Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Mathematical Foundations of Computer Graphics and Vision

EXERCISE 2 - GLOBAL OPTIMIZATION

Handout date: 07.03.2023
Submission deadline: 21.03.2023, 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 a single .zip or .ipynb file of your solution on Moodle. The file must be called "MATHFOUND23-*-firstname-familyname.{ipynb,zip}" (replace * with the assignment number). Write self-documenting code and comment the more complex parts. If submitting a .zip file, it must contain a single folder called "MATHFOUND23-*-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).

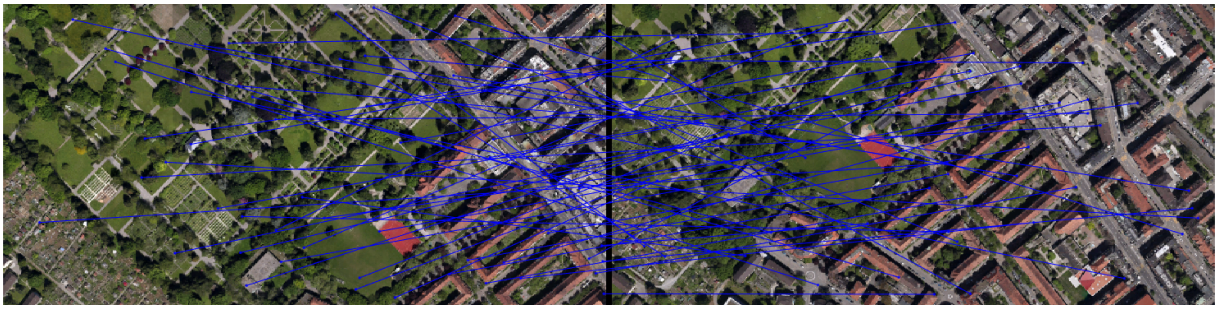
Goal of this exercise

In this exercise you will apply what you learned about global optimization, especially branch and bound (B&B), concave and convex envelopes, and reformulation. You will implement branch and bound for consensus set maximization.

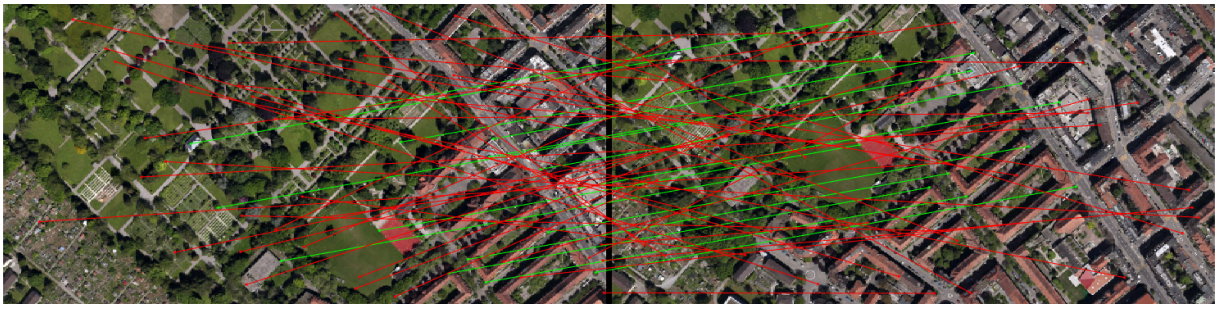
While not required, we encourage that you use the Jupyter notebook that is provided with the exercise.

1. EXERCISE: branch and bound for consensus set maximization

Context and goal. In the context of stereo matching, you will implement consensus set maximization by branch and bound where the model is a 2D translation between two input images. We provide the left and right images, and a set of 2D correspondences composed of inliers and outliers (see Figure 1(a)). The images are related by a 2D translation. Given this input set of correspondences, the goal is to maximize the consensus set (i.e. maximize the number of inlier correspondences), identify the inlier and outlier correspondences, and estimate the model (2D translation). The identified inliers and outliers are shown in Figure 1(b).



(a)



(b)

Figure 1. (a) input correspondences (b) the inlier and outlier correspondences obtained by branch and bound.

As seen in the lecture, the consensus set maximization problem can be formulated as follows. Let the set S of the input data be partitioned into an inlier-set $S_I \subseteq S$ and an outlier-set $S_O = S \setminus S_I$. The cardinality of S_I corresponds to the number of detected inliers. Let q_i represent the i-th data point/sample/information/feature, and Θ be the unknown underlying model. The general consensus set maximization can be mathematically formulated as:

$$\max_{\Theta, S_I} \quad \text{card}(S_I) \tag{1a}$$

$$\text{s.t.} \quad f(\Theta, q_i) \leq \delta, \forall i \in S_I \subseteq S \tag{1b}$$

This formulation simply states that, given a residual tolerance δ (the inlier threshold), the goal is to find the largest consensus set (i.e. maximize the number of inliers) under a model Θ and to estimate this model.

In our context of 2D translation with correspondences between two images, we note:

- the model $\Theta = T = (T_x, T_y) \in \mathbb{R}^2$ where T_x and T_y represent the translation along the x and y axis.
- the i-th input correspondence (p_i, p'_i) where p_i and p'_i represent the points in the left and right images. Their x and y coordinates are written $p_i = (x_i, y_i)$ and $p'_i = (x'_i, y'_i)$. We have n input correspondences, i.e. $i = 1, \ldots, n$
- the cost function f is composed of two parts: error on the x axis and error on the y axis:

$$f_x(\Theta, p_i, p'_i) = |x_i + T_x - x'_i| \tag{2a}$$

$$f_y(\Theta, p_i, p'_i) = |y_i + T_y - y'_i| \tag{2b}$$

A correspondence (p_i, p'_i) is considered inlier if the x and y residuals are below the inlier threshold:

$$|x_i + T_x - x'_i| \leq \delta \text{ and } |y_i + T_y - y'_i| \leq \delta \tag{3}$$

Our consensus set maximization problem can now be formulated as:

$$\max_{\Theta, S_I} \quad \text{card}(S_I) \tag{4a}$$

$$\text{s.t.} \quad f_x(\Theta, p_i, p'_i) \leq \delta, \forall i \in S_I \subseteq S \tag{4b}$$

$$f_y(\Theta, p_i, p'_i) \leq \delta, \forall i \in S_I \subseteq S \tag{4c}$$

i.e.

$$\max_{\Theta, S_I} \quad \text{card}(S_I) \tag{5a}$$

$$\text{s.t.} \quad |x_i + T_x - x'_i| \leq \delta, \forall i \in S_I \subseteq S \tag{5b}$$

$$|y_i + T_y - y'_i| \leq \delta, \forall i \in S_I \subseteq S \tag{5c}$$

What we provide. We provide you:

- the input left and right images: see `InputLeftImage.png` and `InputRightImage.png`. You can read them with `matplotlib.pyplot.imread('InputLeftImage.png')` from the `matplotlib.pyplot` package.

- a set of correspondences composed of inliers and outliers: see `ListInputPoints.mat`. Its size is $n \times 4$ where n is the number of input correspondences. The i-th row contains the coordinates of the points of the i-th correspondence: (x_i, y_i, x'_i, y'_i). In Python you can use `scipy.io.loadmat('file.mat')` from the `scipy.io` package.
- the inlier threshold $\delta = 3$ pixels.
- a Jupyter notebook that implements the reading/plotting functionality and guides the structure of your code. It is not mandatory to use the notebook and you may copy parts from it (indicate that in your code).

Required deliverables

You are required to provide the following deliverables

A. Derivation of the problem formulation in the canonical form of Linear Programming (see Section 2). It is fine to derive it in a form that is required for `scipy.optimize.linprog`.

B. The code implementing branch and bound for consensus set maximization with a 2D translation model

C. The results of the translation model, and the indices of the inliers and outliers obtained by branch and bound

D. A figure showing the identified inlier and outlier correspondences, like Figure 1(b)

E. A figure showing the convergence of the cardinality bounds, i.e. the highest lower bound obtained so far, and the highest upper bound still in the list. See Fig. 2.

Discussion (optional, not graded). In the previous exercise, you implemented RANSAC to fit polynomials. Now, can you use RANSAC to solve the consensus set maximization problem for stereo matching? If so, please briefly describe the main steps you would take to implement RANSAC for this task, and compare it with the BnB approach. Additionally, can you identify scenarios where one approach may be more effective than the other? If not, explain why it is not feasible to adapt RANSAC for this application. Can you also combine RANSAC and BnB in one pipeline?

2. Hints and Notation

Here is some additional information to help you with branch and bound:

- the model Θ is 2D translation (T_x, T_y). Lets note the lower and upper (definition) bounds of Θ as $\underline{T_x}, \overline{T_x}, \underline{T_y}, \overline{T_y}$.
- given a space of Θ (i.e. $T_x \in [\underline{T_x}, \overline{T_x}]$ and $T_y \in [\underline{T_y}, \overline{T_y}]$)
  - compute the upper bound of the nb of inliers by solving the LP
  - compute the lower bound of the nb of inliers by simply testing the model Θ obtained by LP (it is inside the current definition space) or a specific Θ in the space, for example the one at the center, i.e. test $\Theta = (T_x, T_y) = \Big((\underline{T_x} + \overline{T_x})/2, (\underline{T_y} + \overline{T_y})/2\Big)$.
- conduct a greedy search: at each iteration take the best candidate in the list, split it into two children, compute their cardinality bounds, and remove all the elements in the list with

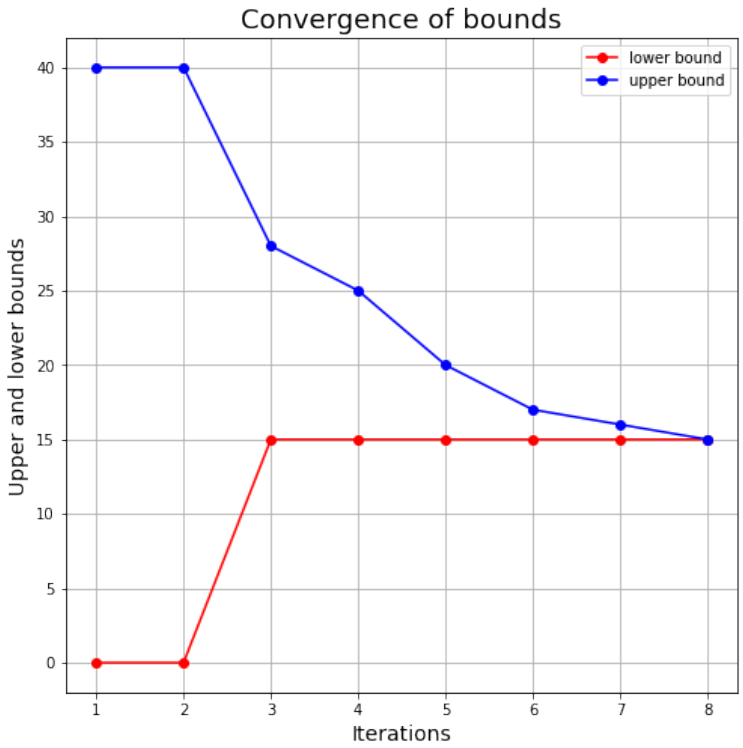


FIGURE 2. Evolution of the lower and upper bounds of the number of inliers along the branch and bound iterations.

a bad bound. When you take the best candidate, remove it from the list, and put its two children into the list

- iterate and remove the spaces that definitely do not contain the optimal solution. For example, let m^* be the highest lower bound of the number of inliers obtained so far. If the upper cardinality bound of a space is less than m^*, it can be safely removed, because even in the best case it cannot contain a better solution.
- when branching, split the current space into two children subspaces in half along the longest dimension
- the iterations stop when the lower and upper bound are nearer than 1, because they will lead to the same integer number of inliers

Reformulation. The first task is to reformulate Eq. (5a) to obtain a linear programming. For this, as seen in the lecture:

- introduce the identification binary variable z_i with $i = 1, \dots, n$. Let $z_i = 1$ if the i-th correspondence is an inlier, i.e. if $f_x(\Theta, p_i, p_i') \leq \delta$ and $f_y(\Theta, p_i, p_i') \leq \delta$. Otherwise $z_i = 0$, i.e. the i-th correspondence is an outlier.
- relax the binary variables z_i

- to avoid the bilinear terms, introduce the auxiliary variables $w_{ix} = z_i T_x$ and $w_{iy} = z_i T_y$
- replace the bilinear terms using the concave and convex envelopes
- reformulate the constraints containing absolute terms, i.e. $|a| \leq b \Rightarrow a \leq b$ and $-a \leq b$
- write down the final linear program system in canonical form

Implementation hints. Here are some implementation hints (feel free to use or ignore them):

- create a list that contains the lower and upper (definition) bounds of Θ, i.e. $\underline{T_x}, \overline{T_x}, \underline{T_y}, \overline{T_y}$, and its lower and upper cardinality bounds. The list can be a structure with the fields ThetaLowerBound (i.e. $(\underline{T_x}, \underline{T_y})$), ThetaUpperbound (i.e. $(\overline{T_x}, \overline{T_y})$), ObjLowerBound (lower bound of the nb of inliers, obtained by testing a certain model), ObjUpperBound (upper bound of the nb of inliers, obtained by LP, "obj" stands for objective function), ThetaOptimizer (the model obtained by LP);
- You can use Python's `heapq` to build a priority queue to efficiently find the next subproblem;
- To avoid issues with BnB implementation that are caused by incorrect LP derivation or implementation, you can use a naïve upper bound estimation by checking box constraints (implemented for you in the provided notebook).
- for LP,
  - we have a long unknown vector that is the concatenation of Θ, $\mathbf{z}$ and $\mathbf{w}$. If we note $\mathbf{x}$ the unknown vector of LP, then we have something like $\mathbf{x} = (\Theta, \mathbf{z}, \mathbf{w})$, for example:

$$\mathbf{x} = (T_x, T_y, z_1, \ldots, z_n, w_{1x}, w_{1y}, \ldots, w_{nx}, w_{ny}) \tag{6}$$

    or for example:

$$\mathbf{x} = (T_x, T_y, z_1, \ldots, z_n, w_{1x}, \ldots, w_{nx}, w_{1y}, \ldots, w_{ny}) \tag{7}$$

  - construct c, A and b, and build the canonical form (in Python for `linprog`):

$$\begin{aligned} &\min_{\mathbf{x}} c^T \mathbf{x} \\ &\text{s.t. } A\mathbf{x} \leq b \\ &\text{and } l_b \leq \mathbf{x} \leq u_b \end{aligned}$$

  - if using `scipy.optimize.linprog` solver, `method="highs-ds"` should be fast and stable, but you may use different methods
- to compute the lower bound with LP, create a function whose inputs are the definition bounds of Θ, the data points and the inlier threshold, and returns the LP results, i.e. the optimized model Θ and the objective cost (the floating point (not integer) sum of the relaxed z_i)
- We will not grade the execution speed, but here are some hints if you want to have a fast execution. Your code can get slow when the matrix size increases, so set the matrix size at the beginning, for example the matrix A and b of the linear programming. Note also that some parts of the matrix are always the same throughout the iterations (i.e. the equations independent of the Θ space bounds), so instead of setting them at each new iteration, you can set them only once during the initialization. Python make use of vectorization, so avoiding loops as much as you can is usually a good idea. Again, we will

not grade the execution time. A good implementation will run in a few second. A very naive implementation might run in minutes.

Linear Programming (not graded). This part is not graded. It is only meant to help you get familiar with setting up a linear programming system.

For example, solve the following Linear Programming using linprog:

$$\begin{aligned} \min_{x_1,x_2,x_3} \; & 3x_1 + -7x_2 9x_3 \\ \text{s.t. } & 3x_1 4x_2 + 8x_3 \leq -12 \\ & x_1 + 2x_2 + -7x_3 \leq 3 \\ & -5 \leq x_i \leq 7 \text{ for } i = 1, \ldots, 3 \end{aligned}$$

The following example needs conversion to use linprog (note the max, the inequality signs, and that some variables do not exist in all the constraints):

$$\begin{aligned} \max_{x,y,z} \; & 3x - 2y + 4z \\ \text{s.t. } & 4x - z \geq -3y + 12 \\ & 5x \leq 4 + z \\ & 0 \leq x \leq 3, 4 \leq y, 1 \leq z \leq 7 \end{aligned}$$