COS 445 — PSet 2
Due online Monday, March 5th at 11:59 pm

Please reference the course infosheet http://www.cs.princeton.edu/~smattw/Teaching/infosheet445sp18.pdf for the complete homework/collaboration policy. Highlights below:

- You must write up your solutions by yourself, without any collaborators or external references.
- Unless otherwise stated, you may collaborate with other students and consult external references.
- You must list all collaborators and external references consulted and upload it as a separate file to Mechanical TA (do not include collaborators in your problem submission).
- Some problems will be marked as no collaboration problems. This is to make sure you have experience solving a problem start-to-finish by yourself in preparation for the midterms/final.
- Please upload your solutions for 1–3 as separate PDF files to Mechanical TA. The name of your file will be visible to graders, so if you would like to remain anonymous to graders, give your PDF a generic name. Please upload your code and writeup for problem 4 to CS Dropbox.

Problem 1: Consistency (10 points, no collaboration)
A voting rule \( F \) is said to be consistent if whenever it selects the same candidate on two sub-populations, it also outputs that candidate on their union. Formally, \( F \) is consistent if whenever \( F(\succ_1, \ldots, \succ_n) = x = F(\succ_{n+1}, \ldots, \succ_{n+k}) \), then \( F(\succ_1, \ldots, \succ_{n+k}) = x \) as well.

Prove that Instant-Runoff Voting is not consistent. Your counterexample must not make use of tie-breaking (i.e. every time the rule compares two numbers in your example, they must not be equal).

Problem 2: Range Voting (10 points)
Range voting works as follows: each voter \( i \) assigns a score \( v_i(c) \in (0, 1) \) to candidate \( c \) (for all candidates). Each candidate’s total score is then \( \sum_i v_i(c) \), and the candidate with the highest score is then selected. Voters are not allowed to give the same score to multiple candidates.

Prove that Range Voting is unanimous (if everyone scores candidate \( a \) the highest, then candidate \( a \) is selected), and is not a dictatorship (there does not exist a voter \( i \) such that the output always selects voter \( i \)’s favorite candidate).

Finally, prove that Range Voting is “very weakly strategyproof” in the following sense: if you know exactly how all other voters are voting, then your optimal vote never needs to place a
candidate \( a \) above \( b \) if you prefer \( b \) to \( a \) (but note that this doesn’t tell you how to assign the actual scores).

**Problem 3: Implications of Strategyproofness (20 points)**

You may NOT use the Gibbard-Satterthwaite Theorem for this problem.

**Part a: Strong Monotonicity (10 points)**

Let \( F \) be a strategyproof voting rule. Show that \( F \) satisfies the following condition, termed *Strong Monotonicity*: if \( F(\succ_1, \ldots, \succ_n) = a \), and for all voters \( i \) and all candidates \( b, a \succ_i b \Rightarrow a \succ'_i b \) (that is for all voters \( i \), if they preferred \( a \) to \( b \) originally, they still prefer \( a \) to \( b \)), then \( F(\succ'_1, \ldots, \succ'_n) = a \) as well.

**Part b: No Losers Allowed (10 points)**

Let \( F \) be a strategyproof and unanimous voting rule. Show that \( F \) satisfies the following property: if for any two candidates \( a, b \), and for all \( i \), \( a \succ_i b \), then \( F(\succ_1, \ldots, \succ_n) \neq b \).

**Problem 4: Manipulating the Polls (15 points)**

You’re getting ready to vote in the upcoming election between three candidates, \( A \), \( B \), and \( C \). Before the election, there are several polls. Your goal in this challenge is to try and get candidates you like best elected, either by manipulating the polls or by manipulating the actual vote.

**Setup:**

- There are three candidates \( A, B, C \), and \( N \) voters (you will play the role of a single voter).
- **Everyone has one of the following preferences:** \( A \succ B \succ C, B \succ C \succ A, C \succ A \succ B \), chosen uniformly at random.
- There are \( P \) polls before the election. In each poll, you are asked to report your favorite candidate (note that this implicitly defines your full ranking).
- In the final vote, Plurality is used (so you need only report your favorite candidate). The most votes wins (ties broken randomly).

**Polling and Voting:**

- You know \( P \). You also know your own preferences. You **do not** know any other voters’ preferences (aside from the fact that they are uniformly random among the three possibilities).
- You will know the outcome of poll \( i \) before casting your vote in poll \( i + 1 \) (and the result of all polls before casting your final vote).

**Payoffs:**

- You will get payoff 2 if your favorite candidate is selected, and 1 if your second-favorite is selected (0 otherwise).
Your strategy should take as input a favorite candidate, $F$, a number of voters $N$, a total number of polls $P$, a voter ID $v$, and then over time report votes in polls and receive a history of past poll results (which will take the form of $N$ arrays, one at a time, of the $N$ reported preferences of each voter, where the $i^{th}$ element denotes the history of voter $i$ through all $N$ rounds, and where the $v^{th}$ elements correspond to your history), and then finally output a candidate to vote for in the real election.

Code it up according to the specifications below, and write a brief justification. **Please visit the course website for grading policies with respect to programming challenges.** Recall that all of your main justification should fit in one page. If you wish to include calculations or simulation results, you may do so in-line, but your writeup should contain at most one page of English justification. This will not be strictly enforced, but graders may not read justifications that go significantly beyond the one-page guideline in full.

Your writeup should provide an overview of the main ideas in your code (remember that we also have your code — so you don’t need to provide pseudocode or a step-by-step description of your algorithm), and justify why you think it will perform well. In addition, you should address the following concrete questions:

- What does your strategy do in the “real vote” if $P = 0$?
- What would be the optimal strategy, if you knew that all other voters told the truth in every poll (and $P > 0$)?

**Specifications:**
You will implement the Voter interface provided in Voter.java, which requires the following methods, as documented in Voter.java:

```java
public void setup(int N, int P, int v, Candidate F);
public Candidate getPoll();
public void addResults(List<Candidate> results);
public Candidate getVote();
```

Your file must follow the naming convention **Voter_netID1_netID2.java**, where **netID1** and **netID2** are the Net IDs of the submitters. (If you work alone, use only one name; if we approved you working in a triple, use all three.) Penalties of up to 9 points may be issued if your submission does not precisely follow the API specifications. Examples of violations include: does not compile, or throws exceptions. Here are some tips from the first homework:

- Note that you must name your Voter implementation with the same name as the Java file in which it resides — **Voter_netID1_netID2** for a pair — this is a requirement of all public classes in Java and will cause compilation to fail otherwise. Several students named their file differently from their class in Homework 1. We did not take points off on Homework 1 for this error, but may in the future.

- Some students had difficulty using the testing code. Here are the exact steps you should take to compile and run your code in a Unix-like environment (including Linux, Mac OS, or Bash on Ubuntu on Windows):
  - Write voter strategies in files named, e.g. **Voter_a.java** and **Voter_b.java**, etc...
– Edit voters.csv to contain netID as the first line, and the name of a strategy (e.g. a or b) on each subsequent line, not necessarily distinct. You may include as many rows as you want (whereas for Admissions you had to have at least 10 strategies listed, not necessarily distinct). The simulation will instantiate each strategy once for each time it appears (and if we use the word ‘bot’, we refer to one instance of a strategy).

– You can also edit ElectionsBase. java if you want to change the parameters of the simulation. We’ll definitely assess you on your performance with different parameters, so you should do this to avoid over-fitting.

– After you have updated any of these files, the command make will use voters.csv to write Elections. java from ElectionsBase. java, then use all the strategy files and Elections. java to build Elections. class.

– Then, the command make test will run the Elections simulation (with assertions enabled) and output the results. The result is your reward (0, 1, or 2) averaged across many simulations.

• Several students violated invariants documented in Student. java on the first assignment. The most common error was returning an illegal preference ordering due to repeating a school twice in your preference ordering. Since it was easy to test for these violations by running the Admissions simulation with assertions enabled (which is the default behavior of make test), and since this prevents us from using your strategy at all in our simulation, and since the Problem Set 1 handout mentioned that this would result in point deductions, we will take off points for this.

• Many students’ strategies for the first assignment were designed based on simulation results against the sample strategies. Note that the sample strategies are not very good, so your results against them will probably be highly non-predictive of your results overall! If you want to use your simulation results to inform your strategic decisions, you should write a bunch of strategies that you think will represent the strategies submitted by other students and compare against them. You should also take into account your strategy’s performance when every bot in the room is using your strategy.

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

Extra Credit: Why is nothing Strategyproof?

Recall that extra credit is not directly added to your PSet scores, but will contribute to your participation grade. Some extra credits are quite challenging and will contribute significantly.

For this problem, you may collaborate with any students. You may not consult course resources or external resources. In this problem we will guide you through the proof of a well-known result, so you should not copy the proof from one of the course texts (nor should you try to find a proof from external sources). You must follow the guide below (and not provide an alternative proof).

A Full-Ranking-Function (FRF) \( F \) is given a set of alternatives \( A \) and a profile of preferences over \( n \) voters, \( p \), (just as with voting rules) except that now it must output a full ranking over \( A \) instead of a single winner.

Here are some desirable properties of FRFs:

• Unanimous: a FRF \( F \) is unanimous if whenever \( a \succ_i b \) for all \( i \), \( \succ = F(\succ_1, \ldots, \succ_n) \) has \( a \succ b \) (whenever everyone likes \( a \) better than \( b \), the final ranking has \( a \) above \( b \)).
• Independence of Irrelevant Alternatives: consider two profiles \( p = (\succ_1, \succ_2, \ldots, \succ_n), \ p' = (\succ'_1, \succ'_2, \ldots, \succ'_n) \) and let \( \succ = F(p), \succ' = F(p') \).

A FRF \( F \) satisfies Independence of Irrelevant Alternatives if for any two alternatives \( a, b \in A \), if \( a \succ_i b \iff a \succ'_i b, \forall i \) (every voter has the same preference between \( a \) and \( b \)) then \( a \succ b \iff a \succ' b \) (the ordering output by \( F \) ranks \( a \) vs. \( b \) the same). Intuitively this property suggests that our preferences for \( c \) should not interfere with the ranking of \( a \) and \( b \), and is related to strategyproof-ness.

Here is an undesirable property of FRFs:

• Dictatorship: voter \( i \) is a dictator in \( F \) if for all \( p = (\succ_1, \succ_2, \ldots, \succ_n) \), \( \succ_i = F(p) \).

That is to say, no matter what everyone else submits, the FRF chooses the ordering of the dictator. \( F \) is a dictatorship if some voter is a dictator.

It would be nice to produce FRFs that are unanimous with Independence of Irrelevant Alternatives, and are not dictatorships. Unfortunately, the following theorem says that for \( |A| \geq 3 \), this is not possible:

**Theorem 1** Every unanimous FRF \( F \) satisfying Independence of Irrelevant Alternatives over a set of more than 2 alternatives is a dictatorship.

**Part a**

Prove the following lemma:

**Lemma 2** Let \( p = (\succ_1, \succ_2, \ldots, \succ_n), \ p' = (\succ'_1, \succ'_2, \ldots, \succ'_n) \) be two profiles such that for every player \( i \), \( a \succ_i b \iff c \succ'_i d \). Then if \( F \) is unanimous with Independence of Irrelevant Alternatives, \( a \succ b \iff c \succ' d \), where \( \succ = F(p), \succ' = F(p') \) (when there are \( > 2 \) alternatives).

Note that a complete proof needs to consider all of the following cases:

1. \( a = c, b = d \).
2. \( a = c, b \neq d \).
3. \( a \notin \{c, d\}, b = c \).
4. \( a \notin \{c, d\}, b = d \).
5. \( a \notin \{c, d\}, b \notin \{c, d\} \).
6. \( a = d, b = c \).
7. \( a = d, b \neq c \).

You do not need to provide a proof for all 7 cases, as many are similar. Provide a proof for cases One, Two, and Five.

**Part b**

Take any \( a \neq b \in A \), and for every \( 0 \leq i \leq n \) define a preference profile \( \pi^i \) in which exactly the first \( i \) voters rank \( a \) above \( b \), and the remaining voters rank \( b \) above \( a \).

Prove that, if \( F \) is unanimous with Independence of Irrelevant Alternatives, there must be some \( i^* \in [1, n] \) such that in \( F(\pi^{i^* - 1}), b \succ a \) but in \( F(\pi^{i^*}), a \succ b \) (at this point, possibly not unique).
Part c

Use the lemma from part a to show the following:

**Lemma 3** If $i^*$ is defined as in Part b, and $F$ is unanimous with Independence of Irrelevant Alternatives, then for any $c \neq d \in A$: if $c >_{i^*} d$ then $c \succ d$.

Conclude that $i^*$ is a dictator for $F$. 