Guide : Project of Integer Linear Programming

Arthur LEONARD

2025-2026

1 Introduction

This document serves as a task statement and guide for your project. During this project, you will implement a solver for the p-median problem using ILP techniques. You will be evaluated based on three elements :

- Your written report, which should detail your approach, the challenges you faced, and how you overcame them.
- Your implementation of the solver, which should be efficient and well-documented.
- An oral presentation of your work, where you will explain your methodology and results, and answer questions from the evaluators.

You must send a .zip or a github / gitlab link containing your report (in PDF format) and your code before december 17th 2025 (anywhere on earth), to the email addresses :

- arthur.leonard@math.u-bordeaux.fr and
- francois.clautiaux@math.u-bordeaux.fr.

You can work in teams of two students if you want.

2 Problem Definition

In the p-median problem, you are given a set C of customers, and a set F of potential facility locations. The goal is to choose p facility locations in F to minimize the total distances between customers and their assigned facilities. Each customer must be assigned to exactly one facility. Each facility can serve multiple customers, but each customer $c \in C$ has a demand d_c, and each facility $f \in F$ has a capacity u_f. The total demand assigned to a facility cannot exceed its capacity.

3 Softwares Setup

You will need the following softwares [1]:

- An IDE : we suggest VSCode with the C/C++ extension. On Ubuntu, you can install it with :

  ```
  sudo apt-get install code
  ```

- A C/C++ compiler : we suggest G++. On Ubuntu, you can install it with :

  ```
  sudo apt-get install g++
  ```

[1] If you are on windows, we strongly suggest using WSL (Windows Subsystem for Linux), and installing all the softwares with WSL.

- A Build system : we suggest Cmake. On Ubuntu, you can install it with :

  ```
  sudo apt-get install cmake
  ```

- A MIP solver : we suggest Gurobi. You can follow the basic steps explained here: Gurobi Installation Guide.

To help you starting your project, we provide a simple code template you can use to build your project with Gurobi.

Question 1. *Make sure you are able to compile and run the code template.*

4 Input Format

The instances are provided in the following format:

- The first line contains 4 space-separated integers: $|C|$, $|F|$, p, and U: the number of customers, the number of facility locations, the number of facilities to open, and the maximum capacity of new depots (it will be useful later).
- The c-th of the next $|C|$ lines contains 3 values: reals x_c, y_c and the integer d_c: the position and the demand of customer c.
- The f-th of the next $|F|$ lines contains 3 values: reals X_f, Y_f and the integer u_f: the position and the capacity of facility location f.

Question 2. *Using the provided structure* ***Point2D****, create a structure* ***Instance*** *that stores all the relevant information, and implement:*

- *A function that given an instance* ***inst****, writes it on the output stream* ***out****.*
- *A function that initialises the instance* ***inst*** *from the input stream* ***in****.*

```
ostream &operator<<(ostream &out, const Instance &inst);
istream &operator>>(istream &in, Instance &inst);
```

5 Output Format

The output must contain $|C|$ lines, the c-th line must contain 2 reals : the coordinates of the opened facility associated with the customer c.

Question 3. *Implement:*

- *A function that given a solution* ***sol****, writes it on the output stream* ***out****.*
- *A function that initialises the solution* ***sol*** *from the input stream* ***in****.*

```
using Solution = vector<Point2D>;
ostream &operator<<(ostream &out, const Solution &sol);
istream &operator>>(istream &in, Solution& sol);
```

6 Objective

Question 4. *Implement a method of* ***Instance*** *that given a solution* ***sol****, returns the value of the solution* ***sol*** *if it is valid, or $+\infty$ otherwise. You can check your program on the instance* ***sample.inst*** *and solution* ***sample.sol****.*

7 Visualisation (Optional but recommended)

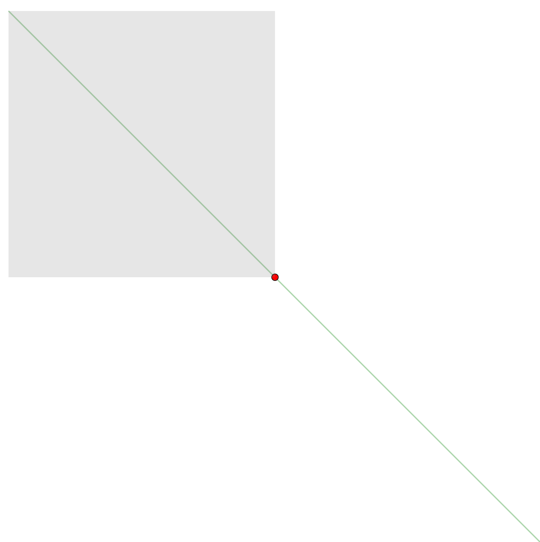
The SVG (Scalable Vector Graphics) is a simple image file format. It uses the following syntax:

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1" width="800" height="800">
<line x1="0.0" y1="0.0" x2="800.0" y2="800.0"
     opacity="0.3" stroke="green" stroke-width="2" />
<circle cx="400.0" cy="400.0" r="5.0"
     fill="red" stroke="black" stroke-width="1" />
<rect x="0.0" y="0.0" width="400.0" height="400.0"
     fill="gray" opacity="0.2" />
</svg>
```

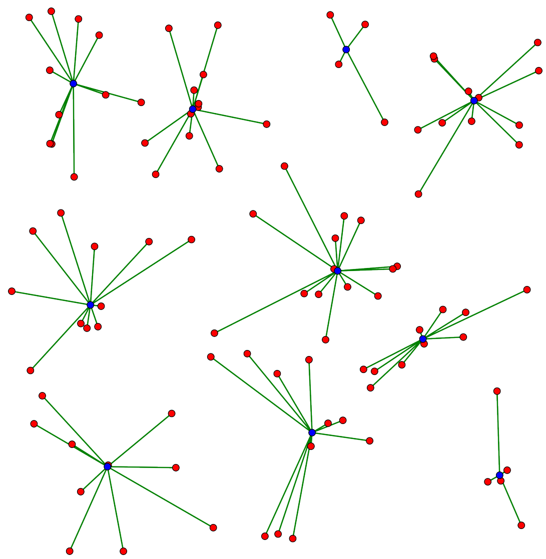
The SVG code above creates a 800 × 800 image, with a semi-transparent green diagonal of width 2, with a red disk drawn above the center of the diagonal. The radius of the disk is 5, and it has a black outline of with 1. The top-left corner is filled with a semi-transparent gray rectangle. You can open the sample provided SVG file with firefox:

```
firefox sample_svg.svg
```

You should see something like this:



Question 5. *Write a simple instance/solution visualizer, which produces an SVG output similar to the one below.*



8 Compact IP Formulation

Question 6. *Write a compact integer programming formulation for the p-median problem, as formulated above, with $|C| \times |F|$ binary variables and $|C| + |F| + 1$ constraints.*

Gurobi is a powerful IP solver that you can use to solve integer programming formulations. To create a model in Gurobi, you can use the following code snippet:

```
#include "gurobi_c++.h"

GRBEnv env = GRBEnv(true);
env.set(GRB_IntParam_LogToConsole, 0); // Optional : disable Gurobi output
GRBModel model = GRBModel(env);
```

You can create variables using:

```
GRBVar x = model.addVar(lb, ub, obj, vtype);
/* vtype is GRB_BINARY, GRB_INTEGER or GRB_CONTINUOUS */;
```

You can create constraints using:

```
GRBLinExpr expr = x + y + z;
GRBConstr constr = model.addConstr(expr <= 10);
```

You can then solve the model, retrieve the objective value and solution using:

```
model.optimize();
double objVal = model.get(GRB_DoubleAttr_ObjVal);
double optX = x.get(GRB_DoubleAttr_X);
```

You can get the dual of a constraint, or the dual of the variable's bound constraints dual value using:

```
double dualVal = constr.get(GRB_DoubleAttr_Pi);
double varDualVal = x.get(GRB_DoubleAttr_RC);
```

Question 7. *Implement this compact formulation with a generic IP solver.*

9 Column generation

Question 8. *Implement a column generation approach to solve the LP relaxation of the problem. A column will represent a facility location $f \in F$, and a set of customers $S \subseteq C$ assigned to the facility location f. You will need to implement:*

- *A master problem that selects at most p columns to cover all customers,*
- *A pricing problem that generates new columns with negative reduced cost.*

For this question, you can use a MIP solver to solve the pricing problem optimally.

Question 9. *Compare the quality of the LP bound obtained with the compact LP formulation and with the column generation approach, in theory and in practice.*

Question 10. *Modify your pricing problem to be solved with a dynamic programming approach.*

Question 11. *Implement a stabilization technique to speed up the convergence of the column generation. Compare the convergence speed with and without stabilization.*

Question 12. *Implement a procedure to obtain integer solutions from the column generation approach. You can use one of the following methods:*

- *Solve the master problem as an integer program after the column generation converges.*
- *Implement a diving heuristic : at each iteration, fix some variables to 1 or 0 based on the LP solution, and re-solve the master problem with column generation.*
- *Implement a branch-and-price algorithm.*