Homework 4: Topology Optimization

6.4420

Computational Design and Fabrication
Assigned:04/20/2024
Due: **05/04/2024 at 11:59 PM**

Please update your codebase by executing following command in your codebase folder:

```
git add ./*.py
git commit -m "My assignment 4 code."
git pull
```

1 Performance Exploration (20 points)

In this part, you will be asked to find and plot pareto fronts, commonly used to understand optimality and trade-offs for performance metrics in multiple objectives. You can imagine plotting the pareto front to be the last step in a pipeline that defines a mapping function from design space to performance space.

For example, a design space could be a parameterization of an arch bridge mesh. You could voxelize the generated mesh and use finite element analysis to compute the compliance under given external force. Then, you could use the compliance and the total mass of the mesh to define the 2D performance space. Finally, you would find and plot the Pareto front.

Let's recall the definition of Pareto optimality. Let $\mathcal{X}$ be some design space and $F : \mathcal{X} \mapsto \mathbb{R}^k$ be a function that computes the k-dimensional objective of a design. Here, lower objective values are better. A point $x \in \mathcal{X}$ is *pareto optimal* iff there does not exists any $\mathbf{x}' \in \mathcal{X}$ such that $F_i(\mathbf{x}') \leq F_i(\mathbf{x})$ for all i and $F_i(\mathbf{x}') < F_i(\mathbf{x})$ for at least one i.

In this task, you will need to come up with an $O(n \log n)$ algorithm to find the Pareto front given points in 2-dimensional objective space. We provide a brute-force reference implementation to help you validate your optimized version, and a test set of points in the `data` folder. You will be graded on the asymptotic properties of your code's runtime, which should be near-instantaneous on the set of 1,000,000 points we have provided (last testcase).

1.1 (20 points) Come up with an $O(n \log n)$ algorithm for finding the Pareto front of 2D points in the `pareto_front` function in `pareto.py`. You can run `python pareto.py -n 10` to run your algorithm against random points or `python pareto.py -f data/points_0.npy` to run against the provided test cases (see the `data` folder). The code will also output an image in the `output` folder. Run `run.sh` to run against all test cases. Describe the algorithm in your report, justifying your runtime bounds and highlighting any edge cases you handled in your implementation. Include all the images in the write-up or include them in the zip file.

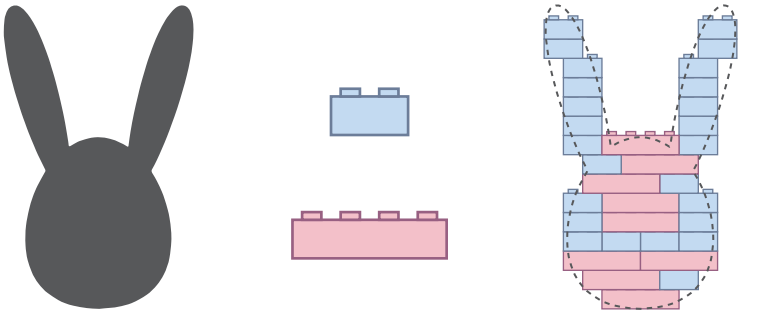
2 End-to-End Design and Optimization Examples (50 points)

The goal of this exercise is to leverage the comprehensive foundations we have built during the course, guiding students to design end-to-end solutions for performance-driven optimization of shapes that will be manufactured. You will be given two examples of design-for-fabrication tasks and asked to create a formalization so that you can search for optimal designs. You don't have to code them up, but doing so will earn extra credit. However, you must specify all variables and algorithms clearly and write detailed pseudocode to demonstrate a deep understanding of how these optimization methods should work.

2.1 Discrete Optimization with MCMC (30 points)

For this task, you will be given an input image of a 2D silhouette, such as the bunny shown in Figure 1, and a set of LEGO blocks of two sizes: 2-by-2 and 2-by-4. Your goal is to assemble LEGO blocks to resemble the image as closely as possible, considering a "2D" composition, as illustrated in Figure 1. You must ensure that all the LEGO pieces are fully connected.
Assumptions you can make: Assume you have an infinite stash. You can also assume there is a "resolution" parameter that can be tweaked by the user, so you can determine how large each LEGO piece will be with respect to the image. You can also assume that the silhouette is a single closed contour (but note that this doesn't matter too much for the solution). If you would like to make any other assumptions, please specify them.

To help you create a pseudocode to solve this problem, we will break it down into a few steps. We suggest using an MCMC method similar to the one we discussed in class for solving the knapsack problem. You can refer to the slides for the algorithm.



(a) Silhouette (b) LEGO Pieces (c) Lego Piece Assembly

Figure 1: Illustration for 2.1

2.1.1 (10 points) Define the design and action space. Your first goal is to define how you will represent a LEGO assembly (Hint: there are two types of LEGOs, and you may consider discretizing the image into a 2D grid to help define the positions of each LEGO). The second thing you want to define is the action space (Hint: you may want to consider adding/removing a LEGO at each iteration block). Importantly, this design space does not need to ensure that anything you specify satisfies physical constraints because you can build checkers for those in the next step (remember how in the knapsack problem if we had an infeasible solution we just rejected it).

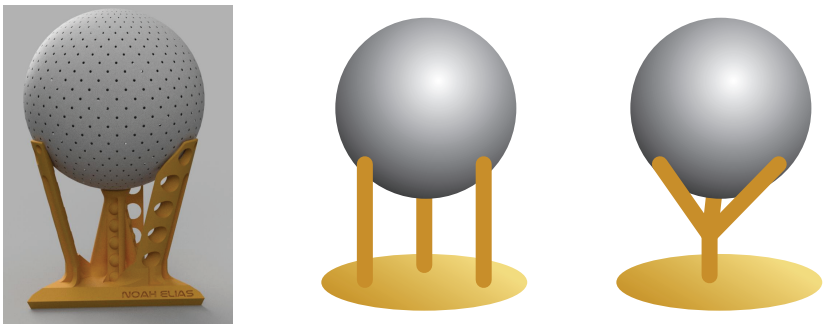
2.1.2 (10 points) Constraint Checking. At each iteration, after performing an action, you might end up with a design that cannot be physically realized or that does not meet the constraints of connecting to one piece. These actions should be rejected. Based on how you defined the design and action space, determine what needs to be checked and propose algorithms for checking them. Write pseudocodes for these algorithms as functions that you will call in Step 3.

2.1.3 (5 points) Write MCMC pseudocode for optimizing the LEGO assembly given a 2D shape (Hint: consider using annealing).

2.1.4 (5 points) Note that this sequence of steps guides you in writing a solution where many of the actions you take will be rejected. This is fine, but it means the algorithm may take a very long time to converge. At a high level, discuss potential solutions for this. Note that your solutions don't have to be perfect, and they can involve trade-offs (e.g., you can suggest something that will converge faster but will cover a smaller design space). Please make sure that you emphasize these trade-offs, however.

2.2 Continuous Optimization for 3D Printing (20 points)

For this task, you will be given an input 3D shape that you want to display in your home, and your goal is to design a 3D printable display stand for this object. Your objective is



(a) (b) (c)

Figure 2: Illustration for 2.2

to use the minimal amount of material and no support material while ensuring the object is stable and at a given input height. We will also assume that you are given three contact points as input, so that if you touch the object at those three points, it will be stable. (see Figure 2a).

Note that the simplest solution would be to just 3D print straight rods connecting the shape to a 3D printable base. However, these may require more material than solutions where you first connect the three rods together and then have only one rod down to the base (compare Figures 2b and c). Our goal is to have a solution that achieves the latter to minimize material, and your task is to come up with an algorithm to find such a design.

Assumptions you can make: Assume that the material you are using is infinitely strong, and that you are given as input a rod radius. You are also given the draft angle requirement of the 3D printer. You can also assume that the shape is convex so that you don't have to worry about it being impossible to insert the shape over the display. Note, however, that this doesn't mean that you don't have to check for collisions at all.

To help you create pseudocode to solve this problem, we will break it down into a few steps. We suggest treating this as a continuous optimization with constraints.

2.2.1 (10 points) Define the Design Space. Your first goal is to define how you will represent the space, which includes the set of variables and constraints. The variables should define both the information of the central rod and the 3 connecting rods. Hint: consider the relationship between the position of the central rod and the center of mass of the object. Consider the two constraints for how each of the three connecting rods must attach to the central rod: you want to consider the draft angle to avoid support material and you must avoid collisions.

2.2.2 (10 points) Express this problem as an optimization problem with pseudocode. This includes expressing a cost function to minimize material usage. Assume you are calling a function that optimizes from a library. Describe what kind of function that will be.

3 High-Level Understanding (30 points)

In this section of the course, we studied how to optimize design spaces to achieve a given performance. We talked about methods for evaluating performance as well as search algorithms to explore this space. In this section, we will ask you what you have learned in class.

3.1 Combining Objectives (10 points)

In multi-objective optimization we need to find not one optimal solution, but a landscape of possible design trade-offs. One could imagine that this would be the result of several single-objective optimizations, for example optimizing for cost, or material weight, on their own.

> Give (i) a simple example of what these objective functions could look like mathematically, (ii) a method by which they might be combined into a single objective that can be optimized to sample points along a pareto frontier, (iii) scenarios where this combination method would produce an undesirable pareto frontier, and (iv) a concrete example of this undesirable scenario.

3.2 Multiple Domains and Bi-level Optimization (10 points)

In designing cyber-physical systems, such as robots, we often need to concurrently optimize across multiple domains (e.g. optimize both the shape *and* motion of a robot), which can translate to a bi-level problem (i.e., nested optimization loops, where we first optimize a shape and then optimize the motion). However, in some scenarios, it is possible to treat co-design across multiple domains as a single-level optimization, if we carefully design our *design space*, i.e. the range of properties we allow to vary.

> Your task is to describe two examples of design spaces for cyber-physical systems: one which can be treated as a single-level optimization and one that cannot, and discuss why. (Hint: consider the case of our walking robot. If we allow the geometry to vary arbitrarily, what does this imply for our motion design space? Can we constrain the design space of valid geometry in a way that makes the overall geometry + motion optimization a single-objective one?)

3.3 Minimizing Expensive Performance Evaluation (10 points)

Consider a scenario when you are trying to optimize over a design space, but the performance cannot be simulated computationally, i.e. in order to evaluate the performance of a point in the design space, you need to fabricate the object and physically measure the performance. You do have the equipment to do this, but the process of fabrication and taking measurements is expensive.

The optimization methods that we have discussed so far involve evaluating the performance objective many times; describe two alternatives to doing so based on the ideas described in class, and discuss when you would use each. (Hint: it may be useful to consider cases where you have some previous expertise or data from performing similar experiments.)

4 Submission Guidelines

Create a zip file containing:

- `writeup.pdf`
- `pareto.py`
- Pareto plot images or include them in the writeup.

Good style and comments are expected. Code which does not compile in the environment specified cannot and will not be graded. If you add code outside of the sections we recommend editing, please tell us where specifically to look in your writeup.

We will grade your assignment based on both your implementation and your write-up.

5 Extra Credit

We don't expect all students to do every extra credit option; instead, we create many extra credit opportunities so that students can explore what they're interested in. Often, these extra credit questions allow students to experiment with software that is used by academics and industry professionals. If you have something in mind to do for extra credit that's not listed here, we would be happy to accommodate; please let us know or post on Ed so other students can also follow up.

5.1 Experimenting with Bayesian Optimization (10 points)

Design your own experiment and use AutoOED (https://www.autooed.org/), an online platform for Bayesian Optimization, to perform optimization and analysis with it. Submit in your writeup a description of the purpose of your experiment, the parameters you were optimizing over, the performance metrics you considered, and your results. Include a discussion of your experience and some relevant screenshots.

5.2 Libraries for Multi-Objective Optimization (10 points)

Pymoo (https://pypi.org/project/pymoo/0.1.2/) is a library with implementations of different multi-objective optimization algorithms. Try out some of the different approaches on a dataset of your choice, and report on your results. Include a discussion of the differences you observed between the algorithms you chose, and some screenshots.

5.3 Experimental Libraries for Optimization (10 points)

Many unusual or unfamiliar approaches to optimization exist beyond what we have discussed in class. Find such a library (the course staff thinks https://gpkit.readthedocs.io/en/latest/index.html looks quite interesting). Try performing optimization on a dataset of your choice, and report on your results and what you learned to complete this extra credit. Include a discussion of your experience and some relevant screenshots.