Probabilistic image segmentation using min (s,t)-cuts

by

Anurag Babruwan Gomsale

Advisor

Dr. Ivona Bezáková
Department of Computer Science
Rochester Institute of Technology

December 2016
# Contents

1 Introduction 1

2 Background  2
   2.1 Directed Acyclic Graph  3
   2.2 Topological Ordering  3
   2.3 Algorithm  4
      2.3.1 Path counting example  5
      2.3.2 Path sampling example  6

3 Implementation Details  7
   3.1 Existing Code Base  7
   3.2 Running Time Analysis  8
   3.3 Test Case Design  9
      3.3.1 Test Case 01  9
      3.3.2 Test Case 02  9
      3.3.3 Test Case 03  10
      3.3.4 Test Case 04  10
      3.3.5 Test Case 05  11

4 Instructions For Running The Software  12
   4.1 Options Available At Run Time  12
   4.2 Sample Command  12
   4.3 Output  13
   4.4 Problem With Big Images  13

5 Experiments And Results  14
   5.1 Experiment 01  15
   5.2 Experiment 02  16
   5.3 Experiment 03  17
   5.4 Experiment 04  18
   5.5 Experiment 05  19
   5.6 Experiment 06  20

6 Future Scope  21

7 Conclusion  22
Chapter 1

Introduction

One of the applications of the Maximum Flow Network graph algorithm and the related min (s,t)-cut algorithm[1] is to produce image segmentation. In this project, we are analyzing an interactive and binary image segmentation approach. In this approach, user selects a pixel (sink-t) from an object to be segmented and a second pixel (source-s) from the background of the object. Before an image can be segmented into an object segment and a background segment, the image is converted into a planar graph. Each image pixel is treated as a vertex in the graph and a function of the difference between intensities of neighboring pixels represents an edge weight. These connected edges form a 4-connected grid.

Treating pixels selected by user as sink vertex and source vertex respectively , maximum flow is found in this graph[2]. Each strongly connected component of the residual maximum flow graph is contracted. Dual graph of this contracted graph is fed to an algorithm for counting minimum cuts in a planar graph[3]. There can be many ways to get minimum capacity cut in this graph, separating a set of object vertices and the corresponding set of background vertices. As there are exponentially many min-cuts possible, we sample few minimum cuts to get the probability of each vertex being in object vertex set or background vertex set. We believe that, in general segmentation produced by this probabilistic approach would be better than a segmentation generated by selecting a particular minimum cut.

In the next chapters we will discuss about the related background and how [2] fits into this project. Further, implementation details of the probabilistic sampling algorithm and few results of the segmentation using this sampling algorithm are discussed.
Chapter 2

Background

The related version [2] of this project segments the input image using a single min (s,t)-cut. Project [2] also calculates the number of min (s,t)-cuts in the residual graph (after finding the maximum flow s-t flow) using Algorithm 1 in [3]. For more details on this min (s,t)-cut counting algorithm and related background information such as dual graph please refer to [2].

In this project we construct a new dual graph optimised for the probabilistic sampling of paths in the dual graph, using the existing dual graph construction method in [2]. Each path from a start vertex (a vertex with 0 indegree) to an end vertex (a vertex with 0 outdegree) in the dual graph, represents a min (s,t)-cut in the parent residual graph. This dual graph is a directed acyclic graph.

As there can be exponentially many min(s,t)-cuts possible for a given image (For example, an 180 X 320 pixel image may have number of min cuts in the order of $10^{22}$), we sample 10,000 of these paths in the dual graph with equal probability.
2.1 Directed Acyclic Graph

Directed acyclic graph (DAG) is a graph with no directed cyclic paths. The graph in the Figure 2.1 is a directed acyclic graph. The graph in the Figure 2.2 is not a DAG.

![Directed Acyclic Graph](image1)

**Figure 2.1:** Directed Acyclic Graph $G$

![Cyclic Graph](image2)

**Figure 2.2:** Cyclic Graph

![Graph G1 with back edges](image3)

**Figure 2.3:** Graph $G^1$ with back edges.

2.2 Topological Ordering

Topological ordering of vertices in a directed graph is a ordering such that, for each directed edge from vertex $u$ to vertex $v$, $u$ is always ordered before $v$. Topological order for vertices in the Figure 2.1 is: $ABCDE$

As we can see in the Figure 2.2, topological ordering is not possible in a cyclic graph.
2.3 Algorithm

Algorithm 1 Path Sampling Algorithm [3]

1: procedure Sample(G) \Comment*[r]{Directed acyclic graph $G = (V, E)$}
2: \hspace{1em} Construct an auxiliary graph $G^1$ using Graph $G$ \Comment*[r]{Figure 2.3}
3: \hspace{1em} $V^1 \leftarrow V$
4: \hspace{1em} $E^1 \leftarrow \emptyset$
5: \hspace{2em} for each edge from $u$ to $v \in E$ do
6: \hspace{3em} $E^1 \leftarrow E^1 \cup \{vu\}$
7: \hspace{2em} end for
8: \hspace{2em} Topologically order the vertices in $G$ using depth first search. \Comment*[r]{For Figure 2.1 topological order is: $ABCDE$}
9: \hspace{2em} for each vertex $v \in V$ do
10: \hspace{3em} if indegree($v$) = 0 then \Comment*[r]{PathCounting Figure 2.4}
11: \hspace{4em} $C_v \leftarrow 1$ \Comment*[r]{$C_v$: Path count at vertex $v$}
12: \hspace{3em} else
13: \hspace{4em} $C_v \leftarrow 0$
14: \hspace{3em} end if
15: \hspace{2em} end for
16: \hspace{2em} for each $v_i$ in topological order $v_1...v_n$ do
17: \hspace{3em} for each edge from $v_i$ to $u \in V$ do
18: \hspace{4em} $C_u \leftarrow C_u + C_{v_i}$
19: \hspace{3em} end for
20: \hspace{2em} end for \Comment*[r]{END PathCounting}
21: \hspace{2em} In $G^1$, $V^1 \leftarrow V^1 \cup \{v_{end}\}$
22: \hspace{2em} for each vertex $u \in V^1$ and outdegree($v$) = 0 do
23: \hspace{3em} $E^1 \leftarrow E^1 \cup \{v_{end}u\}$
24: \hspace{2em} end for
25: \hspace{2em} Random path $S \leftarrow \{}$ \Comment*[r]{ChooseRandomPath}
26: \hspace{2em} From all outgoing edges $v_{end}u$ at $v_{end}$, choose an edge randomly, in proportion to $C_u$.
27: \hspace{2em} $u_{current} \leftarrow u$
28: \hspace{2em} while outdegree($u_{current}$) $\neq 0$ do
29: \hspace{3em} $S \leftarrow S \cup \{u_{current}\}$
30: \hspace{3em} From all outgoing edges $u_{current}u$ at $u_{current}$, choose an edge randomly, in proportion to $C_u$.
31: \hspace{2em} $u_{current} \leftarrow u$
32: \hspace{2em} end while
33: \hspace{2em} return $S$ \Comment*[r]{END ChooseRandomPath}
34: end procedure
2.3.1 Path counting example

![Diagram showing path counting example.](image)

**Figure 2.4:** An example, showing how number of paths passing through each vertex of the given DAG is calculated using topological order. The topological order for the given graph is $A \ B \ C \ D \ E$. 
2.3.2 Path sampling example

As shown in the this example, for the path sampling with equal probability, a dummy End vertex is added to the graph. Outgoing edges from the End vertex is connected to the all existing vertices with indegree 0. Starting at the dummy End vertex, next edge, End to E is chosen with 3 : 3 odds 2.6(a). At vertex E, edge ED or EC is randomly selected with 2 : 1 odds respectively 2.6(b). Suppose edge ED is selected. At vertex D, randomly select edge DC or edge DA, with 1 : 1 odds respectively 2.6(c). Suppose edge DA is selected. As the outdegree(A) is 0, we stop the path discovery process 2.6(d). Thus, out of total 3 available paths from A to E, we have selected the end to end path in the original graph ( path A → D → E ) with probability \( \frac{1}{3} \). Similarly other two paths in the original graph ( path A → B → C → E and path A → B → C → D → E ) are also selected with the probability of \( \frac{1}{3} \) 2.6(d).
Chapter 3

Implementation Details

3.1 Existing Code Base

Following flow chart shows code components in the [2].

[Diagram of flow chart showing code components: Start, countingCutsThroughSchmidt, createGraph, getMinCut, calculateCuts, number of min (s,t) cuts]
3.2 Running Time Analysis

Overall running time: $O(|E| \log |E|)$

Graph construction : $O(|E| + |V|)$

Topological sort : $O(|E| + |V|)$
Topological order is found using $DFS$.

Path Count : $O(|E| + |V|)$

Prepossessing of path count to choose random path : $O(|E| \log |E|)$
At each vertex $u$, outgoing edges $(uv_1, uv_2, \ldots, uv_k)$ are sorted according to path count at $v_k$.

Single random path selection : $O(|V| \log |E|)$
To choose a next edge in the path, at each vertex, binary search is performed on the sorted path count array of the outgoing edges.
3.3 Test Case Design

To validate the correctness of path sampling logic, different graphs with and without duplicate edges were fed to the sampling program.

3.3.1 Test Case 01

Total vertices : 5
Total Samples : 10000
Topological order : A B C D E
Path Count : 1 1 1 2 3

<table>
<thead>
<tr>
<th>Path</th>
<th>Sample Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>A → B → C → D → E</td>
<td>3326</td>
</tr>
<tr>
<td>A → D → E</td>
<td>3330</td>
</tr>
<tr>
<td>A → B → C → E</td>
<td>3344</td>
</tr>
</tbody>
</table>

Table 3.1: Results for Graph 3.1

3.3.2 Test Case 02

Total vertices : 5
Total Samples : 10000
Topological order : A B C D E
Path Count : 1 1 1 3 4

<table>
<thead>
<tr>
<th>Path</th>
<th>Sample Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>A → B → C → D → E</td>
<td>2577</td>
</tr>
<tr>
<td>A → D → E</td>
<td>5029</td>
</tr>
<tr>
<td>A → B → C → E</td>
<td>2394</td>
</tr>
</tbody>
</table>

Table 3.2: Results for Graph 3.2

Figure 3.1: Simple DAG

Figure 3.2: DAG with duplicate edges
3.3.3 Test Case 03

Total vertices : 7
Total Samples : 10000
Topological order : A B C E F D G
Path Count : 1 1 1 2 2 1 4

<table>
<thead>
<tr>
<th>Path</th>
<th>Sample Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>A → B → E → F → G :</td>
<td>2455</td>
</tr>
<tr>
<td>A → C → G          :</td>
<td>2506</td>
</tr>
<tr>
<td>A → C → E → F → G :</td>
<td>2492</td>
</tr>
<tr>
<td>A → F → G          :</td>
<td>2547</td>
</tr>
</tbody>
</table>

Table 3.3: Results for Graph 3.3

Figure 3.3: Simple DAG

3.3.4 Test Case 04

Total vertices : 6
Total Samples : 10000
Topological order : F E D C B A
Path Count : 1 1 1 1 1 2 2

<table>
<thead>
<tr>
<th>Path</th>
<th>Sample Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>F → C → D → B      :</td>
<td>2519</td>
</tr>
<tr>
<td>F → A               :</td>
<td>2482</td>
</tr>
<tr>
<td>E → A               :</td>
<td>2550</td>
</tr>
<tr>
<td>E → B               :</td>
<td>2449</td>
</tr>
</tbody>
</table>

Table 3.4: Results for Graph 3.4

Figure 3.4: DAG with multiple start and end vertices
3.3.5 Test Case 05

Total vertices : 8  
Total Samples : 1000000  
Topological order : B D A C E F G H  
Path Count : 1 1 1 2 2 3 3 2

<table>
<thead>
<tr>
<th>Path</th>
<th>Sample Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>B → C → E → F → G</td>
<td>200357</td>
</tr>
<tr>
<td>B → D → F → G</td>
<td>200285</td>
</tr>
<tr>
<td>A → C → E → H</td>
<td>199762</td>
</tr>
<tr>
<td>B → C → E → H</td>
<td>199781</td>
</tr>
<tr>
<td>A → C → E → F → G</td>
<td>199815</td>
</tr>
</tbody>
</table>

Table 3.5: Results for Graph 3.5

Figure 3.5: Simple DAG
Chapter 4

Instructions For Running The Software

The program is developed using C++ and the same has been tested to run on Mac OS X. This program takes input through command line arguments.

4.1 Options Available At Run Time

1. Considering the top left image corner as origin 0,0 , left direction as +X axis and down direction as +Y axis, indicate the X,Y pixel location of one of the pixel from the object to be segmented Source.

2. Indicate the (X, Y) pixel location of one of the pixel from the background for the object Source.

3. Indicate the number of min(s-t) cuts to be sampled.

4.2 Sample Command

./SamplingCuts ./lena4w256.ppm 17 72 116 167 10000

SamplingCuts : Program name
17 72 : Considering the top left image corner as origin (0,0), left direction as +X axis and down direction as +Y axis, (X,Y) location of one of the background pixel (Sink)
116 167 : (X,Y) location of one of the object pixel (Sink)
10000 : Number of random min(s-t) cuts to be sampled.
4.3 Output

For a given input image, object pixel and background pixel, the program produces four output images similar to what we have in the Experiments and Results chapter.

4.4 Problem With Big Images

Images bigger than 400 * 400 pixel fail to process due to use of recursive DFS. The program runs out of stack memory resulting in a segmentation fault.
Experiments were performed on various images available in online datasets. For each such image we compared segmentation results from [2] with segmentation produced by probabilistic sampling of min (s,t)-cuts. As the segmentation program only accepts .ppm images, the same were converted to .ppm format using [4]. Images bigger than 400X400 pixel were downsized to avoid stack overflow issue.

Following are the results of 6 of the experiments performed.
5.1 Experiment 01

Input image: 5.1(a). Solid red mark in the images below represent the object pixel selected by the user; whereas, solid blue mark represent the background pixel. Binary interactive segmentation using a single min(s,t)-cut is presented in the image 5.1(b) and the corresponding solid segmentation boundary is displayed in the 5.1(d). Image 5.1(c) represents segmentation using sample of 10,000 min(s,t)-cuts out of all possible cuts and image 5.1(e) is the corresponding fuzzy segmentation boundary.

Figure 5.1: Experiment results for the image Lena.[5]
5.2  **Experiment 02**

Input image: 5.2(a). Solid red mark in the images below represent the object pixel selected by the user; where as, solid blue mark represent the background pixel. Binary interactive segmentation using a single min(s,t)-cut is presented in the image 5.2(b) and the corresponding solid segmentation boundary is displayed in the 5.2(d). Image 5.2(c) represents segmentation using sample of 10,000 min(s,t)-cuts out of all possible cuts and image 5.2(e) is the corresponding fuzzy segmentation boundary.

![Figure 5.2: Experiment results for the image Esno.][6]
5.3 Experiment 03

Input image: 5.3(a). Solid red mark in the images below represent the object pixel selected by the user; whereas, solid blue mark represent the background pixel. Binary interactive segmentation using a single min(s,t)-cut is presented in the image 5.3(b) and the corresponding solid segmentation boundary is displayed in the 5.3(d). Image 5.3(c) represents segmentation using sample of 10,000 min(s,t)-cuts out of all possible cuts and image 5.3(e) is the corresponding fuzzy segmentation boundary.

Figure 5.3: Experiment results for the image Bird.[7]
5.4 Experiment 04

Input image: 5.4(a). Solid red mark in the images below represent the object pixel selected by the user; where as, solid blue mark represent the background pixel. Binary interactive segmentation using a single min(s,t)-cut is presented in the image 5.4(b) and the corresponding solid segmentation boundary is displayed in the 5.4(d). Image 5.4(c) represents segmentation using sample of 10,000 min(s,t)-cuts out of all possible cuts and image 5.4(e) is the corresponding fuzzy segmentation boundary.

Figure 5.4: Experiment results for the image Walk.[8]
5.5 Experiment 05

Input image: 5.5(a). Solid red mark in the images below represent the object pixel selected by the user; whereas, solid blue mark represent the background pixel. Binary interactive segmentation using a single min(s,t)-cut is presented in the image 5.5(b) and the corresponding solid segmentation boundary is displayed in the 5.5(d). Image 5.5(c) represents segmentation using sample of 10,000 min(s,t)-cuts out of all possible cuts and image 5.5(e) is the corresponding fuzzy segmentation boundary.

Figure 5.5: Experiment results for the computer generated image. [9]
5.6 Experiment 06

Input image: 5.6(a). Solid red mark in the images below represent the object pixel selected by the user; whereas, solid blue mark represent the background pixel. Binary interactive segmentation using a single min(s,t)-cut is presented in the image 5.6(b) and the corresponding solid segmentation boundary is displayed in the 5.6(e). Image 5.6(c) represents segmentation using sample of 10,000 min(s,t)-cuts out of all possible cuts and image 5.6(e) is the corresponding fuzzy segmentation boundary.

![Diagram](image)

Figure 5.6: Experiment results for the results for the computer generated image.

In the above experiment on a CGI, as the object boundaries are clearly defined, no difference is observed in the segmentation boundaries produced with sampling and without sampling.
Chapter 6

Future Scope

1. Change the image read code to accept all types of images other than .ppm images.

2. Change the recursive code segments into iterative code segments to avoid stack overflow issue while segmenting images greater than 400 X 400 pixels.

3. Test the interactive binary segmentation results against standard image benchmarks such as The Berkeley Segmentation Dataset and Benchmark [10]

4. \texttt{rand}() function of the C++ implementation on Mac OS X generates random numbers in the range \([0, 32767]\). But for the random sampling to code to sample the paths with equal probability, a random generator which can generate the number between \([0, totalMinSTCuts]\) is needed. For this project, we are approximating the random number generation by scaling the \texttt{rand}() function output to \([0, totalMinSTCuts]\).
Chapter 7

Conclusion

As hypothesized, image segmentation using probabilistic sampling of the min (s,t) cuts produced more realistic segmentation boundary as compared to the segmentation produced by using a single min (s,t)-cut. As seen in the results chapter, this behaviour is displayed more prominently in the images where object and background are hard to separate visually.
Bibliography


