COS 445 - Strategy Design 1

Due online Monday, February 20th at 11:59 pm

Instructions:

- **You may not take late days on the Strategy Designs**. If it helps, think of the Strategy Designs as being due on Friday, except we have given everyone three free late days.
- You should aim to work in a team of two, but you are allowed to work alone or in a team of three. Your team should submit a single writeup, using the team feature on codePost. You should also submit a single code solution, using the team feature on TigerFile.
- You are allowed to engage with other teams over Ed or in person (but this is neither encouraged nor discouraged). If this is part of your strategy, you should discuss what you did and why you did it in your writeup. You are allowed to coordinate with other teams, or trick other teams. You are not allowed to promise other teams favors (e.g. monetary rewards) or threaten punishment outside the scope of this assignment. For example, you are allowed to promise "if your code does X, our code will do Y." You are not allowed to promise "if your code does X, I will buy you a cookie." If this is part of your strategy, your justification should explain why it will help you on this assignment.
- Please reference the course collaboration policy here: infosheet445sp23.pdf.
- Please reference the following document for further detail on how these assignments are evaluated: GradesForStrategy.pdf.
- This assignment is open-ended, **please ask questions on Ed to clarify expectations as needed**.

Reminder!

Please read the instructions at GradesForStrategy.pdf to better understand how the strategy design assignments are graded (which in turn should clarify how to answer the prompts).

Alice and Bob go to College (35 points)

Your high school guidance counselor heard you were taking COS 445 and asked you to advise the current seniors on how to decide where to apply for undergrad. You quickly realize that college admissions are a lot like university-proposing deferred acceptance (the universities "propose" to their early admits, and waitlist the rest, only proposing if their initial proposals are rejected), with one important catch: a university cannot propose to a student that didn't apply, and students don't apply everywhere. Fortunately, your guidance counselor is a data whiz and is able to give you the following model. Your team will be responsible for playing the role of one student deciding where to apply to college.

Setup:

- There will be one student and one university (admitting one student) per submitted bot. The number of submissions will henceforth be known as N.
- Every student s has an aptitude A_s drawn independently and uniformly from $[0, S]$. If you are student s, you know S and A_s, but not A_t for any other t.
- Every university u has a quality Q_u drawn independently and uniformly from $[0, T]$. Every student knows T, and Q_u for all universities u.
- Every (student, university) pair has synergy $S_{s,u}$ drawn independently and uniformly from $[0, W]$. If you are student s, you know W, $S_{s,u}$ for all universities u, but not $S_{t,u}$ for any other student t.
- S, T, and W are real numbers and are constant (the same) across students and universities

Admissions:

- Student s forms preferences over universities in decreasing order of $Q_u + S_{s,u}$.
- Every student simultaneously selects 10 universities to apply to.
- University u forms preferences over students who applied in decreasing order of $A_s + S_{s,u}$.
- College-proposing deferred acceptance is performed, where universities only propose to students who applied. That is, when a university is selected to propose, they propose to their favorite student who applied and hasn't yet rejected them. If they have already proposed to all students who applied, they are permanently unmatched.

Payoffs:

- If you are unmatched, you get payoff 0. Otherwise, if there are a total of N universities, and you are matched to your $(k+1)^{th}$ choice (that is, there exist k universities in the entire pool that you prefer to your match), then your payoff is $N - k$.

  To be extra clear, if your true preferences are $\succ_s$ (that is, $u \succ_s u'$ because $Q_u + S_{s,u} > Q_{u'} + S_{s,u'}$), but you submit 10 universities ordered by $\succ'$, your payoff is determined by your true preferences $\succ_s$.

Design a strategy that takes as input $N, S, T, W, A_s, \langle Q_u \rangle_{u \in U}, \langle S_{s,u} \rangle_{u \in U}$, and outputs a list of ten universities to apply to. Code it up according to the specifications below, and and answer the subsequent questions.

Specifications:

You will implement the Student interface provided in `Student.java`, which requires the following method:

- `public int[] getApplications(int N, double S, double T, double W, double aptitude, double[] schools, double[] synergies)`: called with a profile of a student and the potential universities, and with parameters of the distributions from which the profile was created. Note `schools.length == synergies.length`

and there are as many students as schools; and `schools` is sorted in descending order. Implement your strategy for deciding to which schools you shall apply. Return an `int[10]` containing only unique integers which are valid indices into `schools`, which indicate the ten schools you'll apply to (and the preferences over those schools you would like UPDA to use when it executes).

We provide the following sample strategies:

- `Student_usnews`: Applies to the schools with the best overall ranking.
- `Student_synergist`: Applies to the schools with which they have the highest synergy.
- `Student_holist`: Applies to the schools which they like the most.
- `Student_random`: Applies to a uniformly random set of schools.

Your file must follow the naming convention `Student_netid.java`, where `netid` is the NetID of the submitter.

Penalties may be issued if your submission does not precisely follow the API specifications. Examples of violations include: does not compile, or throws exceptions, or violates invariants documented in `Student.java`.

The provided Makefile allow you to test your strategy against the provided strategies and any other strategies you consider. Edit `students.txt` with a list of all the strategies to run, then use `make` to rebuild the testing code with those strategies.

Extra credit may be awarded for reporting substantive bugs in our testing code.

Also submit a single PDF file, containing answers to the following three prompts. Recall that your grade for part c is the maximum of your grade on the writeup and your grade for your strategy's performance.

Part a (10 points)

What should a good strategy do when $T = 0$? Why?

Part b (10 points)

What should a good strategy do when $W = 0$? Why? If you like, you may assume for this part that all students (including you) know A_t for all students t (rather than just knowing S).

Part c (15 points)

Provide a brief justification for your strategy. Focus on convincing the grader that it is a good strategy, by explaining the main ideas and why you chose this strategy. You should aim to keep this under one page. This will not be strictly enforced, but the grader may choose not to read beyond one page. You should not think of this merely as a documentation explaining only what your code does. Instead, try to imagine that it's purpose is to convince your guidance counselor why they should adopt your strategy.