15-251: Great Theoretical Ideas In Computer Science

Recitation 9 Solutions

Computational Social Choice

In this question we'll explore a way to combine voting rules, which we denote with $+$.

Given voting rules $X_1, ..., X_n$, define the voting rule $X_1 + ... + X_n$ as follows:
First, find the winner of each X_i. If they all agree, we select that winner.
If they don't agree, recursively find the winner from the set of winners of each X_i. If this does not decrease the size of the set of candidates, we call some consistent tie-breaking mechanism.

This operation has many interesting algebraic properties. For example, $X+X = X$ and $X+Y = Y+X$ (make sure you know why these are true!).

(a) Prove or disprove that $(X + Y) + Z = X + (Y + Z)$ for all voting rules X, Y, Z.

False-
Consider an election between three candidates a, b, and c and the plurality voting rules X, Y, and Z where in the event of a tie, X chooses a, Y chooses b, and Z chooses c. Support there are three voters, with preferences $a > b > c$, $b > c > a$, and $c > a > b$.

Then, the winner of $(X + Y)$ is a, so the winner of $(X + Y) + Z$ is c. Similarly, the winner of $Y + Z$ is b, so the winner of $X + (Y + Z)$ is a, giving us the desired counterexample.

(b) Prove or disprove that $(X + Y) + X = X + Y$ for all voting rules X and Y.

True-
Let X and Y be voting rules. $X + Y$ either chooses the winner of X or the winner of Y.

Case 1: $X + Y$ chooses the winner of X. Then, $(X + Y) + X$ also chooses the winner of X, so $(X + Y) + X = X + Y$.

Case 2: X and Y initially disagree, and $X + Y$ elects the winner of Y.
Then, $X + Y$ and X also disagree (they choose the same winners as Y and X, respectively), so we must again decide between these two candidates.
Exactly as before (when deciding the winner of $X+Y$), $(X+Y)+X$ will choose the winner of Y since $X + Y$ chooses the winner of Y.

Approximation Algorithms

(a) Given a graph on n vertices that we know to be 3-colorable (but not 2-colorable), we wish to give a coloring that uses as few colors as possible. Give a polynomial algorithm that is a $\sqrt{n}$-approximation for this minimization problem.
Hint: We can 2-color a 2-colorable graph in polynomial time using the greedy algorithm, and if the maximum degree in a graph is Δ, we can greedily $(\Delta + 1)$-color it in polynomial time as well.

Fun fact: Unless $P = NP$, not only is there no c-approximation for any constant c, but also there exists some $\epsilon > 0$ such that there is no $O(n^\epsilon)$-approximation.

Algorithm:
If the max degree of G is at least $\sqrt{n}$, pick a vertex v with degree $\geq \sqrt{n}$. 2-color the neighborhood of v (using two colors that have not been used before). Remove $N(v)$ and continue on the rest of the graph. Note that n is fixed as the number of vertices in the original graph, not the order of the subgraph we recursively call on.
If the max degree is less than $\sqrt{n}$, greedily color it with $\sqrt{n}$ new colors.

This gives a valid coloring because we use new colors at each iteration, and the colorings at each iteration are valid. In particular, we know we can 2-color the neighborhood of any vertex because the input graph is 3-colorable and that vertex must be colored differently than everything in its neighborhood.

Next, notice that we recursively 2-color at most $n/\sqrt{n} = \sqrt{n}$ times, as each time we remove at least $\sqrt{n}$ vertices. Finally, we color the rest of the graph with $\sqrt{n}$ colors, thus coloring the whole graph with at most $3\sqrt{n}$ colors. We know the optimal coloring uses exactly 3 colors, so we have a $\sqrt{n}$-approximation as desired.

Finally, the algorithm is polynomial because there are at most $\sqrt{n}$ iterations, each of which take polynomial time (using greedy algorithms).

Online Algorithms

(a) Given n candidates for a position, we would like to select the best one for the job. However, we have to give a decision to each candidate immediately after interviewing them. Rather than trying to find an approximate solution, we wish to try to find the optimal solution with high probability.

Our algorithm is as follows: first, we interview the first t people. Then, we continue interviewing until we find someone better than everyone we've seen so far and accept that person (if we reach the end, we pick no one).

For what value of t do we maximize the probability that we pick the best candidate (given that we pick some candidate)?

We pick the best applicant if and only if the first applicant with value greater than the maximum of the first t candidates is the best. Thus we can express the probability of selecting the best candidate as follows:

$\sum_{j=t+1}^{n} P(\text{we pick the best applicant} \mid \text{applicant } j \text{ is best}) P(\text{applicant } j \text{ is best})$.

Then, the probability that applicant j is the best is simply $\frac{1}{n}$, and the probability that we pick applicant j given that it is best is exactly the probability that among the first $j-1$ candidates, the maximum was in the first t candidates. The maximum is equally likely to be in any of the $j-1$ spots, so this has probability $\frac{t}{j-1}$. Thus the probability of picking the best person is

$\frac{t}{n}\sum_{j=t+1}^{n}\frac{1}{j-1} = \frac{t}{n}(\sum_{j=1}^{n-1}\frac{1}{j} - \sum_{j=1}^{t-1}\frac{1}{j}) \approx \frac{t}{n}(\text{ln}n - \text{ln}t) = \frac{t}{n}\text{ln}\frac{n}{t}$.

This is maximized at $t = \frac{n}{e}$.