Augur Whitepaper

Brennon Brimhall

August 1, 2014

1 Introduction

Rankings prediction is a problem that to my knowledge has not been solved. It's a tricky proposition, as it depends on the outcomes of matches, which is in itself not considered a solved problem.

But there are a variety of instances where a probabilistic summary of the top teams comes in handy. A rankings predictor helps teams to visualize what teams will be seeded in the top 8. This helps teams cater their strategies and decide what features to showcase the next day, increasing their chances of being picked. Alternatively, teams that are predicted to seed can use this to predict other teams who may seed higher, predict whether or not 'scorched earth' tactics could be a possibility for the next day, and more.

These situations, though, require a probabilistic predictions, not deterministic ones. It is nearly worthless to know only one possible outcome of the 3^n, even if it's the most likely possibility.

In this paper, I'm going to walk through an algorithm that outputs a probabilistic summary of rankings.

2 Theory

A given match has three outcomes: a win for red/loss for blue, a win for blue/loss for red, and a tie. This affects teams' Qualification Score in recent ranking systems (2012, 2013, 2014). This means that the entire tournament has 3^n outcomes, where n is the number of matches played.

We can immediately disregard one of those outcomes, though. A tie is incredibly uncommon. Games have recently sported high point totals, which has reduced the percentage of matches that end with ties. This results in a savings of $14,316,139$ outcomes with $n = 15$, and a savings of $2.05890058 \times 10^{14}$ outcomes at $n = 30$.

Another concept that helps to reduce the possibilities that need to be calculated is that these computations take place the night prior to the day of eliminations – Friday night for regionals, and typically Saturday for districts. This means that we can reduce n to the number of matches yet to be played – around 10 to 30, depending on the event.

Lastly, we can assume that tiebreakers stay the same, normalized to the number of matches that teams have played. This saves us the painstaking time to guess or predict the number of auto points or assist points that each match generates for that alliance.

3 Algorithm

At a high level, here's the algorithm:

1. Find the current QS and tiebreaker values.
2. Normalize the tiebreakers. (Divide by the number of matches played.)
3. Find the set of matches that will be played tomorrow.
4. Assign probabilities to them (either by machine or by human).
5. Generate all unique possibilities, or combinations of match outcomes.
   (a) Make a copy of the teams and associated QS and tiebreakers.
   (b) Assign the probability of this possibility to 1.
   (c) Iterate through the list of matches, adding 2 QS points to each robot that wins. Multiply the probability of this possibility by the probability of each match outcome.
   (d) Sort the teams to generate the rankings from that possibility.
   (e) Persist the rankings. (SQL servers are perfect.)
6. Generate the list of unique teams that seed 1.
7. Sum the probabilities of those possibilities that result in team A seeding 1.
8. Given that team A seeds 1, generate the list of unique teams that seed 2.
9. Sum the probabilities of those possibilities that result in team A seeding 1 and team B 2.
10. Given that team A seeds 1 and team B seeds 2, generate the list of unique teams that seed 3.
11. Sum the probabilities of those possibilities that result in team A seeding 1, team B 2, and team C 3.
12. Continue until desired depth. Remember that in-picking during alliance selection is common, so depths greater than 8 are preferred.
13. Print results to file.

4 Augur

Augur follows the algorithm presented, with minor deviations. It does not normalize tiebreakers, and persists results only to a tree data structure memory. Feel free to take a look at the source code available on GitHub.

4.1 Usage

Augur is a command line utility, meaning that you need to use a terminal emulator to use it. You'll need Python (version 2.7) installed to use it.

To customize the predictions, follow the format of the provided file, substituting in standings and matches to be played with your event.

To install and run the example, run the following commands:

```
cd ~
git clone https://github.com/brennonbrimhall/augur.git
cd augur
python main.py
```