Introduction

Lol is bla bla multiplyer game that is bla bla… (keywords of the game need to be written here, like champions, lanes and more)  
each champion has his own special abilities so it almost impossible to know which combination of champions will give the most advantage.

The main goal in our research is that we want to predict how well 2 champions might play together based on previous games that already played (Without necessarily a game with those two).  
We used Leskovec methods for this prediction and extended it with to be more directed to League Of Legend domain.

The Main Goal  
Right now from our data we could extracted XXX champions with YYY relations between them from ZZZ matches. We will explain on the next chapter how we construct our graph but for now we will say that each champion is a node and relation between 2 nodes mean that they play together. Ideally our graph will be full connected with enough data that each edge is based over multiple games. Well, that’s not the case and this is where our research comes to help.  
A fully connected undirected graph with XXX nodes will have nCp(XXX, 2) = ? edges, means that we don’t know about [nCp(XXX, 2) – YYY ] edges.  
First we want to find out about *if* 2 champions are playing well together, similar to Leskovech research, but after getting these result we decided to extend it and check whether if we can actually predict *how* well these 2 champions play together.

Method

The Steps from Data to Prediction

Lol by Riot give the developers an API with a lot of information about their game including matches of every player we just want.  
The players we use for our research are some of our friend’s accounts names and most of them from the ranking page (<https://eune.op.gg/ranking/ladder/page=30>).  
1. First, we fetch account id from name and then take his recent 100 last matches.

2. After that, from all the data we get we create a graph. Our algorithm is simple:

Algorithm [

For all matches:

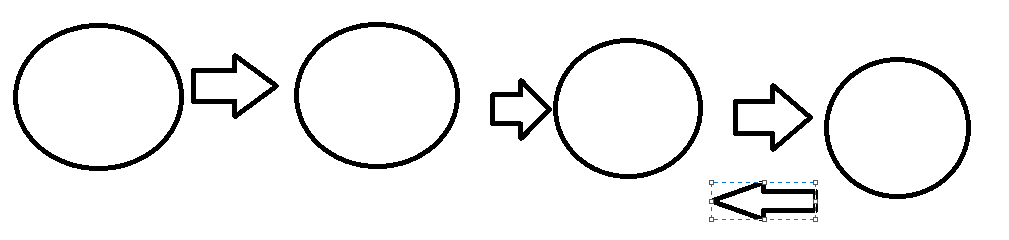
* Get winning champions
* Add 1 to weight between each 2 champions in the graph
* Get losing champions
* Subtract 1 to weight between each 2 losing champions in the graph

]

3. Next, from the graph we extract features to help us to classify/predict the relation between 2 champions and build a DB for a machine learning purpose.

4. Learn the result and try to understand if we can improve that.

[לצייר בלאטח משהו כזה]



Data Source

All our data have been extracted using RIOT API. Following their tutorial [<https://developer.riotgames.com/docs/lol#general>] was easy enough and then with python libraries that make it even simpler we fetch all the data we needed.  
We just had to make sure we regenerate our API Key for new data if we needed.  
The full API can be found here [<https://developer.riotgames.com/apis>]

There are a lot of type of requests, but we used only 2 of them for our purpose.

1. <Account name> to <account id>
2. Last N recent matches of <account id>

In our case we choose N to be 100 to compliance the conditions of riot to developers’ product of 20 requests every 1 second / 100 requests every 2 minutes.

The response from our requests is returned in a JSON format, each match has almost 1500 fields, so we add an example with this report (example.json).

The JSON file is build of array of matches, the fields that used in our cases are only:

* Match[‘gameid’] : Int
* Match[‘participants’] : Array of participant
* Participant[‘stats’][‘win’] : Boolean if win

The Graph Construction

As we already mention

Algorithm [

For all matches:

* Get winning champions
* Add 1 to weight between each 2 champions in the graph
* Get losing champions
* Subtract 1 to weight between each 2 losing champions in the graph
* C(e, m) ++

]

Because we fetch last matches from multiple players, we make sure to remember the matches id to avoid duplications.  
We add a new field named C(e, m) = Count of Matches between edge e.

Each node is a champion and the edges are the relation how well they play together, the higher the weight the better their performance are.

To avoid underfitting on edges with not enough matches we test our model with multiple threshold of matches > *th* to call the link as edge, these is the graphs that were built:

|  |  |  |
| --- | --- | --- |
| th | edges | Missing edges |
| 0 |  |  |
| 1 |  |  |
| 3 |  |  |
| 5 |  |  |
| 8 |  |  |

Our Model

We use ML algorithm to classify whether 2 champions are working well together or not.  
From our simple research the best regressor that exist these days is gradient boosting and XGBoost is the most recommended library for this purpose. Gradient boosting is type of Random Forest algorithm that convert weak learners into strong learners.

The DB is a pandas DataFrame that each row represents an edge with these features:

Phase 1

At the beginning we did like Leskovec’s paper

W(u) = sum all edges weight that connected to node u

להוסיף summary על הטבלה רק עם WU WV..

Because of that the results were quite impressive we decide instead of classify the sign we should try to predict the value.  
This is the results:

Like Heider about balanced structure we thought that if a champion X play well with another champion that play well with another champion Y probably X play well with Y. So we added another 4 features using this idea:

* D2 : average distance of 2 hops from u to v
* D3 : average distance of 3 hops from u to v
* D4 : average distance of 4 hops from u to v

We thought this is enough because the running time to calculate this start to take way too long, and teams in this game is made of 5 champions for each team so 4 hops will be enough.

The results are: