Image Processing Using Graphs
Computer exercises

August 12, 2015

1 Purpose

The purpose of these computer exercises is to make you familiar with the implementation of some of the methods we have encountered in this course. The exercises themselves are not mandatory, but are intended to give you a good starting point for working on your individual projects.

2 Getting started

We will use C++ for the exercises. The code provided with the exercises has been tested with Visual C++, but you are of course free to use any compiler/IDE.

For reading and writing images, we will use *CImg*, a small and free library for working with image data in C++. Documentation for this library can be found at `http://cimg.eu/`. This library is included with the source code for the exercises.

Start by obtaining the source code for the exercises (LINK). The zip file contains three subdirectories, corresponding to the three exercises described below.

3 Exercise 1: Minimum cost paths

Implement Dijkstras algorithm to compute, for all pixels in an image, the cost of the shortest path between the pixel and a user specified set of seeds.

The directory contains the following source files:

- `main.cpp`: Contains the function `main(...)`

- `Dijkstra.cpp`: Contains the function `Dijkstra(...)`. You will write most of your code in this file.
- `Dijkstra.h`: Header file for `Dijkstra.cpp`
- `QueueElement.h`: We will use the priority queue implementation available in the C++ standard library. This header defines a class of *queue elements* to populate the priority queue. A queue element represents a path throught the graph, and contains information about the coordinate of the pixel representing the endpoint of the path, and the cost of the path.

The main function loads two images, specified by the command line arguments, into memory. The first image is the image used to define the graph in which we compute the shortest paths. The second image (which must have the same dimensions as the first one) is used to define the seedpoints. Every pixel in the second image whose intensity is greater than zero is taken to be a seed. Then, the main function calls the `Dijkstra()` function to compute length of the shortest paths from the seeds to all other pixels. The resulting distance values are then normalized to the range $0 - 255$ and written to the file `result.bmp`.

Your task is to complete the implementation of Dijkstras algorithm in the file `Dijkstra.cpp`. Assume that every pixel is adjacent to its four horizontal and vertical neighbors. We will not store the graph edges and their weight explicitly. Instead, we will calculate them implicitly during the execution of Dijkstras algorithm. First, let the weight of each edge be 1. Once you have completed the implementation, try to change the weight of each edge to be the absolute difference in intensity between the pixels spanned by the edge.

4 Exercise 2: Minimum spanning trees

Implement Prim's algorithm to compute a *segmentation by minimum spanning forests* relative to a set of labeled seeds. When completed, your program should output an image where every pixel is assigned the same label as the seed to which it is connected on a minimum spanning forest relative to the seeds.

The directory contains the following source files:

- `main.cpp`: Contains the function `main(...)`
- `MSF.cpp`: Contains the function `MSF(...)`. You will write most of your code in this file.

- `MSF.h`: Header file for `MSF.cpp`
- `QueueElement.h`: We will use the priority queue implementation available in the C++ standard library. This header defines a class of *queue elements* to populate the priority queue. A queue element represents an edge to be added to the forest, and contains information about the coordinate of the pixel representing the endpoint of the edge, the weight of the edge, and the label of the starting point of the edge.

The main function loads three images, specified by the command line arguments, into memory. All images must have the same dimensions.The first image is the image used to define the graph in which we compute the MSF. The second image is used to define the seedpoints. Every pixel in the second image whose intensity is greater than zero is taken to be a seed. The label of the each seed is read from the corresponding pixel in the third image. Then, the main function calls the `MSF()` function to compute an MSF segmentation. The resulting labeled image is written to the file `result.bmp`.

Your task is to complete the implementation of Prim's algorithm in the file `MSF.cpp`. Assume that every pixel is adjacent to its four horizontal and vertical neighbors. We will not store the graph edges and their weights explicitly. Instead, we will calculate them implicitly during the execution of Prim's algorithm. Let the weight of each edge to be the absolute difference in intensity between the pixels spanned by the edge.

5 Exercise 3: Minimal graph cuts

In this exercise we will use the min cut/max flow implementation availble from `http://vision.csd.uwo.ca/code/` to compute minimum graph cut segmentations on pixel adjacency graphs. In addition to the above mentioned max flow implementation, the directory contains a single file: `main.cpp`. Your task is to modify this file to solve the same two-label segmentation problem as in Exercise 2, using minimum graph cuts instead of minimum spanning forests for the actual segmentation step.

The given main function shows how to construct a simple graph and compute a binary segmentation corresponding to minimum cut. Modify this code to contruct a pixel adjacency graph identical to that in Exercise 2, perform minmum graph cut segmentation, and save the resulting labeling to an image file. How do the segmentation results compare to those obtained in Exercise 2?