CS 170 Algorithms
Fall 2015 Prasad Raghavendra and Sanjam Garg

Project

Programming in a Theory Course?

Over the past few weeks, you have seen the basics of the theory of **NP**-completeness. In light of what we saw in this course—and much more mathematical evidence beyond the scope of CS 170—virtually all researchers believe that **NP**-complete problems are intractable.

But for many, there is no substitute for seeing the humbling phenomenon of intractability firsthand. That is why in this project, you will develop, then attempt to solve, instances of an **NP**-hard optimization problem. You will compete against other teams to develop the best algorithms, and also the hardest problem instances that will stump their algorithms.

Problem Description

You are running a tournament with up to 100 players. Through some process, a subset of all possible 2-player matches are played (each possibly more than once), so that for any two players i and j, you may know that i has defeated j in a match, or j has defeated i, or both, or neither.

A natural goal is then to find a ranking of the players, from 1 (weakest player) to n (strongest player), that is most consistent with the match data. This leads us to the following problem:

> MAXIMUM ACYCLIC SUBGRAPH (MAS): Given directed graph $G = (V = \{1,\ldots,n\}, E)$, find an ordering of the nodes $r_1,\ldots,r_n$ (we call r_i the *rank* of player i) that maximizes the number of forward edges:
>
> $$\left(\begin{array}{c}\text{objective value achieved by}\\ \text{ranking } \{r_1,\ldots,r_n\}\end{array}\right) = \sum_{(i,j)\in E} \mathbb{1}\{r_i < r_j\}$$

Note:

- The graphs will have no more than $n \leq 100$ nodes, and never have self-edges. You will find that even at this minuscule size, optimal solutions may still be unattainable.
- There always exists a solution, since any ordering of the nodes gives an objective value. The hard part is finding an *optimal* ordering.

Getting Started

1. Assemble a team of up to 4 students total (working alone is permitted).
2. Get a class account form (see relevant Piazza note). Each team needs only **one**.
3. Read all instructions before starting. There are two phases to this project, with separate deadlines. Plan well ahead.

Phase 1 (Instances + Algorithm Development): due 12/02 at 5pm

In this phase, you have two tasks:

1. Generate **three** instances of MAS that you think will be hard for other teams to solve.
2. Develop and code algorithms that you think will get good solutions to other teams' instances.

Formatting Instances

Generate three files—one per instance. Format each file as follows:

- Line 1: one integer $n \leq 100$, the number of nodes.
- Lines $2, \ldots, n+1$ form the adjacency matrix. Line $i+1$ contains n space-separated 0's and 1's. The jth number is 1 if there is a directed edge (i, j), and 0 otherwise.

Example instance file:

```
4
0 0 1 0
0 0 0 1
0 1 0 1
1 0 0 0
```

How to Submit

Choose a team name consisting of only English letters (capital and lowercase) and numbers.

Using `glookup`, submit the following using the command "`submit phase1`":

1. Your three instances, in files named `TEAMNAME1.in`, `TEAMNAME2.in`, and `TEAMNAME3.in` (with "`TEAMNAME`" replaced by a unique team name of your choosing).
2. A file named `README` (not "`readme`", "`README.txt`", etc.). In it, the first line is your team name, and the following (up to) 4 lines are your team members' student ID numbers.
3. Preliminary code for your solver algorithms (see Phase 2).
   This will not be graded, but is a check to make sure you are on track.

Failure to follow any formatting instructions will result in zero credit without exception. To help you avoid this, we have provided code to validate your inputs (see section on "Starter Code").

Warning: The instances are an important part of the project, but they are worth very little compared to Phase 2 (see section on "Grading"). Make sure you code your algorithms in Phase 1 so that you have enough time to run them during Phase 2.

Phase 2 (Solutions to Instances + Source Code): released 12/03, due 12/06 at 5pm

The day after phase 1 ends, we will pool together all teams' valid instances, add some of our own, and release them as a ZIP file on Piazza.

You then have the next 3 days to run your algorithms on them, to find the best solutions you can.

Input Format

The instances will be given as files named `1.in` through `N.in`. Expect N (number of instances) to be 400-600. Each one will be formatted as per the Phase 1 specification.

Formatting Solutions

Create **one** output file with N lines. Line i is your solution to the ith input file: a space-separated list of vertex numbers by increasing rank.

Example: Suppose the only instances are `1.in` with 4 nodes, and `2.in` with 6 nodes. Then a possible output file is:

```
1 3 2 4
5 6 4 3 1 2
```

What is the objective achieved by the output in the first row? If `1.in` is the Phase 1 example graph, then the number of forward edges—and thus the objective achieved—is 4 (check this!).

How to Submit

Using `glookup`, submit the following using the command "`submit phase2`":

1. Your output file, named `TEAMNAME.out` (with the same team name as in Phase 1).
2. The same `README` file as in Phase 1.
3. All code written by your team for both phases.
4. (If your team used papers or other sources) A file containing citations.

Rules

- You may work in teams of at most 4.
- Failure to follow any submission instructions results in zero credit.
- You may use any programming languages and packages (e.g. optimization software), *as long as such technology is publicly accessible to any student*.
- You are **not permitted to spend money or other currency** on computing resources (e.g. cloud clusters), or to hoard EECS machines in a way that disrupts other students or courses.
  Last semester, all top teams succeeded solely based on algorithmic ingenuity, using only personal computers.
- You may use any student-accessible libraries and research papers, *as long as you cite them in a file and turn this in with your code*.
- Open discussion of strategies is encouraged. But needless to say, copying code from another team is strictly prohibited and will be prosecuted to the fullest extent possible.

Tips and Reminders

- Code for validating submissions is provided on Piazza (for your convenience, but feel free to ignore).
- You are allowed, and encouraged, to "plant" good solutions in your generated instances (so that you know about them) but obfuscate them so that other teams may not find them.
- The staff is unable to give very involved implementation help, as your development tools will vary, and theorists are typically not the best programmers. Choose tools that you are comfortable with.
- We are happy to help, however, with general approaches and strategies, especially in office hours.
- Use the ideas in chapter 9. Look online, talk to peers, and talk to us about other strategies. But be selective, as there is not enough time to implement more than a few.
- Make use of the research literature (there is an interesting paper on MAS co-authored by Professor Raghavendra!).

Starter Code

We have included the following code, which you are free to ignore:

- `instance_validator.py` checks that your instance files are formatted correctly.
- `solutions_validator.py` checks that your solutions file is formatted correctly.
- `scorer_single.py` computes the objective value achieved by an individual solution.

Grading

We will score based only on your instance and solution files. We will **not** automatically run your source code—do not ask us to. At a high level, we will score as follows:

- (10%) How poorly other teams do on your instances. This will be based on the *best* of your three instances, so it is in your interest to submit a diverse set.
- (90%) How well you solve other teams' instances.

These scores are **relative to other teams** (exact mechanism to be determined).

In the end, we will curve the weighted scores to a generous distribution, similar to that of a homework. Teams who put in a reasonable effort will get a good grade. This project is worth the same as **two written homeworks**.

First Prize

The team that scores the highest overall will be treated to dinner with the professors and GSIs.