RIGOR: Reusing Inference in Graph Cuts for generating Object Regions

Ahmad Humayun  Fuxin Li  Jim Rehg

July 2014

http://cpl.cc.gatech.edu/projects/RIGOR
Problem Statement - Finding Figure-Ground Segments

Input Image, $I$

Hierarchy of Object Segments

Motivation

How to find objects?
If there is an object at a small selected location (seed) – what is the best segment

Current methods too slow …

<table>
<thead>
<tr>
<th>Method</th>
<th>Run Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPMC [1]</td>
<td>34.01</td>
</tr>
<tr>
<td>Object Proposals [2]</td>
<td>126.46</td>
</tr>
<tr>
<td>Shape Sharing [3]</td>
<td>410.31</td>
</tr>
</tbody>
</table>

Motivation

**CPMC:** More segments $\rightarrow$ Slower speed $\rightarrow$ Higher recall

Goal

Segments of similar quality in an order of magnitude less time …
Method Overview

Input Image, $I$

Probabilistic Boundaries [1,2,3]

(Superpixels)

Seeds from Superpixels

Method Overview

Input Image, $I$

1. Probabilistic Boundaries
2. (Superpixels)
3. Seeds from Superpixels
4. Parametric Min-Cut produces Segments
5. Filter Segments
Computation Time

When using *Pixel* graphs (CPMC)

Computation Time

Overheads
Segment Filteration
Parametric Min-cut
Unary Potentials
Pairwise Potentials
Structured Edges [3]

Total Time: 18.2s

… use Structured Edges [3]

Computation Time

… woah … from Pixels to Superpixels based Graphs

How can we reduce the parametric Min-cut computation time?

Our Contributions

1. Method to reuse information for MAP inference in different graphs with same pairwise costs, but different unary seeds.

2. Allow sharing information across graph-cut problems before the full cut is computed (allows parallelization).

3. An object segmentation method which is an order of magnitude faster, without loss in accuracy.
Related Work

Boykov and Jolly [1]

*Reusing Flows for Interactive Segmentation*

Kohli and Torr [2]

*Reusing Flows in Dynamic Graph-Cuts*


How to use Parametric Min-Cut?

Input Image, $I$

Labeling of all Superpixels

$$E^i_{\lambda}(X) = \sum_{u \in V} D^i_{\lambda}(x_u) + \sum_{(u,v) \in E} V_{uv}(x_u, x_v)$$

Pairwise Superpixel Potentials

Parametric Unaries

$$D^i_{\lambda}(x_u) = \infty$$

If $u \in S_i$ and assigned to background

$$D^i_{\lambda}(x_u) = f(x_u) + \lambda$$

Parametric unaries otherwise
How to use Parametric Min-Cut?

Input Image with seeds

increasing $\lambda$
Grow Trees

Preliminary: Boykov-Kolmogorov (BK)

Grow Trees

Augment Flow

Preliminary: Boykov-Kolmogorov (BK)

Grow Trees

Augment Flow

Adoption

[T sink tree edge]

[S sink tree edge]

[Augmenting path from S to T]

[Saturated edge on augmenting path]

[Orphan tree]

Problem Statement

Given $N$ energy functions for parametric min-cut, find what information can be shared for their minimization (MAP). $V_{uv}$ remains same across the functions.

**Seed:** $D^i_\lambda(x_u) = \infty$ iff $x_u \in S_i$ and $x_u = 0$. **Condition:** $S_i \cap S_j = \emptyset$, for all $i, j$

$$E^i_\lambda(X) = \sum_{u \in V} D^i_\lambda(x_u) + \sum_{(u,v) \in E} V_{uv}(x_u, x_v)$$

Share information

$$E^j_\lambda(X) = \sum_{u \in V} D^j_\lambda(x_u) + \sum_{(u,v) \in E} V_{uv}(x_u, x_v)$$
Key Insight

Collect all parametric min-cut segments.
  - Count how many times each superpixel edgelet was in the cut

Lots of white edges, which are never used in a cut
Key Insight

~54% boundaries never in the cut
- Share information across Graph-cuts – communicate that some edglets never in cut

53.5% edglets never used
What can we Reuse?

BK spends time creating trees – reusing them is useful
Idea - generate trees that are useful across all seeds

1st step: combine all seeds into one precomputation graph

Seed $S_1$ graph $\cup$ Seed $S_2$ graph $=$ Precomputation graph
Idea - generate trees that are useful across all seeds

1st step: combine all seeds into one precomputation graph

Example

Seed $S_1$ graph

Seed $S_2$ graph

Precomputation graph

$S_0 \cup \cdots \cup S_{n-1} = S_0 \cup \cdots \cup S_{n-1}$
Reparameterization \[1,2\]

Changes flow value but not the cut! As long as \(c_s - c_t\) remains same

---

**Key Idea**

\[
\begin{align*}
S & \quad c_s = 5 \\
\quad c_s^* = 1 \\
\equiv & \quad n \\
T & \quad c_t = 4
\end{align*}
\]

**Eg.**

\[
\begin{align*}
G_1 & \quad f_s = 4 \\
& \quad c_s = 9 \\
& \quad c_s' = 3 \\
G_2 & \quad f_s = 4 \\
& \quad c_s = 9 + 1 \\
& \quad c_s' = 4 \\
\quad c_t = +1 \\
\quad c_t' = 1
\end{align*}
\]

---

1. Add capacity

2. Reparameterize

---


2nd step: Run Boykov-Kolmogorov on precomputation graph

Notice, we now have two $S$ trees and one $T$ tree
3rd step: Convert Precomputation Graph to compute max-flow for each seed

Solved Precomputation graph $\rightarrow$ Reparameterized to use for seed $S_1$ $\rightarrow$ Maxflow for seed $S_1$

Note: we needed to convert tree $S_2$ to $T$
Reusing computation by Precomputation Graph

3rd step: Repeat for all seeds

Solved Precomputation graph $\rightarrow$ Reparameterized to use for seed $S_2$ $\rightarrow$ Maxflow for seed $S_2$

Tree $S_1$ converted to $T$

Note: we get the same cuts, because we are just reparameterizing
Why is the Precomputation Graph useful?

BK spends time creating trees – reusing them is useful

Solved Precomputation graph

Cut for seed $S_1$

Cut for seed $S_2$

Completely reused from Precomputation Graph
Speed-up with Precomputation Graph

Parametric Min Cut time savings

Compared to Boykov Kolmogorov [1]

Compared to Kohli & Torr [2]

Faster Pipeline

Pipeline timing comparison to Object Proposals [1]

## Quantitative Comparison

<table>
<thead>
<tr>
<th>Method</th>
<th>Mean Best Overlap</th>
<th>Mean Best Covering</th>
<th>Run Time (s)</th>
<th># of Segments</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPMC</td>
<td>70.67</td>
<td>82.24</td>
<td>34.01</td>
<td>624.1</td>
</tr>
<tr>
<td>Object Proposals</td>
<td>71.48</td>
<td>80.98</td>
<td>126.46</td>
<td>1544.1</td>
</tr>
<tr>
<td>Shape Sharing</td>
<td>67.82</td>
<td>82.71</td>
<td>410.31</td>
<td>1115.4</td>
</tr>
<tr>
<td>GB, 25 seeds</td>
<td>68.04</td>
<td>79.83</td>
<td>4.62</td>
<td>808.7</td>
</tr>
<tr>
<td>StructEdges, 25 seeds</td>
<td>68.85</td>
<td>79.89</td>
<td>2.16</td>
<td>741.9</td>
</tr>
<tr>
<td>GB, 64 seeds</td>
<td>72.83</td>
<td>82.55</td>
<td>6.99</td>
<td>1490.3</td>
</tr>
<tr>
<td>StructEdges, 64 seeds</td>
<td>73.64</td>
<td>82.84</td>
<td>4.71</td>
<td>1462.8</td>
</tr>
<tr>
<td>GB, 100 seeds</td>
<td>74.22</td>
<td>83.25</td>
<td>9.26</td>
<td>1781.9</td>
</tr>
<tr>
<td>StructEdges, 100 seeds</td>
<td><strong>75.19</strong></td>
<td><strong>83.52</strong></td>
<td><strong>6.84</strong></td>
<td><strong>1828.7</strong></td>
</tr>
</tbody>
</table>
Future Directions

1. Learning unaries which are faster to compute, and accurate.

2. Parametric min-cut for both unaries and pairwise energies. [1]

3. Multiple precomputation graphs, each only dealing with graph with similar unary costs.

4. GPU implementation.

Summary

1. Presented a method to precompute some information for different graph-cuts.

2. Our method can find re-use parts before computing full cuts.

3. A practical object segmenter, running under 2 secs!

4. CODE: http://cpl.cc.gatech.edu/projects/RIGOR