IEOR 169: Problem Set #3

Spring 2022

Due Mar 10 2022 at 2 PM PST

You are encouraged to collaborate with fellow students as you work through the problem set. However, your final submission must be your own work.

If you used any technology to solve a problem (Pyomo, Gurobipy, Excel, AMPL, etc..) make sure to include the relevant details (ipython notebook, pdf of excel set up and solution, pdf of AMPL code and output, etc..). I may request a copy of the original files that you used. A link to a cloud-based file would not be accepted.

Optional problems will not push your grade beyond 100% but it may compensate for your mistakes in the main problem set.

Your solutions must be uploaded on bCourses by Mar 10 2022 at 2 PM PST. Late submissions without prior approval will not be accepted.

Problem 1 tennis problem

A friend (Berkeley EECS alumni) has reached out asking if we could help him to help his father who is running a children tennis team. For practice, they would like to pair together players for multiple rounds of a round-robin tennis doubles tournament. There might be more players than available space to play, so each round might have kids with byes to the next round. They would need a software tool that outputs an optimal pairing schedule when supplied with an input of the number of players available `n_players`, the number of courts `n_courts` (where courts hold 2 teams of 2 players each) and the number of rounds `n_rounds`. To make the training experience diverse and fair, they would like to make sure that the teams are never repeated within a tournament (any player is never paired up with the same player twice, but they can play against each other many times) and that the maximal number of byes by any player is as small as possible. It is assumed that the number of rounds is small enough so that it is possible to never repeat the teams.

For example, if you have 1 court, 6 players, and 3 rounds: 4 people can play at a time, so each round will have 2 people with byes. Round 1 we might pair $[1, 2]$ and $[3, 4]$, giving players 5 and 6 a bye. For the next round, we want to make sure 5 and 6 are playing and don't have byes again, but also that players 1 and 2 don't play

together, and players 5 and 6 don't play together. So our output might be something like $[1, 5]$ playing against $[3, 6]$ with 2 and 4 getting a bye. For the last round, 1 has played with 2 and 5 so we wouldn't want to pair them and 3 has played with 4 and 6 so we want to avoid that pair as well. 1 and 3 havn't gotten a bye yet so they should get that and we end up with something like $[2, 4]$ playing $[5, 6]$ with 1 and 3 getting the bye.

Write and submit the necessary explanations and the code of a software tool that resolves the problem of for the team. If you are using Colab, please download the notebook and pack it into the submitted zip-file. (There is no strict requirements for the methods of input and output)

Example of an input (`n_players`, `n_courts`, `n_rounds`):
10 2 3
Example of an output:
round 0:
court 0 : $(1, 4)$ vs $(2, 8)$
court 1 : $(3, 5)$ vs $(7, 9)$
round 1:
court 0 : $(0, 3)$ vs $(2, 7)$
court 1 : $(4, 8)$ vs $(5, 6)$
round 2:
court 0 : $(0, 9)$ vs $(1, 6)$
court 1 : $(2, 4)$ vs $(5, 7)$

. . . . . .

Problem 2 design a heuristic

Exercise 13.7.2 (i) from the textbook

. . . . . .

Problem 3 apply local search and greedy algorithm

Exercise 13.7.4 from the textbook (apply at most two steps of each heuristic. the exact design is up to you)

. . . . . .

Problem 4 definition of total unimodularity

Exercise 3.9.1 from the textbook

. . . . . .

Problem 5 solve a minimum cost network flow instance

Exercise 3.9.10 from the textbook

. . . . . .

Problem 6 guarantees on greed: knapsack (optional)

Exercise 6.9.11 from the textbook

. . . . . .

Problem 7 guarantees on greed: matching (optional)

Exercise 6.9.12 from the textbook

. . . . . .