# Graph-Based Image Segmentation with NetworKit

Marcel Radermacher, marcel.radermacher@student.kit.edu

15th July 2014

## 1 Introduction

Image segmentation is widely used in computer vision and is mainly a preprocessing step. Therefore, it is desirable that the segmentations does not consume too much time. One class of algorithms works directly on the image data. Another class of algorithms, and this approach, represents the image as a graph. With this it possible to apply graph clustering algorithms on this graphs to segment the images. I implemented a Kruskal-like approach introduced by Felzenszwalb et al. [1]. Further I apply community detection algorithms implemented within the NetworKit on the graph.

## 2 Algorithm

Felzenszwalb et al. [1] introduced a simple Kruskal-like algorithm to segment images. The image is represented as a graph. Each pixel is a node and there exists an edge between two nodes if the corresponding pixels are adjacent (i.e the 8-neighborhood of a pixel). There are several methods to choose the weight of an edge. Two examples are given in the next section. The goal is to find different connected components in die graph, which represent an *important* part of the image.

The basic idea of the algorithm is to use the weight of an edge as criteria of dissimilarity between two nodes. Let $S_0, S_1, \ldots, S_n$ be a series of segmentations. A segmentation $S_i$ is a partition of all nodes. Initially, $S_0$ has a component for each node. Let $C_i(u)$ denote the component of node $u$ in the $i$-th iteration. To build the segmentations $S_i$ the edges are handles in non-decreasing order of their weight. Then the segmentation $S_{i+1}$ is either the same as $S_i$ if the weight $w(e_i)$ of edge $e_i = (v, w)$ is smaller then the threshold $t(C_i(v), C_i(w))$. Or otherwise $S_{i+1}$ yields from joining the sets $C_i(v) \cup C_i(w)$.

$$t(C_1, C_2) = \min(Int(C_1) + k/|C_1|, Int(C_2) + k/|C_2|) \tag{1}$$

To get a good segmentation it is crucial how to choose the threshold. Let $MST(C)$ denote the minimum spanning tree of $G_C = (C, E \cap (C \times C))$. The internal difference $Int(C)$ component $C$ is defined as the largest edge weight of the minimum spanning tree $MST(C)$ of $C$. The threshold $t(C_1, C_2)$ considers the internal difference and the size of the components as well. It is not desirable to have a lot of small components. Therefore, if the internal difference is sufficiently small the size of a component should have a larger impact. As the size of a component grows, the internal difference should get more important. With the factor $k$ this balance between internal difference and component size can be achieved.

### 2.1 Combination

I combined this algorithm with the implementation of PLM in the NetworKit. The first version runs PLM on the image and writes the result into a new image. On this image I applied the Kruskal-like algorithm. I refer to this algorithm as *Com1*

A second implementation runs PLM and computes the communication graph of the components. For each component the color values are accumulated. Based on this new values the edge weights are chosen. One can initially set size the component sizes $|C_i|$ to one or to the actual component size. Then the Kruskal-like algorithm is applied as described before. *Com2* refers to this algorithm with initial component size one. Otherwise I refer to the algorithm as *Com2C*.

| Referrer | Algorithm | Edge Weight | k | Refine | Gamma | MaxIter |
|---|---|---|---|---|---|---|
| KR_M | Kruskal like | mixture | 3000 | - | - | - |
| KR_CH1 | Kruksal like | channel | 3000 | - | - | - |
| KR_CH2 | Kruksal like | channel | 600 | - | - | - |
| PLM1 | PLM | mixture | - | true | 1 | 1 |
| PLM2 | PLM | mixture | - | true | 1 | 32 |
| Com1 | PLM + KR_CH | mixture/channel | 30000 | true | 1 | 1 |
| Com2 | PLM + KR_CH | mixture/channel | 30000 | true | 1 | 1 |
| Com2C | PLM + KR_CH | mixture/channel | 30000 | true | 1 | 1 |

**Table 1** Algorithms and their configurations

| Referrer | Jaccard | $BCE^*$ | Time [s] |
|---|---|---|---|
| KR_M | 0.61 | 0.67 | 0.92 |
| KR_CH1 | 0.65 | 0.85 | 2.5 |
| KR_CH2 | 0.75 | 0.90 | 2.3 |
| PLM1 | 0.94 | 0.96 | 0.4 |
| PLM2 | 0.94 | 0.96 | 0.87 |
| Com1 | 0.67 | 0.85 | 4.2 |
| Com2 | 0.83 | 0.90 | 0.51 |
| Com2C | 0.94 | 0.95 | 0.50 |

**Table 2** Evaluation of the Algorithms

## 3 Implementation

To achieve a good running time of the algorithm it is crucial how to merge two components. The partition datastructure of NetworKit is able to merge two subsets in linear time. This would lead to an overall quadric runtime for this algorithm. I implemented the Union-Find datastructure to represent the components internally. In a final step the components in the Union-Find datastructure are represented within the partition datastructure to keep up compatibility with other NetworKit algorithms.

Within the implementation there are two possible edge weights. The first one was introduced by Jianbo et al. [4]. This edge weights considers the difference of the intensity and the distance between two pixels. I refer to these method as *mixture*. The second way is to represent each color channel as a separate graph. The edge weight is the absolute difference in intensity on the corresponding channel. The algorithm is executed on each of those graphs. The resulting partitions are intersected to form the final segmentation. Further on, this method is called *channel*.

## 4 Experiments

I have evaluated two clusterings algorithms and a combination of both for a segmentation. The base algorithms are the Kruskal-like algorithm introduced before and the PLM clustering algorithm [5]. The configuration is depicted in table 1. The combination PLM_KR runs in the first step the PLM algorithm and writes the segmentation into an image. In a second step this image is segmented with KR_CH. As benchmark instance the 200 train images of the Berkley Segmentation Dataset [2] is used. To measure the quality of an segmentation I used the Jaccard dissimilarity measure implemented within the NetworKit. Additional I implemented the $BCE^*$ measure introduced in [3]. For both measures a value near zero means a high similarity and a value near one a high dissimilarity.

The table 2 shows all evaluation results. This tables suggests that the Kruskal-like algorithm with the mixture edge weight function performs best. Nevertheless, these both quantities are not enough to measure the quality of a segmentation. Figure 1 depicts the segmentation with the best $BCE^*$ score of the KR_M algorithm. The KR_CH1 algorithm has a better $BCE^*$ and Jaccard value. But the KR_M value is quiet good as well. But the actual segmentation of KR_M is not very useful. To get a meaningful measure a combination of further measurement algorithms should be used.

Most algorithms need less then a second to segment an image. The implementation of KR_CH* is not parallel. A parallel implementation should not take much more then one second. The number of iteration for PLM algorithm does not effect the quality of the segmentation.

**(a)** Original image contained in the BSD500 **(b)** KR_M with $BCE^*$ value 0.2, Jaccard value 0.24 **(c)** KR_CH with $BCE^*$ value 0.14, Jaccard value 0.023

**Figure 1** Segmentation with the best score in the $BCE^*$ value.

## References


[1] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient graph-based image segmentation. *Int. J. Comput. Vision*, 59(2):167–181, September 2004.

[2] David Martin, Charless Fowlkes, Doron Tal, and Jitendra Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proc. 8th Int'l Conf. Computer Vision*, volume 2, pages 416–423, July 2001.

[3] David Royal Martin. *An Empirical Approach to Grouping and Segmentation.* PhD thesis, EECS Department, University of California, Berkeley, Aug 2003.

[4] Jianbo Shi and Jitendra Malik. Normalized Cuts and Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[5] Christian Staudt and Henning Meyerhenke. Engineering High-Performance Community Detection Heuristics for Massive Graphs. In *Proceedings of the 2013 International Conference on Parallel Processing.* Conference Publishing Services (CPS), 2013.