Stereo and Energy Minimization

Raquel Urtasun

TTI Chicago

Feb 21, 2013

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

Which Similarity Measure?

- Sum of square intensity differences (SSD), i.e., mean square error
- Absolute intensity differences (SAD), i.e., mean absolute difference

Which Similarity Measure?

- Sum of square intensity differences (SSD), i.e., mean square error
- Absolute intensity differences (SAD), i.e., mean absolute difference
- Robust measures including truncated quadratic functions: they limit the influence of mismatches

Which Similarity Measure?

- Sum of square intensity differences (SSD), i.e., mean square error
- Absolute intensity differences (SAD), i.e., mean absolute difference
- Robust measures including truncated quadratic functions: they limit the influence of mismatches
- Normalized cross-correlation: behaves similarly to the SSD

Which Similarity Measure?

- Sum of square intensity differences (SSD), i.e., mean square error
- Absolute intensity differences (SAD), i.e., mean absolute difference
- Robust measures including truncated quadratic functions: they limit the influence of mismatches
- Normalized cross-correlation: behaves similarly to the SSD
- Binary matching costs based on binary features such as edges (e.g., match or not match)

Which Similarity Measure?

- Sum of square intensity differences (SSD), i.e., mean square error
- Absolute intensity differences (SAD), i.e., mean absolute difference
- Robust measures including truncated quadratic functions: they limit the influence of mismatches
- Normalized cross-correlation: behaves similarly to the SSD
- Binary matching costs based on binary features such as edges (e.g., match or not match)
- Invariant to differences in camera gain or bias, e.g., gradient based measurement, filter responses

Which Similarity Measure?

- Sum of square intensity differences (SSD), i.e., mean square error
- Absolute intensity differences (SAD), i.e., mean absolute difference
- Robust measures including truncated quadratic functions: they limit the influence of mismatches
- Normalized cross-correlation: behaves similarly to the SSD
- Binary matching costs based on binary features such as edges (e.g., match or not match)
- Invariant to differences in camera gain or bias, e.g., gradient based measurement, filter responses
- All sort of feature descriptors that we saw before in class as well as others

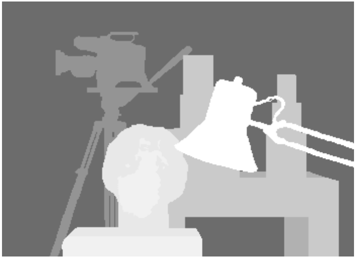
Which Similarity Measure?

- Sum of square intensity differences (SSD), i.e., mean square error
- Absolute intensity differences (SAD), i.e., mean absolute difference
- Robust measures including truncated quadratic functions: they limit the influence of mismatches
- Normalized cross-correlation: behaves similarly to the SSD
- Binary matching costs based on binary features such as edges (e.g., match or not match)
- Invariant to differences in camera gain or bias, e.g., gradient based measurement, filter responses
- All sort of feature descriptors that we saw before in class as well as others

Disparity Estimation

- DSI: Disparity image

Scene



Ground truth

Sparse Correspondences

- Early approaches to stereo were feature-based
- Consists on first extract a set of "matchable" features. How?

Sparse Correspondences

- Early approaches to stereo were feature-based
- Consists on first extract a set of "matchable" features. How?
- This resulted in a sparse disparity computation, and a sparse 3D point cloud

Sparse Correspondences

- Early approaches to stereo were feature-based
- Consists on first extract a set of "matchable" features. How?
- This resulted in a sparse disparity computation, and a sparse 3D point cloud

Denser correspondences

Typical stereo pipeline

1. Matching cost computation
2. Cost aggregation
3. Disparity computation
4. Disparity refinement

Local methods

1. **Matching cost computation**: is the square difference in intensity values at a given disparity
2. **Cost aggregation**: adds matching cost over square window with constant disparity

Local methods

1. **Matching cost computation**: is the square difference in intensity values at a given disparity
2. **Cost aggregation**: adds matching cost over square window with constant disparity
3. **Disparity computation**: select the minimal aggregated value at each pixel

Local methods

1. **Matching cost computation**: is the square difference in intensity values at a given disparity
2. **Cost aggregation**: adds matching cost over square window with constant disparity
3. **Disparity computation**: select the minimal aggregated value at each pixel
4. **Disparity refinement**: consistency check and sub-pixel estimation

Local methods

1. **Matching cost computation**: is the square difference in intensity values at a given disparity
2. **Cost aggregation**: adds matching cost over square window with constant disparity
3. **Disparity computation**: select the minimal aggregated value at each pixel
4. **Disparity refinement**: consistency check and sub-pixel estimation

Aggregation in local methods

- Aggregate the matching cost summing over a **support region**
- The support region can be 2D (i.e., x,y) or 3D (i.e., x,y,d). The latter supports slanted surfaces.

Aggregation in local methods

- Aggregate the matching cost summing over a **support region**
- The support region can be 2D (i.e., x,y) or 3D (i.e., x,y,d). The latter supports slanted surfaces.
- Simplest approach: Aggregation with fixed support can be done by performing 2D or 3D convolution

$$C(x, y, d) = w(x, y, d) * C_0(x, y, d)$$

Aggregation in local methods

- Aggregate the matching cost summing over a **support region**
- The support region can be 2D (i.e., x,y) or 3D (i.e., x,y,d). The latter supports slanted surfaces.
- Simplest approach: Aggregation with fixed support can be done by performing 2D or 3D convolution

$$C(x, y, d) = w(x, y, d) * C_0(x, y, d)$$

- Problems
  - If too small, then ambiguous
  - If too big, bleeding effects at the edges

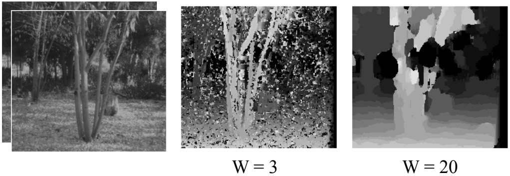


Figure: from N. Snavely

Aggregation in local methods

- Aggregate the matching cost summing over a **support region**
- The support region can be 2D (i.e., x,y) or 3D (i.e., x,y,d). The latter supports slanted surfaces.
- Simplest approach: Aggregation with fixed support can be done by performing 2D or 3D convolution

$$C(x, y, d) = w(x, y, d) * C_0(x, y, d)$$

- Problems
  - If too small, then ambiguous
  - If too big, bleeding effects at the edges

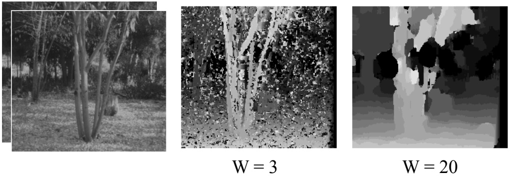
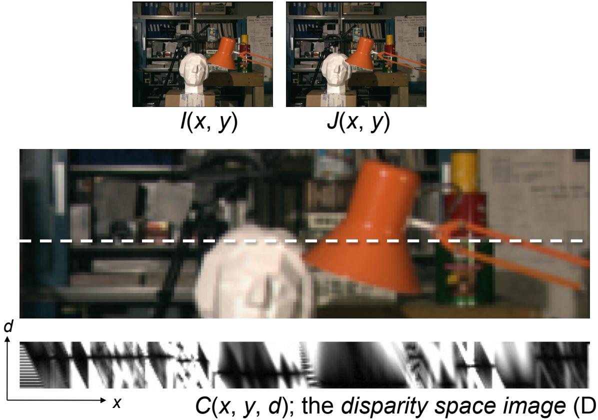


Figure: from N. Snavely

Matching cost computation



$C(x, y, d)$; the *disparity space image* (DSI)

- The disparity is then computed by

$$d(x,y) = arg \min_{d'} C(x,y,d')$$

[Source: N. Snavely]

More complex aggregation

- Solution: make the **window adaptive** to the image evidence (e.g., aggregate from pixels with similar appearance)
- Have we seen something similar in class? When? How would you do this?

More complex aggregation

- Solution: make the **window adaptive** to the image evidence (e.g., aggregate from pixels with similar appearance)
- Have we seen something similar in class? When? How would you do this?
- An alternative solution is to select between different windows. This can be computed efficiently using integral images. How?

More complex aggregation

- Solution: make the **window adaptive** to the image evidence (e.g., aggregate from pixels with similar appearance)
- Have we seen something similar in class? When? How would you do this?
- An alternative solution is to select between different windows. This can be computed efficiently using integral images. How?
- In **iterative diffusion** we repeately add to each pixel's costs the weighted value of their neighbors

More complex aggregation

- Solution: make the **window adaptive** to the image evidence (e.g., aggregate from pixels with similar appearance)
- Have we seen something similar in class? When? How would you do this?
- An alternative solution is to select between different windows. This can be computed efficiently using integral images. How?
- In **iterative diffusion** we repeately add to each pixel's costs the weighted value of their neighbors
  - What happens as a function of the number of iterations?

More complex aggregation

- Solution: make the **window adaptive** to the image evidence (e.g., aggregate from pixels with similar appearance)
- Have we seen something similar in class? When? How would you do this?
- An alternative solution is to select between different windows. This can be computed efficiently using integral images. How?
- In **iterative diffusion** we repeately add to each pixel's costs the weighted value of their neighbors
  - What happens as a function of the number of iterations?
  - Does a strategy like this seem familiar?

More complex aggregation

- Solution: make the **window adaptive** to the image evidence (e.g., aggregate from pixels with similar appearance)
- Have we seen something similar in class? When? How would you do this?
- An alternative solution is to select between different windows. This can be computed efficiently using integral images. How?
- In **iterative diffusion** we repeately add to each pixel's costs the weighted value of their neighbors
  - What happens as a function of the number of iterations?
  - Does a strategy like this seem familiar?
- Global methods are a more principle way to do aggregation

More complex aggregation

- Solution: make the **window adaptive** to the image evidence (e.g., aggregate from pixels with similar appearance)
- Have we seen something similar in class? When? How would you do this?
- An alternative solution is to select between different windows. This can be computed efficiently using integral images. How?
- In **iterative diffusion** we repeately add to each pixel's costs the weighted value of their neighbors
  - What happens as a function of the number of iterations?
  - Does a strategy like this seem familiar?
- Global methods are a more principle way to do aggregation

Local methods

1. Matching cost computation: is the square difference in intensity values at a given disparity
2. Cost aggregation: adds matching cost over square window with constant disparity
3. Disparity computation: select the minimal aggregated value at each pixel. Winner takes all strategy!
4. Disparity refinement:

Local methods

1. Matching cost computation: is the square difference in intensity values at a given disparity
2. Cost aggregation: adds matching cost over square window with constant disparity
3. Disparity computation: select the minimal aggregated value at each pixel. Winner takes all strategy!
4. Disparity refinement:
    - Consistency check: as the previous method check uniqueness of matches only on one direction

Local methods

1. Matching cost computation: is the square difference in intensity values at a given disparity
2. Cost aggregation: adds matching cost over square window with constant disparity
3. Disparity computation: select the minimal aggregated value at each pixel. Winner takes all strategy!
4. Disparity refinement:
   - Consistency check: as the previous method check uniqueness of matches only on one direction
   - Hole filling

Local methods

1. Matching cost computation: is the square difference in intensity values at a given disparity
2. Cost aggregation: adds matching cost over square window with constant disparity
3. Disparity computation: select the minimal aggregated value at each pixel. Winner takes all strategy!
4. Disparity refinement:
   - Consistency check: as the previous method check uniqueness of matches only on one direction
   - Hole filling
   - Sub-pixel estimation

Local methods

1. Matching cost computation: is the square difference in intensity values at a given disparity
2. Cost aggregation: adds matching cost over square window with constant disparity
3. Disparity computation: select the minimal aggregated value at each pixel. Winner takes all strategy!
4. Disparity refinement:
    - Consistency check: as the previous method check uniqueness of matches only on one direction
    - Hole filling
    - Sub-pixel estimation
    - Remove of spurious matches

Local methods

1. Matching cost computation: is the square difference in intensity values at a given disparity
2. Cost aggregation: adds matching cost over square window with constant disparity
3. Disparity computation: select the minimal aggregated value at each pixel. Winner takes all strategy!
4. Disparity refinement:
    - Consistency check: as the previous method check uniqueness of matches only on one direction
    - Hole filling
    - Sub-pixel estimation
    - Remove of spurious matches

Subpixel Estimation

- Most algorithms retrieve a disparity which is an integer
- This is not good enough for some applications, e.g., image-based rendering

Subpixel Estimation

- Most algorithms retrieve a disparity which is an integer
- This is not good enough for some applications, e.g., image-based rendering
- To remedy this, most approaches perform sub-pixel refinement after the disparity is estimated

Subpixel Estimation

- Most algorithms retrieve a disparity which is an integer
- This is not good enough for some applications, e.g., image-based rendering
- To remedy this, most approaches perform sub-pixel refinement after the disparity is estimated
- Gradient descent or fitting a curve is the most common ways to do the subpixel estimation

Subpixel Estimation

- Most algorithms retrieve a disparity which is an integer
- This is not good enough for some applications, e.g., image-based rendering
- To remedy this, most approaches perform sub-pixel refinement after the disparity is estimated
- Gradient descent or fitting a curve is the most common ways to do the subpixel estimation
- What might be the problem?

Subpixel Estimation

- Most algorithms retrieve a disparity which is an integer
- This is not good enough for some applications, e.g., image-based rendering
- To remedy this, most approaches perform sub-pixel refinement after the disparity is estimated
- Gradient descent or fitting a curve is the most common ways to do the subpixel estimation
- What might be the problem?
  - Disparities should be smooth

Subpixel Estimation

- Most algorithms retrieve a disparity which is an integer
- This is not good enough for some applications, e.g., image-based rendering
- To remedy this, most approaches perform sub-pixel refinement after the disparity is estimated
- Gradient descent or fitting a curve is the most common ways to do the subpixel estimation
- What might be the problem?
  - Disparities should be smooth
  - The regions where the disparities are estimated should be on the same (correct) surface, e.g., think of occlusion boundaries

Subpixel Estimation

- Most algorithms retrieve a disparity which is an integer
- This is not good enough for some applications, e.g., image-based rendering
- To remedy this, most approaches perform sub-pixel refinement after the disparity is estimated
- Gradient descent or fitting a curve is the most common ways to do the subpixel estimation
- What might be the problem?
  - Disparities should be smooth
  - The regions where the disparities are estimated should be on the same (correct) surface, e.g., think of occlusion boundaries

Other postprocessing

- **Occluded areas** can be detected using cross-checking, i.e., comparing left-to-right and right-to-left disparity maps
- A **median filter** is typically used to remove spurious mismatches

Other postprocessing

- **Occluded areas** can be detected using cross-checking, i.e., comparing left-to-right and right-to-left disparity maps
- A **median filter** is typically used to remove spurious mismatches
- **Holes** due to occlusion can be filled by surface fitting, or by distributing neighboring disparity estimates

Other postprocessing

- **Occluded areas** can be detected using cross-checking, i.e., comparing left-to-right and right-to-left disparity maps
- A **median filter** is typically used to remove spurious mismatches
- **Holes** due to occlusion can be filled by surface fitting, or by distributing neighboring disparity estimates
- It is interesting sometimes to estimate a notion of confidence for each estimate, e.g.,

$$Var(d) = \frac{\sigma_f^2}{a}$$

with σ_f^2 the image noise and a the curvature of the disparity image (DSI)

Other postprocessing

- **Occluded areas** can be detected using cross-checking, i.e., comparing left-to-right and right-to-left disparity maps
- A **median filter** is typically used to remove spurious mismatches
- **Holes** due to occlusion can be filled by surface fitting, or by distributing neighboring disparity estimates
- It is interesting sometimes to estimate a notion of confidence for each estimate, e.g.,

$$Var(d) = \frac{\sigma_f^2}{a}$$

with σ_f^2 the image noise and a the curvature of the disparity image (DSI)

Global Optimization

- There is no more aggregation step
- The problem is formulated as an energy minimization, i.e., MAP on a Markov random field

Global Optimization

- There is no more aggregation step
- The problem is formulated as an energy minimization, i.e., MAP on a Markov random field
- The MRF is expressed as the level of:

Global Optimization

- There is no more aggregation step
- The problem is formulated as an energy minimization, i.e., MAP on a Markov random field
- The MRF is expressed as the level of:
  - Pixels

Global Optimization

- There is no more aggregation step
- The problem is formulated as an energy minimization, i.e., MAP on a Markov random field
- The MRF is expressed as the level of:
  - Pixels
  - Fronto-parallel planes for sets of pixels

Global Optimization

- There is no more aggregation step
- The problem is formulated as an energy minimization, i.e., MAP on a Markov random field
- The MRF is expressed as the level of:
  - Pixels
  - Fronto-parallel planes for sets of pixels
  - Slanted plances for sets of pixels

Global Optimization

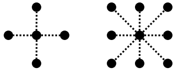
- There is no more aggregation step
- The problem is formulated as an energy minimization, i.e., MAP on a Markov random field
- The MRF is expressed as the level of:
  - Pixels
  - Fronto-parallel planes for sets of pixels
  - Slanted plances for sets of pixels

MRFs on pixels

- Most methods write the energy of the system as

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

where $d_i \in \{0, 1, \cdots, D\}$ represents the disparity of the $i-$th pixel



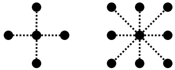
- If the second term does not exist, can we solve this easily? How?

MRFs on pixels

- Most methods write the energy of the system as

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

where $d_i \in \{0, 1, \cdots, D\}$ represents the disparity of the $i-$th pixel



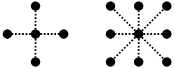
- If the second term does not exist, can we solve this easily? How?
- What about if the second term exists? do you know when we can solve this?

MRFs on pixels

- Most methods write the energy of the system as

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

where $d_i \in \{0, 1, \cdots, D\}$ represents the disparity of the $i-$th pixel



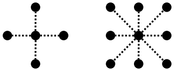
- If the second term does not exist, can we solve this easily? How?
- What about if the second term exists? do you know when we can solve this?
- What are these costs functions?

MRFs on pixels

- Most methods write the energy of the system as

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

where $d_i \in \{0, 1, \cdots, D\}$ represents the disparity of the $i-$th pixel



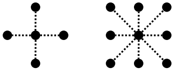
- If the second term does not exist, can we solve this easily? How?
- What about if the second term exists? do you know when we can solve this?
- What are these costs functions?
- C_i is any of the local matching algorithms we have seen

MRFs on pixels

- Most methods write the energy of the system as

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

where $d_i \in \{0, 1, \cdots, D\}$ represents the disparity of the $i-$th pixel



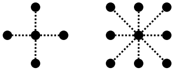
- If the second term does not exist, can we solve this easily? How?
- What about if the second term exists? do you know when we can solve this?
- What are these costs functions?
- C_i is any of the local matching algorithms we have seen
- What about C_{ij}?

MRFs on pixels

- Most methods write the energy of the system as

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

where $d_i \in \{0, 1, \cdots, D\}$ represents the disparity of the $i-$th pixel



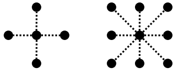
- If the second term does not exist, can we solve this easily? How?
- What about if the second term exists? do you know when we can solve this?
- What are these costs functions?
- C_i is any of the local matching algorithms we have seen
- What about C_{ij}?
- Encode for example that neighbors should be similar, or that for similar appearances, the neighbors should be similar. How do we encode this?

MRFs on pixels

- Most methods write the energy of the system as

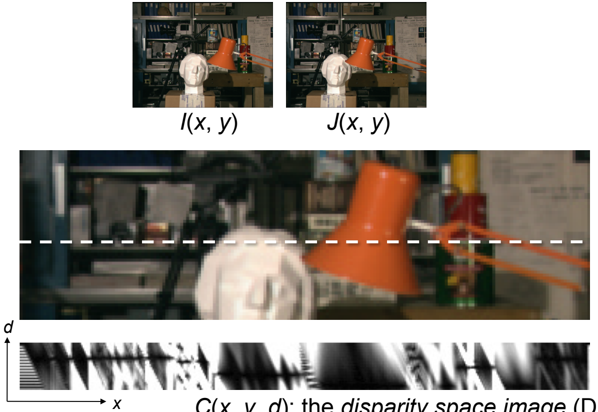
$$E(d_1, \cdots, d_n) = \sum_i C_i(d_i) + \sum_i \sum_{j \in \mathcal{N}(j)} C_{ij}(d_i, d_j)$$

where $d_i \in \{0, 1, \cdots, D\}$ represents the disparity of the $i-$th pixel



- If the second term does not exist, can we solve this easily? How?
- What about if the second term exists? do you know when we can solve this?
- What are these costs functions?
- C_i is any of the local matching algorithms we have seen
- What about C_{ij}?
- Encode for example that neighbors should be similar, or that for similar appearances, the neighbors should be similar. How do we encode this?

Unitary cost functions



$C(x, y, d)$; the *disparity space image* (DSI)

[Source: N. Snavely]

Pairwise cost functions

- A function of the disparity of neighboring pixels

$$C(d_i, d_j) = \rho(d_i - d_j)$$

with ρ a monotonic increasing function of disparity difference

- This can be the L-1 distance

$$C(d_i, d_j) = |d_i - d_j|$$

Pairwise cost functions

- A function of the disparity of neighboring pixels

$$C(d_i, d_j) = \rho(d_i - d_j)$$

  with ρ a monotonic increasing function of disparity difference

- This can be the L-1 distance

$$C(d_i, d_j) = |d_i - d_j|$$

- A popular choice is the Potts model

$$C(d_i, d_j) = \begin{cases} \gamma & \text{if } d_i \neq y_j \\ 0 & \text{if } d_i = y_j \end{cases}$$

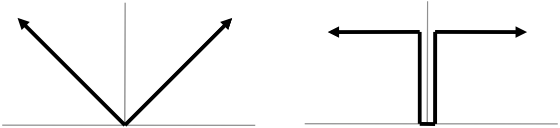


Figure: from N. Snavely

Pairwise cost functions

- A function of the disparity of neighboring pixels

$$C(d_i, d_j) = \rho(d_i - d_j)$$

with ρ a monotonic increasing function of disparity difference

- This can be the L-1 distance

$$C(d_i, d_j) = |d_i - d_j|$$

- A popular choice is the Potts model

$$C(d_i, d_j) = \begin{cases} \gamma & \text{if } d_i \neq y_j \\ 0 & \text{if } d_i = y_j \end{cases}$$

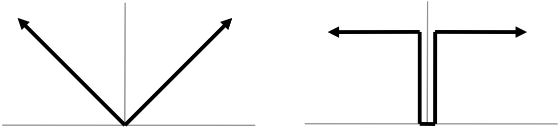


Figure: from N. Snavely

Pairwise cost functions

- This can be made dependent on the image evidence

$$C(d_i, d_j) = \rho_d(d_i, d_j) \cdot \rho_I(||I(u_i, v_i) - I(u_j, v_j)||)$$

where ρ_I is some monotonically decreasing function of intensity differences that lowers smoothness costs at high-intensity gradients

- More sophisticated cost functions can be defined. Any ideas?

Pairwise cost functions

- This can be made dependent on the image evidence

$$C(d_i, d_j) = \rho_d(d_i, d_j) \cdot \rho_I(||I(u_i, v_i) - I(u_j, v_j)||)$$

  where ρ_I is some monotonically decreasing function of intensity differences that lowers smoothness costs at high-intensity gradients

- More sophisticated cost functions can be defined. Any ideas?

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. Do you know any?

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. Do you know any?
- Several ways to get an approximate solution typically
  - Dynamic programming approximations

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. Do you know any?
- Several ways to get an approximate solution typically
  - Dynamic programming approximations
  - Sampling

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. Do you know any?
- Several ways to get an approximate solution typically
  - Dynamic programming approximations
  - Sampling
  - Simulated annealing

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. Do you know any?
- Several ways to get an approximate solution typically
  - Dynamic programming approximations
  - Sampling
  - Simulated annealing
  - Graph-cuts: imposes restrictions on the type of pairwise cost functions

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. Do you know any?
- Several 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?

MRFs on pixels

- The energy is defined as

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

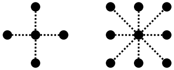
  where $x_i \in \{0, 1, \cdots, D\}$ represents a variable for the disparity of the $i-$th pixel
- This optimization is in general NP-hard.
- Global optima can be obtained in a few cases. Do you know any?
- Several ways to get an approximate solution typically
  - Dynamic programming approximations
  - 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?

A simple DP algorithm

- Neglect the vertical smoothness constraints Scharstein and Szeliski (2002)

- Then simply optimize independent scanlines (one for each direction $\mathbf{j}$) in the global energy function by recursive computation

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

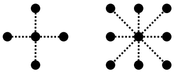


A simple DP algorithm

- Neglect the vertical smoothness constraints Scharstein and Szeliski (2002)

- Then simply optimize independent scanlines (one for each direction **j**) in the global energy function by recursive computation

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



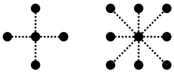
- What are the problems?

A simple DP algorithm

- Neglect the vertical smoothness constraints Scharstein and Szeliski (2002)

- Then simply optimize independent scanlines (one for each direction $\mathbf{j}$) in the global energy function by recursive computation

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



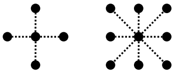
- What are the problems?
- What do we do with each direction?

A simple DP algorithm

- Neglect the vertical smoothness constraints Scharstein and Szeliski (2002)

- Then simply optimize independent scanlines (one for each direction $\mathbf{j}$) in the global energy function by recursive computation

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



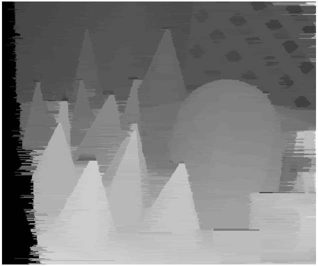
- What are the problems?
- What do we do with each direction?

A simple DP algorithm

- Finds smooth path through DPI from left to right

[Source: N. Snavely]

A simple DP algorithm



[Source: N. Snavely]

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

Global Minimization Techniques

Multiple 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?

Let's look more generaly into MRFs

Structure Prediction

- Input: $x \in \mathcal{X}$, typically an image.
- Output: label $y \in \mathcal{Y}$.
- Consider a score function $\theta(x, y)$ called **potential** or **feature** such that

$$\theta(x, y) = \begin{cases} \text{high} & \text{if } y \text{ is a good label for } x \\ \text{low} & \text{if } y \text{ is a bad label for } x \end{cases}$$

- We want to predict a label as

$$y^* = \arg\max_y \theta(x, y)$$

Score Decomposition

- We assume that the score decomposes

$$\theta(y|x) = \sum_i \theta_i(y_i) + \sum_\alpha \theta_\alpha(y_\alpha)$$

- This represents a (conditional) Markov Random Field (CRF)

$$p(y|x) = \frac{1}{Z} \prod_i \psi_i(x, y_i) \prod_\alpha \psi_\alpha(x, y_\alpha)$$

with $\log \psi_i(x, y_i) = \theta_i(x, y_i)$, and $\log \psi_\alpha(x, y_\alpha) = \theta_\alpha(x, y_\alpha)$.

Score Decomposition

- We assume that the score decomposes

$$\theta(y|x) = \sum_i \theta_i(y_i) + \sum_\alpha \theta_\alpha(y_\alpha)$$

- This represents a (conditional) Markov Random Field (CRF)

$$p(y|x) = \frac{1}{Z} \prod_i \psi_i(x, y_i) \prod_\alpha \psi_\alpha(x, y_\alpha)$$

with $\log \psi_i(x, y_i) = \theta_i(x, y_i)$, and $\log \psi_\alpha(x, y_\alpha) = \theta_\alpha(x, y_\alpha)$.

- $Z = \sum_y \prod_i \psi_i(x, y_i) \prod_\alpha \psi_\alpha(x, y_\alpha)$ is the partition function.

Score Decomposition

- We assume that the score decomposes

$$\theta(y|x) = \sum_i \theta_i(y_i) + \sum_\alpha \theta_\alpha(y_\alpha)$$

- This represents a (conditional) Markov Random Field (CRF)

$$p(y|x) = \frac{1}{Z} \prod_i \psi_i(x, y_i) \prod_\alpha \psi_\alpha(x, y_\alpha)$$

  with $\log \psi_i(x, y_i) = \theta_i(x, y_i)$, and $\log \psi_\alpha(x, y_\alpha) = \theta_\alpha(x, y_\alpha)$.

- $Z = \sum_y \prod_i \psi_i(x, y_i) \prod_\alpha \psi_\alpha(x, y_\alpha)$ is the partition function.
- Prediction also decomposes

$$y^* = \arg\max_y \sum_i \theta_i(y_i) + \sum_\alpha \theta_\alpha(y_\alpha)$$

Score Decomposition

- We assume that the score decomposes

$$\theta(y|x) = \sum_i \theta_i(y_i) + \sum_\alpha \theta_\alpha(y_\alpha)$$

- This represents a (conditional) Markov Random Field (CRF)

$$p(y|x) = \frac{1}{Z} \prod_i \psi_i(x, y_i) \prod_\alpha \psi_\alpha(x, y_\alpha)$$

with $\log \psi_i(x, y_i) = \theta_i(x, y_i)$, and $\log \psi_\alpha(x, y_\alpha) = \theta_\alpha(x, y_\alpha)$.

- $Z = \sum_y \prod_i \psi_i(x, y_i) \prod_\alpha \psi_\alpha(x, y_\alpha)$ is the partition function.
- Prediction also decomposes

$$y^* = \arg\max_y \sum_i \theta_i(y_i) + \sum_\alpha \theta_\alpha(y_\alpha)$$

- This in general is NP hard.

Score Decomposition

- We assume that the score decomposes

$$\theta(y|x) = \sum_i \theta_i(y_i) + \sum_\alpha \theta_\alpha(y_\alpha)$$

- This represents a (conditional) Markov Random Field (CRF)

$$p(y|x) = \frac{1}{Z} \prod_i \psi_i(x, y_i) \prod_\alpha \psi_\alpha(x, y_\alpha)$$

  with $\log \psi_i(x, y_i) = \theta_i(x, y_i)$, and $\log \psi_\alpha(x, y_\alpha) = \theta_\alpha(x, y_\alpha)$.

- $Z = \sum_y \prod_i \psi_i(x, y_i) \prod_\alpha \psi_\alpha(x, y_\alpha)$ is the partition function.
- Prediction also decomposes

$$y^* = \arg\max_y \sum_i \theta_i(y_i) + \sum_\alpha \theta_\alpha(y_\alpha)$$

- This in general is NP hard.

Markov Networks

- A **clique** in an undirected graph is a subset of its vertices such that every two vertices in the subset are connected by an edge.
- A **maximal clique** is a clique that cannot be extended by including one more adjacent vertex.

Markov Networks

- A **clique** in an undirected graph is a subset of its vertices such that every two vertices in the subset are connected by an edge.
- A **maximal clique** is a clique that cannot be extended by including one more adjacent vertex.
- For a set of variables $\mathbf{y} = \{y_1, \cdots, y_N\}$ a **Markov network** is defined as a product of potentials over the maximal cliques y_α of the graph G

$$p(y_1, \cdots, y_N) = \frac{1}{Z} \prod_{\alpha} \psi_\alpha(y_\alpha)$$

Markov Networks

- A **clique** in an undirected graph is a subset of its vertices such that every two vertices in the subset are connected by an edge.
- A **maximal clique** is a clique that cannot be extended by including one more adjacent vertex.
- For a set of variables $\mathbf{y} = \{y_1, \cdots, y_N\}$ a **Markov network** is defined as a product of potentials over the maximal cliques y_α of the graph G

$$p(y_1, \cdots, y_N) = \frac{1}{Z} \prod_\alpha \psi_\alpha(y_\alpha)$$

- Special case: cliques of size 2 – pairwise Markov network

Markov Networks

- A **clique** in an undirected graph is a subset of its vertices such that every two vertices in the subset are connected by an edge.
- A **maximal clique** is a clique that cannot be extended by including one more adjacent vertex.
- For a set of variables $\mathbf{y} = \{y_1, \cdots, y_N\}$ a **Markov network** is defined as a product of potentials over the maximal cliques y_α of the graph G

$$p(y_1, \cdots, y_N) = \frac{1}{Z} \prod_{\alpha} \psi_\alpha(y_\alpha)$$

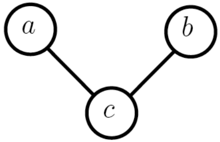
- Special case: cliques of size 2 – pairwise Markov network
- In case all potentials are strictly positive this is called a Gibbs distribution

Markov Networks

- A **clique** in an undirected graph is a subset of its vertices such that every two vertices in the subset are connected by an edge.
- A **maximal clique** is a clique that cannot be extended by including one more adjacent vertex.
- For a set of variables $\mathbf{y} = \{y_1, \cdots, y_N\}$ a **Markov network** is defined as a product of potentials over the maximal cliques y_α of the graph G

$$p(y_1, \cdots, y_N) = \frac{1}{Z} \prod_\alpha \psi_\alpha(y_\alpha)$$

- Special case: cliques of size 2 – pairwise Markov network
- In case all potentials are strictly positive this is called a Gibbs distribution
- Example: $p(a, b, c) = \frac{1}{Z}\psi_{a,c}(a, c)\psi_{bc}(b, c)$

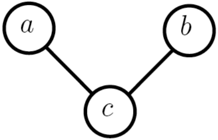


Markov Networks

- A **clique** in an undirected graph is a subset of its vertices such that every two vertices in the subset are connected by an edge.
- A **maximal clique** is a clique that cannot be extended by including one more adjacent vertex.
- For a set of variables $\mathbf{y} = \{y_1, \cdots, y_N\}$ a **Markov network** is defined as a product of potentials over the maximal cliques y_α of the graph G

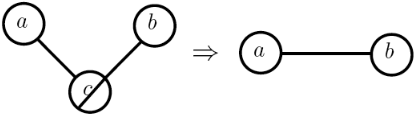
$$p(y_1, \cdots, y_N) = \frac{1}{Z} \prod_\alpha \psi_\alpha(y_\alpha)$$

- Special case: cliques of size 2 – pairwise Markov network
- In case all potentials are strictly positive this is called a Gibbs distribution
- Example: $p(a, b, c) = \frac{1}{Z} \psi_{a,c}(a, c) \psi_{bc}(b, c)$

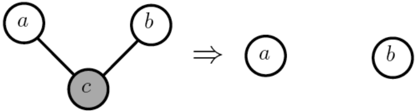


Properties of Markov Network

- **Marginalizing** over c makes *a* and *b* dependent



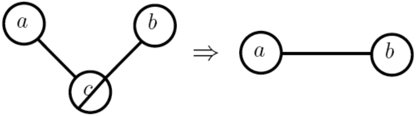
- **Conditioning** on *c* makes *a* and *b* independent



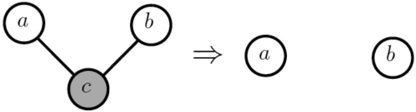
[Source: P. Gehler]

Properties of Markov Network

- **Marginalizing** over c makes *a* and *b* dependent



- **Conditioning** on *c* makes *a* and *b* independent



[Source: P. Gehler]

Local and Global Markov properties

- **Local Markov property**: condition on neighbours makes indep. of the rest

$$p(y_i|\mathbf{y} \setminus \{y_i\}) = p(y|ne(y_i))$$

  Example: $y_4 \perp \{y_1, y_7\} | \{y_2, y_3, y_5, y_6\}$

- **Global Markov Property**: For disjoint sets of variables $(\mathcal{A}, \mathcal{B}, \mathcal{S})$, where $\mathcal{S}$ separates $\mathcal{A}$ from $\mathcal{B}$ then $\mathcal{A} \perp \mathcal{B} | \mathcal{S}$

Local and Global Markov properties

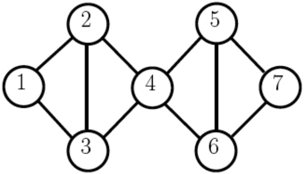
- **Local Markov property**: condition on neighbours makes indep. of the rest

$$p(y_i|\mathbf{y} \setminus \{y_i\}) = p(y|ne(y_i))$$

  Example: $y_4 \perp \{y_1, y_7\} | \{y_2, y_3, y_5, y_6\}$

- **Global Markov Property**: For disjoint sets of variables $(\mathcal{A}, \mathcal{B}, \mathcal{S})$, where $\mathcal{S}$ separates $\mathcal{A}$ from $\mathcal{B}$ then $\mathcal{A} \perp \mathcal{B} | \mathcal{S}$

- $\mathcal{S}$ is called a **separator**.
  Example: $y_1 \perp y_7 | \{y_4\}$



[Source: P. Gehler]

Local and Global Markov properties

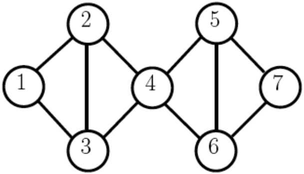
- **Local Markov property**: condition on neighbours makes indep. of the rest

$$p(y_i|\mathbf{y} \setminus \{y_i\}) = p(y|ne(y_i))$$

  Example: $y_4 \perp \{y_1, y_7\} | \{y_2, y_3, y_5, y_6\}$

- **Global Markov Property**: For disjoint sets of variables $(\mathcal{A}, \mathcal{B}, \mathcal{S})$, where $\mathcal{S}$ separates $\mathcal{A}$ from $\mathcal{B}$ then $\mathcal{A} \perp \mathcal{B} | \mathcal{S}$

- $\mathcal{S}$ is called a **separator**.
  Example: $y_1 \perp y_7 | \{y_4\}$



[Source: P. Gehler]

Relationship Potentials to Graphs

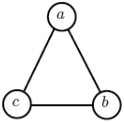
- Consider

$$p(a, b, c) = \frac{1}{Z}\psi(a, b)\psi(b, c)\psi(c, a)$$

- What is the corresponding Markov network (graphical representation)?

Relationship Potentials to Graphs

- Consider
$$p(a, b, c) = \frac{1}{Z}\psi(a, b)\psi(b, c)\psi(c, a)$$
- What is the corresponding Markov network (graphical representation)?



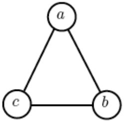
- Which other factorization is represented by this network?

Relationship Potentials to Graphs

- Consider

$$p(a,b,c) = \frac{1}{Z}\psi(a,b)\psi(b,c)\psi(c,a)$$

- What is the corresponding Markov network (graphical representation)?



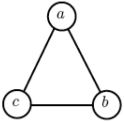
- Which other factorization is represented by this network?

Relationship Potentials to Graphs

- Consider

$$p(a, b, c) = \frac{1}{Z}\psi(a, b)\psi(b, c)\psi(c, a)$$

- What is the corresponding Markov network (graphical representation)?



- Which other factorization is represented by this network?

$$p(a, b, c) = \frac{1}{Z}\psi(a, b, c)$$

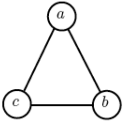
- The factorization is not specified by the graph

Relationship Potentials to Graphs

- Consider

$$p(a, b, c) = \frac{1}{Z}\psi(a, b)\psi(b, c)\psi(c, a)$$

- What is the corresponding Markov network (graphical representation)?



- Which other factorization is represented by this network?

$$p(a, b, c) = \frac{1}{Z}\psi(a, b, c)$$

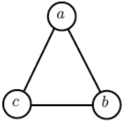
- The factorization is not specified by the graph
- Let's look at Factor Graphs

Relationship Potentials to Graphs

- Consider

$$p(a,b,c) = \frac{1}{Z}\psi(a,b)\psi(b,c)\psi(c,a)$$

- What is the corresponding Markov network (graphical representation)?

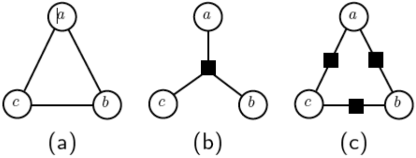


- Which other factorization is represented by this network?

$$p(a,b,c) = \frac{1}{Z}\psi(a,b,c)$$

- The factorization is not specified by the graph
- Let's look at Factor Graphs

Now consider we introduce an extra node (a square) for each factor



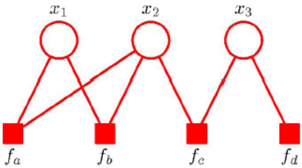
The **factor graph (FG)** has a node (represented by a square) for each factor $\psi(y_\alpha)$ and a variable node (represented by a circle) for each variable x_i.

- Left: Markov Network
- Middle: Factor graph representation of $\psi(a, b, c)$
- Right: Factor graph representation of $\psi(a, b)\psi(b, c)\psi(c, a)$
- Different factor graphs can have the same Markov network

[Source: P. Gehler]

Examples

- Which distribution?



- What factor graph?

$$p(x_1, x_2, x_3) = p(x_1)p(x_2)p(x_3|x_1, x_2)$$

[Source: P. Gehler]

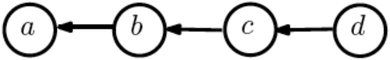
Inference in trees

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



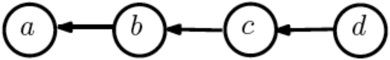
with distribution

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

- 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

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

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

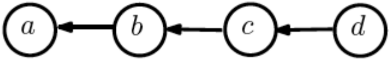
$$p(a) = \sum_{b} p(a \mid b) \gamma_c(b)$$

- 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:



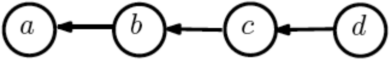
$$p(a,b,c,d) = p(a \mid b)p(b \mid c)p(c \mid d)p(d)$$

- 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]

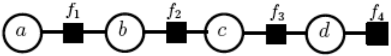
Finding Conditional Marginals



- 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



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

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

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

[Source: P. Gehler]

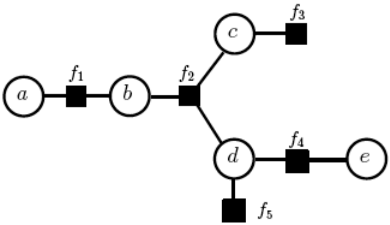
Inference in Chain Structured Factor Graphs

- 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:



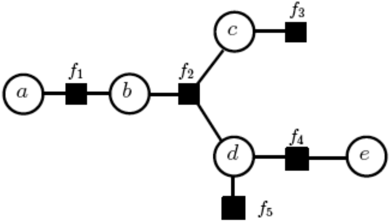
with factors

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

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

[Source: P. Gehler]

General singly-connected factor graphs II

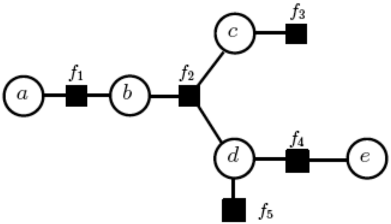


$$p(a,b) = f_1(a,b) \underbrace{\sum_{c,d,e} f_2(b,c,d) f_3(c) f_5(d) f_4(d,e)}_{\mu_{f_2 \to b}(b)}$$

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

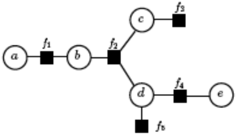
[Source: P. Gehler]

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]



- 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)

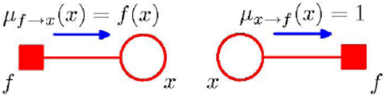
The algorithm is composed of

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

[Source: P. Gehler]

1. Initialization

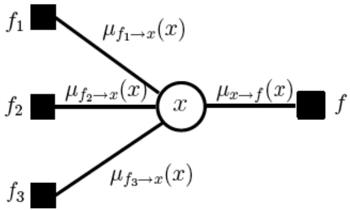
- Messages from extremal (simplical) node factors are initialized to the factor (left)
- Messages from extremal (simplical) variable nodes are set to unity (right)



[Source: P. Gehler]

2. Variable to Factor message

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

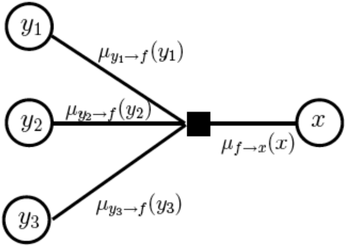


[Source: P. Gehler]

3. Factor to Variable message

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

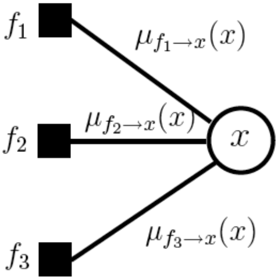
$$\mu_{f \to x}(x) = \sum_{y \in \mathcal{X}_f \setminus x} \phi_f(\mathcal{X}_f) \prod_{y \in \{\mathrm{ne}(f) \setminus x\}} \mu_{y \to f}(y)$$



[Source: P. Gehler]

Marginal computation

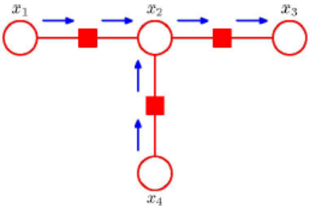
$$p(x) \propto \prod_{f \in \text{ne}(x)} \mu_{f \to x}(x)$$

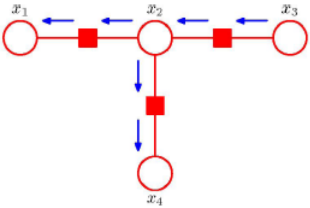


[Source: P. Gehler]

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)

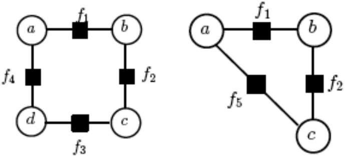




[Source: P. Gehler]

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 $x \in \mathcal{X}$)

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

[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

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

then becomes

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

[Source: P. Gehler]

Log Messages

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

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

then becomes

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

[Source: P. Gehler]

Trick

- Log-Factor-to-Variable Message:

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

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

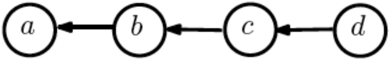
[Source: P. Gehler]

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

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

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

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

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

- 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

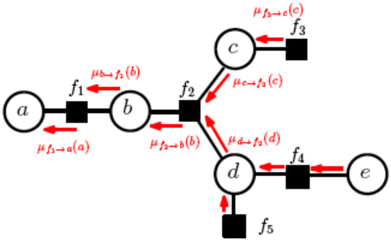
$$\begin{aligned} x_1^* &= \operatorname*{argmax}_{x_1} \gamma(x_1) \\ x_2^* &= \operatorname*{argmax}_{x_2} \phi(x_1^*, x_2)\gamma(x_2) \\ x_3^* &= \operatorname*{argmax}_{x_3} \phi(x_2^*, x_3)\gamma(x_3) \\ x_4^* &= \operatorname*{argmax}_{x_4} \phi(x_3^*, x_4)\gamma(x_4) \end{aligned}$$

- 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

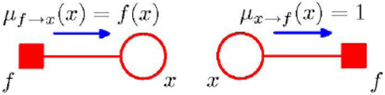
1. Initialisation (same as sum-product)
2. Variable to Factor message (same as sum-product)
3. Factor to Variable message

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



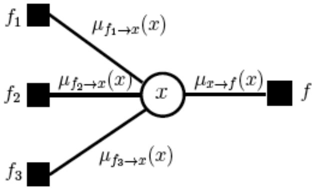
- Same as sum product

[Source: P. Gehler]

2. Variable to Factor message

- Same as for sum-product

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

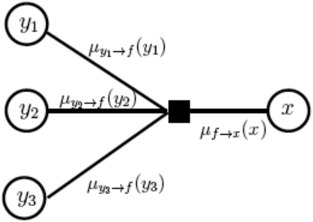


[Source: P. Gehler]

3. Factor to Variable message

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

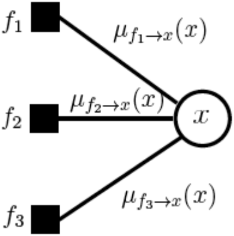
$$\mu_{f \to x}(x) = \max_{y \in \mathcal{X}_f \backslash x} \phi_f(\mathcal{X}_f) \prod_{y \in \{\text{ne}(f) \backslash x\}} \mu_{y \to f}(y)$$



[Source: P. Gehler]

Maximal state of Variable

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



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

[Source: P. Gehler]

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