ISYE 4133: Advanced Optimization

Swiss Chess Project

Name: Leonardo Calizaya

**Introduction:**

A Swiss-system chess tournament is a type of tournament different than traditional tournaments styles. It is characterized by a set of rules that will determine the pairings of each of the players, their game color, the results, etc.

Using data from previous rounds and the set of rules that define the Swiss-system chess tournaments, we can create an integer program formulation to obtain the pairings for the next round as well as the color each player is assigned. The objective of this system is to maximize the number of pairings, it may have other minor objectives as well. The set of rules can be transformed into constraints that the objective needs to meet.

For instance, some constraints are: Two players shall not play against each other more than once, players are paired with others with the same running score, no player shall receive the same color three times in a row, etc.

**Model**:

To model our integer program, we have to determine our variables that will help us model our constraints and objectives. In this case we are going to divide our variables in two: decision variables, and coefficient variables. Decision Variables are the ones that will be determined by optimizing our integer program. In our program, all of our decision variables are integer and most of them are binary. Coefficient variables are predetermined variables, in this case, data from previous swiss chess rounds. The combination of our two types of variables will allow us to optimize our integer program.

**Sets:**

I is the set of all swiss chess players since there are 94 participants

J is the set of all possible number of pairs that there can be with 94 participants, in this case, there could be a maximum of 47 pairs.

Q is a big set of numbers that we will use to create decision variables. The set is large because we will need large numbers for creatings constraints. For instance, when we create a constraint for each pair of players and for each pair.

slot available, we would need (4371)x(48) = 2090808. Some sets of constraints will use 500000 different types of variables or less, but no more.

“W” stands for white, “B” for Black and “N” for Not assigned color.

**Decision Variables:**

**Coefficient Variables**

1. C7, C8, C9, C10, C11, C12, C13, C14, C15, C16 and C17, are binary decision variables representing equations, counters, or simply variables that arise after linearizing logical expressions. Each of them has an index that can go up to 500000, depending on each case. Some of them are related with the objective function in a way that either we want to maximize or minimize the sum of any of these set of variables. The number associated corresponds to which constraint they are related.

**Constraints**:

1. Two players shall not play against each other more than once:

If P\_mn = 1, that means player m has played with player n previously, so they cannot play again. We will add this set of constraints for each P\_mn that is equal to 1.

For a specific j, if P\_mn = 1, then:

This means that the sum of X\_mj and X\_nj cannot be equal to 2, so they cannot be in the same pair. This constraint will be for all J for the pair (m, n).

1. For each pair of players there can only be 2 or 0 players:

We introduce the variable Y\_j so that the sum can only be 2 or 0.

If Y\_j = 1, then:

Knowing that each X\_ij can only have a value of 0 or 1, there should be only two players whose value is 1 for each pair.

If Y\_j = 0, then:

In this case, the pair is totally empty because the sum all X\_ij for a specific j will be 0. We introduce the above constraint to avoid having pairs with only 1 player, which is not possible. Using Y\_j also allows us to count how many completed pairs there are.

1. Each participant can be at most in one pair:

We do a similar thing now for each participant. We sum all the participation each participant has in all pairs. The sum can be 0 if the participant is not assigned to any pair, or it could be 1 if the participant is assigned to one pair, but no more.

1. A player can only be assigned “White”, “Black”, or “Not assigned color”.

For each player i, the sum of c\_iW + C\_iB + C\_iN should be 1. Knowing that C\_ik is a binary variable, there is only one color option for each player.

1. No player shall receive the same color three times in a row.

For each participant “i” and color “k”, if participant “i” has played with color ”k” two times in a row previously, on this round that player cannot play with the same color (“k”). So c\_ik = 0 for that participant.

1. The difference between the number of black and the number of white games shall not be greater than 2 or less than -2.
2. In general, players with the same running score are paired.

This is a soft constraint, meaning it is not mandatory but we want it as much as possible to be considered. Therefore, we will have to connect it to the objective function.

d = 0

The introduction of variable d will be better understood in the code. Basically, for each iteration we want a variable “C7” different than the previous one. To achieve that purpose, we add 1 to the variable d each time we iterate. This will be used in the following constraints too.

1. In general, player with a running score difference of +0.5 or -0.5 are paired.

This is a soft constraint, meaning it is not mandatory but we want it as much as possible to be considered. Therefore, we will have to connect it to the objective function. Moreover, this constraint has less importance than the previous constraint, so we will add it a lower weight in the objective function. We want first to maximize the pairs that have a 0 score difference, then the ones that have 0.5 or -0.5 score difference.

d = 1

1. For each pair of players that will play against each other, one plays with white and the other with black.

d = 1

1. In general, a player is given the color with which they played fewer games
2. If colors are already balanced, then, in general, the player is given the color that alternates from the last one with which they played.
3. Maximize the number of players who where unpaired in the previous round.
4. A strong colour preference for black occurs when a player's colour difference is +1.
5. A strong colour preference for white occurs when a player's colour difference is -1-
6. Minimize the number of players that have a color difference of 2.

d = 1

1. Minimize the number of players that have a color difference of -2.

d = 1

OBJECTIVE FUNCTION:

Where L\_i is the number of players that have a color difference of 2 and M\_i is the number of players that have a color difference of -2. We introduce a different weight to each of the sums to give more importance to some objectives than other. In the case of the substraction, we want to minimize the difference between the running scores of players assigned to each other.

Implementation:

**Solution and Analysis:**

As we can see in the pairing results in the appendix, most players are matched with someone with a similar score, one is assigned the black color and the other the white color. Moreover, the color assigned to each player is following the guidelines of the Swiss Chess system. However, we gave priority first to the objectives, and gave less weight to the “soft” constraints. Within the soft constraints we also gave more weight to some than other.

In general, coming up with an integer program formulation dealing with a big amount of decision variables and constraints, requires going back and forth on the different groups of decision variables that can be applied to solve the problem.

Decision Variables is a crucial step into developing the integer program because all the subsequent constraints will be modeled taking them into account. On first instance, my decision variables were different, but as I continued trying to transform the rules of the Swiss Chess game into the final constraints, i found that it was extremely hard with the decision variables that I had. I had to erase all my formulation. It was on my second try of decision variables that I could finally formulate all constraints for the rules of the game.

Once I had my decision variables decided, I had to formulate the constraints. There were 3 constraints that gave me a lot of trouble. They had double for loops, they had logical expressions that involved creating more variables, and took many updates. These were constraints 5 & 6, 7 and 8. I mention 5 & 6 together because on my first formulation, there was only one constraint involving both of them, but they became two as I later identified a problem in that constraint. Problems were identified while running the program on Gurobi and testing the if the results gave the desired output. These previous formulations are also available on the first report.

Update 1:

Previous formulation of 5 & 6: No player shall receive the same color three times in a row.

When creating this constraint, I was assumming the following that if R\_ik was 0 for some color k, that color would automatically be impossible. Lets say, participant 45 has played color white two times in a row previously, then:

In this case, I was assuming that C\_45W had to be zero. However, when solving the program, I ended up with players having two colors. I realized that there was no restriction on the overall sum of C\_ik to be 1. This resulted in the creation of two constraints. One had to be general to set the sum of C\_ik per each player to be 1.

The other constraint could be the same one we used previously, however, there was a more straightforward way to achieve the same result. As you can see in constraint 5, we basically set C\_ik to 0 if a player has received a color two times in a row previously. With these constraint we dont have to worry about any other posibilities because we directly set C\_ik to be 0.

Update 2:

Constrain 7: In general, players with the same running score are paired.

∀𝑖 ∈ {(𝑎, 𝑏) ∈ 𝐼 𝑥 𝐼|𝑎 ≠ 𝑏}, ∀𝑗 ∈ 𝐽

min (|S2X2(- S3X3(|)

which is equivalently as saying:

∀(𝑎,𝑏)∈𝐼𝑥𝐼, ∀𝑗∈𝐽 𝑚𝑖𝑛 H4

Hz ≥ SaXaj - SbXbj

H4 ≥ -(SaXaj- SbXbj)

When trying to implement this constraint into Gurobi, the model would take up to much time to solve the objective function, and when I had to stop it there was no solution. Basically, this constraint aimed to minimize the difference of running scores of all players in the same slot j. However, I realized ther was a more straightforward way to achieve a similar result. This is how we came up with constraint 7.

d = 0

First of all, I limited the amount of constraints created by setting up an if statement that would only create a constraint were the running score of a pair of players was the same. If this was true for a pair of players, then we wanted to maximize the number of slots with these players, so these players had a greater chance to play together. We introduce the variable “C7” also in the objective function to maximize it.

Update 3:

For constraint 9, the previous formulation was:

The problem with this formulation was that it led to a player a play with color white and a player b play with color white too, or both having black. If Z\_p was 1, then:

We know that from constraint 4, the sum of colors per each player must be 1. It can be white, black, or not assigned.

Having these two constraints together, leads that if two participants are paired together, the sum of their colors must exceed 2. However, there is also the possibility that one player is given white and the other player is also given white, in that case, both constraints would still be achievable with no problem.

To deal with that problem we had to update this constraint:

d = 1

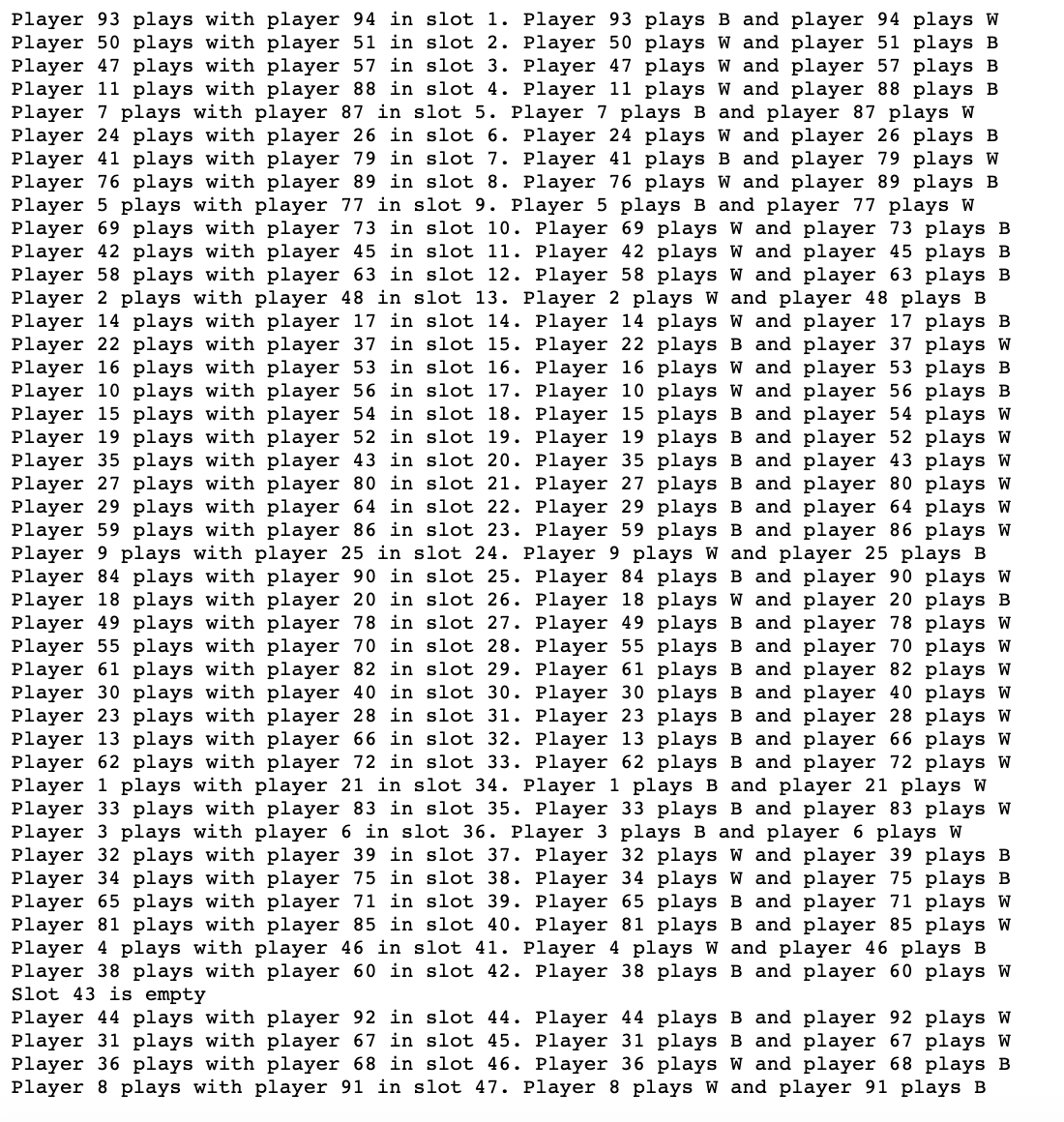
C9 is a binary decision variable denoting if player a is given white and player b is given black, and C9\_d+1 is the same but vice versa. More of the way we got the constraints through linearization of logical expressions are in the appendix.

In conclusion, formulating a linear program for this type of problem involves a lot of iteration to create various types of constraints. It involves always going back and forth trying to correct the errors and to update our formulations.

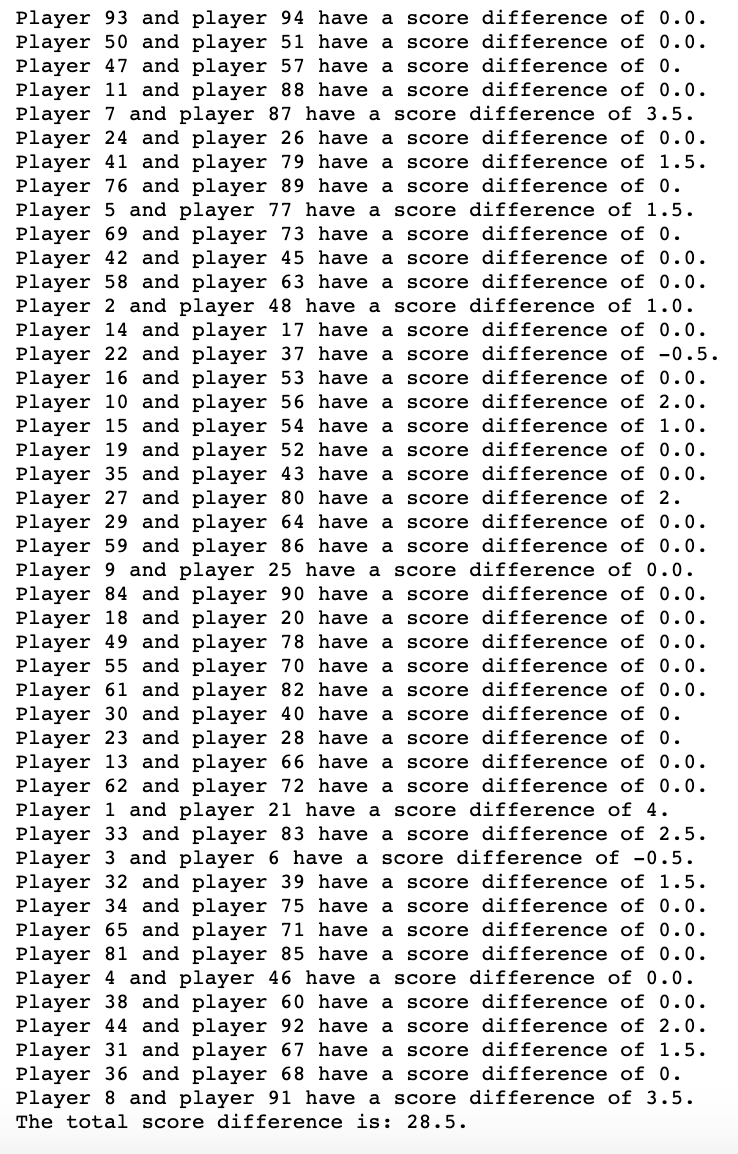
**Appendix:**

Pairings and color:

W stands for white, and B for Black.



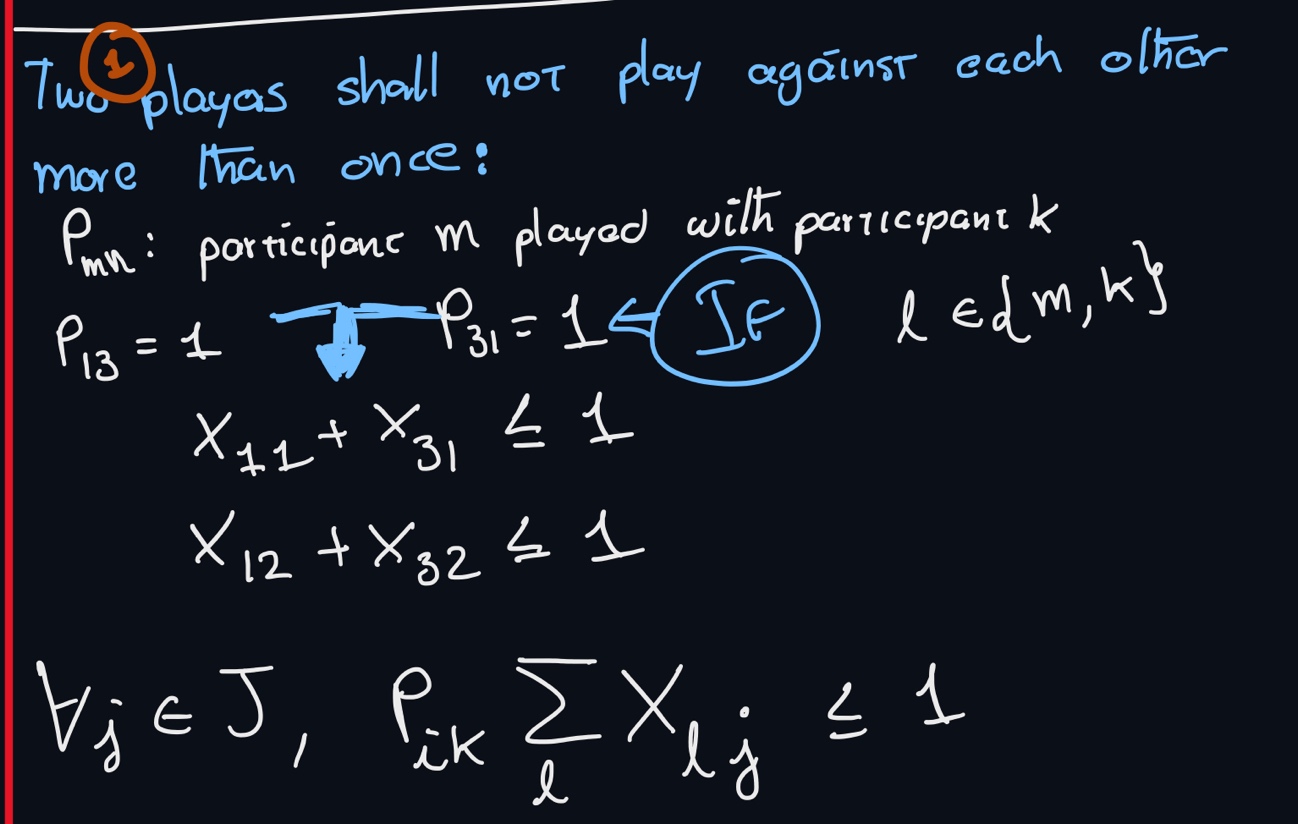
Score differences:



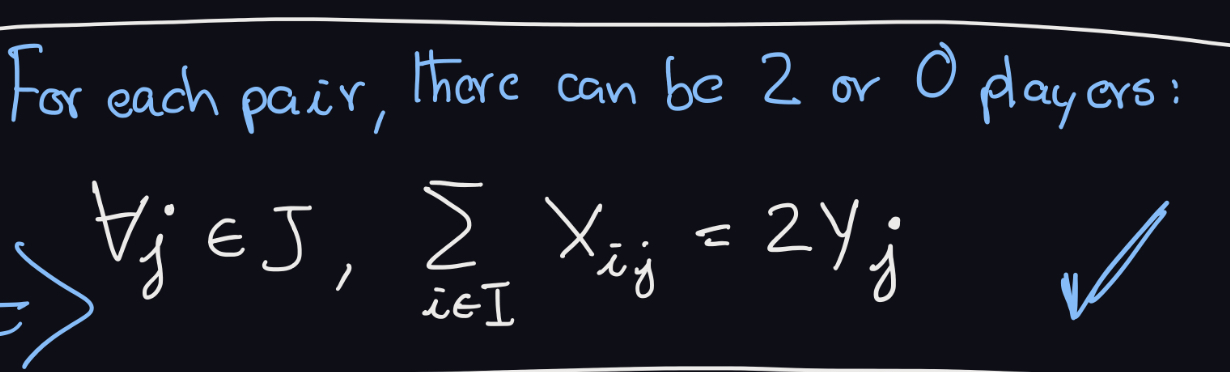
Unpaired players:



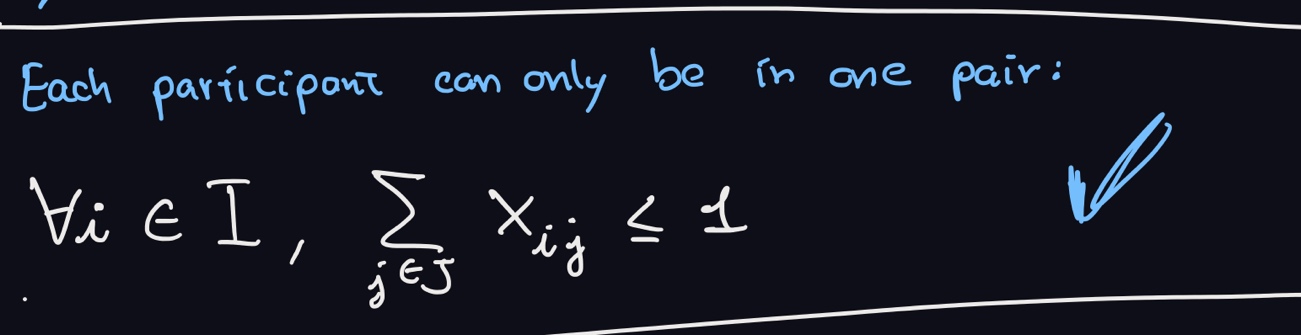
Constraint 1:



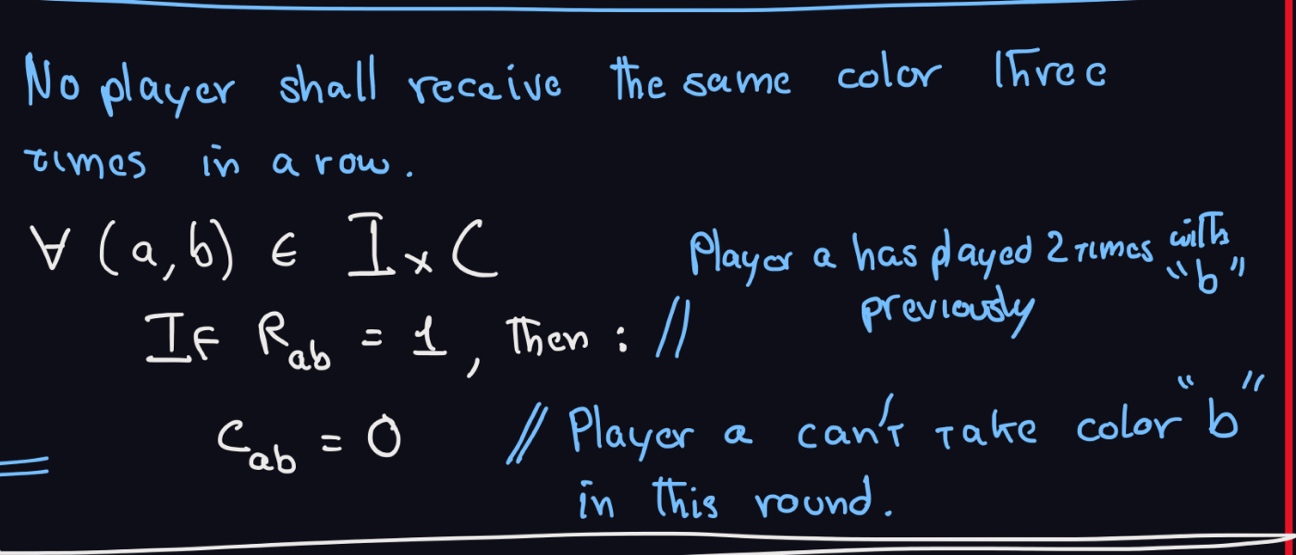
Constraint 2:



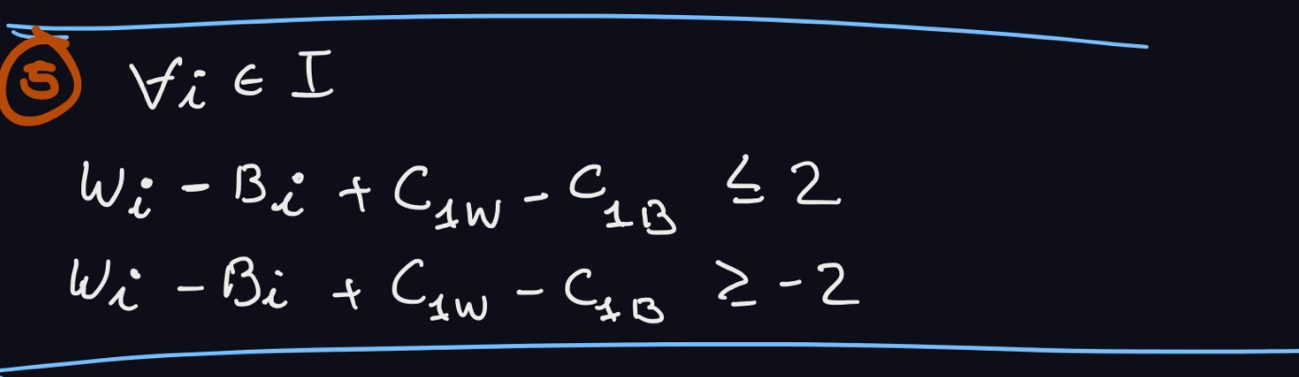
Constraint 3:



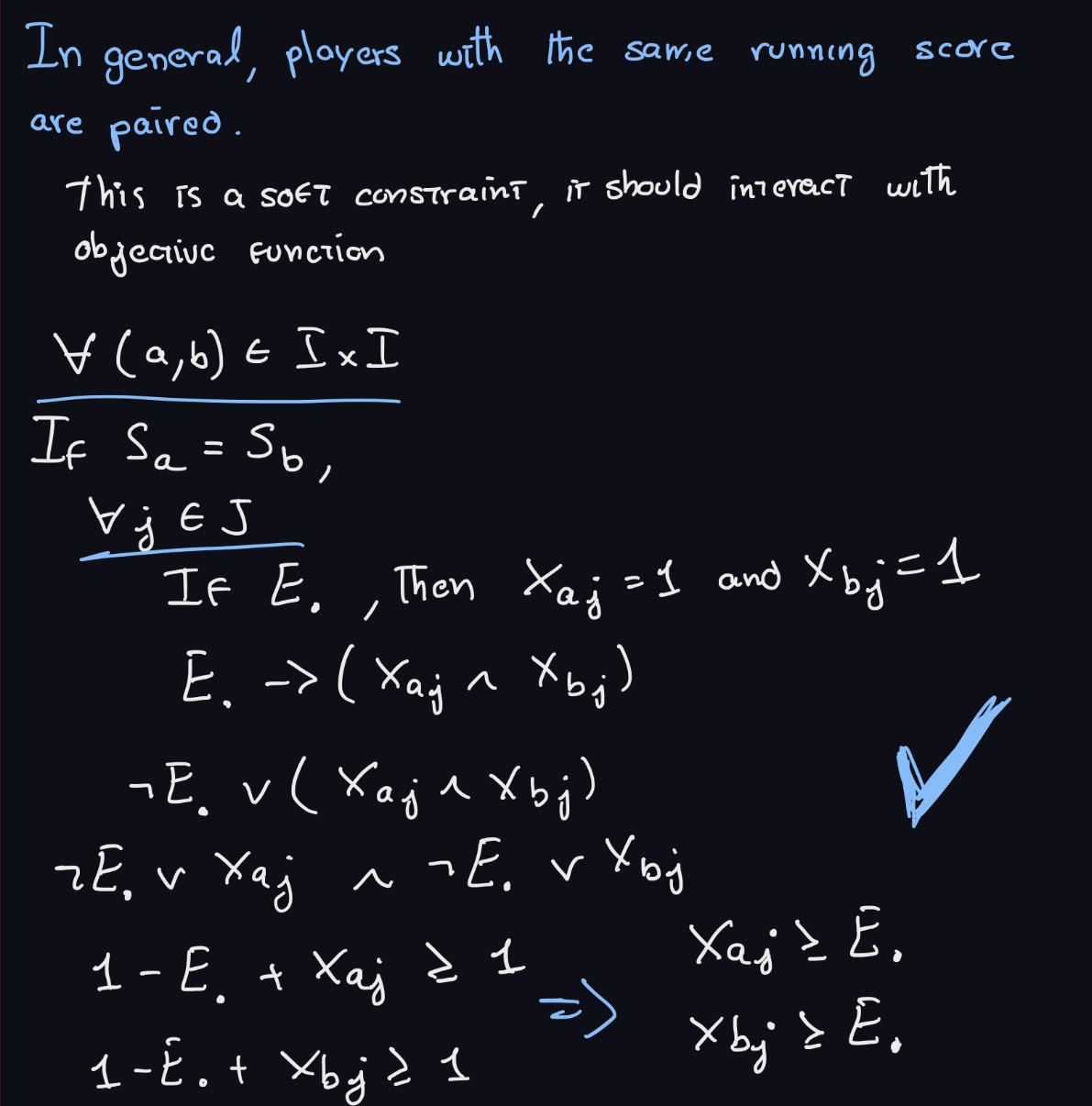
Constraint 5:



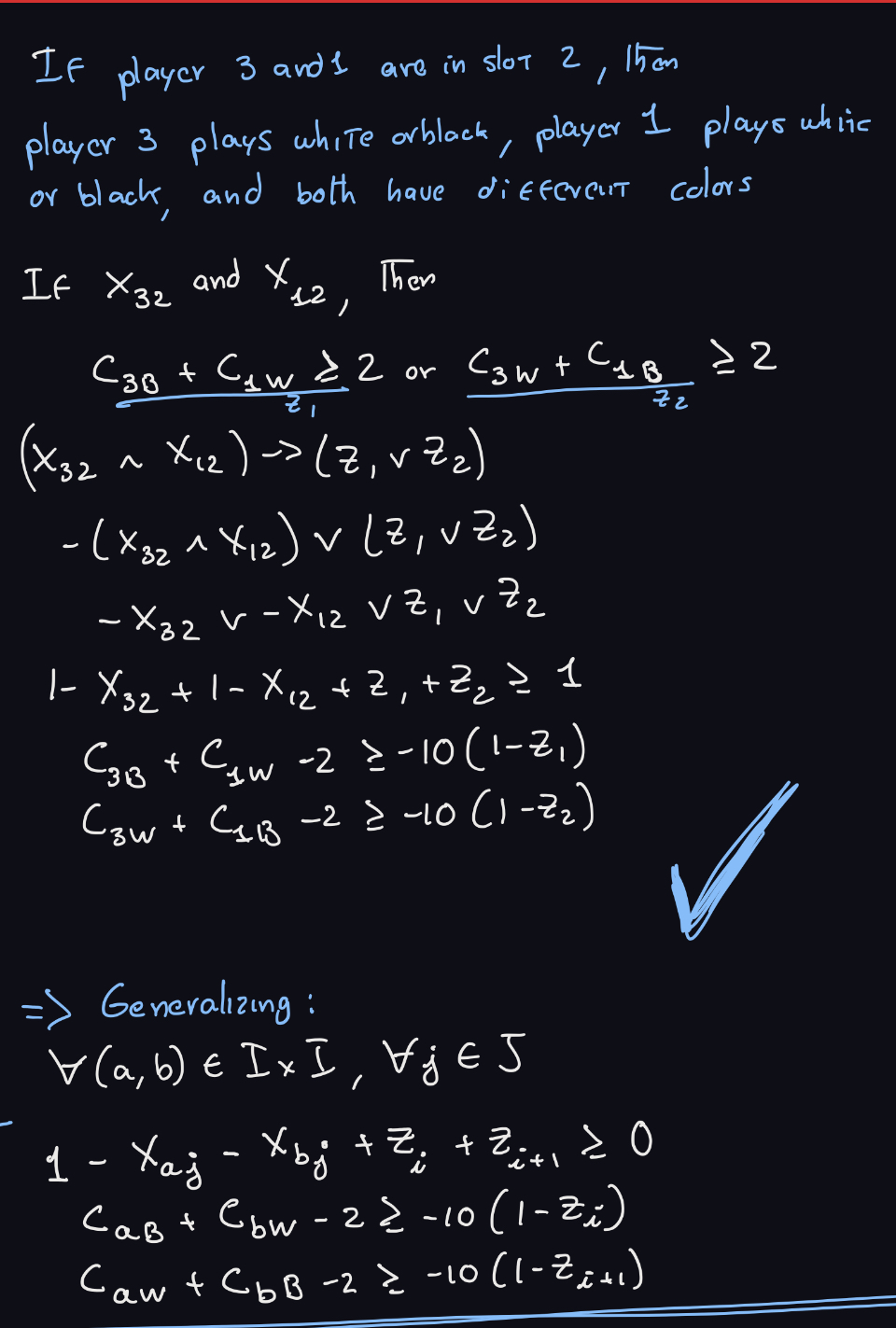
Constraint 6:



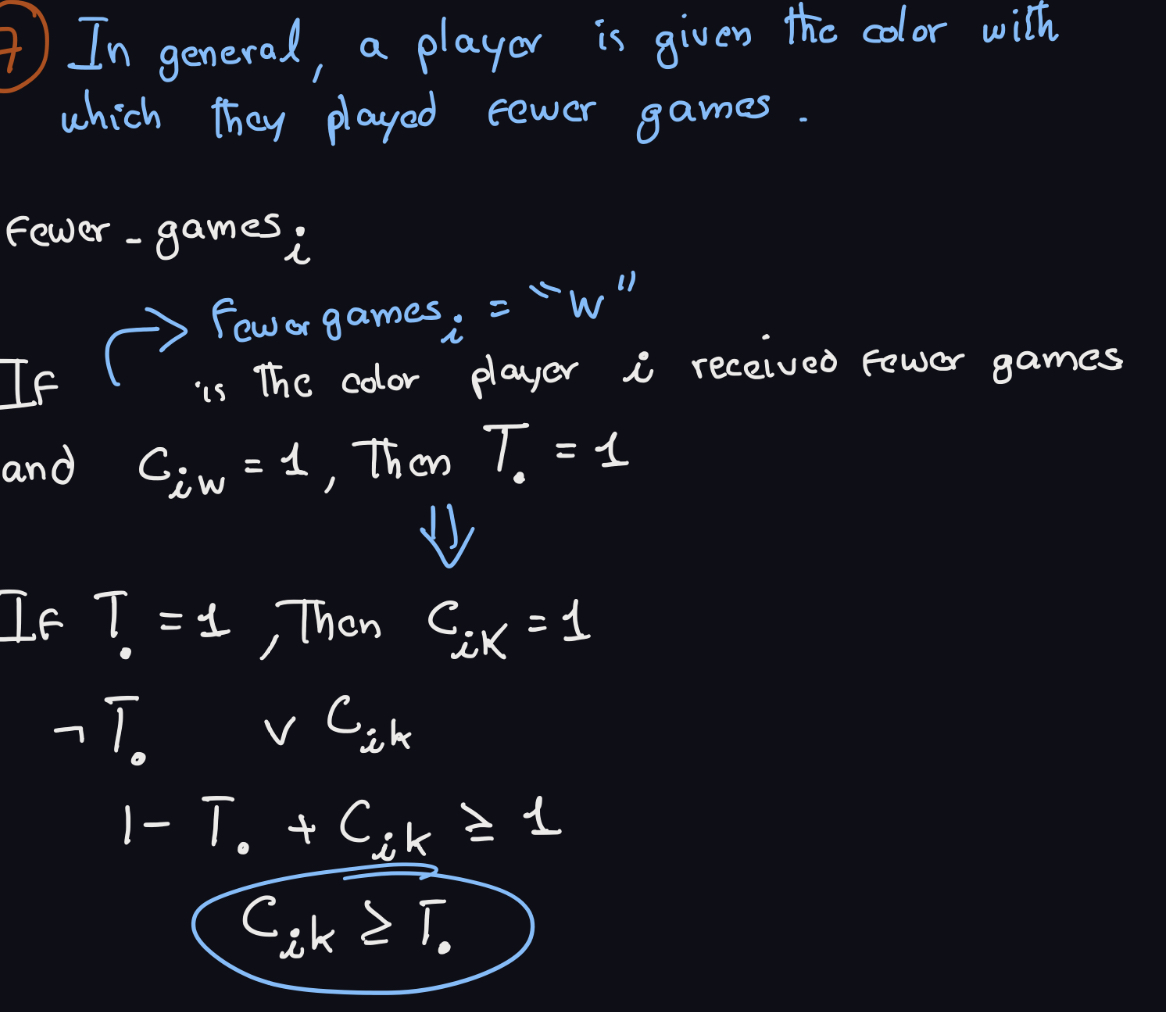
Constraint 7:



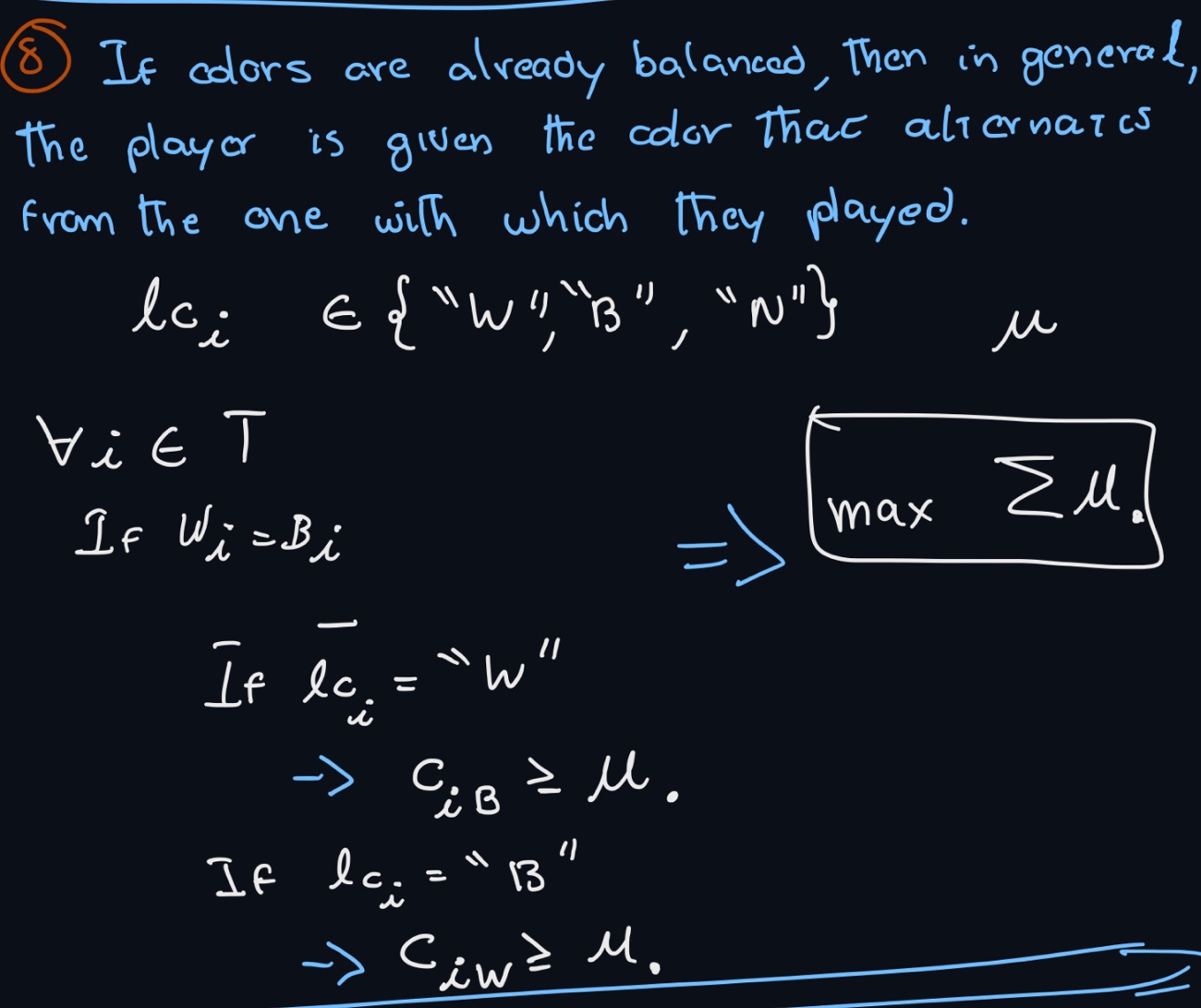
Constraint 9:



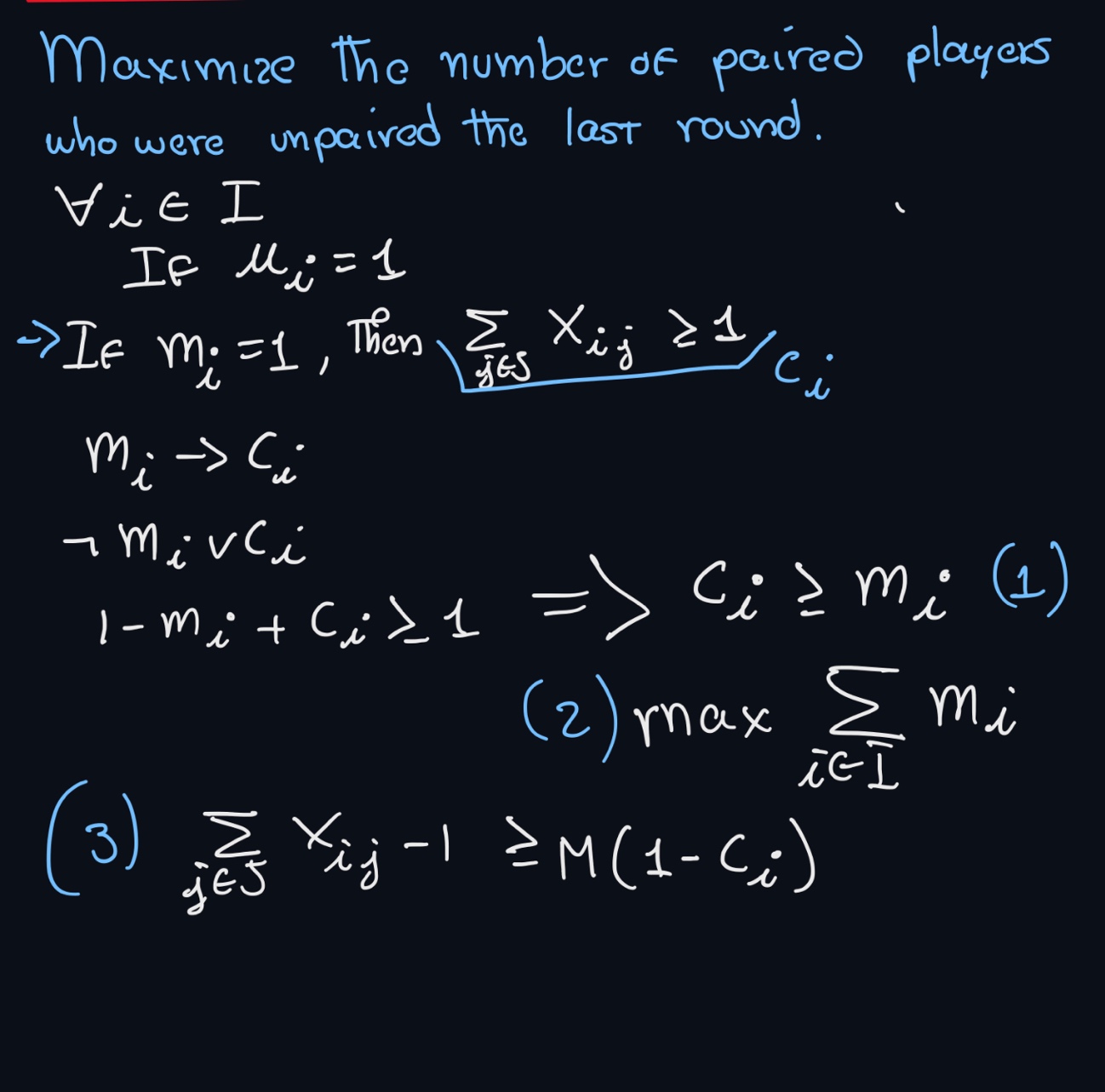
Constraint 10:



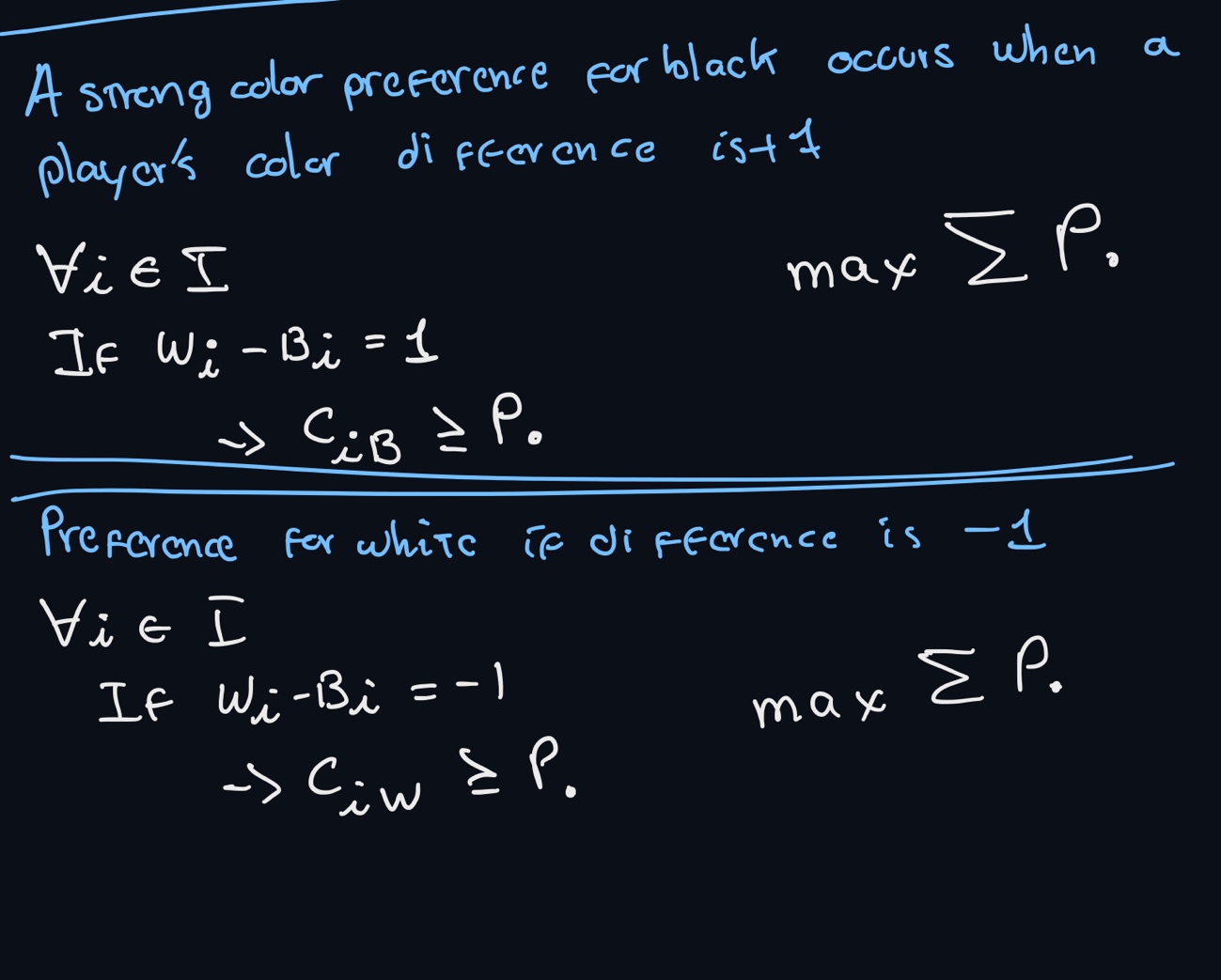
Contraint 11:



Constraint 12:



Constraint 13 and 14:



Constraint 15 and 16:

