}(h'n)$ rather than $\mathcal{O}(h^2 n)$.

Pairwise energies (cont.) *

Let $\mathbf{R}$ be the matrix that performs a rotation of θ radians about the origin. Then,

$$\begin{bmatrix} x_i' \\ y_i' \end{bmatrix} = \begin{bmatrix} x_i \\ y_i \end{bmatrix} + s_i \mathbf{R}_{\theta_i} \begin{bmatrix} x_{ij} \\ y_{ij} \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} x_j' \\ y_j' \end{bmatrix} = \begin{bmatrix} x_j \\ y_j \end{bmatrix} + s_j \mathbf{R}_{\theta_j} \begin{bmatrix} x_{ji} \\ y_{ji} \end{bmatrix} ,$$

where (x_i, y_i), (x_j, y_j) and (x_{ij}, y_{ij}), (x_{ji}, y_{ji}) are the positions of the joints in image and local coordinates, respectively.

We assume the following joint distributions:

- $\mathcal{N}(x_i - x_j, \mathbf{0}, \sigma_x^2)$ and $\mathcal{N}(y_i - y_j, \mathbf{0}, \sigma_y^2)$ which measures the horizontal and vertical distances, respectively, between the observed joint positions.
- $\mathcal{N}(s_i - s_j, 0, \sigma_s^2)$ measures the difference in foreshortening between the two parts.
- $\mathcal{M}(\theta_i - \theta_j, \theta_{ij}, k) \propto \exp(k \cos(\theta_i - \theta_j - \theta_{ij}))$ measures the difference between the relative angle of the two parts and the ideal relative angle.

These parameters can be also obtained by maximum likelihood estimation.

Inference

MAP inference provides a single (best) prediction of the overall pose. The factor–to–varaible messages can be written as

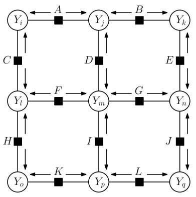
$$r_{F \to v_i}(l_i) = \max_{\substack{(l_i', l_j') \in \mathcal{Y}_F, \\ l_i' = l_i}} \left(\exp(-m_i(l_i') - d_{ij}(l_i', l_j')) + \sum_{k \in N(F) \setminus \{i\}} q_{v_k \to F}(l_k') \right) .$$

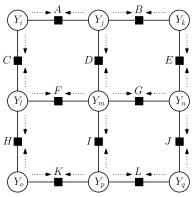
$\mathcal{Y}$ could be quite large ($\approx 1.5M$ possible states), hence $\mathcal{Y}_i \times \mathcal{Y}_j$ is too big. However a special form of pairwise energies is used, so that a message can be calculated in $\mathcal{O}(|\mathcal{Y}_i|)$ time.

Loopy belief propagation

Message passing in cyclic graphs

When the graph has cycles, then there is no well-defined *leaf–to–root* order. However, one can apply message passing on cyclic graphs, which results in **loopy belief propagation**.





1. Initialize all messages as constant 1
2. Pass factor–to–variables and variables–to–factor messages alternately until convergence
3. Upon convergence, treat **beliefs** μ_F as approximate marginals

Messages

The factor–to–variable messages $r_{F\to Y_i}$ remain well-defined and are computed as before.

$$r_{F\to Y_i}(y_i) = \sum_{\substack{y'_F\in\mathcal{Y}_F,\\ y'_i=y_i}} \left(\exp(-E_F(y'_F)) \prod_{j\in N(F)\setminus\{i\}} q_{Y_j\to F}(y'_j)\right)$$

The variable–to–factor messages are normalized at every iteration as follows:

$$q_{Y_i\to F}(y_i) = \frac{\prod_{F'\in M(i)\setminus\{F\}} r_{F'\to Y_i}(y_i)}{\sum_{y'_i\in\mathcal{Y}_i}\prod_{F'\in M(i)\setminus\{F\}} r_{F'\to Y_i}(y'_i)} .$$

In case of tree structured graphs, in the sum–product algorithm these normalization constants are equal to 1, since the marginal distributions, calculated in each iteration, are exact.

Beliefs

The approximate marginals, i.e. beliefs, are computed as before but now a factor-specific normalization constant z_F is also used.

The factor marginals are given by

$$\mu_F(y_F) = \frac{1}{z_F}\exp(-E_F(y_F)) \prod_{i\in N(F)} q_{Y_i\to F}(y_i) ,$$

where the factor specific constant is given by

$$z_F = \sum_{y_F\in\mathcal{Y}_F} \exp(-E_F(y_F)) \prod_{i\in N(F)} q_{Y_i\to F}(y_i) .$$

Beliefs (cont.)

In addition to the factor marginals the algorithm also computes the variable marginals in a similar fashion.

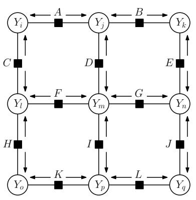
$$\mu_i(y_i) = \frac{1}{z_i}\prod_{F'\in M(i)} r_{F'\to Y_i}(y_i) ,$$

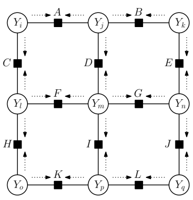
where the normalizing constant is given by

$$z_i = \sum_{y_i\in\mathcal{Y}_i}\prod_{F'\in M(i)} r_{F'\to Y_i}(y_i) .$$

Since the local normalization constant z_F differs at each factor for loopy belief propagation, the exact value of the normalizing constant Z cannot be directly calculated. Instead, an approximation to the log partition function can be computed.

Remarks on loopy belief propagation





Loopy belief propagation is very popular, but has some problems:

- It might not converge (e.g., it can oscillate).
- Even if it does, the computed probabilities are only *approximate*.
- If there is a single cycle only in the graph, then it converges.

Literature *

- Sebastian Nowozin and Christoph H. Lampert. **Structured Prediction and Learning in Computer Vision**. In *Foundations and Trends in Computer Graphics and Vision*, Volume 6, Number 3-4. Note: Chapter 3.
- Daphne Koller and Nir Friedman. **Probabilistic Graphical Models: Principles and Techniques**. The MIT Press, 2009. Note: Chapters 9,10 and 13.
- Christopher Bishop. **Pattern Recognition and Machine Learning**. Springer, 2006. Note: Chapter 8.
- Judea Pearl. **Probabilistic Reasoning in Intelligent Systems: Network of Plausible Inference**. Morgan Kaufmann, 1988.
- Pedro F. Felzenszwalb and Daniel P. Huttenlocher. **Pictorial Structures for Object Recognition**. *International Journal of Computer Vision*, Vol. 61(1), pp. 55-79, January 2005.