Energy Minimization

Raquel Urtasun

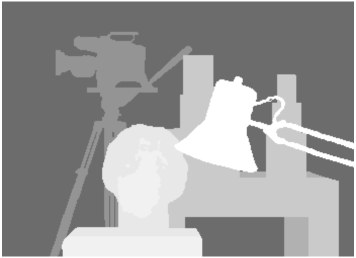
TTI Chicago

Feb 26, 2013

Disparity Estimation

- DSI: Disparity image

Scene



Ground truth

Stereo Estimation Methods

- Local methods
- Grow and seed methods: use a few good correspondences and grow the estimation from them
- Adaptive Window methods (AW)
- Global methods: define a Markov random field over
  - Pixel-level
  - Fronto-parallel planes
  - Slanted planes

MRFs on pixels

- The energy is defined as

$$E(d_1, \cdots, d_n) = \sum_i C(d_i) + \sum_i \sum_{j \in \mathcal{N}(j)} C(d_i, d_j)$$

where $x_i \in \{0, 1, \cdots, D\}$ represents a variable for the disparity of the $i-$th pixel

- This optimization is in general NP-hard.

MRFs on pixels

- The energy is defined as

$$E(d_1, \cdots, d_n) = \sum_i C(d_i) + \sum_i \sum_{j \in \mathcal{N}(j)} C(d_i, d_j)$$

where $x_i \in \{0, 1, \cdots, D\}$ represents a variable for the disparity of the $i-$th pixel

- This optimization is in general NP-hard.
- Global optima can be obtained in a few cases.

MRFs on pixels

- The energy is defined as

$$E(d_1, \cdots, d_n) = \sum_i C(d_i) + \sum_i \sum_{j \in \mathcal{N}(j)} C(d_i, d_j)$$

  where $x_i \in \{0, 1, \cdots, D\}$ represents a variable for the disparity of the $i-$th pixel

- This optimization is in general NP-hard.
- Global optima can be obtained in a few cases.

Semiglobal block matching [Hirschmueller08]

- The energy is defined as

$$E(d_1, \cdots, d_n) = \sum_i C(d_i) + \sum_i \sum_{j \in \mathcal{N}(j)} C(d_i, d_j)$$

with the following pairwise term

$$C(d_i, d_j) = \begin{cases} 0 & \text{if } d_i = d_j \\ \lambda_1 & \text{if } |d_i - d_j| = 1 \\ \lambda_2 & \text{otherwise} \end{cases}$$

- It computes the costs in each direction

$$D_j(\mathbf{p}; d) = C(\mathbf{p}; d) + min_{d'}\{D(\mathbf{p} - \mathbf{j}, d') + \rho_d(d - d')\}$$

Semiglobal block matching [Hirschmueller08]

- The energy is defined as

$$E(d_1, \cdots, d_n) = \sum_i C(d_i) + \sum_i \sum_{j \in \mathcal{N}(j)} C(d_i, d_j)$$

with the following pairwise term

$$C(d_i, d_j) = \begin{cases} 0 & \text{if } d_i = d_j \\ \lambda_1 & \text{if } |d_i - d_j| = 1 \\ \lambda_2 & \text{otherwise} \end{cases}$$

- It computes the costs in each direction

$$D_j(\mathbf{p}; d) = C(\mathbf{p}; d) + min_{d'}\{D(\mathbf{p} - \mathbf{j}, d') + \rho_d(d - d')\}$$

- And aggregate the costs

$$D(\mathbf{p}; d) = \sum_j L_j(\mathbf{p}, d)$$

Semiglobal block matching [Hirschmueller08]

- The energy is defined as

$$E(d_1, \cdots, d_n) = \sum_i C(d_i) + \sum_i \sum_{j \in \mathcal{N}(j)} C(d_i, d_j)$$

with the following pairwise term

$$C(d_i, d_j) = \begin{cases} 0 & \text{if } d_i = d_j \\ \lambda_1 & \text{if } |d_i - d_j| = 1 \\ \lambda_2 & \text{otherwise} \end{cases}$$

- It computes the costs in each direction

$$D_j(\mathbf{p}; d) = C(\mathbf{p}; d) + min_{d'}\{D(\mathbf{p} - \mathbf{j}, d') + \rho_d(d - d')\}$$

- And aggregate the costs

$$D(\mathbf{p}; d) = \sum_j L_j(\mathbf{p}, d)$$

- Then do winner take all

Semiglobal block matching [Hirschmueller08]

- The energy is defined as

$$E(d_1, \cdots, d_n) = \sum_i C(d_i) + \sum_i \sum_{j \in \mathcal{N}(j)} C(d_i, d_j)$$

with the following pairwise term

$$C(d_i, d_j) = \begin{cases} 0 & \text{if } d_i = d_j \\ \lambda_1 & \text{if } |d_i - d_j| = 1 \\ \lambda_2 & \text{otherwise} \end{cases}$$

- It computes the costs in each direction

$$D_j(\mathbf{p}; d) = C(\mathbf{p}; d) + min_{d'}\{D(\mathbf{p} - \mathbf{j}, d') + \rho_d(d - d')\}$$

- And aggregate the costs

$$D(\mathbf{p}; d) = \sum_j L_j(\mathbf{p}, d)$$

- Then do winner take all

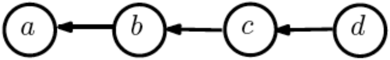
Inference in trees

- Given distribution $p(y_1, \cdots, y_n)$
- Inference: computing functions of the distribution
  - mean
  - marginal
  - conditionals
- Marginal inference in singly-connected graph (trees)
- Later: extensions to loopy graphs

[Source: P. Gehler]

Variable Elimination

- Consider Markov chain ($a, b, c, d \in \{0, 1\}$)



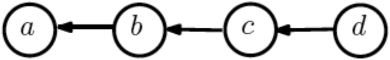
with distribution

$$p(a, b, c, d) = p(a \mid b)p(b \mid c)p(c \mid d)p(d)$$

- Task: compute the marginal $p(a)$

[Source: P. Gehler]

Variable Elimination



$$
\begin{aligned}
p(a) &= \sum_{b,c,d} p(a,b,c,d) \\
&= \sum_{b,c,d} p(a \mid b) p(b \mid c) p(c \mid d) p(d)
\end{aligned}
$$

- Naive: $2 \times 2 \times 2 = 8$ states to sum over
- Re-order summation:

$$p(a) = \sum_{b,c} p(a \mid b) p(b \mid c) \underbrace{\sum_{d} p(c \mid d) p(d)}_{\gamma_d(c)}$$

[Source: P. Gehler]

Variable Elimination

$$p(a) = \sum_{b,c} p(a \mid b)p(b \mid c) \underbrace{\sum_{d} p(c \mid d)p(d)}_{\gamma_d(c)}$$

$$p(a) = \sum_{b} p(a \mid b) \underbrace{\sum_{c} p(b \mid c)\gamma_d(c)}_{\gamma_c(b)}$$

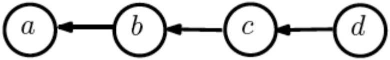
$$p(a) = \sum_{b} p(a \mid b)\gamma_c(b)$$

- We need $2 + 2 + 2 = 6$ calculations
- For a chain of length T scale linearly $n * 2$, cf naive approach 2^n

[Source: P. Gehler]

Finding Conditional Marginals

- Again:



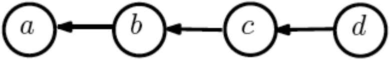
$$p(a, b, c, d) = p(a \mid b)p(b \mid c)p(c \mid d)p(d)$$

- Now find $p(d \mid a)$

$$\begin{aligned} p(d \mid a) &\propto \sum_{b,c} p(a \mid b)p(b \mid c)p(c \mid d)p(d) \\ &= \sum_{c} \underbrace{\sum_{b} p(a \mid b)p(b \mid c)}_{\gamma_b(c)} p(c \mid d)p(d) \\ &\stackrel{def}{=} \gamma_c(d) \text{ not a distribution} \end{aligned}$$

[Source: P. Gehler]

Finding Conditional Marginals



- Found that

$$p(d \mid a) = k\gamma_c(d)$$

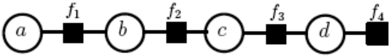
- and since $\sum_d p(d \mid a) = 1$

$$k = \frac{1}{\sum_d \gamma_c(d)}$$

- Again $\gamma_c(d)$ is not a distribution (but a message)

[Source: P. Gehler]

Now with factor graphs



$$p(a,b,c,d) = f_1(a,b) f_2(b,c) f_3(c,d) f_4(d)$$

$$\begin{aligned} p(a,b,c) &= \sum_d p(a,b,c,d) \\ &= f_1(a,b) f_2(b,c) \underbrace{\sum_d f_3(c,d) f_4(d)}_{\mu_{d\rightarrow c}(c)} \end{aligned}$$

$$p(a,b) = \sum_c p(a,b,c) = f_1(a,b) \underbrace{\sum_c f_2(b,c) \mu_{d\rightarrow c}(c)}_{\mu_{c\rightarrow b}(b)}$$

[Source: P. Gehler]

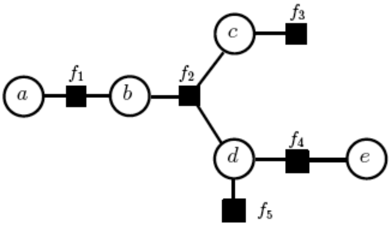
Inference in Chain Structured Factor Graphs

- Simply recurse further
- $\gamma_{m \to n}(n)$ carries the information beyond m
- We did not need the factors in general (next) we will see that making a distinction is helpful

[Source: P. Gehler]

General singly-connected factor graphs I

- Now consider a branching graph:



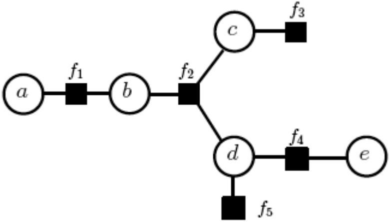
with factors

$$f_1(a,b)f_2(b,c,d)f_3(c)f_4(d,e)f_5(d)$$

- For example: find marginal $p(a,b)$

[Source: P. Gehler]

General singly-connected factor graphs II

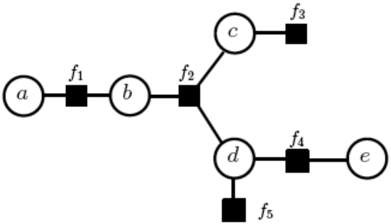


$$p(a,b) = f_1(a,b) \underbrace{\sum_{c,d,e} f_2(b,c,d) f_3(c) f_5(d) f_4(d,e)}_{\mu_{f_2 \to b}(b)}$$

$$\mu_{f_2 \to b}(b) = \sum_{c,d} f_2(b,c,d) \underbrace{f_3(c)}_{\mu_{c \to f_2}(c)} \underbrace{f_5(d) \sum_{e} f_4(d,e)}_{\mu_{d \to f_2}(d)}$$

[Source: P. Gehler]

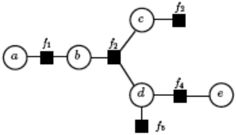
General singly-connected factor graphs III



$$\mu_{d \to f_2}(d) = \underbrace{f_5(d)}_{\mu_{f_5 \to d}(d)} \underbrace{\sum_e f_4(d,e)}_{\mu_{f_4 \to d}(d)}$$

[Source: P. Gehler]

General singly-connected factor graphs IV



- If we want to compute the marginal $p(a)$:

$$p(a) = \underbrace{\sum_b f_1(a,b)\mu_{f_2 \to b}(b)}_{\mu_{f_1 \to a}(a)}$$

- which we could also view as

$$p(a) = \sum_b f_1(a,b) \underbrace{\mu_{f_2 \to b}(b)}_{\mu_{b \to f_1}(b)}$$

[Source: P. Gehler]

Summary

- Once computed, messages can be re-used
- All marginals $p(c), p(d), p(c, d), \cdots$ can be written as a function of messages
- We need an algorithm to compute all messages: Sum-Product algorithm

[Source: P. Gehler]

Sum-product algorithm overview

- Algorithm to compute all messages efficiently, assuming the graph is singly-connected
- It can be used to compute any desired marginals
- Also known as belief propagation (BP)

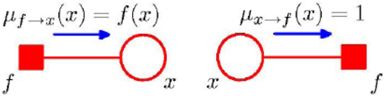
The algorithm is composed of

1. Initialization
2. Variable to Factor message
3. Factor to Variable message

[Source: P. Gehler]

1. Initialization

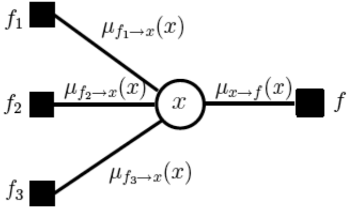
- Messages from extremal (simplical) node factors are initialized to the factor (left)
- Messages from extremal (simplical) variable nodes are set to unity (right)



[Source: P. Gehler]

2. Variable to Factor message

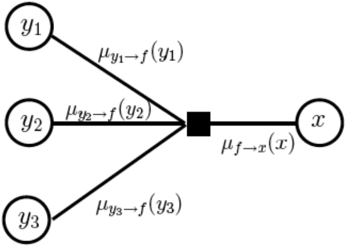
$$\mu_{x \to f}(x) = \prod_{g \in \{\mathrm{ne}(x) \setminus f\}} \mu_{g \to x}(x)$$



[Source: P. Gehler]

- We sum over all states in the set of variables
- This explains the name for the algorithm (sum-product)

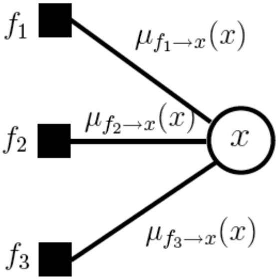
$$\mu_{f\to x}(x) = \sum_{y \in \mathcal{X}_f \backslash x} \phi_f(\mathcal{X}_f) \prod_{y \in \{\text{ne}(f)\backslash x\}} \mu_{y\to f}(y)$$



[Source: P. Gehler]

Marginal computation

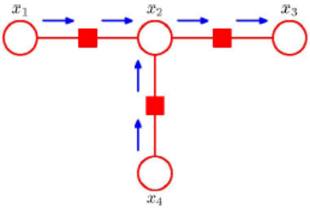
$$p(x) \propto \prod_{f \in \mathrm{ne}(x)} \mu_{f \to x}(x)$$



[Source: P. Gehler]

Message Ordering

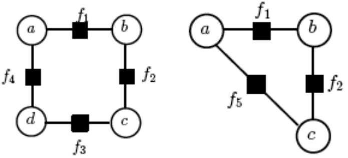
- Messages depend on previous computed messages
- Only extremal nodes/factors do not depend on other messages
- To compute all messages in the graph
  1. leaf-to-root: (pick root node, compute messages pointing towards root)
  2. root-to-leave: (compute messages pointing away from root)



[Source: P. Gehler]

Problems with loops

- Marginalizing over d introduces new link (changes graph structure – in contrast to singly connected graphs)



$$p(a,b,c,d) = f_1(a,b) f_2(b,c) f_3(c,d) f_4(d,a)$$

and marginal

$$p(a,b,c) = f_1(a,b) f_2(b,c) \underbrace{\sum_d f_3(c,d) f_4(d,a)}_{f_5(a,c)}$$

[Source: P. Gehler]

What to infer?

- Mean

$$\mathbb{E}_{p(x)}[x] = \sum_{x \in \mathcal{X}} x p(x)$$

- Mode

$$x^* = \operatorname*{argmax}_{x \in \mathcal{X}} p(x)$$

- Conditional Distributions

$$p(x_i, x_j \mid x_k, x_l) \text{or} p(x_i \mid x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$$

- Max-Marginals

$$x_i^* = \operatorname*{argmax}_{x_i \in \mathcal{X}_i} p(x_i) = \cdots dx_n \operatorname*{argmax}_{x_i \in \mathcal{X}_i} \int_{(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)} p(x) dx_1$$

[Source: P. Gehler]

Computing the Partition Function

- The partition function ($p(x) = \frac{1}{Z} \prod_f \Phi_f(\mathcal{X}_f)$) (normalization constant) Z can be computed after the leaf-to-root step (no need for the root-to-leaf step) (choose any $x \in \mathcal{X}$)

$$
\begin{aligned}
Z &= \sum_{\mathcal{X}} \prod_f \phi_f(\mathcal{X}_f) && (10)\\
&= \sum_x \sum_{\mathcal{X}\setminus\{x\}} \prod_{f \in \mathrm{ne}(x)} \prod_{f \notin \mathrm{ne}(x)} \phi_f(\mathcal{X}_f) && (11)\\
&= \sum_x \prod_{f \in \mathrm{ne}(x)} \sum_{\mathcal{X}\setminus\{x\}} \prod_{f \notin \mathrm{ne}(x)} \phi_f(\mathcal{X}_f) && (12)\\
&= \sum_x \prod_{f \in \mathrm{ne}(x)} \mu_{f \to x}(x) && (13)
\end{aligned}
$$

[Source: P. Gehler]

Log Messages

- In large graphs, messages may become very small
- Work with log-messages instead $\lambda = \log \mu$
- Variable-to-factor messages

$$\mu_{x \to f}(x) = \prod_{g \in \{\text{ne}(x) \setminus f\}} \mu_{g \to x}(x)$$

then becomes

$$\lambda_{x \to f}(x) = \sum_{g \in \{\text{ne}(x) \setminus f\}} \lambda_{g \to x}(x)$$

[Source: P. Gehler]

Log Messages

- Work with log-messages instead $\lambda = \log \mu$
- Factor-to-Variable messages

$$\mu_{f \to x}(x) = \sum_{y \in \mathcal{X}_f \setminus x} \Phi_f(\mathcal{X}_f) \prod_{y \in \{\mathrm{ne}(f) \setminus x\}} \mu_{y \to f}(y) \tag{16}$$

then becomes

$$\lambda_{f \to x}(x) = \log \left(\sum_{y \in \mathcal{X}_f \setminus x} \Phi(\mathcal{X}_f) \exp \left[\sum_{y \in \{\mathrm{ne}(f) \setminus x\}} \lambda_{y \to f}(y) \right] \right) \tag{17}$$

[Source: P. Gehler]

Trick

- Log-Factor-to-Variable Message:

$$\lambda_{f \to x}(x) = \log \sum_{y \in \mathcal{X}_f \setminus x} \Phi_f(\mathcal{X}_f) \exp \sum_{y \in \{\mathrm{ne}(f) \setminus x\}} \lambda_{y \to f}(y) \quad (18)$$

- large numbers lead to numerical instability
- Use the following equality

$$\log \sum_i \exp(v_i) = \alpha + \log \sum_i \exp(v_i - \alpha) \quad (19)$$

- With $\alpha = \max \lambda_{y \to f}(y)$

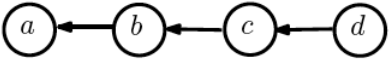
[Source: P. Gehler]

Finding the maximal state: Max-Product

- For a given distribution $p(x)$ find the most likely state:

$$x^* = \underset{x_1,\ldots,x_n}{\operatorname{argmax}}\, p(x_1, \ldots, x_n)$$

- Again use factorization structure to distribute the maximisation to local computations
- Example: chain



$$f(x_1, x_2, x_3, x_4) = \phi(x_1, x_2)\phi(x_2, x_3)\phi(x_3, x_1)$$

[Source: P. Gehler]

Be careful: not maximal marginal states!

- The most likely state

$$x^* = \operatorname*{argmax}_{x_1,\ldots,x_n} p(x_1, \ldots, x_n)$$

  does not need to be the one for which the marginals are maximized:

- For all $i = 1, \ldots, n$

$$x_i^* = \operatorname*{argmax}_{x_i} p(x_i)$$

- Example:

	$x = 0$	$x = 1$
$y = 0$	0.3	0.4
$y = 1$	0.3	0.0

[Source: P. Gehler]

Example chain

$$\begin{aligned}
\max_x f(x) &= \max_{x_1,x_2,x_3,x_4} \phi(x_1,x_2)\phi(x_2,x_3)\phi(x_3,x_4) \\
&= \max_{x_1,x_2,x_3} \phi(x_1,x_2)\phi(x_2,x_3) \underbrace{\max_{x_4} \phi(x_3,x_4)}_{\gamma(x_3)} \\
&= \max_{x_1,x_2} \phi(x_1,x_2) \underbrace{\max_{x_3} \phi(x_2,x_3)\gamma(x_3)}_{\gamma(x_2)} \\
&= \max_{x_1} \underbrace{\max_{x_2} \phi(x_1,x_2)\gamma(x_2)}_{\gamma(x_1)} \\
&= \max_{x_1} \gamma(x_1)
\end{aligned}$$

[Source: P. Gehler]

Example chain

- Once computed the messages ($\gamma(\cdot)$) find the optimal values

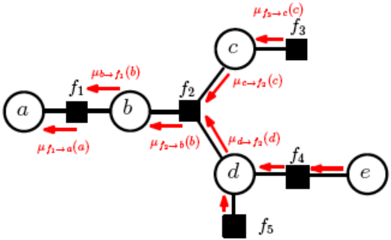
$$
\begin{aligned}
x_1^* &= \operatorname*{argmax}_{x_1} \gamma(x_1) \\
x_2^* &= \operatorname*{argmax}_{x_2} \phi(x_1^*, x_2)\gamma(x_2) \\
x_3^* &= \operatorname*{argmax}_{x_3} \phi(x_2^*, x_3)\gamma(x_3) \\
x_4^* &= \operatorname*{argmax}_{x_4} \phi(x_3^*, x_4)\gamma(x_4)
\end{aligned}
$$

- this is called backtracking (an application of dynamic programming)
- can choose arbitrary start point

[Source: P. Gehler]

Trees

- Spot the messages:



$$\begin{aligned}\max_x f(x) &= \max_{a,b,c,d,e} f_1(a,b) f_2(b,c,d) f_3(c) f_4(d,e) f_5(d) \\ &= \max_a \mu_{f_2 \to a}(a)\end{aligned}$$

[Source: P. Gehler]

Max-Product Algorithm

Pick any variable as root and

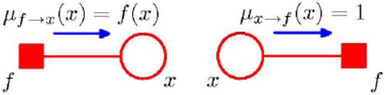
1. Initialisation (same as sum-product)
2. Variable to Factor message (same as sum-product)
3. Factor to Variable message

Then compute the maximal state

[Source: P. Gehler]

1. Initialization

- Messages from extremal node factors are initialized to the factor
- Messages from extremal variable nodes are set to unity



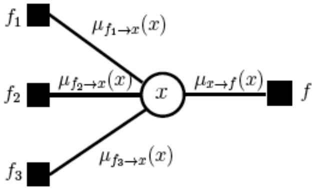
- Same as sum product

[Source: P. Gehler]

2. Variable to Factor message

- Same as for sum-product

$$\mu_{x \to f}(x) = \prod_{g \in \{\text{ne}(x) \setminus f\}} \mu_{g \to x}(x)$$

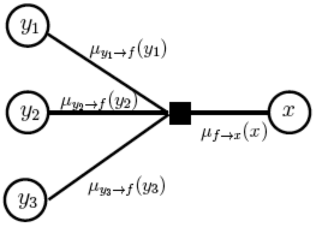


[Source: P. Gehler]

3. Factor to Variable message

- Different message than in sum-product
- This is now a max-product

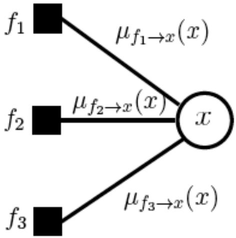
$$\mu_{f \to x}(x) = \max_{y \in \mathcal{X}_f \backslash x} \phi_f(\mathcal{X}_f) \prod_{y \in \{\mathrm{ne}(f) \backslash x\}} \mu_{y \to f}(y)$$



[Source: P. Gehler]

Maximal state of Variable

$$x^* = \underset{x}{\operatorname{argmax}} \prod_{f \in \text{ne}(x)} \mu_{f \to x}(x)$$



- This does not work with loops
- Same problem as the sum product algorithm

[Source: P. Gehler]

Algorithm 1: Belief Propagation on Trees

$(\log Z, \mu) = \text{BeliefPropagation}(V, \quad F, E, E)$
Input:
  (V, F, E), tree-structured factor graph,
  E, energies E_F for all $F \quad F$.
Output:
  $\log Z$, log partition function of $p(y)$,
  μ, marginal distributions μ_F for all $F \quad F$.
Algorithm:
Fix an element of V arbitrarily as tree root
Compute leaf-to-root order R as sequence of directed
  edges of E
for $i = 1, \ldots, |R|$ **do**
  if $(v, F) = R(i)$ is variable-to-factor edge **then**
    Compute $q_{Y_i \to F}$ using (3.2)
  else
    $(F, v) = R(i)$ is factor-to-variable edge
    Compute $r_{F \to Y_i}$ using (3.3)
  end if
end for
Compute $\log Z$ by (3.4)
Compute root-to-leaf order $R' = \text{reverse}(R)$
for $i = 1, \ldots, |R'|$ **do**
  if $(v, F) = R'(i)$ is variable-to-factor edge **then**
    Compute $q_{Y_i \to F}$ using (3.2)
    Compute μ_F using (3.5)
  else
    $(F, v) = R'(i)$ is factor-to-variable edge
    Compute $r_{F \to Y_i}$ using (3.3)
    Compute $p(y_i)$ using (3.6)
  end if
end for

Dealing with loops

- Keep on doing this iterations, i.e., loopy BP
- The problem with loopy BP is that it is not guaranteed to converge
- Message-passing algorithms based on LP relaxations have been developed
- These methods are guaranteed to converge
- Perform much better in practice

Algorithm 2: Loopy Belief Propagation (sum–product)

$(\log Z, \mu) = \textsc{SumProductLoopyBP}(V, \mathcal{F}, \mathcal{E}, E, \varepsilon, T)$

Input:

    $(V, \mathcal{F}, \mathcal{E})$, factor graph,

    E, energies E_F for all $F \in \mathcal{F}$,

    ε, convergence tolerance,

    T, maximum number of iterations.

Output:

    $\log Z$, approximate log partition function of $p(y)$,

    μ, approximate marginal distributions μ_F for all $F \in \mathcal{F}$.

Algorithm:

$q_{Y_i \to F}(y_i) \leftarrow 0$, for all $(i, F) \in \mathcal{E}, y_i \in \mathcal{Y}_i$

$\mu_F(y_F) \leftarrow 0$, for all $F \in \mathcal{F}, y_F \in \mathcal{Y}_F$

for $t = 1, \dots, T$ **do**

    for $(v, F) \in \mathcal{F}$ **do**

        for $y_i \in \mathcal{Y}_i$ **do**

            Compute $r_{F \to Y_i}(y_i)$ using (3.3)

        end for

    end for

    for $(v, F) \in \mathcal{F}$ **do**

        for $y_i \in \mathcal{Y}_i$ **do**

            Compute $q_{Y_i \to F}(y_i)$ using (3.9) to (3.11)

        end for

    end for

    Compute approximate marginals μ' using (3.12) to (3.17)

    $\mathrm{u} \leftarrow \|\mu' - \mu\|_\infty$ {Measure change in beliefs}

    $\mu \leftarrow \mu'$

    if $u \le \varepsilon$ **then**

        break {Converged}

    end if

end for

Compute $\log Z$ using (3.18)

Global Minimization Techniques

Ways to get an approximate solution typically

- Dynamic programming approximations
- Sampling
- Simulated annealing
- Graph-cuts: imposes restrictions on the type of pairwise cost functions
- Message passing: iterative algorithms that pass messages between nodes in the graph. Which graph?

Inference with graph cuts

Submodular Functions

- A Pseudo-boolean function $f : \{0,1\}^n \rightarrow \Re$ is **submodular** if

$$f(A) + f(B) \geq \underbrace{f(A \vee B)}_{OR} + \underbrace{f(A \wedge B)}_{AND} \qquad \forall A, B \in \{0,1\}^n$$

- Example: $n = 2$, $A = [1,0]$, $B = [0,1]$

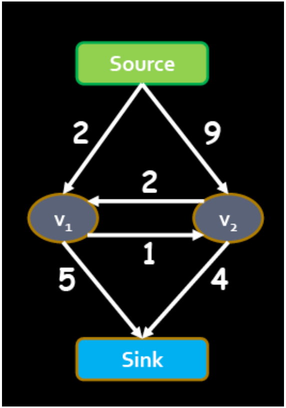
$$f([1,0]) + f([0,1]) \geq f([1,1]) + f([0,0])$$

- Sum of submodular functions is submodular $\rightarrow$ Easy to proof.
- Some energies in computer vision can be submodular

Minimizing submodular Functions

- Pairwise submodular functions can be transformed to st-mincut/max-flow [Hammer, 65].
- Very low running time $\sim \mathcal{O}(n)$

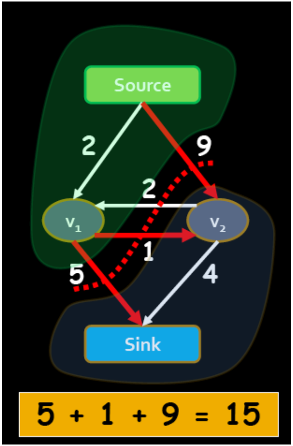
- Suppose we have a graph $G = \{V, E, C\}$, with vertices V, Edges E and costs C.



[Source: P. Kohli]

The ST-mincut problem

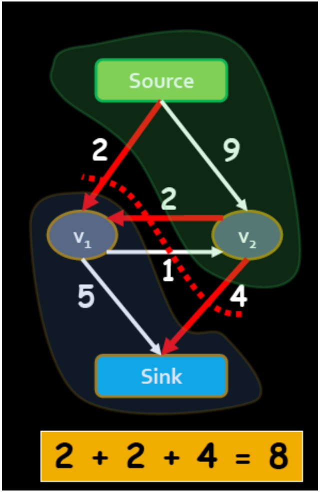
- An st-cut (S,T) divides the nodes between source and sink.
- The cost of a st-cut is the sum of cost of all edges going from S to T



[Source: P. Kohli]

The ST-mincut problem

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

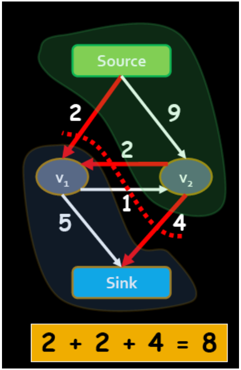


[Source: P. Kohli]

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

Equivalent (transformable)

$$E(x) = \sum_i c_i x_i + \sum_{i,j} c_{ij} x_i(1 - x_j) \qquad c_{ij} \geq 0$$

[Source: P. Kohli]

How are they equivalent?

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

$$\begin{array}{cc} & x_j \\ & \begin{array}{cc} 0 & 1 \end{array} \\ x_i \begin{array}{c} 0 \\ 1 \end{array} & \begin{bmatrix} A & B \\ C & D \end{bmatrix} \end{array} = A + \begin{array}{cc} & \begin{array}{cc} 0 & 1 \end{array} \\ \begin{array}{c} 0 \\ 1 \end{array} & \begin{bmatrix} 0 & 0 \\ C\text{-}A & C\text{-}A \end{bmatrix} \end{array} + \begin{array}{cc} & \begin{array}{cc} 0 & 1 \end{array} \\ \begin{array}{c} 0 \\ 1 \end{array} & \begin{bmatrix} 0 & D\text{-}C \\ 0 & D\text{-}C \end{bmatrix} \end{array} + \begin{array}{cc} & \begin{array}{cc} 0 & 1 \end{array} \\ \begin{array}{c} 0 \\ 1 \end{array} & \begin{bmatrix} 0 & B{+}C{-}A{-}D \\ 0 & 0 \end{bmatrix} \end{array}$$

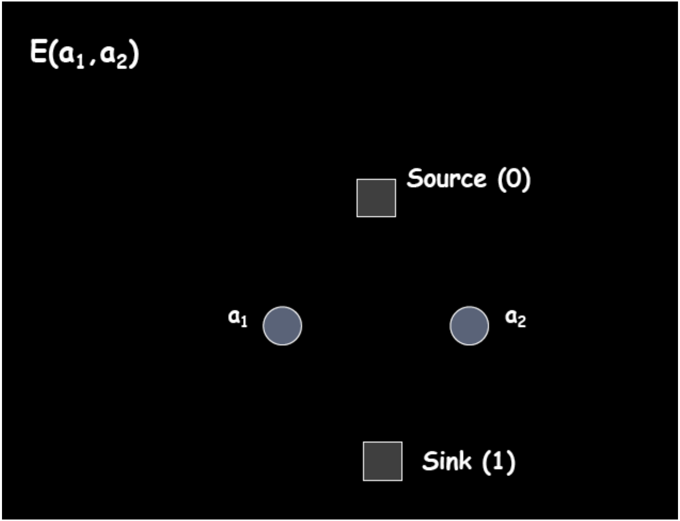
if x_1=1 add C-A **if x_2 = 1 add D-C**

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

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

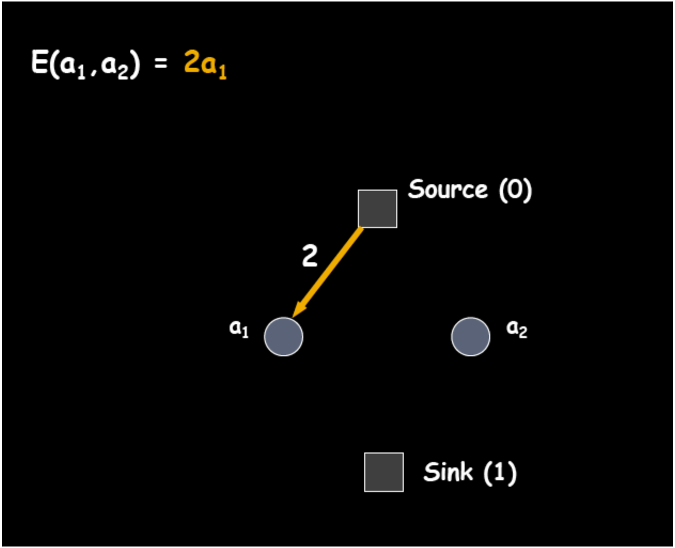
[Source: P. Kohli]

Graph Construction



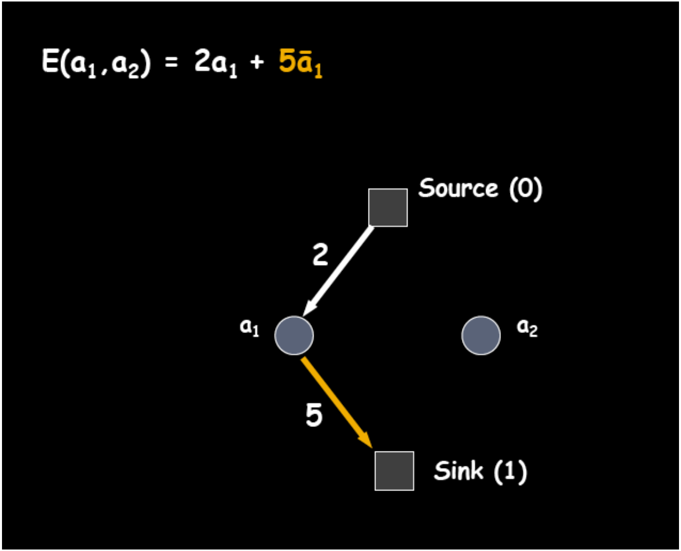
[Source: P. Kohli]

Graph Construction



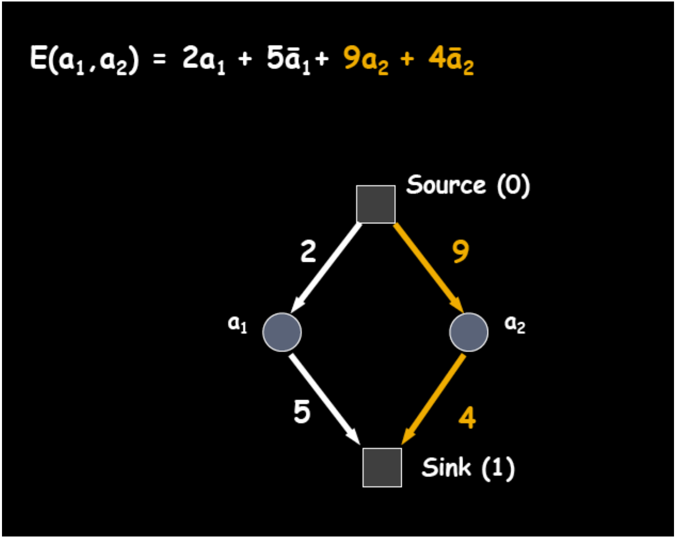
[Source: P. Kohli]

Graph Construction



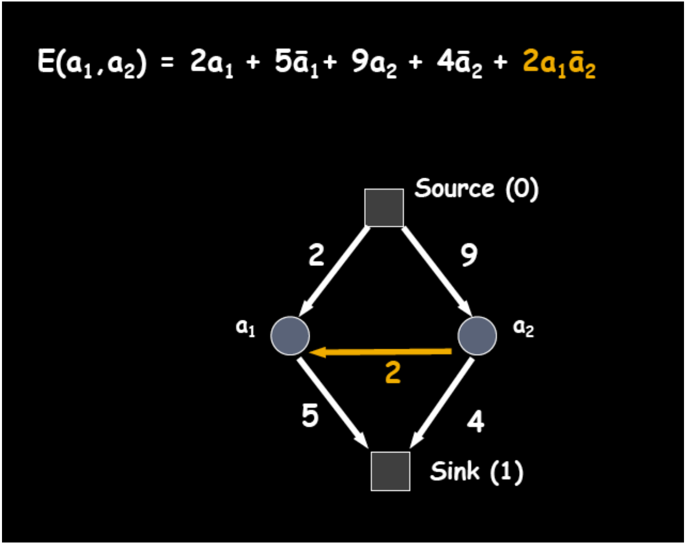
[Source: P. Kohli]

Graph Construction



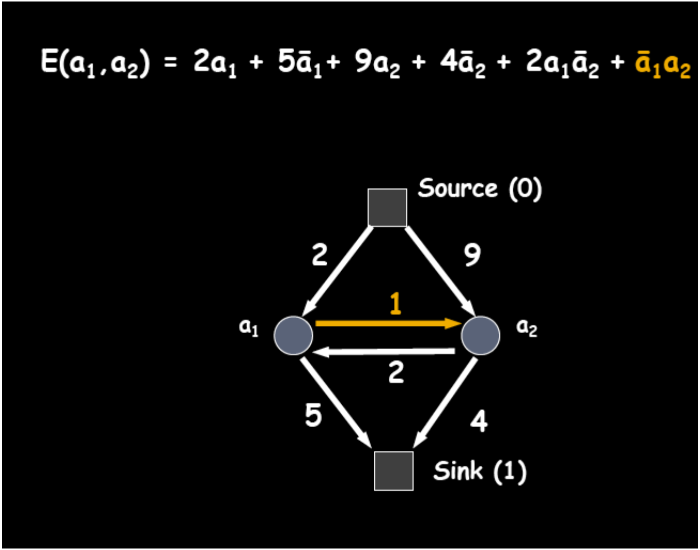
[Source: P. Kohli]

Graph Construction



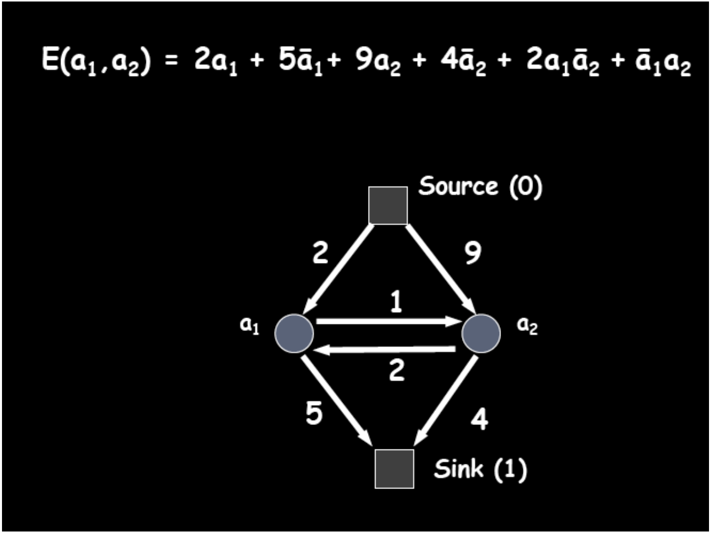
[Source: P. Kohli]

Graph Construction



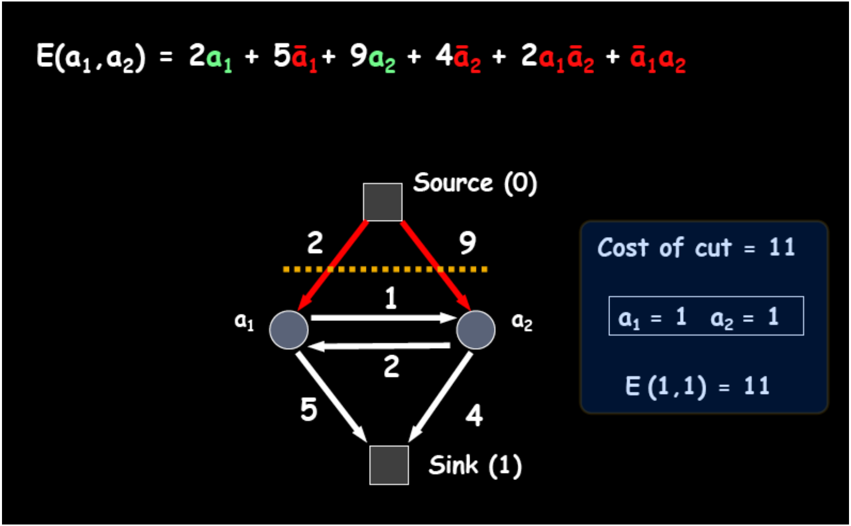
[Source: P. Kohli]

Graph Construction



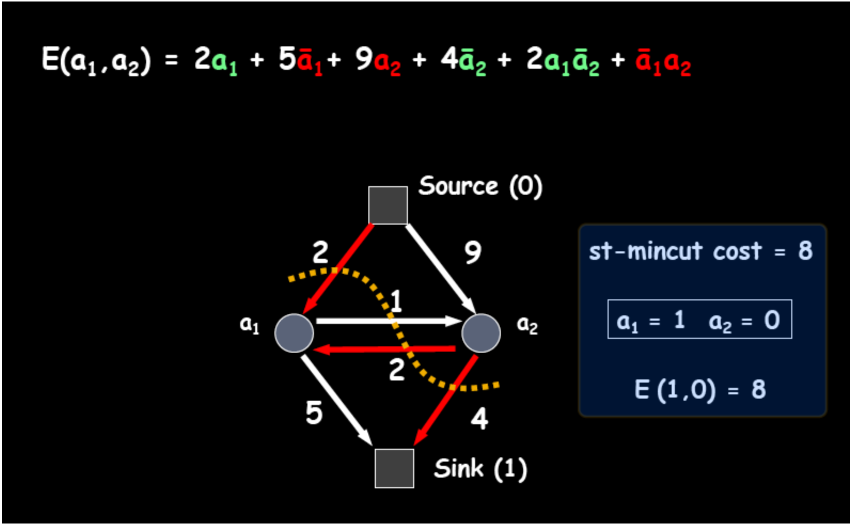
[Source: P. Kohli]

Graph Construction



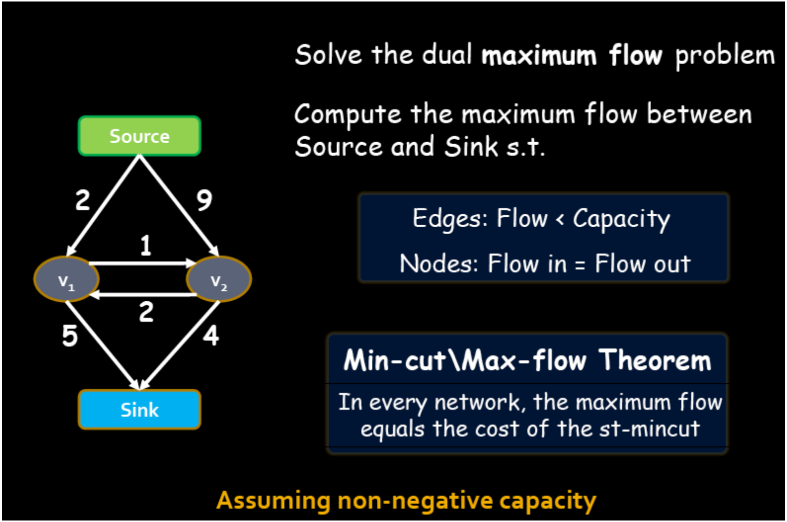
[Source: P. Kohli]

Graph Construction



[Source: P. Kohli]

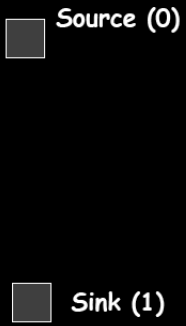
How to compute the St-mincut?



[Source: P. Kohli]

How does the code look like

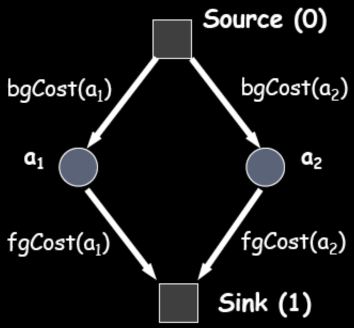
```
Graph *g;

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

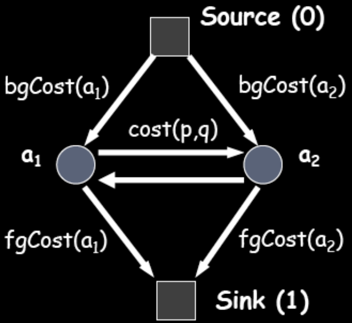
```
Graph *g;

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

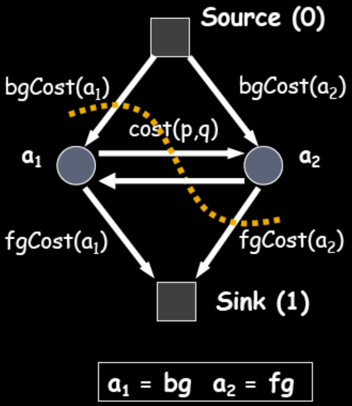
```
Graph *g;

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

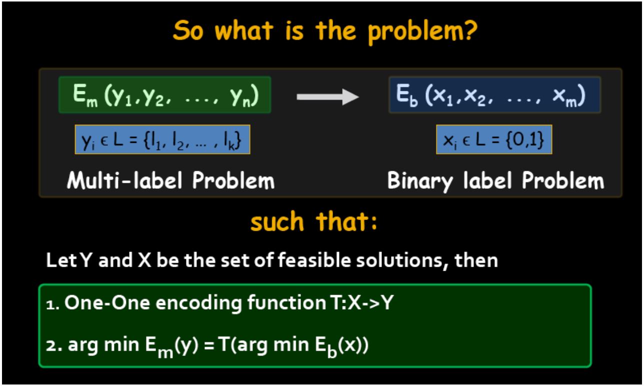
```
Graph *g;

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]

Graph cuts for multi-label problems

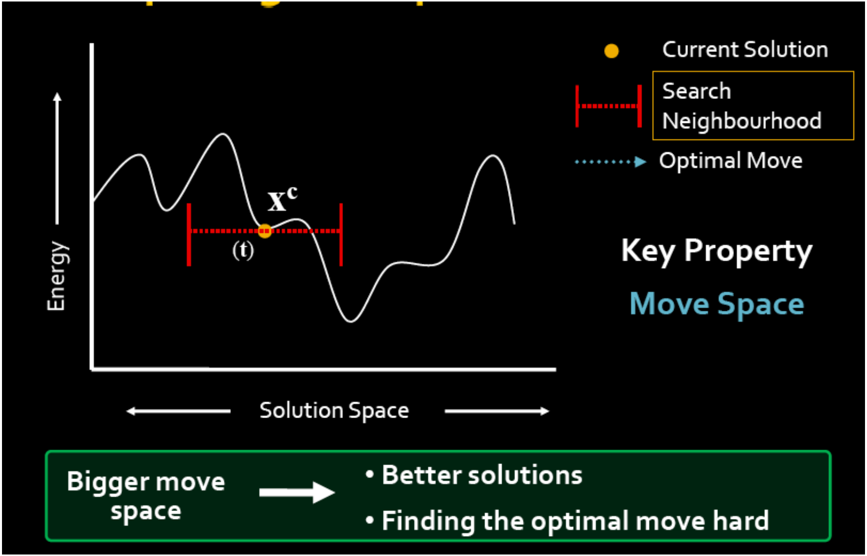
- Exact Transformation to QPBF [Roy and Cox 98] [Ishikawa 03] [Schlesinger et al. 06] [Ramalingam et al. 08]



- Very high computational cost

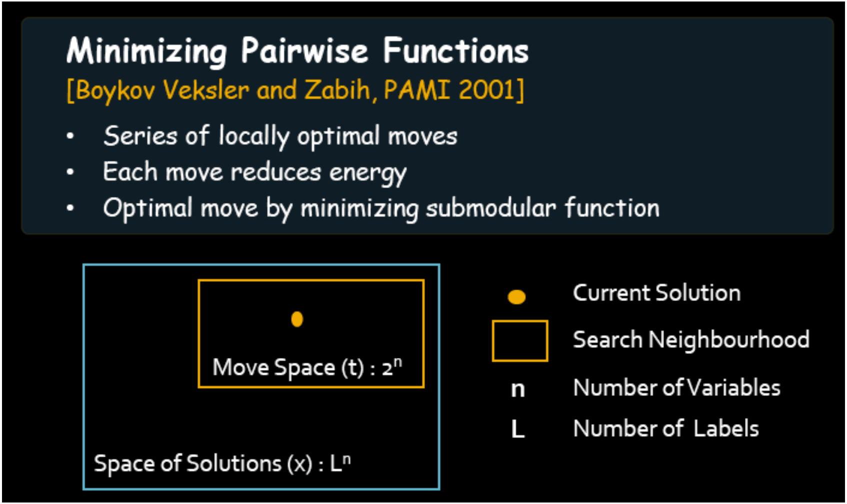
[Source: P. Kohli]

Computing the Optimal Move



[Source: P. Kohli]

Move Making Algorithms



[Source: P. Kohli]

Energy Minimization

- Consider pairwise MRFs

$$E(f) = \sum_{\{p,q\} \in \mathcal{N}} V_{p,q}(f_p, f_q) + \sum_p D_p(f_p)$$

  with $\mathcal{N}$ defining the interactions between nodes, e.g., pixels

- D_p non-negative, but arbitrary.

Energy Minimization

- Consider pairwise MRFs

$$E(f) = \sum_{\{p,q\} \in \mathcal{N}} V_{p,q}(f_p, f_q) + \sum_{p} D_p(f_p)$$

  with $\mathcal{N}$ defining the interactions between nodes, e.g., pixels
- D_p non-negative, but arbitrary.
- This is the graph-cuts notation.
- Important to notice it's the same thing.

Energy Minimization

- Consider pairwise MRFs

$$E(f) = \sum_{\{p,q\} \in \mathcal{N}} V_{p,q}(f_p, f_q) + \sum_p D_p(f_p)$$

  with $\mathcal{N}$ defining the interactions between nodes, e.g., pixels
- D_p non-negative, but arbitrary.
- This is the graph-cuts notation.
- Important to notice it's the same thing.

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

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

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.

- 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.

- For multi-dimensional, replace $|\cdot|$ by any norm.

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.

- For multi-dimensional, replace $|\cdot|$ by any norm.
- 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.

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.

- For multi-dimensional, replace $|\cdot|$ by any norm.
- 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.

Binary Moves

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

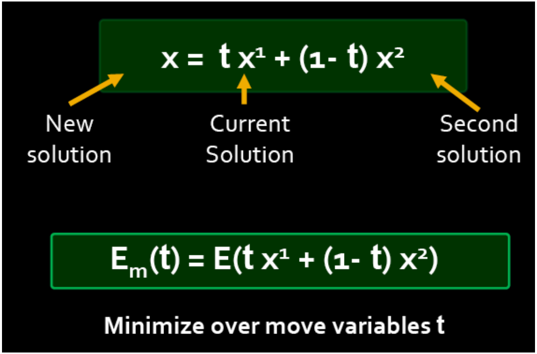


Figure: Figure from P. Kohli tutorial on graph-cuts

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

Swap Move

[Source: P. Kohli]

Swap Move

- Variables labeled α, β can swap their labels

- Move energy is submodular if:
  - Unary Potentials: Arbitrary
  - Pairwise potentials: Semi-metric

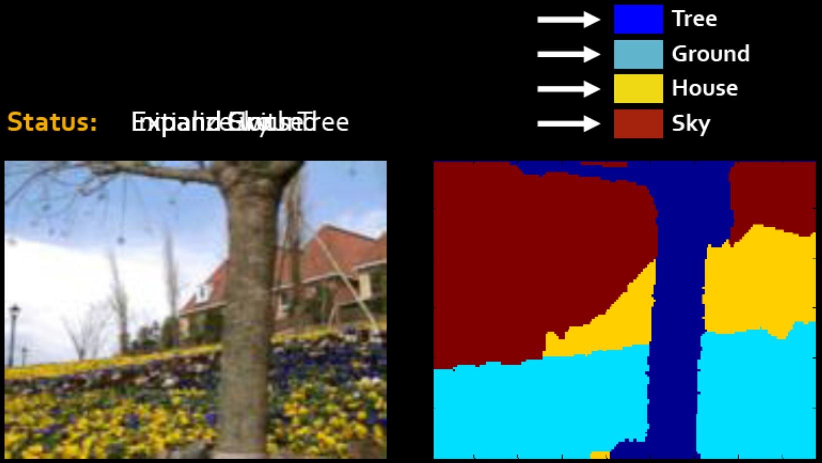
$$\boxed{\begin{array}{c} \theta_{ij}(l_a, l_b) \geq 0 \\ \theta_{ij}(l_a, l_b) = 0 \longleftrightarrow a = b \end{array}}$$

Examples: Potts model, Truncated Convex

[Source: P. Kohli]

Expansion Move

- Variables take label α or retain current label



[Source: P. Kohli]

Expansion Move

- Variables take label α or retain current label

- Move energy is submodular if:
  - Unary Potentials: Arbitrary
  - Pairwise potentials: Metric

Semi metric
+
Triangle Inequality

$$\boxed{\theta_{ij}(l_a, l_b) + \theta_{ij}(l_b, l_c) \geq \theta_{ij}(l_a, l_c)}$$

Examples: Potts model, Truncated linear

Cannot solve truncated quadratic

[Source: P. Kohli]

More formally

- Any labeling can be uniquely represented by a partition of image pixels $\mathbf{P} = \{\mathcal{P}_l | l \in \mathcal{L}\}$, where $\mathcal{P}_l = \{p \in \mathcal{P} | f_p = l\}$ is a subset of pixels assigned label l.
- There is a one to one correspondence between labelings f and partitions $\mathcal{P}$.

More formally

- Any labeling can be uniquely represented by a partition of image pixels $\mathbf{P} = \{\mathcal{P}_l | l \in \mathcal{L}\}$, where $\mathcal{P}_l = \{p \in \mathcal{P} | f_p = l\}$ is a subset of pixels assigned label l.
- There is a one to one correspondence between labelings f and partitions $\mathcal{P}$.
- Given a pair of labels α, β, a move from a partition $\mathcal{P}$ (labeling f) to a new partition $\mathcal{P}$' (labeling f') is called an $\alpha - \beta$ **swap** if $\mathcal{P}_l = \mathcal{P}'$ for any label $l \neq \alpha, \beta$.

More formally

- Any labeling can be uniquely represented by a partition of image pixels $\mathbf{P} = \{\mathcal{P}_l | l \in \mathcal{L}\}$, where $\mathcal{P}_l = \{p \in \mathcal{P} | f_p = l\}$ is a subset of pixels assigned label l.
- There is a one to one correspondence between labelings f and partitions $\mathcal{P}$.
- Given a pair of labels α, β, a move from a partition $\mathcal{P}$ (labeling f) to a new partition $\mathcal{P}$' (labeling f') is called an $\alpha - \beta$ **swap** if $\mathcal{P}_l = \mathcal{P}'$ for any label $l \neq \alpha, \beta$.
- The only difference between $\mathcal{P}$ and $\mathcal{P}'$ is that some pixels that were labeled in $\mathcal{P}$ are now labeled in $\mathcal{P}'$, and vice-versa.

More formally

- Any labeling can be uniquely represented by a partition of image pixels $\mathbf{P} = \{\mathcal{P}_l | l \in \mathcal{L}\}$, where $\mathcal{P}_l = \{p \in \mathcal{P} | f_p = l\}$ is a subset of pixels assigned label l.
- There is a one to one correspondence between labelings f and partitions $\mathcal{P}$.
- Given a pair of labels α, β, a move from a partition $\mathcal{P}$ (labeling f) to a new partition $\mathcal{P}$' (labeling f') is called an $\alpha - \beta$ **swap** if $\mathcal{P}_l = \mathcal{P}'$ for any label $l \neq \alpha, \beta$.
- The only difference between $\mathcal{P}$ and $\mathcal{P}'$ is that some pixels that were labeled in $\mathcal{P}$ are now labeled in $\mathcal{P}'$, and vice-versa.
- Given a label l, a move from a partition $\mathcal{P}$ (labeling f) to a new partition $\mathcal{P}'$ (labeling f') is called an α-**expansion** if $\mathcal{P}_\alpha \subset \mathcal{P}'_\alpha$ and $\mathcal{P}'_l \subset \mathcal{P}_l$.

More formally

- Any labeling can be uniquely represented by a partition of image pixels $\mathbf{P} = \{\mathcal{P}_l | l \in \mathcal{L}\}$, where $\mathcal{P}_l = \{p \in \mathcal{P} | f_p = l\}$ is a subset of pixels assigned label l.
- There is a one to one correspondence between labelings f and partitions $\mathcal{P}$.
- Given a pair of labels α, β, a move from a partition $\mathcal{P}$ (labeling f) to a new partition $\mathcal{P}$' (labeling f') is called an $\alpha - \beta$ **swap** if $\mathcal{P}_l = \mathcal{P}'$ for any label $l \neq \alpha, \beta$.
- The only difference between $\mathcal{P}$ and $\mathcal{P}'$ is that some pixels that were labeled in $\mathcal{P}$ are now labeled in $\mathcal{P}'$, and vice-versa.
- Given a label l, a move from a partition $\mathcal{P}$ (labeling f) to a new partition $\mathcal{P}'$ (labeling f') is called an α-**expansion** if $\mathcal{P}_\alpha \subset \mathcal{P}'_\alpha$ and $\mathcal{P}'_l \subset \mathcal{P}_l$.
- An α-**expansion** move allows any set of image pixels to change their labels to α.

More formally

- Any labeling can be uniquely represented by a partition of image pixels $\mathbf{P} = \{\mathcal{P}_l | l \in \mathcal{L}\}$, where $\mathcal{P}_l = \{p \in \mathcal{P} | f_p = l\}$ is a subset of pixels assigned label l.
- There is a one to one correspondence between labelings f and partitions $\mathcal{P}$.
- Given a pair of labels α, β, a move from a partition $\mathcal{P}$ (labeling f) to a new partition $\mathcal{P}$' (labeling f') is called an $\alpha - \beta$ **swap** if $\mathcal{P}_l = \mathcal{P}'$ for any label $l \neq \alpha, \beta$.
- The only difference between $\mathcal{P}$ and $\mathcal{P}'$ is that some pixels that were labeled in $\mathcal{P}$ are now labeled in $\mathcal{P}'$, and vice-versa.
- Given a label l, a move from a partition $\mathcal{P}$ (labeling f) to a new partition $\mathcal{P}'$ (labeling f') is called an α-**expansion** if $\mathcal{P}_\alpha \subset \mathcal{P}'_\alpha$ and $\mathcal{P}'_l \subset \mathcal{P}_l$.
- An α-**expansion** move allows any set of image pixels to change their labels to α.

Example

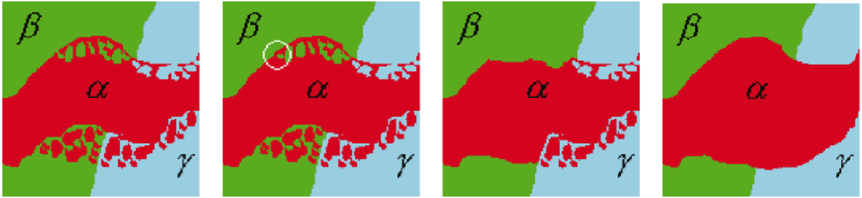


Figure: (a) Current partition (b) local move (c) $\alpha - \beta$-swap (d) α-expansion.

Algorithms

1. Start with an arbitrary labeling f
2. Set success := 0
3. For each pair of labels $\{\alpha, \beta\} \subset \mathcal{L}$
   3.1. Find $\hat{f} = \arg\min E(f')$ among f' within one α-β swap of f
   3.2. If $E(\hat{f}) < E(f)$, set f := $\hat{f}$ and success := 1
4. If success = 1 goto 2
5. Return f

---

1. Start with an arbitrary labeling f
2. Set success := 0
3. For each label $\alpha \in \mathcal{L}$
   3.1. Find $\hat{f} = \arg\min E(f')$ among f' within one α-expansion of f
   3.2. If $E(\hat{f}) < E(f)$, set f := $\hat{f}$ and success := 1
4. If success = 1 goto 2
5. Return f

Finding optimal Swap move

- Given an input labeling f (partition $\mathcal{P}$) and a pair of labels α, β we want to find a labeling $\hat{f}$ that minimizes E over all labelings within one $\alpha - \beta$-swap of f.
- This is going to be done by computing a labeling corresponding to a minimum cut on a graph $\mathcal{G}_{\alpha\beta} = (\mathcal{V}_{\alpha\beta}, \mathcal{E}_{\alpha\beta})$.

Finding optimal Swap move

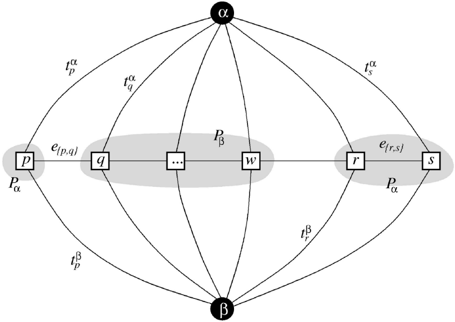
- Given an input labeling f (partition $\mathcal{P}$) and a pair of labels α, β we want to find a labeling $\hat{f}$ that minimizes E over all labelings within one $\alpha - \beta$-swap of f.
- This is going to be done by computing a labeling corresponding to a minimum cut on a graph $\mathcal{G}_{\alpha\beta} = (\mathcal{V}_{\alpha\beta}, \mathcal{E}_{\alpha\beta})$.
- The structure of this graph is dynamically determined by the current partition $\mathcal{P}$ and by the labels α, β.

Finding optimal Swap move

- Given an input labeling f (partition $\mathcal{P}$) and a pair of labels α, β we want to find a labeling $\hat{f}$ that minimizes E over all labelings within one $\alpha - \beta$-swap of f.
- This is going to be done by computing a labeling corresponding to a minimum cut on a graph $\mathcal{G}_{\alpha\beta} = (\mathcal{V}_{\alpha\beta}, \mathcal{E}_{\alpha\beta})$.
- The structure of this graph is dynamically determined by the current partition $\mathcal{P}$ and by the labels α, β.

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

Computing the Cut

- Any cut must have a single *t*-link not cut.
- This defines a labeling

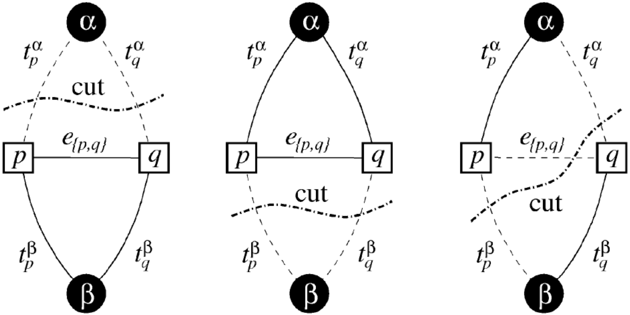
$$f_p^{\mathcal{C}} = \begin{cases} \alpha & \text{if } t_p^{\alpha} \in \mathcal{C} \text{ for } p \in \mathcal{P}_{\alpha\beta} \\ \beta & \text{if } t_p^{\beta} \in \mathcal{C} \text{ for } p \in \mathcal{P}_{\alpha\beta} \\ f_p & \text{for } p \in \mathcal{P},\ p \notin \mathcal{P}_{\alpha\beta}. \end{cases}$$

- There is a one-to-one correspondences between a cut and a labeling.
- The energy of the cut is the energy of the labeling.
- See Boykov et al, "*fast approximate energy minimization via graph cuts*" PAMI 2001.

Properties

- For any cut, then

$$\begin{array}{lllll}
(a) & \mathit{If} & t_p^\alpha, t_q^\alpha \in \mathcal{C} & \mathit{then} & e_{\{p,q\}} \notin \mathcal{C}. \\
(b) & \mathit{If} & t_p^\beta, t_q^\beta \in \mathcal{C} & \mathit{then} & e_{\{p,q\}} \notin \mathcal{C}. \\
(c) & \mathit{If} & t_p^\beta, t_q^\alpha \in \mathcal{C} & \mathit{then} & e_{\{p,q\}} \in \mathcal{C}. \\
(d) & \mathit{If} & t_p^\alpha, t_q^\beta \in \mathcal{C} & \mathit{then} & e_{\{p,q\}} \in \mathcal{C}.
\end{array}$$



Finding the optimal α expansion

- Given an input labeling f (partition $\mathcal{P}$) and a label α we want to find a labeling $\hat{f}$ that minimizes E over all labelings within one α-expansion of f.
- This is going to be done by computing a labeling corresponding to a minimum cut on a graph $\mathcal{G}_\alpha = (\mathcal{V}_\alpha, \mathcal{E}_\alpha)$.

Finding the optimal α expansion

- Given an input labeling f (partition $\mathcal{P}$) and a label α we want to find a labeling $\hat{f}$ that minimizes E over all labelings within one α-expansion of f.
- This is going to be done by computing a labeling corresponding to a minimum cut on a graph $\mathcal{G}_\alpha = (\mathcal{V}_\alpha, \mathcal{E}_\alpha)$.
- The structure of this graph is dynamically determined by the current partition $\mathcal{P}$ and by the label α.

Finding the optimal α expansion

- Given an input labeling f (partition $\mathcal{P}$) and a label α we want to find a labeling $\hat{f}$ that minimizes E over all labelings within one α-expansion of f.
- This is going to be done by computing a labeling corresponding to a minimum cut on a graph $\mathcal{G}_\alpha = (\mathcal{V}_\alpha, \mathcal{E}_\alpha)$.
- The structure of this graph is dynamically determined by the current partition $\mathcal{P}$ and by the label α.
- Different graph than the $\alpha - \beta$ swap.

Finding the optimal α expansion

- Given an input labeling f (partition $\mathcal{P}$) and a label α we want to find a labeling $\hat{f}$ that minimizes E over all labelings within one α-expansion of f.
- This is going to be done by computing a labeling corresponding to a minimum cut on a graph $\mathcal{G}_\alpha = (\mathcal{V}_\alpha, \mathcal{E}_\alpha)$.
- The structure of this graph is dynamically determined by the current partition $\mathcal{P}$ and by the label α.
- Different graph than the $\alpha - \beta$ swap.

Graph Construction

- The set of vertices includes the two terminals α and $\bar{\alpha}$, as well as all image pixels $p \in \mathcal{P}$.
- Additionally, for each pair of neighboring pixels p, q such that $f_p \neq f_q$ we create an auxiliary node $a_{p,q}$.

Graph Construction

- The set of vertices includes the two terminals α and $\bar{\alpha}$, as well as all image pixels $p \in \mathcal{P}$.
- Additionally, for each pair of neighboring pixels p, q such that $f_p \neq f_q$ we create an auxiliary node $a_{p,q}$.
- Each pixel p is connected to the terminals α and $\bar{\alpha}$, called t-links.

Graph Construction

- The set of vertices includes the two terminals α and $\bar{\alpha}$, as well as all image pixels $p \in \mathcal{P}$.
- Additionally, for each pair of neighboring pixels p, q such that $f_p \neq f_q$ we create an auxiliary node $a_{p,q}$.
- Each pixel p is connected to the terminals α and $\bar{\alpha}$, called t-links.
- Each set of pixels p, q which are neighbors and $f_p = f_q$, we connect with and n-link.

Graph Construction

- The set of vertices includes the two terminals α and $\bar{\alpha}$, as well as all image pixels $p \in \mathcal{P}$.
- Additionally, for each pair of neighboring pixels p, q such that $f_p \neq f_q$ we create an auxiliary node $a_{p,q}$.
- Each pixel p is connected to the terminals α and $\bar{\alpha}$, called t-links.
- Each set of pixels p, q which are neighbors and $f_p = f_q$, we connect with and n-link.
- For each pair of neighboring pixels such that $f_p \neq f_q$, we create a triplet $\{e_{p,a}, e_{a,q}, t_a^{\bar{\alpha}}\}$.

Graph Construction

- The set of vertices includes the two terminals α and $\bar{\alpha}$, as well as all image pixels $p \in \mathcal{P}$.
- Additionally, for each pair of neighboring pixels p, q such that $f_p \neq f_q$ we create an auxiliary node $a_{p,q}$.
- Each pixel p is connected to the terminals α and $\bar{\alpha}$, called t-links.
- Each set of pixels p, q which are neighbors and $f_p = f_q$, we connect with and n-link.
- For each pair of neighboring pixels such that $f_p \neq f_q$, we create a triplet $\{e_{p,a}, e_{a,q}, t_a^{\bar{\alpha}}\}$.
- The set of edges is then

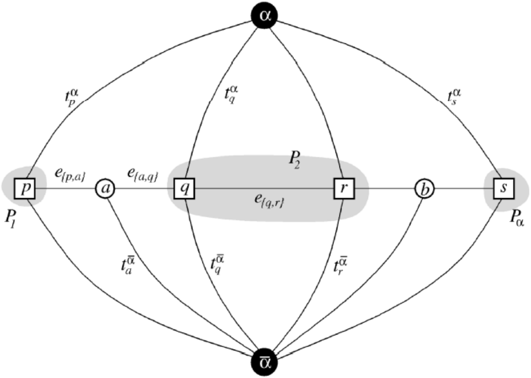
$$\mathcal{E}_\alpha = \left\{ \bigcup_{p \in \mathcal{P}} \{t_p^\alpha, t_p^{\bar{\alpha}}\}, \bigcup_{\substack{\{p,q\} \in \mathcal{N} \\ f_p \neq f_q}} \mathcal{E}_{\{p,q\}}, \bigcup_{\substack{\{p,q\} \in \mathcal{N} \\ f_p = f_q}} e_{\{p,q\}} \right\}$$

Graph Construction

- The set of vertices includes the two terminals α and $\bar{\alpha}$, as well as all image pixels $p \in \mathcal{P}$.
- Additionally, for each pair of neighboring pixels p, q such that $f_p \neq f_q$ we create an auxiliary node $a_{p,q}$.
- Each pixel p is connected to the terminals α and $\bar{\alpha}$, called t-links.
- Each set of pixels p, q which are neighbors and $f_p = f_q$, we connect with and n-link.
- For each pair of neighboring pixels such that $f_p \neq f_q$, we create a triplet $\{e_{p,a}, e_{a,q}, t_a^{\bar{\alpha}}\}$.
- The set of edges is then

$$\mathcal{E}_\alpha = \left\{ \bigcup_{p \in \mathcal{P}} \{t_p^\alpha, t_p^{\bar{\alpha}}\}, \bigcup_{\substack{\{p,q\} \in \mathcal{N} \\ f_p \neq f_q}} \mathcal{E}_{\{p,q\}} \ , \bigcup_{\substack{\{p,q\} \in \mathcal{N} \\ f_p = f_q}} e_{\{p,q\}} \right\}$$

Graph Construction



edge	weight	for
$t_p^{\bar{\alpha}}$	∞	$p \in \mathcal{P}_\alpha$
$t_p^{\bar{\alpha}}$	$D_p(f_p)$	$p \notin \mathcal{P}_\alpha$
t_p^{α}	$D_p(\alpha)$	$p \in \mathcal{P}$
$e_{\{p,a\}}$	$V(f_p, \alpha)$	$\{p,q\} \in \mathcal{N},\ f_p \neq f_q$
$e_{\{a,q\}}$	$V(\alpha, f_q)$	
$t_a^{\bar{\alpha}}$	$V(f_p, f_q)$	
$e_{\{p,q\}}$	$V(f_p, \alpha)$	$\{p,q\} \in \mathcal{N},\ f_p = f_q$

Properties

- There is a one-to-one correspondences between a cut and a labeling.

$$f_p^{\mathcal{C}} = \begin{cases} \alpha & \text{if} \quad t_p^{\alpha} \in \mathcal{C} \\ f_p & \text{if} \quad t_p^{\bar{\alpha}} \in \mathcal{C} \end{cases} \qquad \forall p \in \mathcal{P}.$$

- The energy of the cut is the energy of the labeling.
- See Boykov et al, "*fast approximate energy minimization via graph cuts*" PAMI 2001.

Property 5.2. *If* $\{p, q\} \in \mathcal{N}$ *and* $f_p \neq f_q$, *then a minimum cut* $\mathcal{C}$ *on* $\mathcal{G}_\alpha$ *satisfies:*

$$\begin{array}{llllll} (a) & \textit{If} & t_p^{\alpha}, t_q^{\alpha} \in \mathcal{C} & \textit{then} & \mathcal{C} \cap \mathcal{E}_{\{p,q\}} = \emptyset. \\ (b) & \textit{If} & t_p^{\bar{\alpha}}, t_q^{\bar{\alpha}} \in \mathcal{C} & \textit{then} & \mathcal{C} \cap \mathcal{E}_{\{p,q\}} = t_a^{\bar{\alpha}}. \\ (c) & \textit{If} & t_p^{\bar{\alpha}}, t_q^{\alpha} \in \mathcal{C} & \textit{then} & \mathcal{C} \cap \mathcal{E}_{\{p,q\}} = e_{\{p,a\}}. \\ (d) & \textit{If} & t_p^{\alpha}, t_q^{\bar{\alpha}} \in \mathcal{C} & \textit{then} & \mathcal{C} \cap \mathcal{E}_{\{p,q\}} = e_{\{a,q\}}. \end{array}$$

Global Minimization Techniques

Ways to get an approximate solution typically

- Dynamic programming approximations
- Sampling
- Simulated annealing
- Graph-cuts: imposes restrictions on the type of pairwise cost functions
- Message passing: iterative algorithms that pass messages between nodes in the graph. Which graph?

Now we can solve for the MAP (approximately) in general energies. We can solve for other problems than stereo