Image Segmentation via Network Flow

Li, Peixin	Nie, Yuchen	Wang, Chunxi	Wang, Sihan	Xiao, Lijun
pxli@bu.edu	ycnie@bu.edu	tracycxw@bu.edu	shwang95@bu.edu	ljxiao@bu.edu

May 3, 2018

1 Overview

1.1 Introduction

The main goal of this project is to segment the foreground and background of an image. The pre-processing of image is basically down sampling in order to improve the speed of image processing. After down sampling, use k-means and naïve Bayes classifier to separate pixels into two clusters with the probability that whether they belong to are foreground or background. In addition, GMM is another choice that also implements this function, which is also used in the project to compare two different ways for a better choice. According to probability of each pixel, build a flow and use Edmonds-Karp Algorithm to find the maximum flow, which represent the optimal way to separate the foreground and background. Finally, do mask processing and create the image for output.

As for the extension, the GUI is built by PyQt5. The GUI allows users to select an image from the directory and start segmentation. Additionally, it improves the running time by converting Python code to C code via Cython.

The result of segmentation is basically as expected. The foreground and background can be separated in a smooth way after modifying our algorithm. The running time is improved. And the accuracy is also satisfying.

1.2 High-level Design

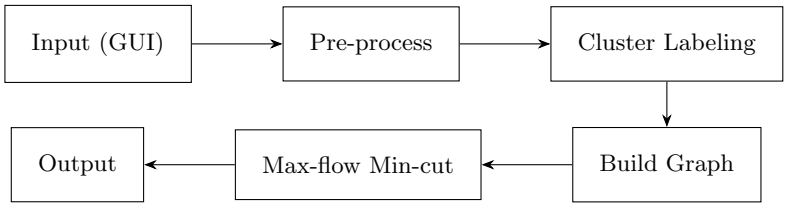


Figure 1: Flow Chart of the Project

(a) **Input**: The image is opened as an array of pixels, each pixel contains three value of R, G and B for representing the color. GUI enables users to input from files and users can click on the image to select the area needed to be segmented.

(b) **Pre-process**: The original image is too large for doing minimum cut fast, so it has to be down-sampled.

(c) **Cluster labeling**: Before building the flow, we need to first cluster the image, i.e., find the background and foreground part by using kinds of clustering algorithm. We implement two methods: Gaussian mixture model (GMM) and k-means with naïve Bayes classifier.

(d) **Build graph**: After clustered, we need to build the flow and calculate the minimum cut. The capacities of each edges on the graph need to be specified by several penalty functions.

(e) **Minimum Cut**: Calculate the maximum flow for the generated network flow, then find the minimum cut set which are foreground pixels and background pixels. Segmentation is done in this step. A mask image is generated for output.

(f) **Output**: Four images are outputted: the original image, the foreground, the background and the up-sampled mask. In GUI mode, the area user selected will be output.

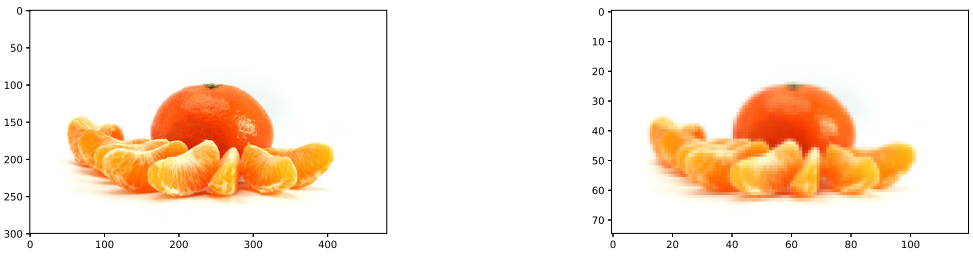
2 Implementations

2.1 Input

To input, we use `opencv` library for Python. The function provided by the library enables us to read an image as an array of pixels, e.g., an RGB image with hight 480 pixels and width 300 pixels will be read as an array with shape $(300, 480, 3)$.

2.2 Pre-process

We use bilinear interpolation to down-sample the image, this function is provided by existing Python scientific computing library `scipy`. The method we use for down-sampling the image is a bilinear interpolation. The following figures show the image before and after pre-processing, the size of the processed image shrinks to 25% of the original image on both hight and width.



(a) Original Image

(b) Down-sampled Image

Figure 2: Down-sampling Results

2.3 Cluster Labeling

For clustering, we use both GMM and k-means. In statistics, a mixture model is a probabilistic model for representing the presence of subpopulations within an overall population, without requiring that an observed data set should identify the sub-population to which an individual observation belongs. GMM is one of the mixture models. While k-means clustering is a method of vector quantization, originally from signal processing, that is popular for cluster analysis in data mining.

Compares to GMM as a soft clustering method, i.e., it returns an explanation of points (pixels) mix of multiple Gaussian probabilities, k-means is a hard clustering method, which means it assigns each point to a certain cluster. Thus k-means cannot return probabilities of pixels belong to the foreground and background, which is critical for building the graph. So we implement naïve Bayes classifier for help.

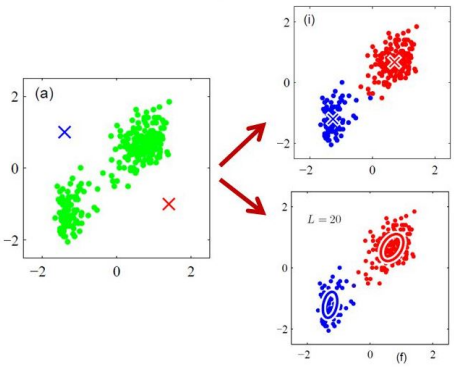
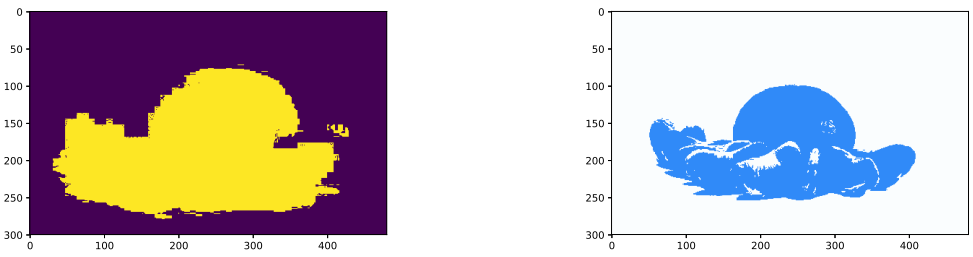


Figure 3: Difference between GMM and k-means

Naive Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes' theorem with strong independence assumptions between the features. It is a supervised probabilistic classifier, the features (or labels) is needed for training. So we provide the classifier with the labels generated by the k-means function, then we will get the probabilities for pixels belong to the foreground and background. In our project, we use Gaussian naive Bayes as the probabilistic calssifier.

The GMM function is provided by `scipy` library and the k-means function is provided by opencv library. The following figures shows the result of clustering using GMM and k-means.



(a) GMM

(b) K-means

Figure 4: Clustering Results

2.4 Build Graph

We use the simplest *4-neighborhood*, which is where a pixel is connected to the pixels directly above, below, left, and right, to define the graph.

To determine the weights on these edges, we need a segmentation model that determines the edge weights between pixels and between pixels and the *source* and *sink*. Let w_{ij} be the weight between pixel i and j, and s and t represent the source and the sink, then we use the penalty functions:

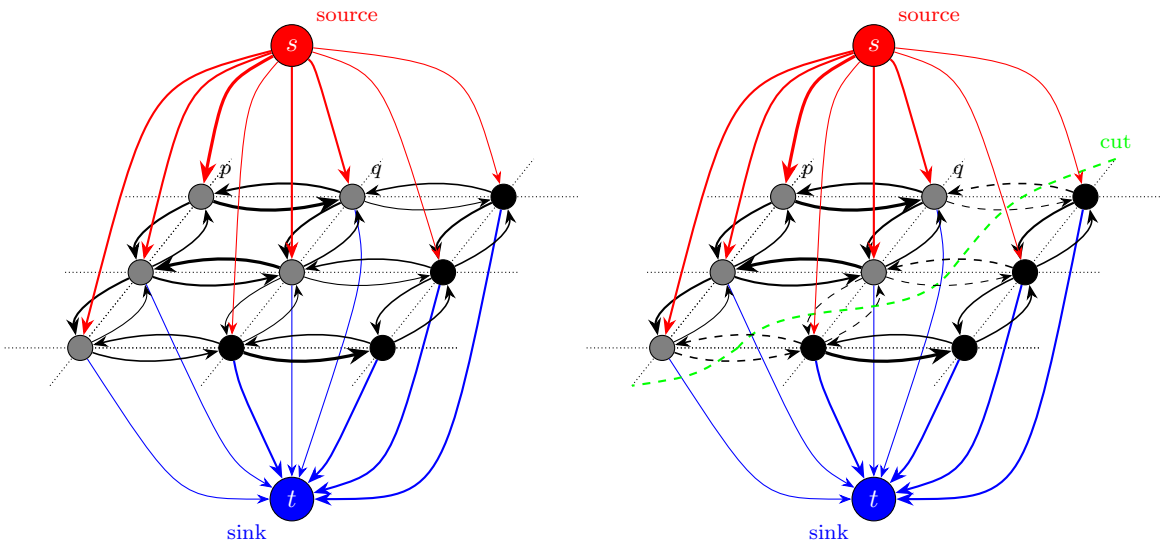
$$
\begin{aligned}
w_{si} &= \frac{p_F(I_i)}{p_F(I_i) + p_B(I_i)}, \\
w_{it} &= \frac{p_B(I_i)}{p_F(I_i) + p_B(I_i)}, \\
w_{ij} &= \kappa e^{-|I_i - I_j|^2/\sigma},
\end{aligned}
$$

where $p_F(I_i)$ and $p_B(I_i)$ indicate the probabilities for the pixel be in foreground and background that provided by the previous step.

The w_{ij} describe the pixel similarity between neighbors, similar pixels have weight close to κ, dissimilar close to 0. The parameter σ determines how fast the values decay toward zero with increasing dissimilarity. For our final implementation, we use $\kappa = 2$ and $\sigma = 100$.

2.5 Max-flow Min-cut

According to the max-flow min-cut theorem, finding the minimum cut is equivalent to finding the maximum flow between the source s and the sink t. Here we use the Edmonds-Karp (E-K) algorithm for finding the minimum cut. The running time for E-K algorithm in a directed graph $G = (V, E)$ is $O(VE^2)$. E-K algorithm is identical to Ford-Fulkerson (F-F) algorithm. The different of the E-K algorithm and the F-F algorithm is how they find the s-t path. The E-K algorithm uses the breadth-first search (BFS) wile the F-F algorithm doesn't specify an algorithm.



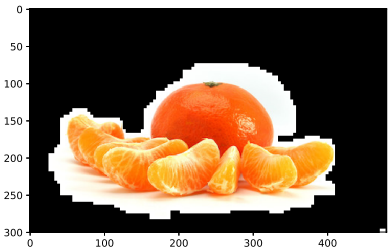
(a) A Graph G (b) A Cut on G

Figure 5: Example of a Graph and a Cut

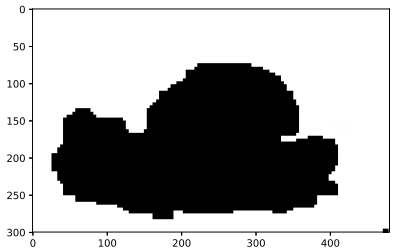
After finding the maximum flow, we need to determine the minimum cut set, i.e., the nodes that can reach the source and the nodes that can reach the sink. Since the BFS used in the E-K algorithm is a first in, first out ($FIFO$) method, the terminal point i.e., the source or the sink, that can reach/be reached by a node in the last iteration is the set where it should belong to.

2.6 Output

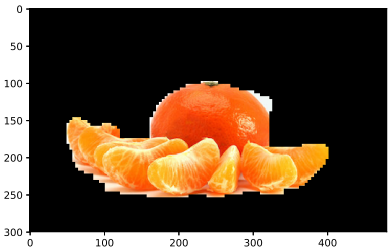
By the minimum cut, we obtained pixels of the foreground and the background, which is an array of 0 and 1 (the *mask*). Then we can then up-sampled the mask and apply it on the original image, then print it out. The following figures show the output of different cluster labeling method.



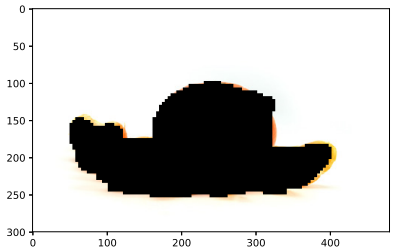
(a) Foreground (GMM)



(b) Background (GMM)



(c) Foreground (k-means)



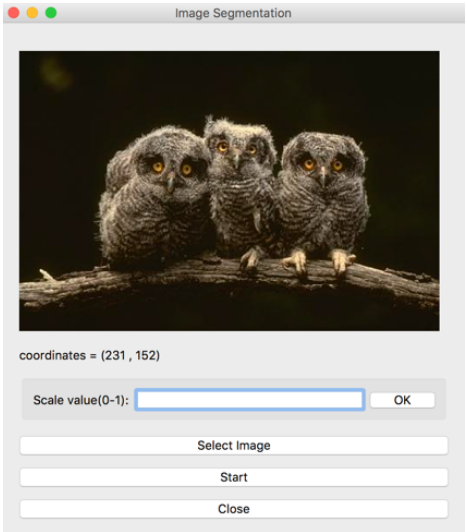
(d) Background (k-means)

Figure 6: Output of GMM and K-means (scale 0.25)

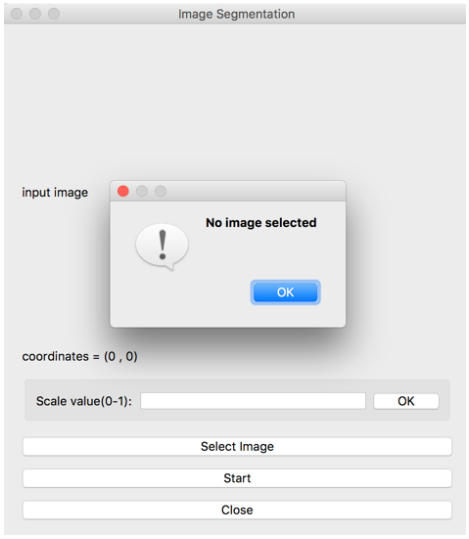
2.7 GUI

We choose to build the Graphical User Interface using `PyQt5` as an extension of the project.

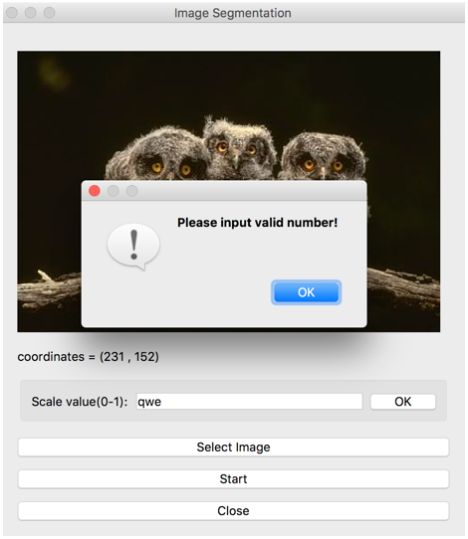
(a)



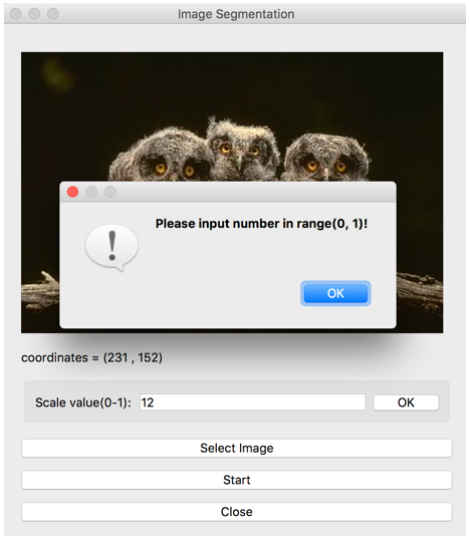
(b)



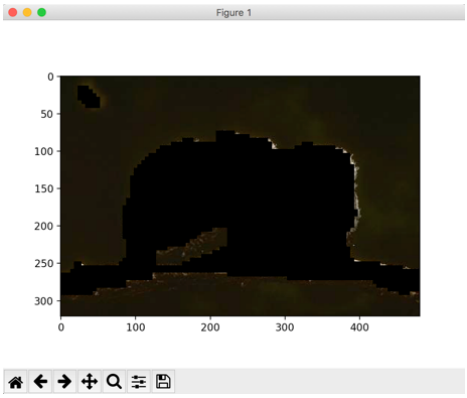
(c)



(d)



(e)



(f)

Figure 7: GUI

(a) The UI allows users to select an image from a local directory. We also limit the input file formats to `jpg` and `png`.

(b) The coordinates label reflects the coordinates of the pixel spot users clicked previously. The default value is set to $(0, 0)$, and it will only change when clicking in the range of the image.

(c) The UI checked if there is a valid picture chosen, if the start button is clicked without any image chosen then the main program wont start.

(d) (e) The scale value should be a valid number and the number users type it should be in the range between $(0, 1)$, this validation is also checked by the UI.

(f) If every parameter meets the requirement, then our program will start. Finally, according to the pixel user clicked on, another window will pop up and display the image our program returns of either foreground or background.

The way we found the area the user selected is by detecting the color where user clicked after the segmentation, since the part cut off will be shown as black, so if the program find the pixel user select is black, i.e., $(0, 0, 0)$ in RGB, then it will output another segmented image.

3 Results

3.1 Running Time Analysis

The following figures show some test results for running time analysis.

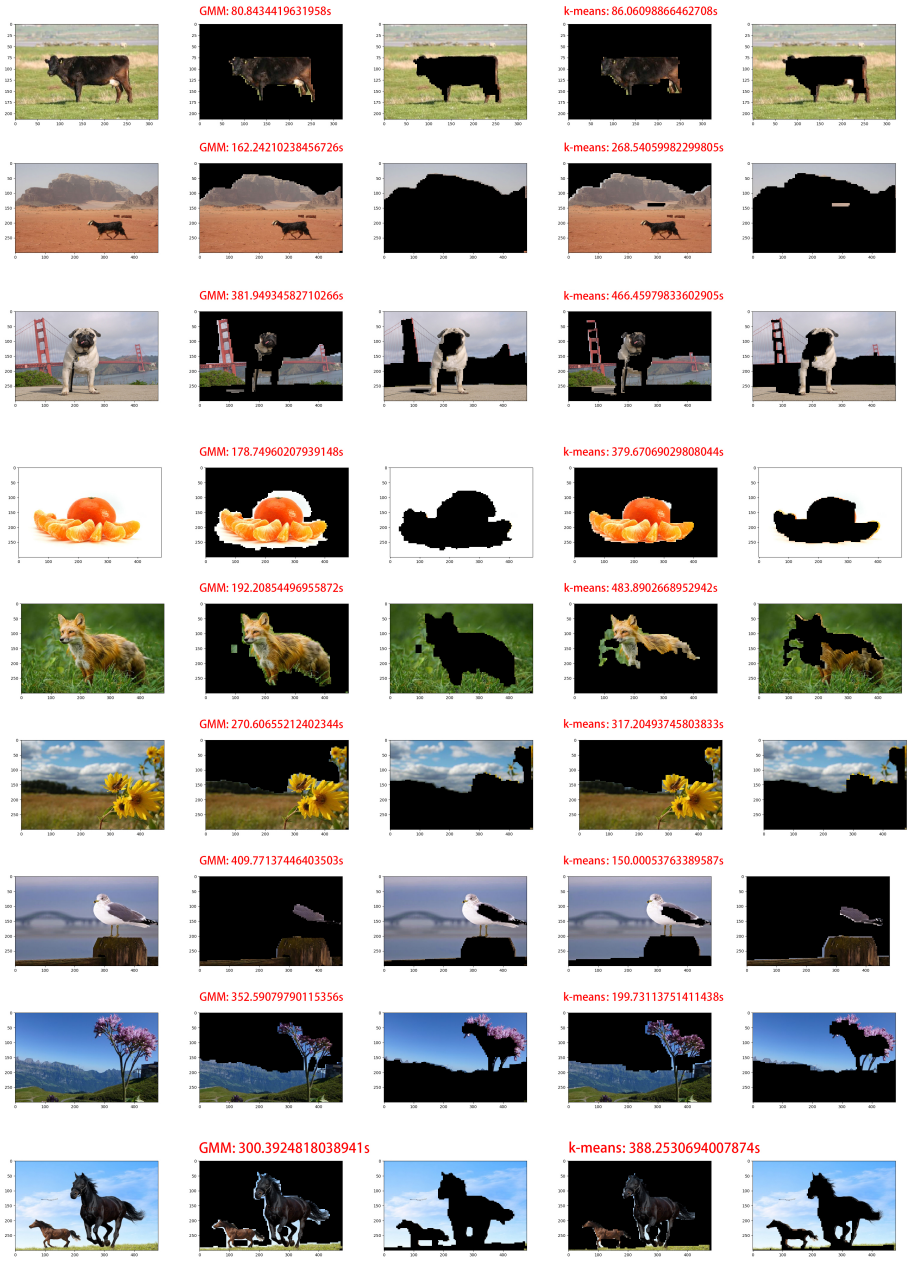


Figure 8: Test Results

For most of the images, GMM has better performance than k-means, you can see the outline of foreground and background is smoother. However, for the white background like the picture of an orange, k-means will achieve a better result.

3.2 Performance Improvement

We utilize Cython to improve the speed performance.

Cython is a compiled language that generates CPython extension modules. These extension modules can then be loaded and give us C-like performance with code that is mostly written in Python.

The following chart shows the running time results with sizes of images.

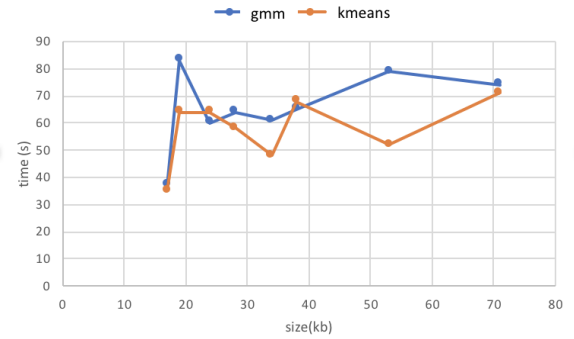


Figure 9: Run Time Results

For each of the example pictures, we test the speed performance using both GMM and k-means. We down-sample the image by setting the scale to 0.2. From the chart, we can see there is no obvious relationship between the speed performance and image size. But overall, the k-means method needs less time than GMM. With scale 0.2, running time of both methods are about 60s.

We also test the relationship between speed performance with scale size. The fact is: the smaller scale is, the more we down-sampled, the less running time we need.

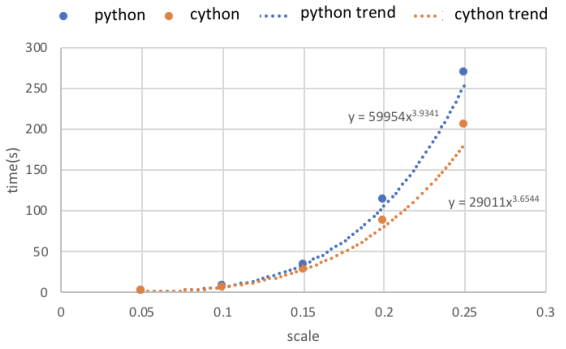


Figure 10: Cython and CPython Results

According to the previous chart, we can see that Cython reduce the running time about 30%.

4 Conclusions

The results prove that our design of combining clustering algorithm and network flow for image segmentation is feasible. Our works also prove that GMM clustering algorithm as well as k-means with naïve Bayes classifier can be used as the same with a minor difference. For the max-flow min-cut algorithm, the *E-K* algorithm is not efficient enough, a push-relabel algorithm may be better. Also, using Cython instead of traditional CPython does reduce the total running time.

GitHub Link

`https://github.com/shwang95/EC504_Project`

References

[1] "Mixture model", *En.wikipedia.org*, 2018. [Online]. Available: `https://en.wikipedia.org/wiki/Mixture_model`. [Accessed: 03-May-2018].

[2] "K-means clustering", *En.wikipedia.org*, 2018. [Online]. Available: `https://en.wikipedia.org/wiki/K-means_clustering`. [Accessed: 03-May-2018].

[3] J. Edmonds and R. Karp, "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems", *Journal of the ACM*, vol. 19, no. 2, pp. 248-264, 1972.

[4] M. Goodrich and R. Tamassia, *Algorithm design: foundations, analysis, and Internet examples*. New York: Wiley, 2002.

[5] T. Cormen, C. Leiserson, R. Rivest and C. Stein, *Introduction to algorithms*. Cambridge, Massachusetts: MIT Press, 2014.

[6] J. Solem, *Programming computer vision with Python*. Farnham: O'Reilly, 2012.

[7] Y. Boykov and V. Kolmogorov, "An experimental comparison of min-cut/max- flow algorithms for energy minimization in vision", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 9, pp. 1124-1137, 2004.