The Chinese University of Hong Kong, Shenzhen
SDS · School of Data Science
Andre Milzarek · Summer Semester 2020

MAT 3007 – Optimization

Midterm Project

Image Inpainting and Mosaics

The goal of this project is to investigate different optimization models and to utilize minimization methodologies that were introduced in the lecture to solve two imaging tasks. Specifically, we consider two classes of imaging problems that are known as *inpainting* and *mosaic generation*.

Background: Images. A *grey-scale* image $U \in \mathbb{R}^{m\times n}$ is typically represented as an $m \times n$ matrix where each entry U_{ij} represents a specific pixel of the image containing the color information. If the columns of $U = (u_{[1]}, u_{[2]}, \ldots, u_{[n]})$ are stacked on top of each other, we obtain the vector form

$$u = \mathrm{vec}(U) = (u_{[1]}^\top, u_{[2]}^\top, \ldots, u_{[n]}^\top)^\top \in \mathbb{R}^{mn}$$

of the image U. General color images $U \in \mathbb{R}^{m\times n\times 3}$ consist of three channels $U_1, U_2, U_3 \in \mathbb{R}^{m\times n}$ and each channel either contains the *red*, *green*, or *blue* color information to form the `RGB` color code of the image.

Part I – Image Inpainting. Image inpainting describes the task of recovering an image U from partial data and observations. In particular, we assume that parts of the image U are missing or damaged, (e.g., due to scratches, stains, or compression), and the aim is to reconstruct this missing or damaged information via solving a suitable inpainting or optimization problem.

Figure 1: Examples of different damaged images. We want to recover the missing image information in the red areas.

The images in Figure 1 show several typical situations where inpainting techniques can be applied. Our overall task is to reconstruct the red target areas. In this project, we assume that these target areas are known, i.e., we have access to a binary mask $\mathtt{Ind} \in \mathbb{R}^{m\times n}$ with

$$\mathtt{Ind}_{ij} = \begin{cases} 1 & \text{the pixel } (i,j) \text{ is not damaged,} \\ 0 & \text{the pixel } (i,j) \text{ is damaged.} \end{cases}$$

This mask contains the relevant information about missing or damaged parts in the image.

Let us set $\texttt{ind} := \text{vec}(\texttt{Ind}) \in \mathbb{R}^{mn}$, $s := \sum_{i=1}^{mn} \texttt{ind}_i$, and $\mathcal{I} := \{i : \texttt{ind}_i = 1\}$. Moreover, let $\mathcal{I} = \{q_1, q_2, ..., q_s\}$ denote the different elements of the index set $\mathcal{I}$. Then, we can define the following *selection matrix*

$$A = \begin{pmatrix} e_{q_1}^\top \\ \vdots \\ e_{q_s}^\top \end{pmatrix} \in \mathbb{R}^{s \times mn}, \quad e_j \in \mathbb{R}^{mn}, \quad [e_j]_i = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{otherwise,} \end{cases} \qquad \forall\, j \in \mathcal{I}.$$

The vectors e_j are unit vectors in $\mathbb{R}^{mn}$ and the matrix A selects all undamaged pixels of a stacked image according to the mask $\texttt{ind}$. In particular, if $U \in \mathbb{R}^{m \times n}$ is the input image and $u = \text{vec}(U)$ is its stacked version, then the vector

$$b = Au \in \mathbb{R}^s$$

contains the color information of all undamaged pixels of the original image U. Hence, we now want to find a reconstruction $y \in \mathbb{R}^{mn}$ of u such that:

(i) Original information of the undamaged parts in U is maintained, i.e., y satisfies

$$Ay = b \quad \text{or} \quad Ay \approx b. \tag{1}$$

(ii) The image y can recover the missing parts in U in a "suitable" way.

In this project, we will discuss different models and strategies to achieve these goals.

Part I – Project Tasks.

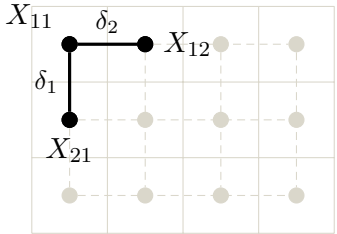
1. *Total Variation Minimization.* The linear system of equations (1) is underdetermined and has infinitely many possible solutions. In order to recover solutions with a "natural image structure", we first consider the so-called total variation image reconstruction problem

$$\min_x \ \|Dx\|_1 \qquad \text{s.t.} \qquad Ax = b, \tag{2}$$

where $A \in \mathbb{R}^{s \times mn}$ and $b \in \mathbb{R}^s$ are given as in (1). Here, $D \in \mathbb{R}^{\kappa \times mn}$, $\kappa = 2mn - m - n$, is a discretized version of the *image gradient*. For an input image $x = \text{vec}(X)$ and at the pixel (i, j), the image gradient Dx computes the difference between the pixel values

$$\delta_1 = X_{(i+1)j} - X_{ij} \quad \text{and} \quad \delta_2 = X_{i(j+1)} - X_{ij}$$

in the x- and y-direction of the image.



The total variation term $\|Dx\|_1$ penalizes the ℓ_1-norm of the differences δ_1 and δ_2 at all pixels. As a consequence, the objective function in (2) is minimized, when neighboring pixels have similar values.

- Reformulate problem (2) as a linear program and derive the associated dual of the linear optimization formulation.
- Write a `MATLAB` program to solve the linear reformulation of the optimization problem (2) and its associated dual problem. You can use the program `linprog` provided in `MATLAB` for the minimization.

– A suitable image quality measure is the so-called PSNR value. Suppose that $u^* = \mathrm{vec}(U^*)$ is the original true (and undamaged) image, then the PSNR value is defined via:

$$\mathrm{PSNR} := 10 \cdot \log_{10}\left[\frac{mn}{\|x - u^*\|^2}\right],$$

where x denotes the reconstructed image. The program `linprog` in `MATLAB` offers the solvers "`dual-simplex`" (default) and "`interior-point`". Test these two approaches and compare the results with respect to the PSNR value, the number of iterations, the required cpu-time, and the achieved duality gap. Plot the reconstructed images. How does the performance change when adjusting some of the parameters of the algorithms, e.g., when using a lower termination tolerance? How does the PSNR value behave for different tolerances?

– Run your code for different test images and masks and report your results.

Hints and Further Guidelines:

– On BB, you can find two datasets containing test images and test inpainting masks. Compare the performance of your methods on several images and different type of masks.

– Typically, the pixel values U_{ij} are represented by integer numbers in $[0, 255]$. Convert the images using the command `double(·)` and scale them to $[0, 1]$ by dividing each entry by 255.

– Under `help` > `linprog`, you can find useful information on the structure of `linprog` and on the different parameters.

– This application is a large-scale problem! In particular, if the input image has resolution 512×512, then the optimization variable will have (more than) 262 144 components and the matrix D has dimensions $523\,264 \times 262\,144$. A naive implementation of a matrix of this size requires over 1 000 GB of memory. Use the `sparse` format in `MATLAB` or other tools to build sparse and memory-efficient versions of D and A. (The program `linprog` allows sparse matrices as input and is able to exploit such a structure).

– The following `MATLAB` commands can be useful:

`imread`, `imshow`, `rgb2gray`, `sparse`, `speye`, `spdiags`, `reshape`.

You can also use the command `imresize` to rescale the images and to first work with smaller sizes.

2. *Sparse Reconstruction.* We now consider a related variant of problem (2) – the so-called ℓ_1-regularized image reconstruction problem. This optimization problem utilizes a slightly different regularization to solve the inpainting task (1); it is given by:

$$\min_{x \in \mathbb{R}^{mn}} \ \|\Psi x\|_1 \quad \text{s.t.} \quad \|Ax - b\|_\infty \leq \delta, \tag{3}$$

where $A \in \mathbb{R}^{s \times mn}$ and $b \in \mathbb{R}^s$ are derived as in (1) and $\delta > 0$ is an additional parameter.

Here, the matrix $\Psi \in \mathbb{R}^{mn \times mn}$ changes the basis and transfers the image x to the *frequency domain*. In this new basis or representation, many images tend to be very sparse and many of the components $[\Psi x]_i$ are zero or close to zero. This motivates the choice of the ℓ_1-norm, $\|x\|_1 = \sum_{i=1}^{mn} |x_i|$, in the model (3). The constraint in (3) corresponds to the condition (1) and is small when the pixels of undamaged parts of u and of the corresponding reconstruction are close. In this project, we want to use a block-wise discrete cosine transformation (Block-DCT) as sparse basis for the images. Such a transformation is, e.g., also used in the `jpeg` format for compressing images.

- Reformulate problem (3) as a linear program. Derive the associated dual of the linear optimization formulation and try to simplify the dual problem (e.g., by reducing the number of variables).
- Write a `MATLAB` program to solve the primal problem. You can again apply `linprog` using the `interior-point` solver. Test your approach on different images and masks. Plot the reconstructed images and discuss the performance of the algorithm, i.e., report the PSNR value, the number of iterations, and the required cpu-time. Run the algorithm with lower termination tolerances and compare the PSNR values and performance? Do you observe significant changes?
- Repeat your experiments for different choices of δ. A suitable initial choice is $\delta = 10^{-2}/s$. How do the reconstructions change when you increase or decrease δ? Compare the PSNR values with the results in part 1.) (on the same example) – does the total variation model achieve better results?
- Test your code in the following *denoising* setting: $A = I$, $b = u + \sigma$`randn(mn,1)`, where u denotes the original (stacked) image and the command `randn` adds Gaussian white noise (scaled by $\sigma > 0$) to the image. Run your code for some $\sigma \in [0.01, 0.1]$ and $\delta \in [0.5\sigma, \sigma]$ (e.g., $\sigma = 0.075$, $\delta = 0.9\sigma$). Plot the image(s) and compare the PNSR value of the reconstruction and the noisy input b.

Hints and Guidelines:

- On BB, we have provided a `MATLAB` function `Psi = get_Psi(m,n,bsz)` to compute the operator Ψ for given dimensions m and n and for a block-size parameter `bsz`. You can use the code to generate Ψ. This code assumes that the image or variable is partitioned in `bsz` $\times$ `bsz` blocks and that each of these blocks is first stacked to form a vector with dimension `bsz`^{2}. These smaller vectors are then stacked to the full image vector x in $\mathbb{R}^{mn}$. This form of stacking is different from the one described at the beginning, i.e., it is a different permutation of the pixels, and it will result in a different definition of A and b.

  In order to resolve the differences in these different formats, you can first apply the block-stacking to the input image U and to the mask `Ind` before forming A and b. This way, A and b will access and compare the right pixels in x.

  We recommend the choice `bsz` $= 8$ (or `bsz` $= 16$ for image sizes 256×256 or smaller).
- The following additional `MATLAB` command can be useful: `blockproc`.
- This problem can be computationally demanding. Try to implement the linear model as efficiently as possible! You can test your implementation first on smaller images (256×256 or smaller). The `MATLAB`-command `imresize(u,0.5)` can be used to scale the original image u to half of its size. We recommend to use the `interior-point` method when applying `linprog`.

Part II – Generating Mosaics. In this part of the project, we want to generate mosaics from an image using a given set of mosaic tiles via linear optimization approaches.

Suppose that we have a set of r individual mosaic tiles of size $\ell \times \ell$. We now want to use these different tiles to form a mosaic of a given target image $U \in \mathbb{R}^{m \times n}$. In order to match the tiles and the image, we assume $\text{mod}(m, \ell) = \text{mod}(n, \ell) = 0$ and first partition the target image into $\ell \times \ell$ blocks of pixels. We focus on a matching that is based on comparing the block brightness of the image and the tile brightness values. The block brightness of the image can be measured by taking the average of the pixel values.

image block (i, j):

0.35	0.4	0.45
0.5	0.55	0.4
0.6	0.6	0.65

$\Longrightarrow$ average brightness: $\beta_{i,j} = 0.5$

This will result in $\frac{m}{\ell} \cdot \frac{n}{\ell}$ block brightness values $\beta_{i,j}$, $i = 1, ..., \frac{m}{\ell}$, $j = 1, ..., \frac{n}{\ell}$ for the image and r brightness values c_k, $k = 1, ..., r$ for the different tiles (we can also simply use $c_k = k$ to differentiate the tiles). An exemplary set of tiles is shown in Figure 2.

We can use the matrix of block brightness values to (linearly) assign a unique tile to each image block. However, in order to obtain a balanced mosaic, we want to enforce that each type of tile is used the same number of times, i.e., $[\frac{m}{\ell} \cdot \frac{n}{\ell}]/r$ many times. (We again assume that the number $[\frac{m}{\ell} \cdot \frac{n}{\ell}]/r$ is an integer).

In this project, we model the decision variable as a tensor $x_{k,i,j}$ with the following constraints:

- We exactly place one (of the r many) tiles in block (i, j).
- We exactly place $[\frac{m}{\ell} \cdot \frac{n}{\ell}]/r$ copies of each tile.
- Each optimization variable is a binary number $x_{k,i,j} \in \{0, 1\}$, i.e., $x_{k,i,j}$ contains the information if we place the k-th tile at block (i, j) or not.

Our goal is to design a mosaic that resembles the target image. Suppose we place a copy of tile k in block (i, j). As the tile k has brightness c_k, and block (i, j) has brightness $\beta_{i,j}$, the cost for placing a copy of tile k in block (i, j) can be defined as the squared error $(c_k - \beta_{i,j})^2$. The task is now to minimize the total error of all combinations of tiles and blocks subject to the given constraints.

In Figure 3 and 4, we show exemplary mosaics by solving the described optimization problem.

Figure 2: An exemplary set of 12 tiles with size 40×40. The color information is gradually increasing from 0 (black) to 1 (white).

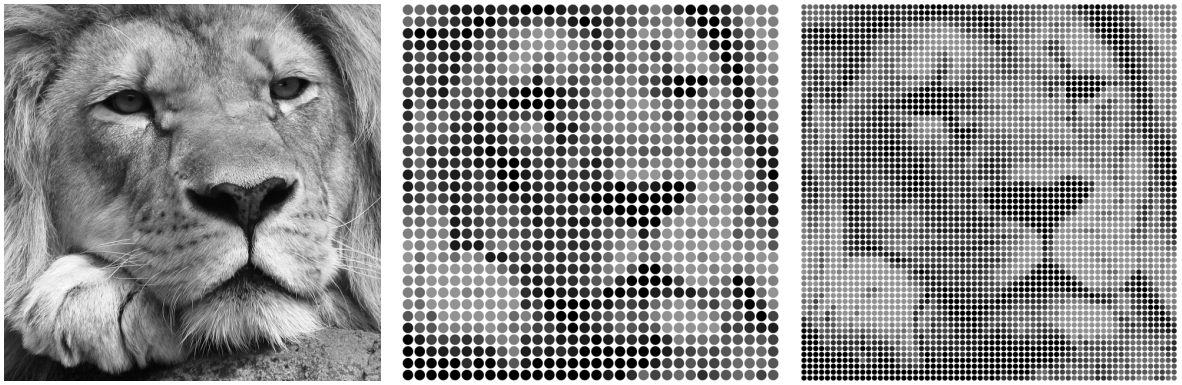


Figure 3: Generating mosaics via linear optimization. Left: ground truth image u. Middle: mosaic with 8 circular tiles of size 20×20. Right: mosaic with 8 circular tiles of size 10×10.

Part II – Project Tasks.

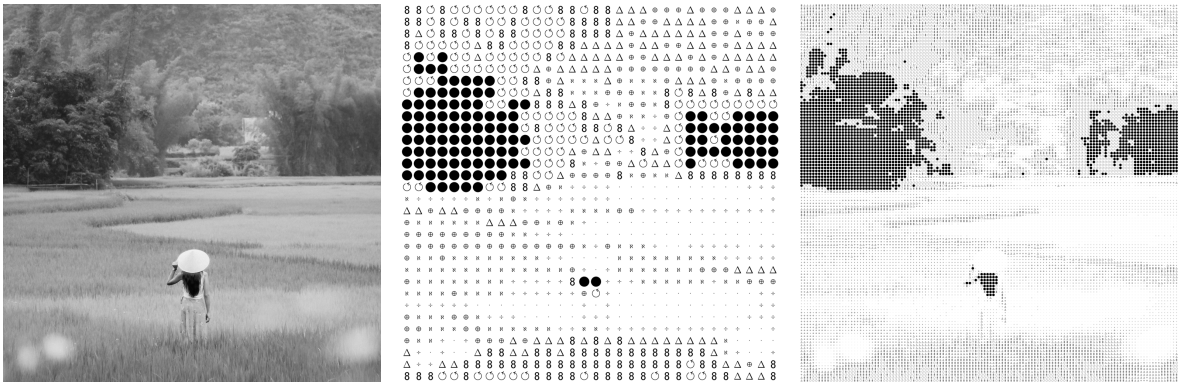


Figure 4: Generating mosaics via linear optimization. Left: ground truth image u. Middle: mosaic with 8 symbol tiles of size 32×32. Right: mosaic with 8 symbol tiles of size 8×8.

- Formulate the described mosaic generation task as a linear optimization problem with binary constraints.

  Write a `MATLAB` program that solves the linear program via the (dual) simplex method. In your implementation, you can use a continuous relaxation of the assignment problem, i.e., change the binary constraints $x_{k,i,j} \in \{0, 1\}$ to the continuous constraints $x_{k,i,j} \in [0, 1]$. Test different sets of tiles and images and plot the resulting mosaics. Report the runtime, the number of iterations, and investigate whether the obtained results are solutions of the original integer problem. Is the found mosaic a degenerate solution? What are your observations?

- Derive the dual of the relaxed linear optimization problem and implement a `MATLAB` program to solve it (using the simplex method). Compare the required cpu-time and number of iterations with the results of the primal problem. Calculate and report the duality gap.

- Suppose we have found a mosaic for the grey-scale version of a color image U. We also want to include the color information in the calculation of the brightness coefficients $\beta_{i,j}$ of U, i.e., $\beta_{i,j}$ is now computed as

$$\begin{aligned}\beta_{i,j} = \tfrac{1}{3} \times \text{ average in } \texttt{red} \text{ channel} \quad &+ \quad \tfrac{1}{3} \times \text{ average in } \texttt{green} \text{ channel} \\ &+ \quad \tfrac{1}{3} \times \text{ average in } \texttt{blue} \text{ channel.}\end{aligned}$$

  Based on the lectures, try to investigate whether the found mosaic is still optimal for this new objective function. Write a `MATLAB` program that checks whether the current mosaic needs to be recomputed when using this brightness measure. Test your code for different examples and report your results.

- Generate your own set of tiles (figures, geometric shapes, colors, smaller pictures, etc.) and experiment with the problem setup. Can you think of possible extensions, adjustments, or other creative applications of the mosaic generation task?

Hints and Guidelines:

- You can follow the hints and guidelines stated in the other project tasks. Implement the optimization model as efficiently as possible and exploit the structure of the LP using the `sparse` format in `MATLAB`.

- We have provided two sets of exemplary tiles on BB. Test different shapes and sizes of tiles and images. Try to use meaningful sets of tiles that allow to recover and maintain main features of the original images.

Project Report. This project is designed for groups of three students. Please send the following information to `huangzhipeng@cuhk.edu.cn` until **July, 8th, 11:00 am**:

- Name and student ID number of the participating students in your group, (group name).

Please contact the instructor in case your group is smaller to allow adjustments of the project outline and requirements.

A report should be written to summarize the project and to collect and present your different results. The report itself should take no more than 15–18 typed pages plus a possible additional appendix. It should cover the following topics and items:

- What is the project about?
- What have you done in the project? Which modeling or algorithmic components have you chosen to implement? What are your main contributions?
- Include the derivation and discussion of the optimization problem and dual formulations in your report.
- Summarize your main results and observations.
- Describe your conclusions about the different problems and methods that you have studied.

You can organize your report following the outlined structure in this project description. As the different parts in this project only depend very loosely on each other, you can choose to distribute the different tasks and parts among the group members. Please clearly indicate the responsibilities and contributions of each student and mention if you have received help from other groups, the teaching assistant, or the instructor.

Try to be brief, precise and fact-based. Main results should be presented by highly condensed and organized data and not just piles of raw data. To support your summary and conclusions, you can put more selected and organized data into an appendix which is not counted in the page limit. Please use a cover page that shows the names and student ID numbers of your group members.

The deadline for the report submission is **Saturday, July 18th, 11:00 pm**. Please submit your report (as pdf-document) and all supporting materials and code online using Blackboard.