Optimization of Team Assignment Based on a Kill-Death Ratio Match Making System as Applied to Online Battle Royale-Style Video Games Using SAS

Austin S Bohlin, Nilabh Chaturvedy

Introduction to the Problem

Our team was tasked with building a skill-based matchmaking system (SBMM) using kill-death ratio (K/D) as the sole match making rating (MMR) factor. We were provided a list of 300 players and their respective K/D's, 3 lobbies and their respective K/D targets, and a list of premade player teams. This system is built with the given assumption that each queue will be served as a batch of players, with no players joining the queue after the pairing process has started. This specific model was built and tested using the given batch player size of 300. It was given that each player has the option of preselecting their teammate and all other players will be assigned 1 teammate. We assume that there is no exception to the team size and this model will only work for teams of 2. Each team must be assigned to a lobby that best matches that team's combined K/D. Each lobby requires between 48 and 52 teams.

We made a few additional assumptions before creating this model. We assume that each player must have played the game before and has a K/D, as no base value is assigned for missing K/D values. This means that the model will not work if a player has not played the game and a default K/D value was not created to pass to the solver. Additionally, we assume that each list of players passed to the solver will be an *even* number between 2*48*(number of lobbies) and 2*52*(number of lobbies) as the lobbies strictly require between 48 and 52 teams, and a non-even number of players will break the *"each player needs a team"* requirement. There is no limit on the number of players that can be in a premade team, as long as those teams only have 2 players. With these assumptions in mind, we can now discuss our approach to solving this problem.

Methods

We began this problem by identifying how a to structure the problem so a MILP could solve it. We imported the data so *Players* would be a set of all 300 players and *Lobbies* would be a set of all 3 lobbies. Our first attempt at this was to make 2 binary matrixes, 1 to keep track of teams and another to keep track of lobbies:

$$Teams[i,j] = \begin{matrix} \\ \mathrm{P}_1 \\ \mathrm{P}_\mathrm{i} \\ \mathrm{P}_{300} \end{matrix} \begin{matrix} \begin{matrix} \mathrm{P}_1 & \mathrm{P}_\mathrm{j} & \mathrm{P}_{300} \end{matrix} \\ \begin{pmatrix} t_{1,1} & \cdots & t_{1,300} \\ \vdots & \ddots & \vdots \\ t_{300,1} & \cdots & t_{300,300} \end{pmatrix} \end{matrix} \mid i \in Players, j \in Players$$

$$LobbyAssignments[i,k] = \begin{matrix} \\ \mathrm{P}_1 \\ \mathrm{P}_\mathrm{i} \\ \mathrm{P}_{300} \end{matrix} \begin{matrix} \begin{matrix} \mathrm{L}_\mathrm{A} & \mathrm{L}_\mathrm{B} & \mathrm{L}_\mathrm{C} \end{matrix} \\ \begin{pmatrix} A_{1,A} & A_{1,B} & A_{1,C} \\ \vdots & \vdots & \vdots \\ A_{300,A} & A_{300,B} & A_{300,C} \end{pmatrix} \end{matrix} \mid i \in Players, k \in Lobbies$$

Using this approach, we realized it would make it impossible for the solver to work, as it would be duplicating data, and would have too many decision variables for a LP to work. We decided the best way to get around this is to combine the above matrixes into one of a higher complexity. We arrived at using a binary 3-Dimensional solution space that combined all the players on 2 axes and the lobbies on the 3rd:

$$Groups[i,j,k] = \begin{array}{c} \\ P_1 \\ P_i \\ P_{30} \end{array} \begin{array}{c} \begin{array}{ccc} & & L_C \\ & L_B & \\ L_A & & \end{array} \\ \begin{pmatrix} g_{1,1,A} & \cdots & g_{1,300,A} \\ \vdots & \ddots & \vdots \\ g_{300,1,A} & \cdots & g_{300,300,A} \end{pmatrix} \begin{matrix} \\ \\ \\ \end{matrix} \begin{matrix} \begin{pmatrix} g_{1,1,B} & \cdots & g_{1,300,B} \\ \vdots & \ddots & \vdots \\ & & g_{300,300,B} \end{pmatrix} \\ \\ \\ \end{matrix} \begin{matrix} \begin{pmatrix} g_{1,1,C} & \cdots & g_{1,300,C} \\ \vdots & \ddots & \vdots \\ & & g_{300,300,C} \end{pmatrix} \\ \\ \\ \end{matrix} \\ \begin{matrix} P_1 & P_j & P_{300} \end{matrix} \end{array} \quad | \; i \in Players, \; j \in Players, \; k \in Lobbies$$

Since this matrix would be set up as a binary, where Groups[i,j,k]=1 will show not only who is assigned on the same team, but also what lobby they are assigned to. While this matrix is more difficult to look at, it is one that allows the MILP to solve the system. Once we had a matrix that could be assigned solutions, we needed to work on the objective function. Before we discuss the objective function, we must first discuss something special about our matrix *Groups*.

If we examine what will happen at the end of the solver, we will see that $Groups = Groups^T$ which means that *Groups* is symmetrical. We can use this property to our advantage when it comes to our objective function. If we think about each value in *Groups* each team appears twice, once in *Groups[i,j,k]* and again in *Groups[j,i,k]* since each teammate in a team is a teammate with the other. This means that all the information we need is in the top right corner of the matrix, for example let's examine a matrix of just 4 players:

$$\begin{array}{c} \\ P_1 \\ P_2 \\ P_3 \\ P_4 \end{array} \begin{array}{c} \begin{matrix} P_1 & P_2 & P_3 & P_4 \end{matrix} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

We can see that player1 is paired with player2 and player3 with player4. But this information is in the table twice. We can figure out who is on whose team by looking only at the top right half of the matrix, or put another way, where $i < j$. This means we can narrow our criteria for almost all our constraints and objective function by more than half! If you can already get the information of who's on what team from half the matrix, you don't need to even consider the other half. Now with this insight revealed, we can move on to discussing the objective function.

We came up with 2 different interpretations for the objective function and will discuss the merits of both in the results section. The first solution is minimizing the absolute difference between *team K/D* (given as $Team\ KD = KD_{Player1} + KD_{Player2}$) and *lobby target K/D*:

$$minimize\ Z = \sum_{\substack{(i,j) \in Players^2 : i<j \\ k \in Lobbies}} Groups_{i,j,k} * |TargetKD_k - (PlayerKD_i + PlayerKD_j)|)$$

Please note the $(i,j) \in Players^2 : i < j$ binds i and j to only the square matrix space of the players list where $i < j$ as discussed in the above section. Our other idea for the function was using a minimization of the square of the difference between the *lobby target K/D* and the *Team K/D*:

$$minimize\ Z = \sum_{\substack{(i,j)\in Players^2 : i<j \\ k\in Lobbies}} Groups_{i,j,k} * (TargetKD_k - (PlayerKD_i + PlayerKD_j))^2)$$

Now that we have discussed how we formed the objective function, we need to discuss the creation of the constraints we can pull from the problem statement. We will start with the constraints which we could not figure out how to incorporate the symmetrical matrix insight into, each player only being on 1 team and 1 lobby. This means that each slice of the *Groups* matrix i-k plane and j-k plane must sum to 1 which means we cannot us the i<j method:

$$s.t. \sum_{\substack{i\in Players \\ k\in Lobbies}} Groups_{i,j,k} = 1\,, \qquad \forall j \in Players$$

$$s.t. \sum_{\substack{j\in Players \\ k\in Lobbies}} Groups_{i,j,k} = 1\,, \qquad \forall i \in Players$$

This constraint will hold true with the identity matrix, which would mean players in a team with themselves, so we must set the diagonal to 0 to fix this:

$$s.t.\ Groups_{i,i,k} = 0, \qquad \forall i \in Players, k \in Lobbies$$

Now we must fix the number of teams per lobby to between 48 and 52. For this one we can use the i<j method. We will take each i-j plane and the sum of the top right half of that plane:

$$s.t. \quad 48 \leq \sum_{(i,j)\in Players^2 : i<j} Groups_{ijk} \leq 52, \qquad \forall k \in Lobbies$$

Now we must make sure each team has both teammates in the same lobby:

$$s.t. \quad Groups_{i,j,k} = Groups_{j,i,k}, \qquad \forall (i,j) \in Players^2 : i < j, k \in Lobbies$$

The last thing we needed was to fix the teammates as partners, while still allowing the program to pick their lobbies. For this we had a data frame *Premades* that stored the *teamid*, *teammate1*, and *teammate2*. Using this we could establish our last constraint:

$$s.t. \sum_{k\in lobbies} Groups_{x,y,k} = 1\,, \qquad \forall (x,y) \in Premades$$

Please note, when passing team information, the player with the lower ID needs to be mentioned first, otherwise the solver might not pair them together. Once all these constraints are put into the SAS OptModel package, we can solve our system with MILP. We created a data frame from our solution *Groups* matrix to output the player pairings, lobby assignments, and team K/D using a short SAS code so we could export the solution to csv for analysis:

```
create data solution from [player1=i player2=j lobby=a]=
        {i in PLAYERS, j in Players, a in lobbies:
             ((groups[i,j,a].sol = 1) & i<j)}
        TeamKD=(player_kd[i]+player_kd[j]);
```

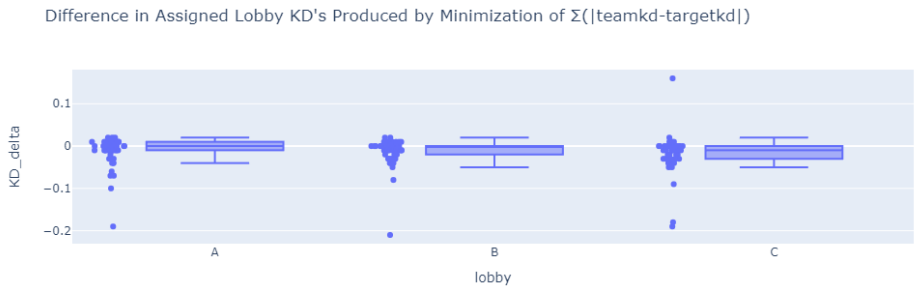
Results

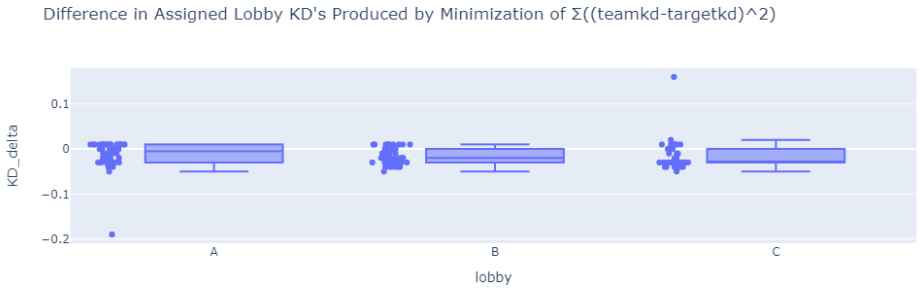
The only comparison between the 2 different objective functions in SAS is that the absolute function ran slightly slower (.2 second difference). Now we needed to compare the solutions outside of SAS.

A quick way to see how the optimization went is to view them on box plots. To make this comparison easier to look at we calculate a new value:

$$KDdelta_s = TargetKD_s - TeamKD_s, \qquad \forall s \in solution$$

This new value represents the difference in the team's K/D from the assigned lobby's target K/D. We calculated these knew variables in excel for each of the two objective functions, and then imported them to pandas for plotting purposes. Using this we plotted the 2 different objective functions:





We can see from these charts that Absolute solver keeps a tighter IQR but has a larger spread of outliers. In fact, the Squared solver only has 2 outliers in the whole chart, both of which are the premade teams. If I was a player in these lobbies, I would want to be assigned to the Squared lobbies as everyone would be closer in skill level as opposed to the majority that occurred with the Absolute solver. There is such little difference between the highest and lowest scores, not counting premade teams, that either operation would be sufficient. However, the Squared was the fastest and had the fewest outliers, so we selected this one as the final objective function.

Our speed performance increased dramatically once we applied the i<j rule to most of the constraints. Solve time went from around 12 seconds to the 6.58 seconds shown below:

```
NOTE: Optimal.
NOTE: Objective = 0.1437618717.
145
146   /* Creates the dataframe with the solution */
147   create data solution from [player1=i player2=j lobby=a]={i in PLAYERS
147 ! i<j)}
148   TeamKD=(player_kd[i]+player_kd[j]);
NOTE: The data set WORK.SOLUTION has 150 observations and 4 variables.
149
150   file log;
151   put "Team Members: Austin Bohlin, Nilabh Chaturvedy";
Team Members: Austin Bohlin, Nilabh Chaturvedy
152   put "Date and Time: %sysfunc(date(),date9.) %sysfunc(time(),timeampm8
Date and Time: 04SEP2022  1:19 AM
153   quit;
NOTE: The PROCEDURE OPTMODEL printed pages 1-2.
NOTE: PROCEDURE OPTMODEL used (Total process time):
      real time           6.58 seconds
```

Regarding this solver's accuracy, how do we know it found the optimal solution? The graph sure makes it seem like this is the best solution as all lobbies have a spread of less than 0.1 not including outliers. We can try to quantify an unrealistic optimal solution and then compare ours to that. First,

we will start by assuming we can move K/D points around to other players to get an optimal fit. We can take a sum of all the non-premade players K/D's and try to fit them into lobbies based on that. For this data set the average non-premade team KD is 4.96483 and we can calculate the average K/D of the lobbies by averaging out the sum of targetKD*(# of teams per lobby):

$$averageMoveableKD = \frac{51 * 4.25 + 50 * 5 + 47 * 5.75}{148} = 4.97973$$

If we compare this to the average team K/D, we see it is off by 0.015. We can then take this value and calculate the Z value of the solution:

$$Z = 148 * (.015)^2 + (5.9053 - 5.75)^2 + (4.05683 - 4.25)^2 = 0.09473$$

Compare this to our solutions Z value of 0.14376 and we can see what factor we are off by:

$$ImpossibleAccuracyFactor = \left(1 - \frac{0.09473}{0.14376}\right) = 0.341$$

So out solution is 34.1% away from being the optimal solution that breaks the rules of how KD's work. This number taken with how tight the lobby distributions are in the box plots, makes us conclude that this solver worked with a high accuracy.

Discussion

So, the big question is how useful will this model be in deployment? Unfortunately, after a few uses, this model will not be very useful at all. The problem comes with the underlying assumption that K/D, by itself, is useful as a **Match Making Rating (MMR)**, which we can run through a quick example to show that it is not. Say we take our lobby of stars with an average team K/D of 5.75, what happens at the end of that match? What are the new team K/D's? Well, if K/D was a good indicator of skill, then they should all be equally matched, and the average score should be 1 kill and 1 death. This makes a K/D of 1, which will now bring down the average K/D of each player in that lobby. But that makes no sense, did every player somehow lose skill by playing the game with those of comparable skill level? No, what's happening is you are asking a metric from each round to also hold the information about the skill of players across all rounds. A metric that, if lobbies are truly matched and equal, will always tend toward 1. To further illustrate the absurdity of using K/D as the only MMR variable, think for a minute how it came to be that the average K/D of the players in the queue got to be substantially higher than 1. Where are all the players that have the lower than 1 K/D's that would allow for a higher than 1 K/D to exist at?

So clearly K/D will not work. Instead, they should use some new MMR variable that stays with each player and only changes based on the outcomes of games given other players MMR. If a player gets a high K/D in a game with other players of similar MMR, the high K/D player should gain a lot of MMR and the other players should lose some. Similarly, if a player gets a high K/D in a game with other players of much higher MMR, they should be rewarded with much more MMR and the loses should lose a larger amount of MMR. This way game designers are using K/D and post-game statistics to change the match making variables, as opposed to using those post-game statistics as the match making variables. However, if an MMR system is fed into our algorithm, we will still be able to match make teams as there is no special requirement on the input player metric. So, if the lobby targets where say changed to MMR targets of 800, 1200, and 1600, and we were given player MMR's we could adequately pair players as we have demonstrated in this report. However, a new algorithm will need to be developed to adjust these MMRs after the games conclude using end game statistics.