COS 445 - Strategy Design 2

Due online Friday February 25th 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: infosheet445sp22.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).

Strategic Gerrymandering (35 points)

In Candyland, all resources are allocated fairly, and all congressional districts are drawn via careful protocols. Of course, Candyland still has a strong two-party system, and every ten years the two parties participate in the I-cut-you-freeze protocol developed here (`https://arxiv.org/pdf/1710.08781.pdf`, by Pegden, Procaccia, and Yu) to redistrict.[1] Your political party of choice gerrymandered poorly last cycle, and is looking to up their game. They heard you were

[1] **Note:** You are not expected to read this paper — it's just included as a reference. This assignment is self-contained, and you only need to understand what's written in the assignment.

taking COS 445 and offered you a consulting gig to maximize the number of districts they win in the upcoming election.

Your team will be playing the role of one political party, against one other team (at a time, you will play all teams) in the following protocol.

Setup:

- There are two teams, Alpha and Beta.
- There are N blocks of voters. A block cannot be further subdivided (think of this like a neighborhood).
- Each block i has α_i constituents who will vote for Alpha, and β_i constituents who will vote for Beta.
- α^* and β^* are drawn independently and uniformly at random from $[T, 2T]$.
- Each α_i is drawn independently and uniformly at random from $[0, \alpha^*]$, and each β_i is drawn independently and uniformly at random from $[0, \beta^*]$.
- Both Alpha and Beta know N, $\vec{\alpha}, \vec{\beta}$.

Districting:

- A districting of the voters is a partition into d disjoint sets, each containing exactly N/d blocks (N will always be an integer multiple of d).
- Alpha wins a district D if the number of voters in all blocks in D who prefer Alpha exceed the number of voters in all blocks in D who prefer Beta. That is, Alpha wins iff $\sum_{i \in D} \alpha_i > \sum_{i \in D} \beta_i$.[2] Beta wins if Alpha does not.

I-Cut-You-Freeze:

Note: You are certainly welcome to visit the linked paper for any insight,[3] but our model is slightly simpler than in the paper, so the formal algorithmic descriptions may not line up. The one in this handout is what will be used.

1. Initialize $r = d$. Initialize $R = \{1, \dots, N\}$. Initialize Districts = an empty list. Initialize activePlayer = Alpha, otherPlayer = Beta.
2. While $r > 1$ (while there are still districts left to make):
3. activePlayer proposes a partition of R into r disjoint districts $X_1, \dots, X_r$, each of size $|R|/r$ (activePlayer proposes a full districting of all remaining blocks into r districts of exactly $|R|/r$ blocks).

[2] We will set T large enough so that a tie is extremely unlikely in any possible district.

[3] This is another reminder that you are not expected to visit the linked paper, and that the assignment is self-contained. If you choose to visit that paper and find any ideas helpful, you're certainly free to use them.

4. otherPlayer picks any X_i, and adds X_i to Districts (otherPlayer picks a district to finalize, the rest are reset).

5. Remove all blocks in X_i from R (R is the remaining blocks, and all blocks in X_i are now districted).

6. Swap activePlayer and otherPlayer. Decrease r by one.

7. Go back to step 2.

In other words, Alpha and Beta alternate between proposing a districting of the remaining blocks. Every time one of them proposes a districting, the other one picks one district to finalize. Then they swap roles and repeat.

Payoffs:

- For one game, Alpha's payoff is the number of districts they win. Beta's payoff is the number of districts they win.
- You will be matched against every other submission, and against each other submission you will play multiple rounds to remove noise due to randomness.[4]

Your job is to design a strategy that plays I-Cut-You-Freeze, and your goal is to maximize your payoff (number of districts won). Code it up according to the specifications below, and answer the subsequent questions.

Specifications:

We provide a Block class which methods `alpha()` and `beta()` to get you the number of votes for alpha and beta in that block.

You will implement the Party interface provided in `Party.java`, which requires the following methods:

- `public static Party New(bool isBeta, int numDistricts, List<Block> blocks)` must construct and return a `Party` based on the provided blocks. Do any initialization here.
- `public List<List<Block>> cut(int numDistrictsRemaining, List<Block> remaining)` must partition the remaining blocks into numDistricts, each with `remaining.size() / numDistrictsRemaining` elements. We will issue a penalty if your strategy outputs the wrong number of districts, outputs districts with different numbers of blocks, does not include all the remaining blocks, or includes any other blocks.
- `public List<Block> choose(List<List<Block>> districts)` must chose and return one of the provided districts. We will issue a penalty if your strategy does not return one of the provided lists.
- `public void accept(List<Block> chosen)` is used to inform the active party of the choice made by the nonactive party.

[4]For instance, note that if $\alpha^* \gg \beta^*$, Alpha should do much better than Beta. So we will play multiple rounds to level the field.

We guarantee that we will always call the methods in this order:

- `New` on the class of each of alpha and beta
- Repeating `numDistricts` times, with the active player initially alpha:
  - `cut` on the active party
  - `choose` on the nonactive party
  - `accept` on the active party
  - swap the active and nonactive parties

We provide the following sample strategies:

- `Party_pack_cut_pack_choose`: A strategy which makes the most uneven districts possible and always freezes the district with the most voters for the opponent.
- `Party_even_cut_pack_choose`: A strategy which uses the greedy algorithm to create fair-ish districts (not the fairest, but as good as possible with the greedy algorithm) and always freezes the district with the most voters for the opponent.
- `Party_even_cut_even_choose`: A strategy which uses the greedy algorithm to create fair-ish districts (not the fairest, but as good as possible with the greedy algorithm) and always freezes the district it wins by the smallest margin (if it wins no districts, freezing the district it loses by the largest margin).

Your file must follow the naming convention `Party_netid.java`, where `netid` is the NetID of the primary submitter. Your class must also be named `Party_netid`, or else it will not compile. **Please follow the naming convention correctly so that we do not need to modify your submission.** Because filenames differ, we have to use the "Additional Files" zone on TigerFile. However, only upload one file (your Party). If you want to include other classes, declare them as private inner classes within your Party.

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 above and in `Party.java`.

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

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 (for Alpha) do when $d = 2$ and all $\vec{\alpha}, \vec{\beta}$ satisfy: $\alpha_i + \beta_i = 1$, $\alpha_i, \beta_i \in \{0, 1\}$ for all i (that is, exactly one of α_i or β_i is one, and the other is zero)? Make sure to consider the case where Alpha has more total votes than Beta, less total votes than Beta, and less than a quarter of the total votes.

What makes this hard for general $\vec{\alpha}, \vec{\beta}$? (**Hint:** Google SUBSET-SUM or PARTITION).

Note: It is OK to be informal with calculations and to ignore off-by-one errors. It is also OK just to write a few sentences explaining what makes this hard in general. The staff solutions are (much) less than half a page.

Part b (10 points)

What should a good strategy do (for both Alpha and Beta) when $d = 3$ and all $\vec{\alpha}, \vec{\beta}$ satisfy: $\alpha_i + \beta_i = 1$, $\alpha_i, \beta_i \in \{0, 1\}$ for all i (that is, exactly one of α_i or β_i is one, and the other is zero)? You may want to first reason about what Beta should do, and then reason about what Alpha should do conditioned on this.

Note: It is OK to be informal with calculations and to ignore off-by-one errors. It is OK to explicitly consider casework for what you would do as Alpha. It is also OK to describe a clear and well-defined optimization problem that you would solve (without going through all the cases to solve it). The staff solutions are less than half a page.

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 its purpose is to convince your political party of choice why they should adopt your strategy.