**ISyE4133: Advanced Optimization**

# Project #1: Circuit Design

Checkpoint: *May 30 (11:59pm EST)* Project: Jun 11 (11:59pm EST)

Adapted from previous projects by Drs. Mathieu Dahan and Johannes Milz.

## Problem Description

The traveling salesman problem (TSP) is a fundamental problem from combinatorial optimization that has several real-world routing applications, including transportation, DNA sequencing, astronomy, and circuit design. Formally, we are given an undirected graph $G = (V, E)$, an arbitrary starting vertex $v_s \in V$, and a set of edge weights $w : E \to \mathbb{R}_{\geq 0}$. The goal is to find a tour starting at $v_s \in V$ (i.e., a route that starts and ends at $v_s$ and visits each node in $V$ exactly once) and this tour should be the shortest path tour in terms of the cumulative edge weights.

Below, we give a preview of applying TSP towards the circuit design problem. Given a set of locations we want to drill holes (i.e. nodes), we want to determine the most efficient drilling route. In particular, in this problem we will work with 280 node graph, representing 280 drill holds. The file, `a280.tsp.gz`, can be downloaded from here

http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/.

In the image below, one feasible drilling route is shown, where the thin black lines represents the route the drill can take. For more details, see this tutorial: https://developers.google.com/optimization/routing/tsp.

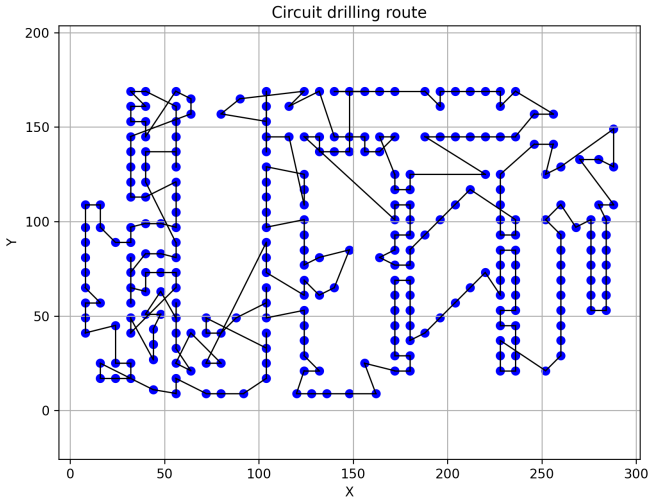



Figure 1: Circuit drilling route. Blue circles are nodes or holes while black lines are the route.

One can see this problem can be formulated as a TSP problem, and as shown in class, can be formulated as a integer (possibly mixed-integer linear) program. However, there are an exponential

number of routes, so direct enumeration is impossible. In this project, we will discuss ways we can solve this seemingly intractable integer program using different formulations and Gurobi callbacks. Students will:

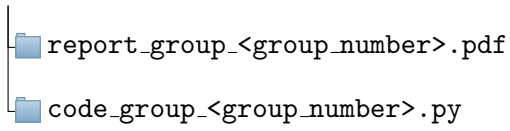
1. Utilize basic I/O tools to interact with problem specific files
2. Practice basic visualization of graph problems
3. Learn a couple mixed integer linear programming (MILP) formulations of TSP
4. Implement callbacks with Gurobi for improving and monitoring performance for a MILP solver

## Submission Instructions

- Please upload:
    1. A single **typed** PDF report with your answers (when applicable) to the questions below. Moreover, all plots in your report must be digitally created with a clear and appropriate title, x and y-axis label, several x and y-axis tick labels[1], and a legend when appropriate. Your submission should also contain a brief paragraph explaining the contribution of each member.
    2. Python code. Your code must be able to run if we directly copy and paste the contents into a Jupyter/Colab notebook. **Code that does not run may automatically receive a zero for the code portion**[2]. You can assume when I run the code, I have a local copy of `a280.tsp` available and a full Gurobi license.
- Outside references, including generative AI are allowed. But the submitted work must be one's own original work and effort. Use of outside reference must be properly cited, i.e., name of the reference, a link to the reference (if possible), and how the reference was used. Collaborations are only permitted within the group.

### File Directory Structure

Your submission should consist of one PDF and one Python file. I will assign each group a number when they submit the project checkpoint. When archiving your Python files and preparing your PDF report, please use the following file directory structure for your submission:




For example, `report_group_<group_number>.pdf` should be submitted as `report_group_1.pdf` if your group number is 1. You may lose a small number of points if your submission does not adhere to this format.

[1] Matplotlib will usually have these automatically generated.
[2] I will try to be lenient for small bugs due to issues between running code on different machines.

## Details

- Use the dataset described in the "Problem Description" section. In particular, we will work with the file `a280.tsp`. Within the file, you will notice about 6 lines of the file description, followed by a sequences of lines containing 3 integer values each. The first is the node id, the second and third are the x- and y-coordinate of the corresponding node. The library `tsplib95` will automatically parse this for you.
- In your code, use comments to identify which codes correspond to answering which questions. This will make it easier for staff to grade the coding portion. Code that is written poorly and does not have comments to guide reader may thus lose points due to low code "read-ability". **Note**: You can complete your work in a Colab notebook and convert to a Python file afterwards. Text cells are automatically converted to multi-line comments.
- Since we will not cover mixed integer linear programming and TSP until the second week of Unit 2, Canvas will contain some lectures on TSP formulations from previous courses and basic code for invoking callbacks within Gurobi as a guideline.
- When the MIP gap is mentioned, we refer to the same relative MIP gap Gurobi uses. See https://support.gurobi.com/hc/en-us/articles/8265539575953-What-is-the-MIPGap.
- If you run your code on Colab, the free license may not be sufficient. Refer to HW#1 on instructions to get the full license.
- Grading is roughly 50% report (includes correctness and presentation, i.e., writing is mostly clear and not too many grammatical errors), 45% for code (includes ability to run, correct, and "read-able", i.e., should be able to be read and understood by someone who has not seen your code before and should not contain excess comments nor commented out code), and 5% for completing the project checkpoint.
- If you feel some group members did not sufficiently contribute to the project and caused the overall quality to go down and do not know how to resolve it, you can privately email the instructor to find alternative grading guidelines.

To get you started, you may refer to the following starter code with hints:

https://colab.research.google.com/drive/1agEELNnmHCsUCz2fPmtuViBS04qVVfdO?usp=sharing

Additionally, since we will not cover TSP nor callbacks until the middle of Week 4, you can find previous notes on TSP formulations and high-level details on callbacks for lazy constraints. These will be on Canvas under `files/lecture_unfilled`.

## Questions

1. (10 points) Using the library `tsplib95` (src: https://tsplib95.readthedocs.io/en/stable/index.html), load the circuit printing problem data `a280.tsp.gz` and plot the circuit drill holes (without the routes) similarly to Figure 1. Put the plot in your report. The plot should be a scatter plot with clearly labeled X and Y-axes with tick values, plus a title.

   **Hint**: You may need to access the `node_coords` attribute of the TSP problem object.

2. (10 points) Formulate the TSP the problem but without any type of subtour elimination constraints. Clearly identify and explain the variables, objective, constraints, and data. In particular, explain how you will construct the data for the TSP problem using the TSP file.

3. (15 points) Design a heuristic method that does not require solving a mixed integer linear program (MILP) for approximately solving TSP. Write pseudocode (not a screenshot of your code) that clearly explains how to find this approximate solution.

   **Hint**: Consider a greedy approach.

4. (15 points) Implement your method from the previous part. Afterwards, complete the following:

   - Plot the circuit drilling route on top of the circuit drilling holds from Part 1.
   - Print out and put in your report the final objective value and total runtime.

5. (15 points) Implement the TSP formulation with Miller-Tucker-Zemlin (MTZ) constraints as described in class. Solve with Gurobi with a time limit of 10 minutes or until optimality. Then

   - Plot the circuit drilling route on top of the circuit drilling holds from Part 1.
   - Print out and put in your report the final objective value, MIP gap, and total runtime.

6. (20 points) Repeat Part 5 but without a time limit. Instead, implement a callback to early stop the solver when the relative MIP gap falls below 50%, 20%, and 5%.

   In addition, provide a qualitative comparison between the route provided at each checkpoint.

   **Note**: When solving the MILP multiple times, make sure to remake the Gurobi model between solves. This is because Gurobi may save the solution and warm-start the solver with the previous solution if the Guorbi model is unchanged.

7. (15 points) Recall the subtour elimination constraints. We call a subtour undesirable if it does not visit all nodes. Write a pseudocode (not a screenshot of your code) that clearly explains how to determine if your current solution contains an undesirable subtour. If it does, your pseudocode should output the undesirable subtour.

8. (25 points) Repeat Part 5 but use subtour elimination (together with your pseudocode for subtour detection in the previous part) with lazy constraints via callbacks in Gurobi. What do you notice in runtime performance between MTZ constraints and lazy constraints?

9. (10 points) Due to technical issues in the circuit printer, the route cannot include any edges that move vertically along the column $x = 104$ (i.e., an edge between two nodes that share the same x-coordinate of $y = 104$). Explain what constraints you will add to the TSP formulation to avoid such routes.

10. (15 points) Repeat Part 8 but include the modifications to avoid routes that use edges vertically along column $x = 104$. What do you notice in runtime performance compared to Part 8? Provide one possible reason for this phenomena.

    **Hint**: Your reasoning does not need to be too complicated nor require deep knowledge of combinatorial/discrete optimization.