Optimization of Markov Random Field in Computer Vision

Thalaiyasingam Ajanthan

The Australian National University

Data61, CSIRO

Data61, April 2017
Collaborators

Richard Hartley (primary supervisor)

Hongdong Li (panel chair)

Mathieu Salzmann (co-supervisor)

Philip Torr
Pawan Kumar
Alban Desmaison
Rudy Bunel
Outline

Introduction

Memory Efficient Max Flow

Iteratively Reweighted Graph Cut

Efficient Linear Programming

Conclusion
Outline

Introduction

Memory Efficient Max Flow

Iteratively Reweighted Graph Cut

Efficient Linear Programming

Conclusion
A Pairwise Markov Random Field

\[ E(\mathbf{x}) = \sum_{i \in \mathcal{V}} \theta_i(x_i) + \sum_{(i,j) \in \mathcal{E}} \theta_{ij}(x_i, x_j), \]

where \( x_i \in \mathcal{L} \) for all \( i \in \mathcal{V} \).

- \( \theta_i \) Unary potentials (data)
- \( \theta_{ij} \) Pairwise potentials (regularizer)
- \( \mathcal{V} \) Set of vertices (\( n \))
- \( \mathcal{E} \) Set of edges (\( m \))
- \( \mathcal{L} \) Set of labels (\( \ell \))

**Optimization**

\[ \mathbf{x}^* = \arg\min_{\mathbf{x} \in \mathcal{L}^n} E(\mathbf{x}). \]
A Pairwise Markov Random Field

\[ E(\mathbf{x}) = \sum_{i \in \mathcal{V}} \theta_i(x_i) + \sum_{(i,j) \in \mathcal{E}} \theta_{ij}(x_i, x_j) , \]

where \( x_i \in \mathcal{L} \) for all \( i \in \mathcal{V} \).

- \( \theta_i \) Unary potentials (data)
- \( \theta_{ij} \) Pairwise potentials (regularizer)

\( \mathcal{V} \) Set of vertices (\( n \))
\( \mathcal{E} \) Set of edges (\( m \))
\( \mathcal{L} \) Set of labels (\( \ell \))

Optimization

\[ \mathbf{x}^* = \arg\min_{x \in \mathcal{L}^n} E(\mathbf{x}) . \]
A Pairwise Markov Random Field

\[ E(\mathbf{x}) = \sum_{i \in \mathcal{V}} \theta_i(x_i) + \sum_{(i,j) \in \mathcal{E}} \theta_{ij}(x_i, x_j), \]

where \( x_i \in \mathcal{L} \) for all \( i \in \mathcal{V} \).

- \( \theta_i \) Unary potentials (data)
- \( \theta_{ij} \) Pairwise potentials (regularizer)
- \( \mathcal{V} \) Set of vertices (\( n \))
- \( \mathcal{E} \) Set of edges (\( m \))
- \( \mathcal{L} \) Set of labels (\( \ell \))

**Optimization**

\[ \mathbf{x}^* = \arg\min_{\mathbf{x} \in \mathcal{L}^n} E(\mathbf{x}). \]

Intractable
Computer Vision Applications

Stereo

\[ V \] Set of pixels
\[ E \] 4-connected neighbourhood
\[ \mathcal{L} \] Set of disparities, \( \{0, \ldots, \kappa\} \)
Computer Vision Applications

Inpainting

\( \mathcal{V} \)  Set of pixels
\( \mathcal{E} \)  4-connected neighbourhood
\( \mathcal{L} \)  Set of intensities, \( \{0, \ldots, 255\} \)
Computer Vision Applications

Segmentation

\[ \mathcal{N} \quad \text{Set of pixels} \]
\[ \mathcal{E} \quad \text{Fully connected neighbourhood} \]
\[ \mathcal{L} \quad \text{Set of object classes} \]
Contributions

Three new algorithms.

Memory Efficient Max Flow (MEMF)

- A max-flow algorithm with $O(\ell)$ memory reduction for multi-label submodular MRFs.

Iteratively Reweighted Graph Cut (IRGC)

- A move-making algorithm that can handle robust non-convex priors.

Efficient Linear Programming (PROX-LP)

- An LP minimization algorithm for dense CRFs that has linear time iterations.
Contributions

Three new algorithms.

Memory Efficient Max Flow (MEMF)

▶ A max-flow algorithm with $\mathcal{O}(\ell)$ memory reduction for multi-label submodular MRFs.

Iteratively Reweighted Graph Cut (IRGC)

▶ A move-making algorithm that can handle robust non-convex priors.

Efficient Linear Programming (PROX-LP)

▶ An LP minimization algorithm for dense CRFs that has linear time iterations.
Contributions

Three new algorithms.

Memory Efficient Max Flow (MEMF)

- A max-flow algorithm with $O(\ell)$ memory reduction for multi-label submodular MRFs.

Iteratively Reweighted Graph Cut (IRGC)

- A move-making algorithm that can handle robust non-convex priors.

Efficient Linear Programming (PROX-LP)

- An LP minimization algorithm for dense CRFs that has linear time iterations.
Contributions

Three new algorithms.

Memory Efficient Max Flow (MEMF)

- A max-flow algorithm with $\mathcal{O}(\ell)$ memory reduction for multi-label submodular MRFs.

Iteratively Reweighted Graph Cut (IRGC)

- A move-making algorithm that can handle robust non-convex priors.

Efficient Linear Programming (PROX-LP)

- An LP minimization algorithm for dense CRFs that has linear time iterations.
Outline

Introduction

Memory Efficient Max Flow

Iteratively Reweighted Graph Cut

Efficient Linear Programming

Conclusion
Introduction

Minimize

\[ E(x) = \sum_{i \in V} \theta_i(x_i) + \sum_{(i,j) \in E} \theta_{ij}(x_i, x_j) , \]

where \( x_i \in \mathcal{L} = \{0, 1, \cdots, \ell - 1\} \).

Multi-label submodular

\[ \theta_{ij}(\lambda', \mu) + \theta_{ij}(\lambda, \mu') - \theta_{ij}(\lambda, \mu) - \theta_{ij}(\lambda', \mu') \geq 0 , \]

for all \( \lambda, \lambda', \mu, \mu' \) where \( \lambda < \lambda' \) and \( \mu < \mu' \) [Schlesinger-2006].

E.g. \( \theta_{ij} \) is convex.

Current method

- Ishikawa algorithm [Ishikawa-2003, Schlesinger-2006].
Introduction

Minimize

\[ E(x) = \sum_{i \in V} \theta_i(x_i) + \sum_{(i,j) \in E} \theta_{ij}(x_i, x_j), \]

where \( x_i \in \mathcal{L} = \{0, 1, \ldots, \ell - 1\} \).

Multi-label submodular

\[ \theta_{ij}(\lambda', \mu) + \theta_{ij}(\lambda, \mu') - \theta_{ij}(\lambda, \mu) - \theta_{ij}(\lambda', \mu') \geq 0, \]

for all \( \lambda, \lambda', \mu, \mu' \) where \( \lambda < \lambda' \) and \( \mu < \mu' \) [Schlesinger-2006].

E.g. \( \theta_{ij} \) is convex.

Current method

- Ishikawa algorithm [Ishikawa-2003, Schlesinger-2006].
Introduction

Minimize

\[ E(x) = \sum_{i \in V} \theta_i(x_i) + \sum_{(i,j) \in E} \theta_{ij}(x_i, x_j) , \]

where \( x_i \in \mathcal{L} = \{0, 1, \ldots, \ell - 1\} \).

Multi-label submodular

\[ \theta_{ij}(\lambda', \mu) + \theta_{ij}(\lambda, \mu') - \theta_{ij}(\lambda, \mu) - \theta_{ij}(\lambda', \mu') \geq 0 , \]

for all \( \lambda, \lambda', \mu, \mu' \) where \( \lambda < \lambda' \) and \( \mu < \mu' \) [Schlesinger-2006].

E.g. \( \theta_{ij} \) is convex.

Current method

- Ishikawa algorithm [Ishikawa-2003, Schlesinger-2006].
The Ishikawa Algorithm

The Ishikawa graph
The Ishikawa Algorithm

The Ishikawa graph
The Ishikawa Algorithm

Drawback

- Huge memory complexity: $O(m\ell^2)$.

Contribution

- An algorithm with memory complexity $O(m\ell)$. 
The Ishikawa Algorithm

Drawback

- Huge memory complexity: $O(ml^2)$.

Contribution

- An algorithm with memory complexity $O(ml)$.

$E.g. \ n = 10^6, \ \ell = 256$
$m \approx 2 \times 10^6$
Edges $\approx 2 \times 10^6 \times 2 \times 256^2$
Memory $\approx 1000$ GB

$\downarrow$

Memory $\approx 4$ GB
The Ishikawa Algorithm

Drawback

- Huge memory complexity: $\mathcal{O}(m\ell^2)$.

Contribution

- An algorithm with memory complexity $\mathcal{O}(m\ell)$.

<table>
<thead>
<tr>
<th>E.g. $n = 10^6$, $\ell = 256$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m \approx 2 \times 10^6$</td>
</tr>
<tr>
<td>Edges $\approx 2 \times 10^6 \times 2 \times 256^2$</td>
</tr>
<tr>
<td>Memory $\approx 1000$ GB</td>
</tr>
</tbody>
</table>

⇓

| Memory $\approx 4$ GB |

Memory reduction: $\mathcal{O}(\ell)$. 
The Ishikawa Algorithm

**Drawback**
- Huge memory complexity: $O(m\ell^2)$.

**Contribution**
- An algorithm with memory complexity $O(m\ell)$.

*E.g. $n = 10^6$, $\ell = 256$
$m \approx 2 \times 10^6$
Edges $\approx 2 \times 10^6 \times 2 \times 256^2$
Memory $\approx 1000$ GB

\[ \Downarrow \]

Memory $\approx 4$ GB

Memory reduction: $O(\ell)$. 
Max Flow on the Ishikawa Graph

Initial Ishikawa graph

Flow = 0
Max Flow on the Ishikawa Graph

Max-flow in progress

Flow = 0
Max Flow on the Ishikawa Graph

Max-flow in progress

Flow = 2
Max Flow on the Ishikawa Graph

Max-flow in progress

Flow = 2
Max Flow on the Ishikawa Graph

Max-flow in progress

Flow = 4
Max Flow on the Ishikawa Graph

\[ x_i = 2 \]
\[ x_j = 1 \]

Flow = 7

Min-cut
Memory Efficient Flow Encoding

**Idea:** Don’t store the residual graph but *exit-flows* between each pair of neighbouring columns.
Memory Efficient Flow Encoding

**Idea:** Don’t store the residual graph but exit-flows between each pair of neighbouring columns.

**Exit-flow:** Given flow $\psi$, an exit-flow is defined as

$$\Sigma_{ij;\lambda} = \sum_{\mu} \psi_{ij;\lambda\mu}.$$
Memory Efficient Flow Encoding

**Idea:** Don’t store the residual graph but exit-flows between each pair of neighbouring columns.

**Exit-flow:** Given flow $\psi$, an exit-flow is defined as

$$
\Sigma_{ij:\lambda} = \sum_{\mu} \psi_{ij:\lambda\mu}.
$$

The residual graph can be rapidly computed from the exit-flows.
Flow Equivalence - An Example

\[ U_i:2 \quad 1 \quad U_j:2 \quad -1 \]

\[ U_i:1 \quad 1 \quad U_j:1 \quad -1 \]

Exit-flows
Flow Equivalence - An Example

A reconstructed flow
Flow Equivalence - An Example

Another reconstructed flow
Flow Equivalence - An Example

Both reconstructions are equivalent

Both reconstructions are equivalent
Flow Equivalence - An Example

Both reconstructions are equivalent

Flow-loop $\equiv$ reparametrization.
Flow Reconstruction / Computing Residual Edges

\[ \phi_{ij}^0 \]
Flow reconstruction as a small max-flow problem
Memory Efficient Max Flow (MEMF)

Algorithm

**Require:** \( \phi^0 \triangleright \text{Initial Ishikawa capacities} \)
\( \Sigma \leftarrow 0 \triangleright \text{Initialize exit-flows} \)

repeat

\( P \leftarrow \text{augmenting-path}(\phi^0, \Sigma) \)
\( \Sigma \leftarrow \text{augment}(P, \phi^0, \Sigma) \)

until no augmenting paths possible

**Assumption:**
\( \phi^0 \) can be stored in an efficient manner.
Memory Efficient Max Flow (MEMF)

**Algorithm**

**Require:** $\phi^0$ $\triangleright$ Initial Ishikawa capacities

$\Sigma \leftarrow 0$ $\triangleright$ Initialize exit-flows

repeat

\[
P \leftarrow \text{augmenting-path}(\phi^0, \Sigma)
\]

\[
\Sigma \leftarrow \text{augment}(P, \phi^0, \Sigma)
\]

until no augmenting paths possible

**Assumption:**

$\phi^0$ can be stored in an efficient manner.

Memory complexity: $O(m\ell)$. 
Memory Efficient Max Flow (MEMF)

Algorithm

Require: \( \phi^0 \triangleright \) Initial Ishikawa capacities
\[ \Sigma \leftarrow 0 \] \( \triangleright \) Initialize exit-flows
repeat
\[ P \leftarrow \text{augmenting-path}(\phi^0, \Sigma) \]
\[ \Sigma \leftarrow \text{augment}(P, \phi^0, \Sigma) \]
until no augmenting paths possible

Assumption:
\( \phi^0 \) can be stored in an efficient manner.

Memory complexity: \( \mathcal{O}(m\ell) \).
Efficiently Finding an Augmenting Path

Simplified graph

- Unweighted sparse graph.
- Fewer augmenting paths.

Search-tree-recycling

- Good empirical performance.

Simplified graph representation
Efficiently Finding an Augmenting Path

Simplified graph

- Unweighted sparse graph.
- Fewer augmenting paths.

Search-tree-recycling

- Good empirical performance.

Simplified graph representation
Efficiently Finding an Augmenting Path

Simplified graph

- Unweighted sparse graph.
- Fewer augmenting paths.

Search-tree-recycling

- Good empirical performance.

Image from [Boykov-2004]
Augmentation

Maximum flow can be pushed using dynamic programming.
Augmentation

Augmenting path

Directed acyclic graph

Maximum flow can be pushed using dynamic programming.
Augmentation

Maximum flow can be pushed using dynamic programming.
## Results

<table>
<thead>
<tr>
<th>Problem Name</th>
<th>$\ell$</th>
<th>Memory [MB]</th>
<th>Time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BK</td>
<td>EIBFS</td>
<td>MEMF</td>
</tr>
<tr>
<td>Tsukuba 16</td>
<td>3195</td>
<td>2495</td>
<td>211</td>
</tr>
<tr>
<td>Venus 20</td>
<td>7626</td>
<td>5907</td>
<td>396</td>
</tr>
<tr>
<td>Sawtooth 20</td>
<td>7566</td>
<td>5860</td>
<td>393</td>
</tr>
<tr>
<td>Map 30</td>
<td>6454</td>
<td>4946</td>
<td>219</td>
</tr>
<tr>
<td>Cones 60</td>
<td>*72303</td>
<td>*55063</td>
<td>1200</td>
</tr>
<tr>
<td>Teddy 60</td>
<td>*72303</td>
<td>*55063</td>
<td>1200</td>
</tr>
<tr>
<td>KITTI 40</td>
<td>*88413</td>
<td>*67316</td>
<td>2215</td>
</tr>
<tr>
<td>Penguin 256</td>
<td>*173893</td>
<td>*130728</td>
<td>663</td>
</tr>
<tr>
<td>House 256</td>
<td>*521853</td>
<td>*392315</td>
<td>1986</td>
</tr>
</tbody>
</table>

*Comparison with other max-flow implementations*

- **BK** [Boykov-2004]
- **EIBFS** [Goldberg-2015]
Empirical time complexity:

\[ O(n \ell^3) \]
Empirical time complexity: $\mathcal{O}(n\ell^3)$. 
Summary

▶ We have introduced a memory efficient alternative to the Ishikawa algorithm.

**Publication:** CVPR, 2016 and submitted to PAMI, 2017

**Code:** https://github.com/tajanathan/memf
Outline

Introduction

Memory Efficient Max Flow

Iteratively Reweighted Graph Cut

Efficient Linear Programming

Conclusion
Introduction

Minimize

\[
E(x) = \sum_{i \in \mathcal{V}} \theta_i(x_i) + \sum_{(i,j) \in \mathcal{E}} \theta_{ij}(|x_i - x_j|),
\]

where \( x_i \in \mathcal{L} = \{0, 1, \ldots, \ell - 1\} \).

Graph cut algorithms

- \( \theta_{ij} \) convex \( \Rightarrow \) Ishikawa algorithm [Ishikawa-2003].
- \( \theta_{ij} \) concave \( \Rightarrow \) \( \alpha \)-expansion [Boykov-2001].
- \( \theta_{ij} \) non-convex \( \Rightarrow \) IRGC [Ajanthan-2015].
Introduction

Minimize

\[ E(x) = \sum_{i \in V} \theta_i(x_i) + \sum_{(i,j) \in E} \theta_{ij}(|x_i - x_j|) , \]

where \( x_i \in \mathcal{L} = \{0, 1, \cdots, \ell - 1\} \).

Graph cut algorithms

- \( \theta_{ij} \) convex ⇒ Ishikawa algorithm [Ishikawa-2003].
- \( \theta_{ij} \) concave ⇒ \( \alpha \)-expansion [Boykov-2001].
- \( \theta_{ij} \) non-convex ⇒ IRGC [Ajanthan-2015].
Introduction

Minimize

\[ E(x) = \sum_{i \in V} \theta_i(x_i) + \sum_{(i,j) \in E} \theta_{ij}(|x_i - x_j|) , \]

where \( x_i \in \mathcal{L} = \{0, 1, \cdots, \ell - 1\} \).

Graph cut algorithms

- \( \theta_{ij} \) convex \( \Rightarrow \) Ishikawa algorithm [Ishikawa-2003].
- \( \theta_{ij} \) concave \( \Rightarrow \) \( \alpha \)-expansion [Boykov-2001].
- \( \theta_{ij} \) non-convex \( \Rightarrow \) IRGC [Ajanthan-2015].

\begin{align*}
\text{Trun. quad.} & \quad \text{Cauchy} & \quad \text{Cor. Gauss.}
\end{align*}
Optimal expansion move is found using max-flow.
α-expansion

Pairwise potential

Expand green

- Optimal expansion move is found using max-flow.
Optimal expansion move is found using \textit{max-flow}.
\(\alpha\)-expansion

Pairwise potential

Expand *light-green*

- Optimal expansion move is found using *max-flow*. 
Optimal expansion move is found using \textit{max-flow}.
Iteratively Reweighted Graph Cut (IRGC)

A move-making algorithm

- Minimizes the original MRF energy, by iteratively minimizing a multi-label submodular surrogate energy.
- Monotonic decrease of the original energy.

**Special case:** Iteratively Reweighted Least Squares (IRLS).
Iteratively Reweighted Graph Cut

Assumption

\[ \theta_{ij} (|x_i - x_j|) = h \circ g (|x_i - x_j|). \]

Non-decreasing concave \hspace{2cm} Convex

Minimize

\[ \tilde{E}(x) = \sum_{i \in V} \theta_i(x_i) + \sum_{(i,j) \in E} w_{ij}^t g (|x_i - x_j|). \]

Depends on the function \( h \) and the current labelling \( x^t \).
Iteratively Reweighted Graph Cut

Assumption

\[ \theta_{ij} (|x_i - x_j|) = h \circ g (|x_i - x_j|). \]

Non-decreasing concave \hspace{2cm} Convex

Minimize

\[ \tilde{E}(x) = \sum_{i \in V} \theta_i(x_i) + \sum_{(i,j) \in E} w^t_{ij} g (|x_i - x_j|). \]

Depends on the function \( h \) and the current labelling \( x^t \).
Iteratively Reweighted Graph Cut

Assumption

\[ \theta_{ij} (|x_i - x_j|) = h \circ g (|x_i - x_j|). \]

Non-decreasing concave \hspace{1cm} Convex

\[ \text{Trun. quad.} \hspace{1cm} \text{Cauchy} \hspace{1cm} \text{Cor. Gauss.} \]

Minimize

\[ \tilde{E}(x) = \sum_{i \in V} \theta_i(x_i) + \sum_{(i,j) \in E} w_{ij}^t \cdot g (|x_i - x_j|). \]

Depends on the function \( h \) and the current labelling \( x^t \).
Iteratively Reweighted Graph Cut

Assumption

\[ \theta_{ij}(|x_i - x_j|) = h \circ g(|x_i - x_j|). \]

Non-decreasing concave \hspace{5cm} Convex

\begin{align*}
\text{Trun. quad.} & \quad \text{Cauchy} & \quad \text{Cor. Gauss.}
\end{align*}

Minimize

\[ \tilde{E}(\mathbf{x}) = \sum_{i \in V} \theta_{i}(x_i) + \sum_{(i,j) \in E} w_{ij}^t g(|x_i - x_j|). \]

Depends on the function \( h \) and the current labelling \( \mathbf{x}^t \).

\[ \tilde{E}(\mathbf{x}) \text{ is multi-label submodular.} \]
Choice of Functions $g$ and $h$

\[ \theta(z) = h \circ g(z) \, . \]

- Choose $g$ such that the number of edges in the Ishikawa graph is minimized.

$\theta$ - Cauchy function
Hybrid Strategy

- Updates $x^t \rightarrow x^{t+1}$ in two steps:
  1. $x^t \rightarrow x'$ ⇒ Ishikawa algorithm.
  2. $x' \rightarrow x^{t+1}$ ⇒ One pass of $\alpha$-expansion.

- Effective to overcome local minima.
Results

- We evaluated on stereo and inpainting problems.

Map, Trunc. linear

Cones, Cauchy

Venus, Trunc. quad.

Penguin, Trunc. quad.
## Results

<table>
<thead>
<tr>
<th>Problem</th>
<th>$\alpha$-exp. (QPBO)</th>
<th>$\alpha\beta$ swap</th>
<th>TRWS</th>
<th>IRGC</th>
<th>IRGC+exp.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>1.05%</td>
<td>4.59%</td>
<td>0.05%</td>
<td>0.74%</td>
<td>0.17%</td>
</tr>
<tr>
<td>Teddy</td>
<td>0.75%</td>
<td>2.40%</td>
<td>0.30%</td>
<td>1.63%</td>
<td>0.21%</td>
</tr>
<tr>
<td>Venus</td>
<td>4.26%</td>
<td>4.85%</td>
<td>0.91%</td>
<td>0.35%</td>
<td>0.26%</td>
</tr>
<tr>
<td>Sawtooth</td>
<td>3.42%</td>
<td>4.58%</td>
<td>0.65%</td>
<td>0.96%</td>
<td>0.26%</td>
</tr>
<tr>
<td>Cones</td>
<td>7.80%</td>
<td>95.11%</td>
<td>0.16%</td>
<td>0.01%</td>
<td>0.01%</td>
</tr>
<tr>
<td>Tsukuba</td>
<td>1.99%</td>
<td>3.45%</td>
<td>0.09%</td>
<td>0.47%</td>
<td>0.17%</td>
</tr>
<tr>
<td>Penguin</td>
<td>6.71%</td>
<td>8.53%</td>
<td>1.56%</td>
<td>11.72%</td>
<td>0.83%</td>
</tr>
<tr>
<td>House</td>
<td>4.59%</td>
<td>3.71%</td>
<td>0.02%</td>
<td>0.01%</td>
<td>0.01%</td>
</tr>
<tr>
<td>Average</td>
<td>3.82%</td>
<td>15.90%</td>
<td>0.47%</td>
<td>1.99%</td>
<td>0.24%</td>
</tr>
</tbody>
</table>

**Quality of the minimum energies**

TRWS [Kolmogorov-2006]
We have introduced a move-making algorithm that is effective on multi-label MRFs with non-convex priors.

Publication: CVPR, 2015

Code: https://github.com/tajanthan/irgc
Outline

Introduction

Memory Efficient Max Flow

Iteratively Reweighted Graph Cut

Efficient Linear Programming

Conclusion
Introduction

Minimize

\[ E(x) = \sum_{i \in \mathcal{V}} \theta_i(x_i) + \sum_{(i,j) \in \mathcal{E}} \theta_{ij}(x_i, x_j), \]

where \( x_i \in \mathcal{L}, \mathcal{V} = \{1, \ldots, n\} \) and \( \mathcal{E} = \{(i, j) \mid i, j \in \mathcal{V}, i \neq j\} \).

Gaussian pairwise potentials

\[ \theta_{ij}(x_i, x_j) = 1[x_i \neq x_j] \exp \left( -\frac{\|f_i - f_j\|^2}{2} \right), \]

where \( f_i \in \mathbb{R}^d \).

Why?

- Captures long-range interactions and provides fine grained segmentations [Krähenbühl-2011].
Introduction

Minimize

\[ E(\mathbf{x}) = \sum_{i \in \mathcal{V}} \theta_i(x_i) + \sum_{(i,j) \in \mathcal{E}} \theta_{ij}(x_i, x_j), \]

where \( x_i \in \mathcal{L} \), \( \mathcal{V} = \{1, \ldots, n\} \) and \( \mathcal{E} = \{(i,j) \mid i, j \in \mathcal{V}, i \neq j\} \).

Gaussian pairwise potentials

\[ \theta_{ij}(x_i, x_j) = \mathbb{1}[x_i \neq x_j] \exp \left( \frac{-\|\mathbf{f}_i - \mathbf{f}_j\|^2}{2} \right), \]

where \( \mathbf{f}_i \in \mathbb{R}^d \).

Why?

- Captures long-range interactions and provides fine grained segmentations [Krähnenbühl-2011].
Introduction

Minimize

\[ E(x) = \sum_{i \in V} \theta_i(x_i) + \sum_{(i,j) \in E} \theta_{ij}(x_i, x_j), \]

where \( x_i \in \mathcal{L} \), \( V = \{1, \ldots, n\} \) and \( E = \{(i,j) \mid i, j \in V, i \neq j\} \).

Gaussian pairwise potentials

\[ \theta_{ij}(x_i, x_j) = \begin{cases} \mathbb{1}[x_i \neq x_j] & \text{Label compatibility} \\ \exp \left( -\frac{||f_i - f_j||^2}{2} \right) & \text{Pixel compatibility} \end{cases}, \]

where \( f_i \in \mathbb{R}^d \).

Why?

- Captures long-range interactions and provides fine grained segmentations [Krähenbühl-2011].
Dense CRF

\[ E(x) = \sum_{i=1}^{n} \theta_i(x_i) + \sum_{i=1}^{n} \sum_{j=1}^{n} 1[x_i \neq x_j] \exp \left( -\frac{\|f_i - f_j\|^2}{2} \right). \]

Difficulty

- Requires \( O(n^2) \) computations \( \Rightarrow \) Infeasible.

Idea

- Approximate using the filtering method [Adams-2010] \( \Rightarrow O(n) \) computations.
Dense CRF

\[ E(\mathbf{x}) = \sum_{i=1}^{n} \theta_i(x_i) + \sum_{i=1}^{n} \sum_{j=1}^{n} \mathbb{1}[x_i \neq x_j] \exp \left( -\frac{\|\mathbf{f}_i - \mathbf{f}_j\|^2}{2} \right). \]

**Difficulty**

- Requires \( \mathcal{O}(n^2) \) computations \( \Rightarrow \) Infeasible.

**Idea**

- Approximate using the filtering method [Adams-2010] \( \Rightarrow \mathcal{O}(n) \) computations.
Dense CRF

\[ E(\mathbf{x}) = \sum_{i=1}^{n} \theta_i(x_i) + \sum_{i=1}^{n} \sum_{\substack{j=1 \atop j \neq i}}^{n} \mathbb{1}[x_i \neq x_j] \exp \left( \frac{-(\|f_i - f_j\|^2)}{2} \right) \]

Difficulty

- Requires \( \mathcal{O}(n^2) \) computations ⇒ Infeasible.

Idea

- Approximate using the filtering method [Adams-2010] ⇒ \( \mathcal{O}(n) \) computations.
Current Algorithms for MAP Inference in Dense CRFs

- Rely on the **efficient filtering method** [Adams-2010].

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time complexity per iteration</th>
<th>Theoretical bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean Field (MF) [1]</td>
<td>$\mathcal{O}(n)$</td>
<td>No</td>
</tr>
<tr>
<td>Quadratic Programming (QP) [2]</td>
<td>$\mathcal{O}(n)$</td>
<td>Yes</td>
</tr>
<tr>
<td>Difference of Convex (DC) [2]</td>
<td>$\mathcal{O}(n)$</td>
<td>Yes</td>
</tr>
<tr>
<td>Linear Programming (LP) [2]</td>
<td>$\mathcal{O}(n \log(n))$</td>
<td>Yes (best)</td>
</tr>
</tbody>
</table>

**Contribution**

- LP in $\mathcal{O}(n)$ time per iteration
  - An order of magnitude speedup.

[1] [Krähenbühl-2011]
[2] [Desmaison-2016]
Rely on the efficient filtering method [Adams-2010].

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time complexity per iteration</th>
<th>Theoretical bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean Field (MF) [1]</td>
<td>$O(n)$</td>
<td>No</td>
</tr>
<tr>
<td>Quadratic Programming (QP) [2]</td>
<td>$O(n)$</td>
<td>Yes</td>
</tr>
<tr>
<td>Difference of Convex (DC) [2]</td>
<td>$O(n)$</td>
<td>Yes</td>
</tr>
<tr>
<td>Linear Programming (LP) [2]</td>
<td>$O(n \log(n))$</td>
<td>Yes (best)</td>
</tr>
</tbody>
</table>

Contribution

- LP in $O(n)$ time per iteration
  ⇒ An order of magnitude speedup.

[1] [Krähenbühl-2011]
[2] [Desmaison-2016]
Current Algorithms for MAP Inference in Dense CRFs

- Rely on the efficient filtering method [Adams-2010].

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time complexity per iteration</th>
<th>Theoretical bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean Field (MF) [1]</td>
<td>$O(n)$</td>
<td>No</td>
</tr>
<tr>
<td>Quadratic Programming (QP) [2]</td>
<td>$O(n)$</td>
<td>Yes</td>
</tr>
<tr>
<td>Difference of Convex (DC) [2]</td>
<td>$O(n)$</td>
<td>Yes</td>
</tr>
<tr>
<td>Linear Programming (LP) [2]</td>
<td>$O(n \log(n))$</td>
<td>Yes (best)</td>
</tr>
</tbody>
</table>

**Contribution**

- LP in $O(n)$ time per iteration

  $\Rightarrow$ An order of magnitude speedup.

$\text{E.g. } n = 10^6 \Rightarrow 20$ times speedup.
\[ y_{i:\lambda} = 1 \quad \Rightarrow \quad x_i = \lambda. \]

\[
\min_y \tilde{E}(y) = \sum_i \sum_{\lambda} \theta_{i:\lambda} y_{i:\lambda} + \sum_{i,j \neq i} \sum_{\lambda} K_{ij} \frac{|y_{i:\lambda} - y_{j:\mu}|}{2},
\]

s.t. \( y \in \mathcal{S} = \left\{ y \left| \sum_{\lambda} y_{i:\lambda} = 1, \ i \in \mathcal{V} \right. \right. \left. \left. y_{i:\lambda} \in [0, 1], \ i \in \mathcal{V}, \ \lambda \in \mathcal{L} \right\}, \)

where \( K_{ij} = \exp \left( \frac{-\|f_i - f_j\|^2}{2} \right). \)

▶ Provides integrality gap of 2 [Kleinberg-2002].
LP Relaxation of a Dense CRF

\[ y_{i: \lambda} = 1 \quad \Rightarrow \quad x_i = \lambda. \]

\[
\min_{\tilde{y}} \quad \tilde{E}(\tilde{y}) = \sum_i \sum_{\lambda} \theta_{i: \lambda} y_{i: \lambda} + \sum_{i,j \neq i} \sum_{\lambda} K_{ij} \frac{|y_{i: \lambda} - y_{j: \mu}|}{2},
\]

s.t. \[ y \in \mathcal{S} = \left\{ y \left| \sum_{\lambda} y_{i: \lambda} = 1, \: i \in \mathcal{V} \right. \right. \]
\[ \left. \left. \left. \left. \sum_{\lambda} y_{i: \lambda} \in [0, 1], \: i \in \mathcal{V}, \: \lambda \in \mathcal{L} \right. \right. \right. \}

where \[ K_{ij} = \exp \left( -\frac{\|f_i - f_j\|^2}{2} \right). \]

- Provides integrality gap of 2 [Kleinberg-2002].
LP Relaxation of a Dense CRF

\[ y_{i: \lambda} = 1 \Rightarrow x_i = \lambda. \]

\[
\begin{align*}
\min_y \quad & \tilde{E}(y) = \sum_i \sum_{\lambda} \theta_{i: \lambda} y_{i: \lambda} + \sum_{i,j \neq i} \sum_{\lambda} K_{ij} \frac{|y_{i: \lambda} - y_{j: \mu}|}{2}, \\
\text{s.t.} \quad & y \in S = \left\{ y \left| \sum_{\lambda} y_{i: \lambda} = 1, \ i \in V \right. \right. \\
& \left. \left. y_{i: \lambda} \in [0, 1], \ i \in V, \ \lambda \in \mathcal{L} \right\} , \right.
\end{align*}
\]

where \( K_{ij} = \exp \left( \frac{-||f_i - f_j||^2}{2} \right) \).

\[ \Rightarrow \] Provides integrality gap of 2 [Kleinberg-2002].

Standard solvers would require \( O(n^2) \) variables.
LP Minimization

Current method

- Projected subgradient descent ⇒ Too slow.
  - Linearithmic time per iteration.
  - Expensive line search.
  - Requires large number of iterations.

Our algorithm

- Proximal minimization using block-coordinate descent.
  - One block: Significantly smaller subproblems.
  - The other block: Efficient conditional gradient descent.
    - Linear time per iteration.
    - Optimal step size.
  - Guarantees optimality and converges faster.
LP Minimization

Current method

- Projected subgradient descent ⇒ Too slow.
  - Linearithmic time per iteration.
  - Expensive line search.
  - Requires large number of iterations.

Our algorithm

- Proximal minimization using block-coordinate descent.
  - One block: Significantly smaller subproblems.
  - The other block: Efficient conditional gradient descent.
    - Linear time per iteration.
    - Optimal step size.
  - Guarantees optimality and converges faster.
Proximal Minimization of LP (PROX-LP)

\[
\min_y \tilde{E}(y) + \frac{1}{2\eta} \| y - y^r \|^2,
\]

s.t. \( y \in S \),

where \( \eta > 0 \) and \( y^r \) is the current estimate.

Why?

- Initialization using MF or DC.
- Smooth dual \( \Rightarrow \) Sophisticated optimization.

Approach

- Block-coordinate descent tailored to this problem.
Proximal Minimization of LP (PROX-LP)

\[
\min_y \tilde{E}(y) + \frac{1}{2\eta} \|y - y^r\|^2,
\]

\text{s.t.} \quad y \in S,

where \( \eta > 0 \) and \( y^r \) is the current estimate.

Why?

- Initialization using MF or DC.
- \textbf{Smooth dual} \( \Rightarrow \) Sophisticated optimization.

Approach

- Block-coordinate descent tailored to this problem.
Proximal Minimization of LP (PROX-LP)

\[
\min_y \tilde{E}(y) + \frac{1}{2\eta} \|y - y^r\|^2,
\]

\[\text{s.t.} \quad y \in S,\]

where \(\eta > 0\) and \(y^r\) is the current estimate.

Why?

- Initialization using MF or DC.
- **Smooth dual** ⇒ Sophisticated optimization.

Approach

- Block-coordinate descent tailored to this problem.
Dual of the Proximal Problem

\[
\begin{align*}
\min_{\alpha, \beta, \gamma} \, g(\alpha, \beta, \gamma) &= \frac{\eta}{2} \|A\alpha + B\beta + \gamma - \theta_u\|^2 \\
&\quad + \langle A\alpha + B\beta + \gamma - \theta_u, y^r \rangle - \langle 1, \beta \rangle,
\end{align*}
\]

s.t. \( \gamma_i: \lambda \geq 0 \quad \forall \, i \in \mathcal{V} \quad \forall \, \lambda \in \mathcal{L}, \)

\(\alpha \in \mathcal{C} = \left\{ \alpha \mid \begin{array}{l}
\alpha_{1ij}: \lambda + \alpha_{2ij}: \lambda = \frac{K_{ij}}{2}, \forall \, i, j \neq i, \lambda \in \mathcal{L} \\
\alpha_{1ij}: \lambda, \alpha_{2ij}: \lambda \geq 0, \forall \, i, j \neq i, \lambda \in \mathcal{L}
\end{array} \right\}.\)

Block-coordinate descent

- \(\beta\): Unconstrained \(\Rightarrow\) Set derivative to zero.
- \(\gamma\): Unbounded and separable \(\Rightarrow\) Small QP for each pixel.
- \(\alpha \in \mathcal{C}\): Compact domain \(\Rightarrow\) Conditional gradient descent.
Dual of the Proximal Problem

\[
\min_{\alpha, \beta, \gamma} g(\alpha, \beta, \gamma) = \frac{\eta}{2} \| A\alpha + B\beta + \gamma - \theta_u \|^2 \\
+ \langle A\alpha + B\beta + \gamma - \theta_u, y^r \rangle - \langle 1, \beta \rangle,
\]

s.t. \( \gamma_{i: \lambda} \geq 0 \) \( \forall i \in V \) \( \forall \lambda \in \mathcal{L} \),

\( \alpha \in \mathcal{C} \) = \( \left\{ \alpha \mid \alpha_{i:j: \lambda}^{1} + \alpha_{i:j: \lambda}^{2} = \frac{K_{ij}}{2}, \forall i, j \neq i, \lambda \in \mathcal{L} \right\} \).

Block-coordinate descent

- \( \beta \): Unconstrained \( \Rightarrow \) Set derivative to zero.
- \( \gamma \): Unbounded and separable \( \Rightarrow \) Small QP for each pixel.
- \( \alpha \in \mathcal{C} \): Compact domain \( \Rightarrow \) Conditional gradient descent.
Dual of the Proximal Problem

\[
\min_{\alpha, \beta, \gamma} g(\alpha, \beta, \gamma) = \frac{\eta}{2} \| A\alpha + B\beta + \gamma - \theta_u \|^2 \\
+ \langle A\alpha + B\beta + \gamma - \theta_u, y^r \rangle - \langle 1, \beta \rangle ,
\]

s.t. \( \gamma_{i: \lambda} \geq 0 \) \( \forall \ i \in V \ \forall \ \lambda \in \mathcal{L} \) ,

\( \alpha \in \mathcal{C} \) = \( \left\{ \alpha \mid \frac{\alpha_{1ij: \lambda}}{2} + \frac{\alpha_{2ij: \lambda}}{2} = \frac{K_{ij}}{2}, \ \forall \ i, j \neq i, \ \lambda \in \mathcal{L} \right\} \).

Block-coordinate descent

▶ \( \beta \): Unconstrained \( \Rightarrow \) Set derivative to zero.

▶ \( \gamma \): Unbounded and separable \( \Rightarrow \) Small QP for each pixel.

▶ \( \alpha \in \mathcal{C} \): Compact domain \( \Rightarrow \) Conditional gradient descent.
Dual of the Proximal Problem

\[
\min_{\alpha, \beta, \gamma} g(\alpha, \beta, \gamma) = \frac{\eta}{2} \|A\alpha + B\beta + \gamma - \theta_u\|^2 \\
+ \langle A\alpha + B\beta + \gamma - \theta_u, y^r \rangle - \langle 1, \beta \rangle,
\]

s.t. \( \gamma_i: \lambda \geq 0 \quad \forall \ i \in V \quad \forall \ \lambda \in \mathcal{L} \),

\[
\alpha \in \mathcal{C} = \left\{ \alpha \mid \alpha_{ij: \lambda}^1 + \alpha_{ij: \lambda}^2 = \frac{K_{ij}}{2}, \forall \ i, j \neq i, \lambda \in \mathcal{L} \right\}.
\]

Block-coordinate descent

- **\( \beta \): Unconstrained \( \Rightarrow \) Set derivative to zero.
- **\( \gamma \): Unbounded and separable \( \Rightarrow \) Small QP for each pixel.
- **\( \alpha \in \mathcal{C} \): Compact domain \( \Rightarrow \) Conditional gradient descent.
Dual of the Proximal Problem

\[
\min_{\alpha, \beta, \gamma} g(\alpha, \beta, \gamma) = \frac{\eta}{2} \| A\alpha + B\beta + \gamma - \theta_u \|^2 \\
+ \langle A\alpha + B\beta + \gamma - \theta_u, y^r \rangle - \langle 1, \beta \rangle,
\]

s.t.  \quad \gamma_{i:\lambda} \geq 0 \quad \forall \, i \in \mathcal{V} \quad \forall \, \lambda \in \mathcal{L},

\[
\alpha \in \mathcal{C} = \left\{ \alpha \mid \begin{array}{l}
\alpha^1_{ij:\lambda} + \alpha^2_{ij:\lambda} = \frac{K_{ij}}{2}, \forall \, i,j \neq i, \lambda \in \mathcal{L} \\
\alpha^1_{ij:\lambda}, \alpha^2_{ij:\lambda} \geq 0, \forall \, i,j \neq i, \lambda \in \mathcal{L}
\end{array} \right\}.
\]

Block-coordinate descent

\begin{itemize}
  \item \( \beta \): Unconstrained \quad \Rightarrow \quad \text{Set derivative to zero.}
  
  \item \( \gamma \): Unbounded and separable \quad \Rightarrow \quad \text{Small QP for each pixel.}
  
  \item \( \alpha \in \mathcal{C} \): Compact domain \quad \Rightarrow \quad \text{Conditional gradient descent.}
\end{itemize}

Guarantees optimality since \( g \) is strictly convex and smooth.
Dual of the Proximal Problem

\[
\min_{\alpha, \beta, \gamma} g(\alpha, \beta, \gamma) = \frac{\eta}{2} \| A\alpha + B\beta + \gamma - \theta_u \|^2 \\
+ \langle A\alpha + B\beta + \gamma - \theta_u, y^r \rangle - \langle 1, \beta \rangle ,
\]

s.t. \[ \gamma_{i:\lambda} \geq 0 \quad \forall i \in V \quad \forall \lambda \in L , \]

\[
\alpha \in \mathcal{C} = \left\{ \alpha \left| \alpha_{ij: \lambda}^1 + \alpha_{ij: \lambda}^2 = \frac{K_{ij}}{2}, \forall i, j \neq i, \lambda \in L \right. \right\}.
\]

Block-coordinate descent

▷ \( \beta \): Unconstrained \( \Rightarrow \) Set derivative to zero.

▷ \( \gamma \): Unbounded and separable \( \Rightarrow \) Small QP for each pixel.

▷ \( \alpha \in \mathcal{C} \): Compact domain \( \Rightarrow \) Conditional gradient descent.

Guarantees optimality since \( g \) is strictly convex and smooth.
Conditional Gradient Descent

\[
\min_{\alpha \in C} g(\alpha).
\]

Requirements

▶ \( g : C \rightarrow \mathbb{R} \) is differentiable.
▶ \( C \subseteq \mathbb{R}^N \) is convex and compact.

Conditional gradient (s)

▶ Minimize the first order Taylor approximation.

In our case

▶ Linear time conditional gradient computation.
▶ Optimal step size.

Image from [Lacoste-2012]
Conditional Gradient Descent

\[
\min_{\alpha \in C} g(\alpha) .
\]

Requirements

- \( g : C \rightarrow \mathbb{R} \) is differentiable.
- \( C \subset \mathbb{R}^N \) is convex and compact.

Conditional gradient (s)

- Minimize the first order Taylor approximation.

In our case

- Linear time conditional gradient computation.
- Optimal step size.

Image from [Lacoste-2012]
Conditional Gradient Descent

\[ \min_{\alpha \in C} g(\alpha). \]

Requirements

- \( g : C \to \mathbb{R} \) is differentiable.
- \( C \subset \mathbb{R}^N \) is convex and compact.

Conditional gradient (s)

- Minimize the first order Taylor approximation.

In our case

- Linear time conditional gradient computation.
- Optimal step size.

*Image from [Lacoste-2012]*
Conditional Gradient Computation

\[ \forall i \in \mathcal{V}, \quad \tilde{s}_i = \sum_j K_{ij} \mathbb{1}[y_i \geq y_j], \]

where \( K_{ij} = \exp\left(-\frac{\|f_i - f_j\|^2}{2}\right) \), \( y_i \in [0, 1] \) and \( f_i \in \mathbb{R}^d \).

Difficulty

- The permutohedral lattice based filtering method of [Adams-2010] cannot handle the ordering constraint.

Current method [Desmaison-2016]

- Repeated application of the original filtering method using a divide-and-conquer strategy \( \Rightarrow \mathcal{O}(d^2n \log(n)) \) computations.

Our idea

- Discretize the interval \([0, 1]\) to \( H \) levels and instantiate \( H \) permutohedral lattices \( \Rightarrow \mathcal{O}(Hdn) \) computations (\( H = 10 \)).
Conditional Gradient Computation

\[ \forall i \in V, \quad \tilde{s}_i = \sum_j K_{ij} \mathbb{1}[y_i \geq y_j], \]

where \( K_{ij} = \exp \left( -\frac{\|f_i - f_j\|^2}{2} \right) \), \( y_i \in [0, 1] \) and \( f_i \in \mathbb{R}^d \).

Difficulty

- The permutohedral lattice based filtering method of [Adams-2010] cannot handle the ordering constraint.

Current method [Desmaison-2016]

- Repeated application of the original filtering method using a divide-and-conquer strategy \( \Rightarrow \mathcal{O}(d^2 n \log(n)) \) computations.

Our idea

- Discretize the interval \([0, 1]\) to \( H \) levels and instantiate \( H \) permutohedral lattices \( \Rightarrow \mathcal{O}(Hdn) \) computations (\( H = 10 \)).
Conditional Gradient Computation

\[ \forall i \in \mathcal{V}, \quad \tilde{s}_i = \sum_j K_{ij} \mathbb{1}[y_i \geq y_j], \]

where \( K_{ij} = \exp \left( -\frac{\|f_i - f_j\|^2}{2} \right) \), \( y_i \in [0, 1] \) and \( f_i \in \mathbb{R}^d \).

Difficulty

- The permutohedral lattice based filtering method of [Adams-2010] cannot handle the ordering constraint.

Current method [Desmaison-2016]

- Repeated application of the original filtering method using a divide-and-conquer strategy \( \Rightarrow \mathcal{O}(d^2 n \log(n)) \) computations.

Our idea

- Discretize the interval \([0, 1]\) to \( H \) levels and instantiate \( H \) permutohedral lattices \( \Rightarrow \mathcal{O}(Hdn) \) computations (\( H = 10 \)).
Conditional Gradient Computation

\[ \forall i \in V, \quad \tilde{s}_i = \sum_j K_{ij} \mathbb{1}[y_i \geq y_j], \]

where \( K_{ij} = \exp \left( -\frac{\|f_i - f_j\|^2}{2} \right) \), \( y_i \in [0, 1] \) and \( f_i \in \mathbb{R}^d \).

Difficulty

▶ The permutohedral lattice based filtering method of [Adams-2010] cannot handle the ordering constraint.

Current method [Desmaison-2016]

▶ Repeated application of the original filtering method using a divide-and-conquer strategy \( \Rightarrow \mathcal{O}(d^2 n \log(n)) \) computations.

Our idea

▶ Discretize the interval \([0, 1]\) to \( H \) levels and instantiate \( H \) permutohedral lattices \( \Rightarrow \mathcal{O}(Hdn) \) computations (\( H = 10 \)).
Energy vs time plot for an image in (left) MSRC and (right) Pascal

- Both LP minimization algorithms are initialized with DC\textsubscript{neg}.
## Segmentation Results

<table>
<thead>
<tr>
<th></th>
<th>Avg. E ($\times 10^3$)</th>
<th>Avg. T (s)</th>
<th>Acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MSRC</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MF5</td>
<td>8078.0</td>
<td>0.2</td>
<td>79.33</td>
</tr>
<tr>
<td>MF</td>
<td>8062.4</td>
<td>0.5</td>
<td>79.35</td>
</tr>
<tr>
<td>DC\textsubscript{neg}</td>
<td>3539.6</td>
<td>1.3</td>
<td>83.01</td>
</tr>
<tr>
<td>SG-LP\textsubscript{\ell}</td>
<td>3335.6</td>
<td>13.6</td>
<td>83.15</td>
</tr>
<tr>
<td>PROX-LP\textsubscript{acc}</td>
<td><strong>1340.0</strong></td>
<td><strong>3.7</strong></td>
<td><strong>84.16</strong></td>
</tr>
<tr>
<td><strong>Pascal</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MF5</td>
<td>1220.8</td>
<td>0.8</td>
<td>79.13</td>
</tr>
<tr>
<td>MF</td>
<td>1220.8</td>
<td>0.7</td>
<td>79.13</td>
</tr>
<tr>
<td>DC\textsubscript{neg}</td>
<td>629.5</td>
<td>3.7</td>
<td>80.43</td>
</tr>
<tr>
<td>SG-LP\textsubscript{\ell}</td>
<td>617.1</td>
<td>84.4</td>
<td>80.49</td>
</tr>
<tr>
<td>PROX-LP\textsubscript{acc}</td>
<td><strong>507.7</strong></td>
<td><strong>14.7</strong></td>
<td><strong>80.58</strong></td>
</tr>
</tbody>
</table>

Results on the MSRC and Pascal datasets
Segmentation Results

Qualitative results on MSRC
Modified Filtering Method

Spatial kernel \((d = 2)\)

Bilateral kernel \((d = 5)\)

*Speedup of our modified filtering algorithm on a Pascal image*
Modified Filtering Method

Bilateral kernel ($d = 5$)

**Speedup of our modified filtering algorithm on a Pascal image**

Speedup is around **45 – 65** on the standard image.
Summary

- We have introduced the first LP minimization algorithm for dense CRFs whose iterations are linear in the number of pixels and labels.

Publication: CVPR, 2017

Code: https://github.com/oval-group/DenseCRF
Outline

Introduction

Memory Efficient Max Flow

Iteratively Reweighted Graph Cut

Efficient Linear Programming

Conclusion
Conclusion

- We have introduced three new algorithms for MRF optimization, targeting computer vision applications.

**MEMF**: A max-flow algorithm with $O(\ell)$ memory reduction for Ishikawa type graphs.

**IRGC**: A move-making algorithm that can handle robust non-convex priors.

**PROX-LP**: An LP minimization algorithm for dense CRFs that has linear time iterations.
Thank you!
Flow vs Reparametrization

Flow vs reparametrization
Finding an Augmenting Path

Find augmenting paths on a subgraph
Finding an Augmenting Path

Find augmenting paths on a subgraph

Utilize upward infinite capacity edges in each column.
Finding an Augmenting Path

Find augmenting paths on a subgraph

Overall time complexity: $O(nm\ell^6)$
Iteratively Reweighted Minimization

- Minimize the original energy $E(x) = \sum_k h_k \circ f_k(x)$, by iteratively minimizing a surrogate energy $	ilde{E}(x) = \sum_k w_k f_k(x)$.

Lemma (Monotonic decrease)

Given a set $\mathcal{X}$, functions $f_k : \mathcal{X} \to \mathcal{D}$ and concave functions $h_k : \mathcal{D} \to \mathbb{R}$, with $\mathcal{D} \subseteq \mathbb{R}$, such that,

\[
\sum_k w_k^t f_k(x^{t+1}) \leq \sum_k w_k^t f_k(x^t),
\]

where $w_k^t = h_k^s(f_k(x^t))$ and $x^t$ is the estimate of $x$ at iteration $t$, then

\[
\sum_k h_k \circ f_k(x^{t+1}) \leq \sum_k h_k \circ f_k(x^t).
\]
Permputohedral Lattice

A 2-dimensional hyperplane tessellated by the permutohedral lattice.
Modified Filtering Algorithm

Assignment energy as a function of time for an image in (left) MSRC and (right) Pascal