Memory Efficient Max Flow for Multi-label Submodular MRFs

Thalaiyasingam Ajanthan    Richard Hartley    Mathieu Salzmann

The Australian National University

Data61, CSIRO

Data61, May 2016
Introduction

Minimize

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

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

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]

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 $x_i \in \{0, 1, \cdots, \ell - 1\}$.

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]

Current method

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

Minimize

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

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

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]

Current method

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

The Ishikawa graph
The Ishikawa algorithm

The Ishikawa graph
The Ishikawa algorithm

**Drawback**
- Stores $2\ell^2$ edges for each pair of neighbours.

**Idea**
- Stores $2\ell$ values for each pair of neighbours.
The Ishikawa algorithm

Drawback

▶ Stores $2\ell^2$ edges for each pair of neighbours.

\[ E.g. \quad |V| = 10^6, \ell = 256 \]
\[ \text{Edges} \approx 2 \times 10^6 \times 2 \times 256^2 \]
\[ \text{Memory} \approx 1000 \text{ GB} \]

Idea

▶ Stores $2\ell$ values for each pair of neighbours.

\[ \Downarrow \]

\[ \text{Memory} \approx 4 \text{ GB} \]
The Ishikawa algorithm

Drawback
- Stores $2\ell^2$ edges for each pair of neighbours.

Idea
- Stores $2\ell$ values for each pair of neighbours.

Example:
- $|\mathcal{V}| = 10^6$, $\ell = 256$
- Edges $\approx 2 \times 10^6 \times 2 \times 256^2$
- Memory $\approx 1000$ GB

↓

Memory $\approx 4$ GB
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 in progress
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

Max-flow in progress

Flow = 4

Max-flow in progress
Max flow on the Ishikawa graph

Max-flow in progress
Max flow on the Ishikawa graph

Max-flow in progress

Flow = 5
Max flow on the Ishikawa graph

Flow = 7

Max-flow in progress
Max flow on the Ishikawa graph

Min-cut

Flow = 7
Memory efficient flow encoding

Initial exit-flows
Memory efficient flow encoding

Update exit-flows
Memory efficient flow encoding

Update exit-flows
Memory efficient flow encoding

Update exit-flows
Memory efficient flow encoding

Update exit-flows
Memory efficient flow encoding

Update exit-flows
Memory efficient flow encoding

Update exit-flows
Memory efficient flow encoding

Update exit-flows
Memory efficient flow encoding

Update exit-flows
Memory efficient max flow

**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

**Space complexity:** \( O(|E| \ell) \)
Memory efficient max flow

Algorithm

Require: $\phi^0$ \textgreater Initial Ishikawa capacities
$\Sigma \leftarrow 0$ \textgreater Initialize exit-flows
repeat
$P \leftarrow$ augmenting path($\phi^0$, $\Sigma$)
$\Sigma \leftarrow$ augment($P$, $\phi^0$, $\Sigma$)
until no augmenting paths possible

Space complexity: $\mathcal{O}(|E|\ell)$
Memory efficient max flow

**Algorithm**

**Require:** $\phi^0 \triangleright$ Initial Ishikawa capacities
$\Sigma \leftarrow 0$  \hspace{1cm} $\triangleright$ 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

**Space complexity:** $\mathcal{O}(|E|\ell)$
Flow reconstruction / Computing residual edges

\[ T(\ell^3) \]

\[ \phi_{ij}^0 \]
Flow reconstruction / Computing residual edges

Flow reconstruction as a small max-flow problem
Flow reconstruction as a small max-flow problem

All flow-reconstructions are equivalent.
Flow reconstruction / Computing residual edges

Flow reconstruction as a small max-flow problem

Time complexity: $O(\ell^3)$
Flow equivalence - an example

\[ U_i:2 \quad \begin{array}{c} 1 \end{array} \quad -1 \quad U_j:2 \]

\[ U_i:1 \quad \begin{array}{c} 1 \end{array} \quad -1 \quad U_j:1 \]

Exit-flows
Flow equivalence - an example

\[ U_i:2 \xrightarrow{1} -1 \xrightarrow{} U_j:2 \]

\[ U_i:1 \xrightarrow{1} -1 \xrightarrow{} U_j:1 \]

A reconstructed flow
Flow equivalence - an example

Another reconstructed flow
Flow equivalence - an example

Both reconstructions are equivalent
Finding an augmenting path

Find augmenting paths on a subgraph

Overall time complexity: $O(|V||E|^{\ell/6})$
Finding an augmenting path

Find augmenting paths on a subgraph

Find augmenting paths on a subgraph
Finding an augmenting path

Overall time complexity: $\mathcal{O}(|V||E|^6)$
Efficiently finding an augmenting path

Simplified graph

- Sparse graph.
- Fewer augmenting paths.

Search-tree-recycling

- Good empirical performance.

Simplified graph representation
Efficiently finding an augmenting path

Simplified graph
- Sparse graph.
- Fewer augmenting paths.

Search-tree-recycling
- Good empirical performance.

Simplified graph representation
Efficiently finding an augmenting path

Simplified graph

▷ Sparse graph.
▷ Fewer augmenting paths.

Search-tree-recycling

▷ Good empirical performance.

Image courtesy of [Boykov-2004]
## Results

<table>
<thead>
<tr>
<th>Problem</th>
<th>Memory [MB]</th>
<th></th>
<th>Time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BK EIBFS</td>
<td>MEMF</td>
<td>BK EIBFS</td>
</tr>
<tr>
<td>Tsukuba</td>
<td>3195 2495</td>
<td>211</td>
<td>14 4</td>
</tr>
<tr>
<td>Venus</td>
<td>7626 5907</td>
<td>396</td>
<td>35 9</td>
</tr>
<tr>
<td>Sawtooth</td>
<td>7566 5860</td>
<td>393</td>
<td>31 8</td>
</tr>
<tr>
<td>Map</td>
<td>6454 4946</td>
<td>219</td>
<td>57 9</td>
</tr>
<tr>
<td>Cones</td>
<td>*72303 *55063</td>
<td>1200</td>
<td>- -</td>
</tr>
<tr>
<td>Teddy</td>
<td>*72303 *55063</td>
<td>1200</td>
<td>- -</td>
</tr>
<tr>
<td>KITTI</td>
<td>*88413 *67316</td>
<td>2215</td>
<td>- -</td>
</tr>
<tr>
<td>Penguin</td>
<td>*173893 *130728</td>
<td>663</td>
<td>- -</td>
</tr>
<tr>
<td>House</td>
<td>*521853 *392315</td>
<td>1986</td>
<td>- -</td>
</tr>
</tbody>
</table>

Comparison with other max-flow implementations

**BK** Boykov-2004

**EIBFS** Goldberg-2015
Empirical time complexity

Tsukuba

Penguin
Empirical time complexity

Tsukuba

Penguin

Empirical time complexity: $O(|V|\ell^3)$
Conclusion

- We have introduced a memory efficient alternative to the max-flow algorithm that can optimally minimize multi-label submodular MRF energies.
Thank you!
Augmentation
Augmentation

$\tilde{m}(1, 0, \alpha_{i,j})$

$\tilde{m}(0, 0, \alpha_{j,k})$

Subtract min

$U_i:0$

$B_i:0$

$U_j:0$

$B_j:0$

$U_k:0$

$B_k:0$

$B_j:1$

$B_k:1$

$B_i:1$

$U_i:5$

$U_j:5$

$U_k:5$

Augmentation
Augmentation