Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Mathematical Foundations of Computer Graphics and Vision

EXERCISE 4 - OPTIMAL TRANSPORT AND GRAPH CUTS

Handout date: 09.04.2024
Submission deadline: 23.04.2024, 23:59

General Rules

Plagiarism note. Copying code (either from other students or from external sources) is strictly prohibited! We will be using automatic anti-plagiarism tools, and any violation of this rule will lead to expulsion from the class.

Late submissions up to 24h will be accepted with a penalty, except for extensions in case of serious illness or emergency. In that case please notify the assistant and provide a relevant medical certificate.

Software. All exercises of this course use Python. See the exercise session slides and this document for hints or specific functions that could be useful for your implementation.

What to hand in. Upload your solution in a .zip file on Moodle. The file must be called "MATHFOUND24-*-firstname-familyname.zip" (replace * with the assignment number). The .zip file MUST contain a single folder called "MATHFOUND24-*-firstname-familyname" with the following data inside:

- A folder named "code" containing your code
- A PDF README / Report file containing a description of what you've implemented and instructions for running it, as well as explanations/comments on your results.
- Screenshots of all your results with associated descriptions in the README file.

Grading. This homework is 8.3% of your final grade. Your submission will be graded according to the quality of the images produced by your program, the conformance of your program to the expected behaviour of the assignment, and your understanding of the underlying techniques used in the assignment. The submitted code must produce exactly the same images included in your submission (up to randomness).

1. PART 1: Optimal Transport

The objective of this section is to give you some basic hands-on experience on optimal transport in simple scenarios and see how even this simple cases can be used for interesting applications in computer graphics.

Given two probability distributions α and β over the space Ω. The optimal transport given the cost function C on $\Omega \times \Omega$ can be defined as

$$\inf_{P \in \Pi(\alpha,\beta)} = \int \int C(x,y) P(dx,dy) \tag{1}$$

1.1. Task 1 - Optimal Transport in 1D.

For simplicity, we will focus on discrete measures that are sampled from two distributions in $\mathbb{R}$. We can write the p-Wasserstein distance as

$$W_p^p = \min_{P \in \Pi(\alpha,\beta)} \sum_{i,j}^{n} \frac{1}{n} ||x_i - y_j||_p^p P_{i,j} \tag{2}$$

Let's consider two Gaussian distributions $\mathcal{N}_1(25, 5^2)$ and $\mathcal{N}_2(100, 10^2)$. We consider that we have n samples. These measures are discrete Diracs centered at points x_i and y_i in $\mathbb{R}$. In this special case, the optimal transport plan is simply obtained by sorting the projections and considering points in the resulting order: first with first, second with second, etc. The optimal transport is obtained through computing the difference between these points.

- Sample $n = 10000$ points from each distribution and plot them
- Compute the optimal transport between the two sets of samples, using the algorithm described above.
- There is a closed form solution for Optimal Transport between Gaussians in $\mathbb{R}^d$:

$$W_2^2(\mu_1, \mu_2) = \|m_1 - m_2\|_2^2 + \text{trace}\big(\Sigma_1 + \Sigma_2 - 2\big(\Sigma_2^{1/2}\Sigma_1\Sigma_2^{1/2}\big)^{1/2}\big) \tag{3}$$

  with $\mu_1 = \mathcal{N}(m_1, \Sigma_1)$ and $\mu_2 = \mathcal{N}(m_2, \Sigma_2)$. Use this equation to verify your OT result.
- Generate samples by interpolating between the two distribution using the obtained matching: $z_\lambda = \lambda * x + (1 - \lambda)y$
- Using Gaussian kernels, estimate the probability density function (PDF) of the resulting samples. For $\lambda \in \{0.25, 0.5, 0.75\}$, plot the results as illustrated in Figure 1

1.2. Task 2 - Color Transfer Using Sliced Optimal Transport.

We have seen in the course that computing the optimal transport in the general setting is computationally expensive (with respect to the number of measures). One approach to this issue is to leverage the efficient algorithm that exists in 1D. As you saw in Task 1, solving OT amounts to sorting the measures which can be done in $O(n \log n)$.

Sliced optimal transport is leveraging the 1D case by projecting the measures onto a 1D line $P_{\vec{\theta}} : \mathbb{R}^d \to \mathbb{R}$. The optimal transport problem is solved on this line. After the update step, the process is iterated, each time a different direction $\vec{\theta}$ is sampled for the projection.

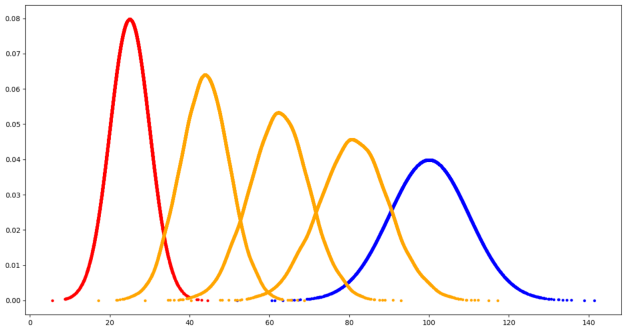


FIGURE 1. Interpolation between two distributions, using the optimal transport plan.

FIGURE 2. Color Transfer using Sliced Optimal Transport.

We will use this technique to compute a color transfer between two images of same size. This problem can be seen as a transportation of point clouds in $\mathbb{R}^3$ (considering the RGB color space).

We provide two images and the objective is to achieve the color transfer illustrated in Figure 2.

- Read the images and transform them into numpy arrays of shape (width, height, channels) with values in range $[0, 1]$. We recommend working at $\frac{1}{4}$ of the original resolution (especially during tests).
- Write a function that creates a random normalized vector $\vec{\theta}$ in 3D.
- Create two multisets (*i.e.* sets that allow duplicate values) of RGB values from the source and target images. Project the two pixel multisets on the direction defined by $\vec{\theta}$, to respectively obtain the multisets $\{x_i\}$ and $\{y_j\}$.
- Using OT in 1D, find the optimal transport plan between the two projected multisets.
- We use the optimal transport plan to get the update to make on the *source* as $u_i \cdot \vec{\theta}$, with $u_i = y_{s_i} - x_i$ and s_i indicating the index of the matching value from $\{y_i\}$
- To make this update step more robust, select multiple random directions and average the obtained update steps.

- To compute the color transfer, this operation needs to be repeated several times. Results in Figure 2 are obtained with 100 iterations each randomly selecting 5 different directions for each update step.

Hints:

- You can use `opencv` to load and display images (`pip install opencv-python` and then `import cv2`).
- `scipy.stats.gaussian_kde` and `np.argsort` might come in handy.

2. PART 2: Interactive Segmentation with Graph Cut

Segmentation is one of the most fundamental application of computer vision. It consists in partitioning an image into segments (a segment being here a set of pixels). A basic task for segmentation is the separation of foreground objects from background. An example of application is medical imaging, where physicians need to be able to immediately detect organs in a scan.

Fully automatic segmentation is difficult to achieve without training a classifier on large datasets. A tradeoff is to consider interactive segmentation. The user labels parts of the image as foreground and parts as background, and lets the computer figure out which segmentation is the best.

In this exercise, you will be asked to implement an interactive segmentation tool inspired by [1].

Since segmenting foreground from the background is a binary labeling problem, we can use graph cut methods to find the optimal labeling. To do so, one needs to build a proper graph that correspond to the problem and define the corresponding unary and pairwise cost. We then look for the minimal cut of the graph which gives the optimal labeling.

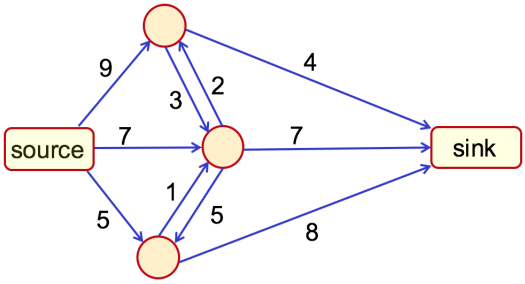


Figure 3. Graphical model

2.1. **Task 1 - Handling Max Flow.** As seen in the lecture, looking for the minimal cut is equivalent to looking for the maximal flow through the graph. In [2] Boykov and Kolmogorov gave a powerful algorithm for computing max flow. You will use an existing implementation of this algorithm.

In this task you will use a max flow implementation in order to solve the trivial graph represented on figure 3. It will enable you to understand how to use the library in order to apply it for the interactive segmentation task.

- Step 1: create the graph corresponding to figure 3
- Step 2: solve it and extract the optimal labeling
- Step 3: solve it manually and verify your answer

Hints:

- Install the `PyMaxflow` library (`pip install --user PyMaxflow`).
- Use the class `GraphCut` defined `graph_cut.py`. Use the methods `set_unary`, `set_pairwise`, `minimize`, `get_labeling`.
- We consider that the source S gives a label of 0 and the sink T a label of 1.

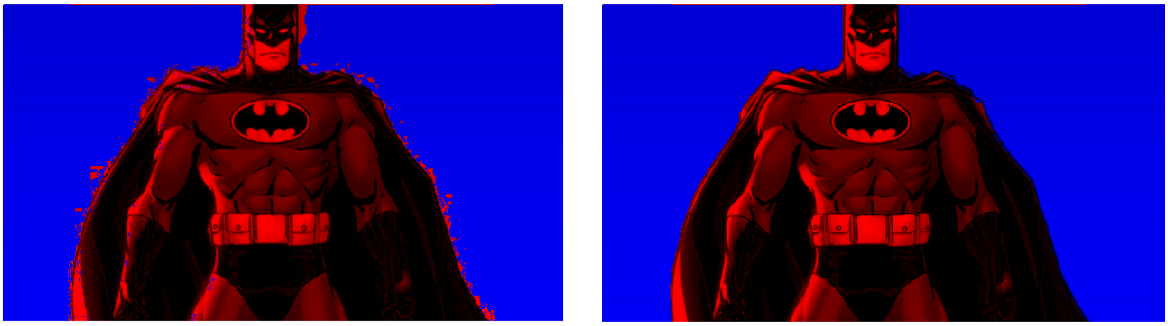
2.2. **Task 2: Interactive Segmentation.** You will use the max flow library in order to implement an interactive segmentation algorithm. We provide you the code which creates the interactive interface and extracts the position of points that are labeled by the user. Your task will be to focus on creating the graph using this information, solving the graph, and extracting the optimal labeling.

To do so, you will need to read [1] especially section 3 and the introduction of section 4 in order to understand what the unary and pairwise costs should be.

- Step 1: build a function that creates a color histogram using data from a given set of pixels.
  - Choose a resolution of 32^3 for your color histograms in order to have consistency (see hints).
  - Before normalizing your histogram such that it sums up to 1, you should smooth it in order to make it more general. You can use `scipy.ndimage.gaussian_filter`. In our examples we use a Gaussian smoothing with $\sigma = 0.1$.
- Step 2: build a function that create the unaries used in [1], using the color histograms you would have previously built.
- Step 3: build a function that creates the pairwise terms used in section 4 of [1]. Use the formula from section 4 of [1], with $\sigma = 5$.
- Step 4: build and solve the graph. Produce segmentation images 4 and 5.
- Step 5: Use the obtained segmentation to change the background of your image.

Hints:

- You will only have to modify `graph_cut_controller.py`. Look for the TODO comments.
- Use `coo_matrix` from the `scipy.sparse` library to initialize large sparse matrices. The `coo_matrix` has the following syntax for initialization:
  `coo_matrix((data, (row, col)), shape=(width, height))`
- Each of the RGB channels in the color images can take a value between 0 and 255, which means that there are 256 possible bins for each dimension that can be filled in order to create the histogram. This is too large for proper generalization considering that we only use few pixels from a single image to build histograms. This is why we choose a resolution of 32. You can use `numpy.histogramdd` to compute the histograms.



(a) Batman $\lambda = 1.0$ (b) Batman $\lambda = 2 \cdot 10^{-3}$

FIGURE 4. The Dark Knight Segmented

- However, some entries might still be never filled. Before applying the logarithm to your histograms when you build the unaries, you should also add a small value, such as 10^{-10}.
- In [1], the authors define a value $K = 1 + \max_{p \in \mathcal{P}} \sum_{\{p,q\} \in \mathcal{N}} \mathcal{B}_{p,q}$ for the unaries of pixels marked by the user as foreground (resp. background) if they are linked to the source $\mathcal{S}$ (resp. the sink $\mathcal{T}$). It is supposed to make sure that these pixels will not be marked as background (resp. foreground) after optimizing. You don't need to compute K, you just need to set it very high. The easiest way is to set it to infinity (`np.inf`).

2.3. **Written output for Part 2.**

- Optimal labeling for the graph of task 1
- Code for getting color histograms
- Code for building the unaries and the pairwise terms of the graphs
- Segmentation of the provided images (Batman and Van Damme), using the provided scribbles for foreground and background, and the following parameters:
  - The batman picture with $\lambda = 1.0$, and with $\lambda = 0.002$.
  - The Van Damme picture with $\lambda = 1.0$, and with $\lambda = 10^{-4}$.
  - You do not need to reproduce the provided reference images exactly.
- The Van Damme picture with a new background
- Segmentation of an image chosen by you.
- Discuss the benefits of graph cut for image segmentation

REFERENCES

[1] Y. Y. Boykov and M. P. Jolly. Interactive graph cuts for optimal boundary amp; region segmentation of objects in n-d images. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 1, pages 105–112 vol.1, 2001.

[2] Yuri Boykov and Vladimir Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(9):1124–1137, September 2004.

(a) JCVD $\lambda = 1.0$

(b) JCVD $\lambda = 10^{-4}$

FIGURE 5. Van Damme Segmented

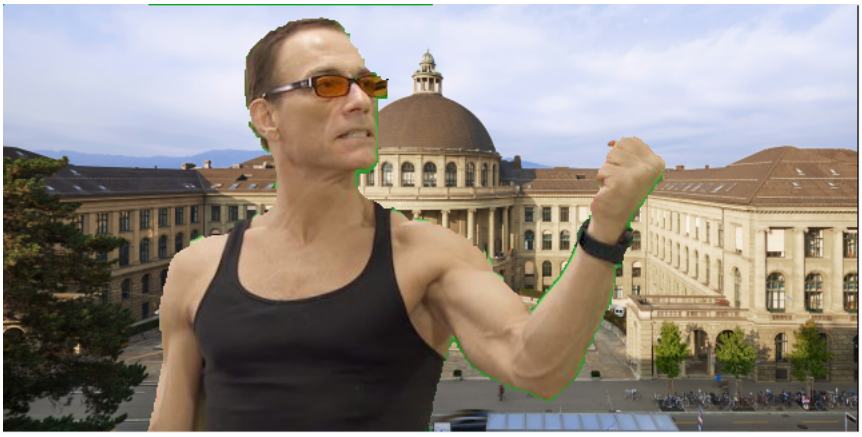


FIGURE 6. Van Damme at ETH