Welcome to the Jungle: Effect of Jungler Pathing on Winrate in League of Legends

Ben Drews
Justin Smilan
Gary Chen

12/16/2016

Abstract

In the popular strategy game League of Legends, two teams of five race to complete objectives and ultimately destroy the opposing team's nexus. The players within these teams are divided into five roles: Top, Mid, ADC, Support, and Jungler. At the beginning of the game, the first four roles are divided between three "lanes" on the game map, whereas the jungler takes a more variable path. The jungler first gains experience and gold by defeating stationary, computer-controlled "jungle camps," then roams into any of the three lanes to give their teammate(s) an advantage against their respective opponents (a process known as "ganking"). In League of Legends, there is a common belief that early advantages tend to snowball into greater leads as the game goes on. We are interested in how the initial path taken by a team's jungler impacts the team's winrate. We attempt to do this by using Fuzzy C-Means (FCM) clustering and k-Nearest Neighbors (kNN) to identify similar jungle paths and generate a predicted winrate for those paths. Our experiments with the clustering algorithm showed some interesting trends in jungle paths, but our attempts to predict a win or loss using kNN showed that this was not very statistically significant. Because of the vast amount of variables involved in a League of Legends game, early jungle pathing alone is a poor predictor of winrate.

Introduction

This is a pattern recognition problem. We want to look at similar early game jungle paths and determine whether they are more or less likely to lead a team to victory.

For this project we must recognize two different patterns. First, we must recognize which jungle paths are essentially the same. In order to sort the positional data gathered from the League of Legends API into a series of general paths, we required a sophisticated clustering algorithm. Second, we wanted to see how (and if) the various paths taken by junglers predicted winrate. In order to solve this problem we needed an machine learning agent to classify the data.

This year, the League of Legends World Championship had a prize pool of over $5 million USD[1]. League of Legends has 100 million monthly players worldwide[2], an active community looking to improve their ability. We believe that the clear passion demonstrated for this game, as well as its overwhelming popularity, merits examining all of its strategic facets scientifically.

[1]"League of legends 2016 world championship - tournament results & prize money: E-sports earnings," e-Sports Earnings, 2016. [Online]. Available: http://www.esportsearnings.com/tournaments/20345-lol-2016-world-championship. Accessed: Dec. 16, 2016.

[2]P. Tassi, "Riot games reveals league of legends has 100 Million monthly players," in Forbes, Forbes, 2016. [Online]. Available: http://www.forbes.com/sites/insertcoin/2016/09/13/riot-games-reveals-league-of-legends-has-100-million-monthly-players/#619cf04610b1. Accessed: Dec. 16, 2016.

Survey of Related Work

In researching work related to our project, we started by observing how machine learning had been applied to League of Legends in the past. While we couldn't find anyone else analyzing jungler pathing, we did find one study that used machine learning to analyze how many other factors more easily obtained from the League of Legends API correlated with winrate[3]. This report didn't include any clustering, but it helped us in assessing what League of Legends data was the most useful to analyze.

In searching for an example of clustering used to analyze data in strategy games, we found an interesting study on the popular card game Magic: The Gathering[4]. This gave us an idea of how we could use the FCM algorithm in particular to separate out the important features in early game jungle pathing.

From this point, the sources we consulted were related to the technical aspects of implementing our project. Since we did not discuss FCM clustering in class, we had to find external resources on it.[5]

Formulation

We defined our problem as trying to label early game jungle paths as either a win or a loss based on what champion was used, and what their positions were for the first 6 minutes of the game (roughly the amount of time it takes to complete a full jungle clear). Simply running k-Nearest Neighbors on the data would be far to sensitive to noise for any reasonable value of k. Additionally, we wanted to analyze what the different jungle paths taken by each champion: what were the features that defined them, which were the most common, and what did each represent. To accomplish both of these goals, we turned to the Fuzzy C-Means clustering algorithm.

[3]Mike, "Playing in random forests in league of legends," in Trail of Papers, 2015. [Online]. Available: http://www.trailofpapers.net/2015/10/playing-in-random-forests-in-league-of.html. Accessed: Dec. 16, 2016.

[4]drogerson, "Mathing Diversity in the Modern Magic Meta," in Drogerson/public-projects, GitHub, 2016. [Online]. Available: https://github.com/drogerson/Public-Projects/blob/master/Mathing%20the%20Modern%20Magic%20Meta.ipynb. Accessed: Dec. 16, 2016.

[5]A. Naik, "Fuzzy c-means clustering algorithm - data clustering Algorithms," 2010. [Online]. Available: https://sites.google.com/site/dataclusteringalgorithms/fuzzy-c-means-clustering-algorithm. Accessed: Dec. 16, 2016.

Fuzzy C-Means

This algorithm is similar to the k-means clustering algorithm, however, it assigns each data point a membership weight to every cluster. The algorithm is initialized by choosing a number of clusters and assigning each cluster a random data point to use as an initial centroid. FCM then calculates the cluster weights and then recalculates the new centroids. This repeats until the evaluation metric converges.

Algorithm 1 Fuzzy C-Means

procedure CLUSTER($dataMatrix$, c)
    $centroidMatrix \leftarrow c$ random data points
    $newEval = -2 * \varepsilon$ ▷ to ensure we don't end immediately
    $oldEval = 0$
    while $|newEval - oldEval| > \varepsilon$ **do** ▷ Iterate until the evaluation converges
        $oldEval \leftarrow newEval$
        $weightMatrix \leftarrow updateWeights(positionMatrix, centroidMatrix, c)$
        $centroidMatrix \leftarrow updateCentroid(positionMatrix, weightMatrix, c)$
        $newEval \leftarrow eval(positionMatrix, weightMatrix, centroidMatrix, c)$
    end while
    return $centroidMatrix$ ▷ The centroids define the clusters
end procedure

k-Nearest Neighbors

We used k-Nearest Neighbors as a classifier to predict the winrate of the various jungle paths we observed. The number of neighbors was set to be the number of clusters, and each cluster was weighted based on the Fuzzy C-Means weights. These weights were used along with the cluster labels to give a predicted winrate of the test path. We then classify the game as a win if the predicted winrate is above 0.5, and a loss otherwise.

Algorithm 2 k-Nearest Neighbors

```
1: procedure K-NN(clusters, testData)
2:     for testPoint in testSet do
3:         clusterWeights ← []                ▷ Find how close each point is to each cluster
4:         for cluster in clusters do
5:             clusterWeights.append(updateWeight(testPoint, cluster)
6:         end for
7:         predictedWinRate ← 0
8:         for weight in clusterWeights do
9:             predictedWinRate ← predictedWinRate + clusterLabel ∗ weight
10:        end for
11:    end for
12:    if predictedWinRate > 0.5 then
13:        return win
14:    else
15:        return loss
16:    end if
17: end procedure
```

Approaches

Since our predicted result varied greatly based on the specific jungle Champion used for each game, we began by stratifying the data by champion. We then took data targeting a specific set of champions, in this case the five most popular junglers. Then we used the Fuzzy C-Means clustering algorithm to separate out our data into clusters. Our fuzziness constant (1.5) was selected through trial and error, finding the value that allowed the clusters enough distinction but also prevented them from easily converging. Since our data was interpreted solely based on map position, we only looked at junglers playing on blue side.

After obtaining the clusters, we found the percentage of those paths in each that corresponded with a win. We used k-Nearest Neighbors to perform a regression and estimate what the likelihood of a win is for any given path and predict if the game was won or lost. Additionally, we output the centroid of all clusters along with their winrate labels.

To implement this, we gathered data from Riot Games, the company that creates League of Legends, through their Riot Developers API. Our data was gathered by taking the match list of a seed player at high rank, and recursively gathering the match lists of all the players the seed player recently played with. We used this over a random sample because we wanted to weight our data toward higher skill games, where gameplay is closer to optimal.

Our application was originally going to be written in Java and make use of Google's GSON library to parse the data obtained. However it became clear early on that Python allowed for much easier manipulation of JSON data. We performed k-fold cross validation, clustering with the training set into c clusters and then using k-Nearest Neighbors (with $k = c$) for regression to predict the win rate of each jungle path in the test set. If the predicted win rate was above 0.5 we labeled the game a win, otherwise we labeled the game a loss. After each clustering session, the application output a JSON object to file containing information about the different clusters defined for the targeted Champion.

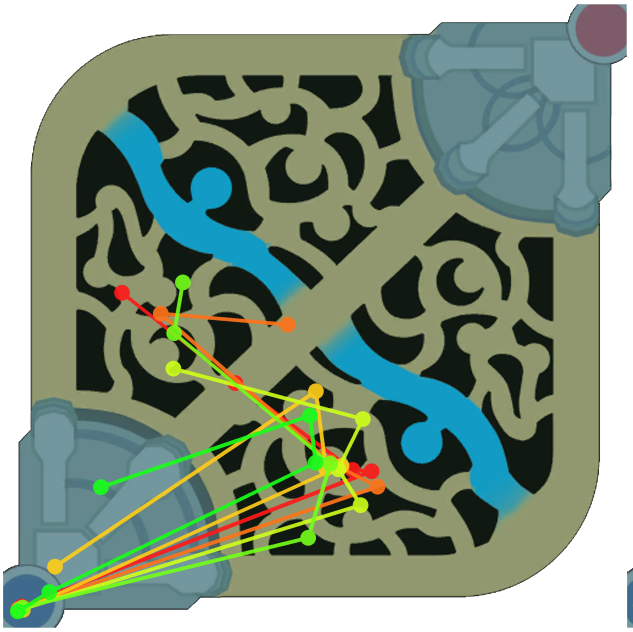
We then built a visualizer to help us interpret the initial jungle paths from our cluster data. We accomplished this using Pillow, a fork of the Python Imaging Library (PIL) that is compatible with Python 3. Our visualizer read the JSON objects containing our cluster data and illustrated the paths taken by the junglers. We then projected these paths onto a blank map of Summoner's Rift we obtained from the League of Legends wiki. Because the Riot Games API only polls player location every minute, we aren't able to see the exact path the jungler took, but our diagrams illustrate the general shape of their path.

Experiments & Analysis

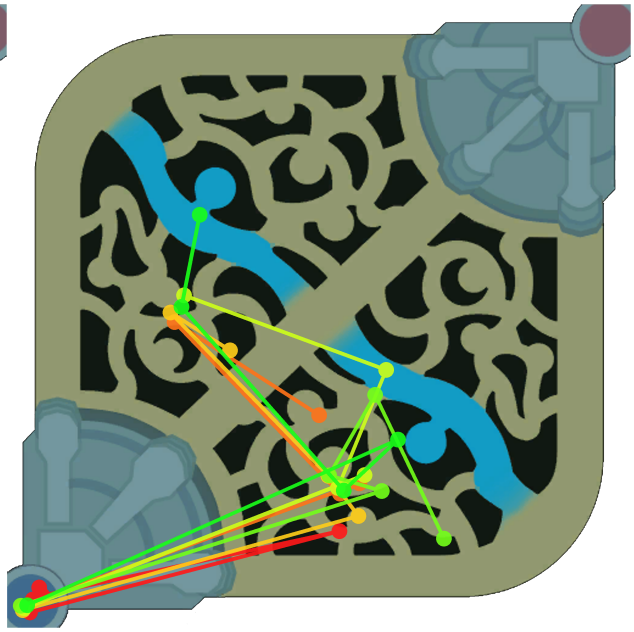
We ran our experiments on data for the five most popular jungle champions at high ELO (ranked around Diamond or higher). The following diagrams show the paths generated from the clusters for each champion, arranged in order of popularity. The red paths represent the lowest success rate, while the green paths represent the highest success rate.

Figure 1: Highest and lowest winrate paths for popular junglers in high ELO

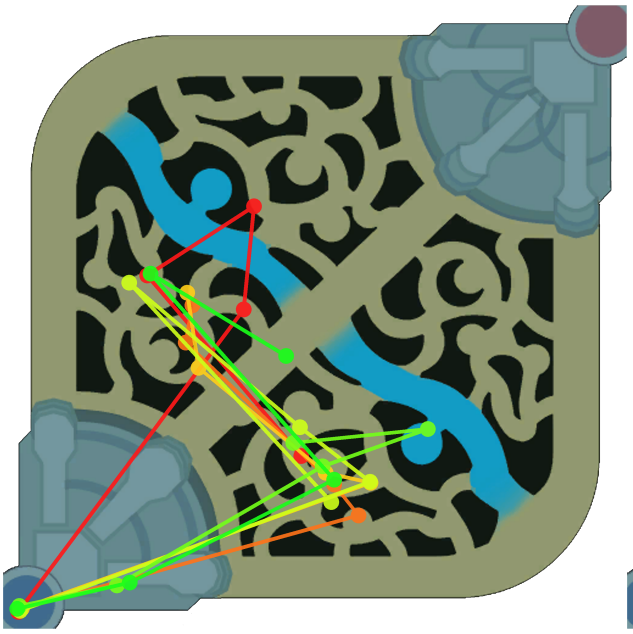
(a) Lee Sin



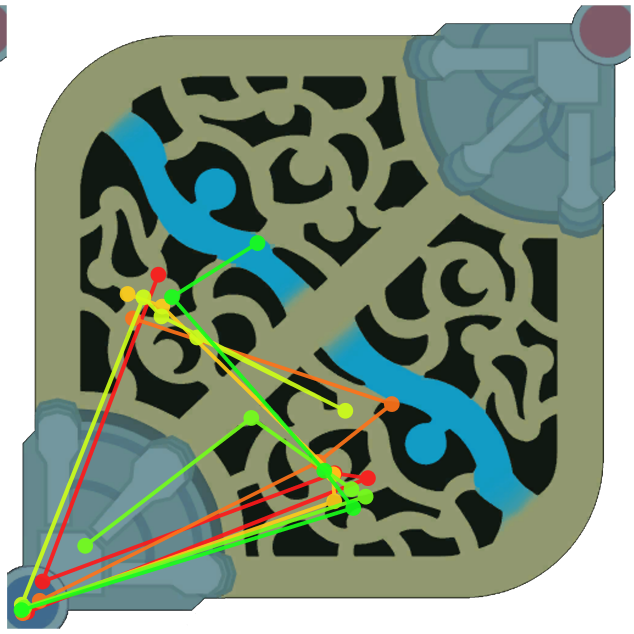
(b) Vi

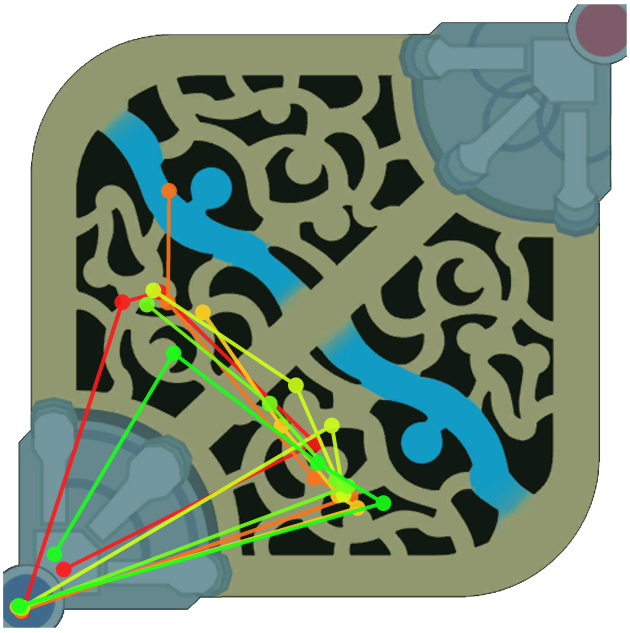


(c) Shyvana



(d) Hecarim





(e) Kha'Zix

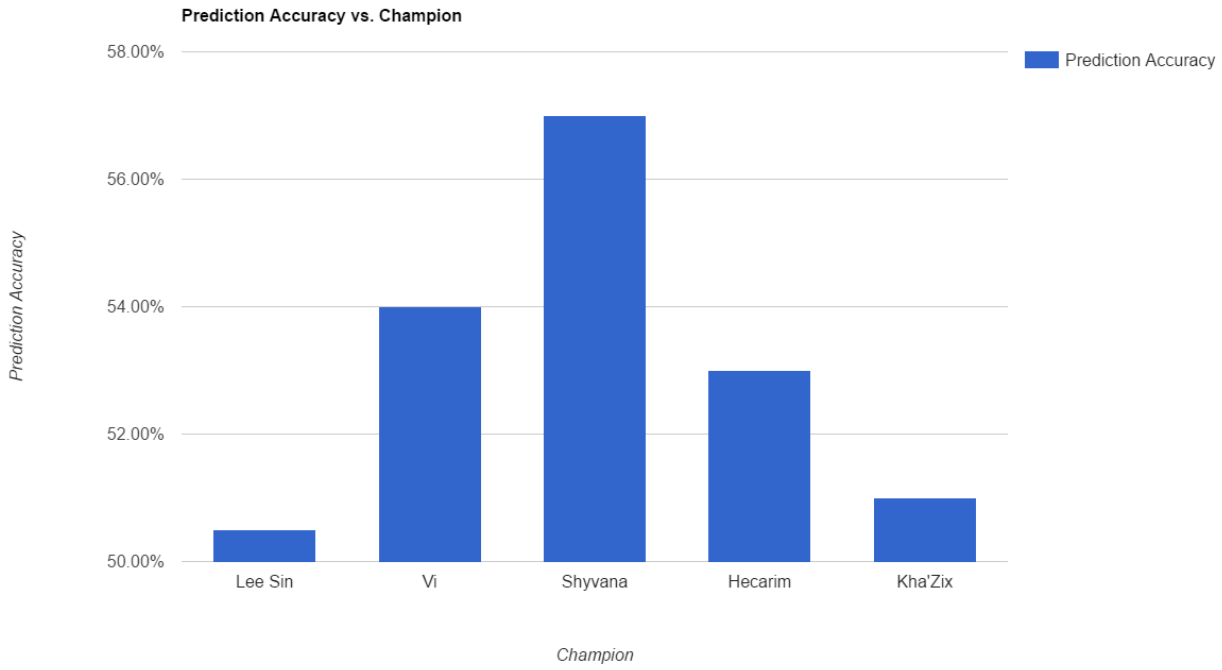
There are some interesting patterns that can be observed from this data. For all of the champions surveyed, most jungle paths began on the bottom side of the jungle, and apparently with good reason. In figures 1c and 1e, the paths in which the junglers began their clear on the top side of the jungle demonstrated the lowest winrate. Looking more closely at the worst-performing cluster in 1c, the rate of failure in the red route may also be explained by a common misconception that Shyvana should be using her exceptional speed at clearing jungle camps to clear her blue side quickly and then start taking enemy jungle camps early on — we see that at three minutes she invades the opposing team's jungle. However the higher winrate paths for Shyvana show that it is preferred to use her clear speed to rush through her own jungle then recall and start again.

We also visualized the paths with the highest average weights. These figures are included in the appendix.

Predicting winrate from early game jungle paths

We also wanted to observe how the paths taken by junglers predicted winrate.

Figure 2: Prediction accuracy results



Our main finding in this section is that early game jungle pathing is a poor predictor of winrate. This is consistent with the data we read about in "Playing in random forests in League of Legends" [3]. This post contains a heatmap which indicates that none of the data available in the first five minutes of the game is a very strong predictor of winrate.

However, our prediction accuracy is significantly stronger for some champions than others. We interpret this data to mean that early game jungle pathing is more important for Shyvana than for Lee Sin, to give an example. To contextualize this in terms of gameplay, Shyvana's performance is more impacted by how she farms the jungle in the first six minutes of the game.

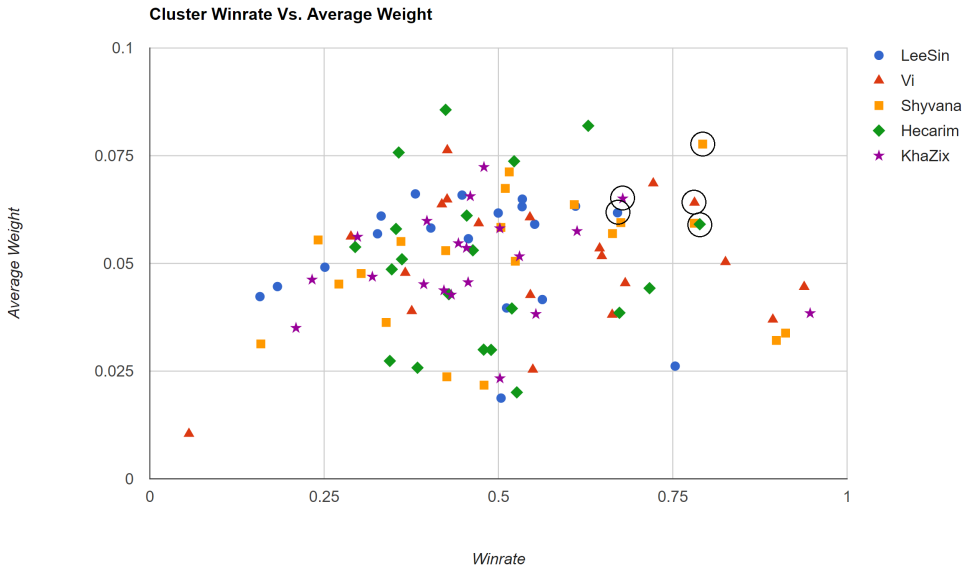
Visualizing relatively successful jungle paths

If we plot all of our clusters for each jungler by their winrates and their average weighting over all datapoints for that champion, we see that our experiment identifies a relatively flat, even distribution of what paths are popular in terms of how successful they are; no regression is easily identifiable, suggesting that players don't necessarily take the best or worst jungling paths. However, there were certain datapoints which stood out; we selected, for the sake of an example, one path for each champion which was particularly successful and also popular

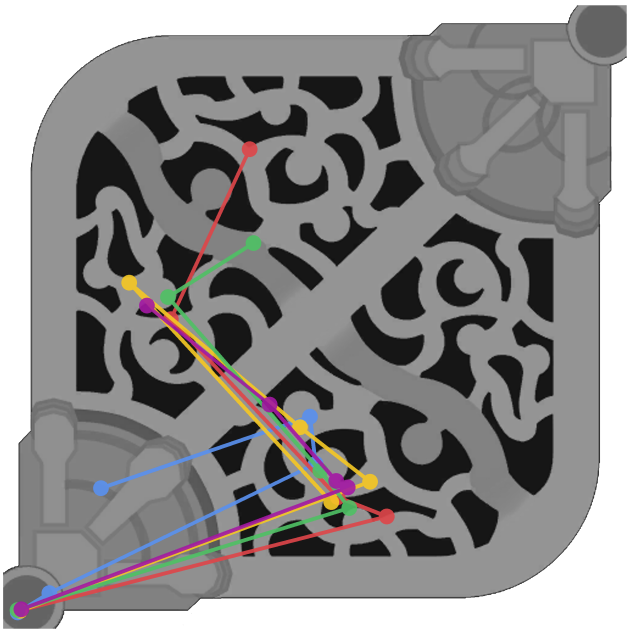
in our training data, circled in the scatter plot. Their corresponding paths are shown in the map below.

Figure 3: Successful clusters for each champion

(a) Scatter plot of all clusters for each champion



(b) Selected paths for each champion plotted on map



Conclusions

Our major finding from our project is that, with all the variables involved in League of Legends, early jungle pathing is a poor predictor of winrate. However, some paths are clearly better than others, and the route a jungler takes matters more for some champions than others. For instance the path taken by Shyvana seems to be a much better predictor of winrate than for Hecarim. Additionally, when we look at Shyvana's clusters, there are highly weighted clusters that have winrates both much higher than average and much lower than average.

The major weakness in our study was the number of data points we were able to use. We had initially wanted to look at the first 10 minutes of gameplay, but because of the curse of dimensionality, when we included that many data points the clusters all converged into one path. Additionally, when we attempt to use larger amounts of data we did not have enough random initial clusters to hit all of the clusters, but using more clusters added significant runtime to our program.

For future work, we could look at data from junglers playing on red side, look at a wider breadth of champions, or try to include more positions in each data point. Alternatively, we could look at pushing the computational burden of our application to the GPU using PyOpenGL to take advantage of the large potential for parallel processing. This would hopefully allow us to process much larger data sets with higher cluster counts with reasonable efficiency.

Appendix

Ben Drews: Wrote the match crawler for the Riot Games API and the majority of the code for the AI agents. Also provided expert knowledge of League of Legends.

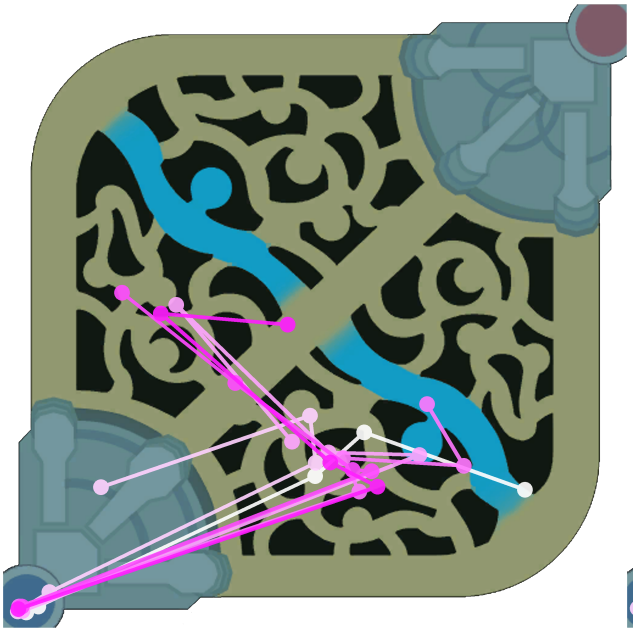
Gary Chen: Wrote the visualizer program and created the figures used to analyze the data we gathered.

Justin Smilan: Researched related works and wrote the majority of the lab report. Also contributed to analysis.

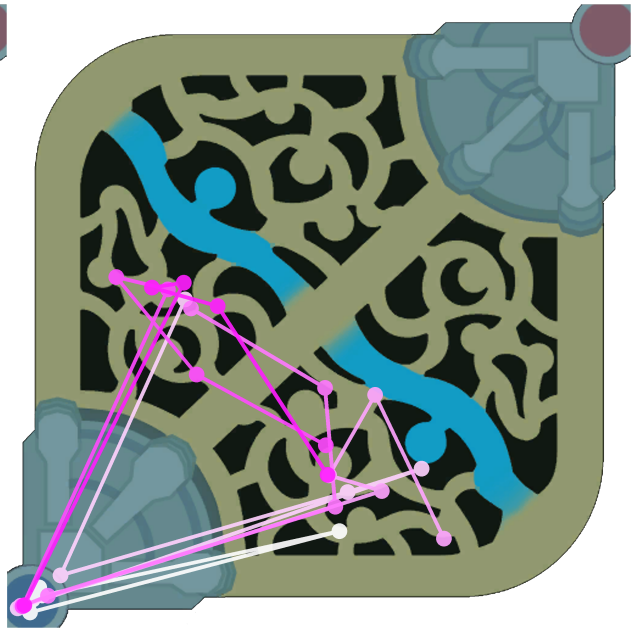
Additional figures

Figure 4: Highest weighted paths for popular junglers in high ELO

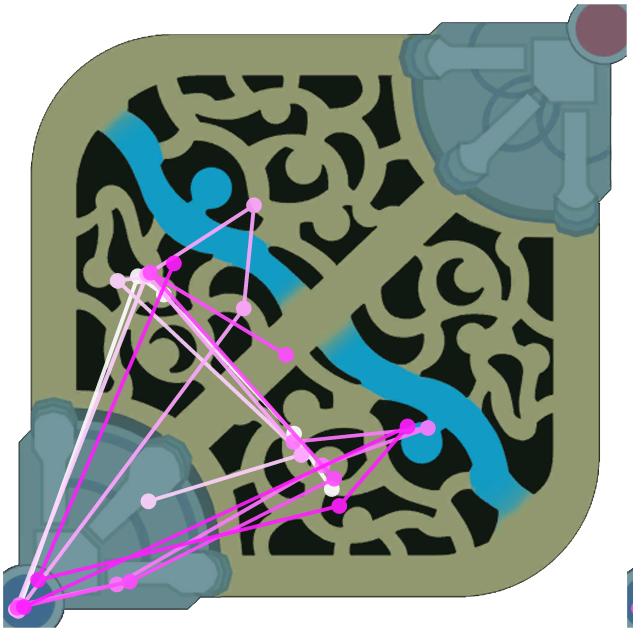
(a) Lee Sin



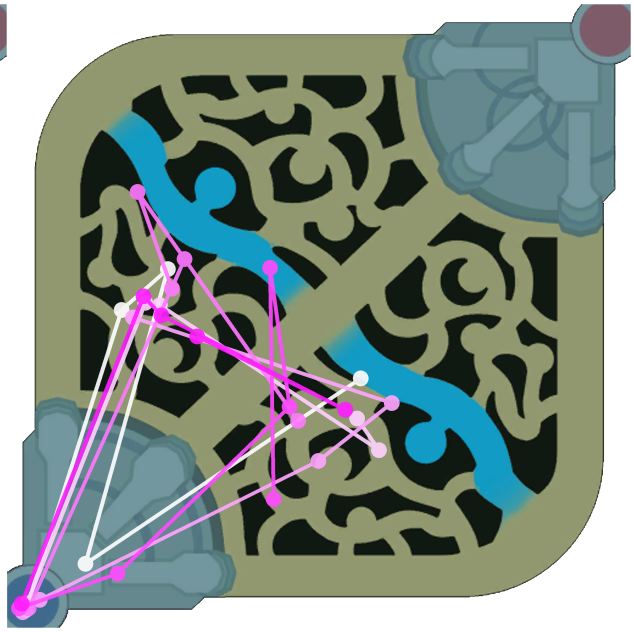
(b) Vi

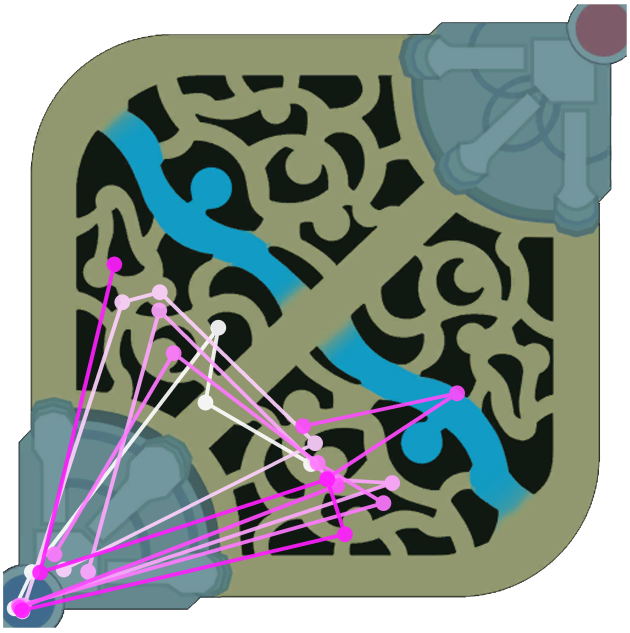


(c) Shyvana



(d) Hecarim





(e) Kha'Zix