Andrew Severance

Prof. Leonidas Kontothanassis

CDS DS 210

15 December 2023

Final Project Write-up

Introduction

As is the case in many sports, Formula 1 and its brothers (F2, F3, F4, Formula E, IndyCar, etc.) collect data on each car and driver's (or the "pilot's") performance at every competitive race. While data on the car's performance and the pilot's record is extremely helpful in determining who is considered the "best" pilot and who will score new contracts with their team for future seasons, it does little to analyze which pilots are the most competitive or who have competed against the most other pilots. This is what I sought to analyze in my final project. The program I built creates a graph of the pilots and connects them based on the number of times they have competed against each other. If a connection exists between two pilots (vertices), it means there is an instance where the pilots have competed against each other, thus creating an edge between them. Shorter distances on these edges between the pilots' nodes indicate more competitiveness, or a higher count of instances where two given pilots have competed against each other. The output provides connections between a handful of the nodes, along with the weight of the connection, which will be explained later on in this write-up.

Data Set Description

The data set used here builds off the combination of several data sets. While I attempted to restrict my data only to Formula 1 races, it would not be possible to adequately complete the

project with this data set alone, as it would only allow for approximately 850 unique vertices, with each vertex indicating a different pilot that has competed in a Formula 1 race. As such, I opted to combine several datasets on Formula 1, Formula 2, and Formula E. This allows for greater comparison between pilots, especially because many competitors in Formula 2 and Formula E have their sights set on Formula 1, meaning there is high interaction for these individuals between both sects of the racing world. The sets, provided by Kaggle, were somewhat corrupted and formatted differently among sects. As such, I took the liberty of cleaning the data outside of Rust and instead in the .csv files directly. After fixing the corruption (most of which was caused by accented characters in driver names not being written properly), I combined all of the data into one file, named *results.csv.* This dataset includes the following columns:

- result_id: This indexing column simply indicates the row of the data set.
- race_id: Each individual race (for example, the 2023 Belgian Grand Prix) is granted a numeric identifier. F1 races are indicated by raceId numbers between 1 and 1999, F2 races between 2000 and 2999, and FE races above 3000.
- driver_Id: This number indicates a unique pilot.
- driver_ref: This string, pegged to driverId, shows the name of that pilot.

In total, this dataset contained 1004 unique pilots, which would become the vertices in our graph.

Repository Structure

Aside from the *main.rs* file, the repository contains a few other script files that house modules that will be used by the main() function. These were separated into sections to ensure that the program can be updated in pieces and, should further improvements be made, can accommodate future optimization.

Models module (models.rs):

The Models module builds a public struct that allows for the main function to import the CSV file and name the fields accordingly. raceId, which was helpful when cleaning the data and when indexing data for other reasons, is discarded. The other fields are classified as the default *usize* type, with the *driver_ref* being classified as a string. These will be used later on.

Graph module (graph.rs):

The graph module builds the graph that we will use for analysis. It begins by building the struct for the graph, making it a String. It creates a blank new graph and makes its first node with the *new()* and *add_pilot()* functions. It then adds nodes for the remaining pilots and, using the *add_race_edge()* function, connects the nodes if those two particular pilots share at least one common *race_Id* value. As the algorithm finds more and more common *race_Id* values, it begins to weigh certain nodes more than others based on the number of connections they have with other pilots and the number of times two particular nodes have competed against each other. Then, distances are calculated using Dijkstra's Algorithm, which allows for certain pathways to be preferred over others based on edge weight. This helps us determine which connections are the closest, or the most weighted.

Analysis module (analysis.rs):

The Analysis module helps us to better calculate the average distance between nodes by way of a Dijkstra algorithm, similar to the *graph* module. The average distance function, *avg_dist,* starts by setting the total distance and number of pairs at zero, which are built up as the algorithm finds more pairings and adds more steps to the graph, breaking from the function when the analysis is complete. The Dijkstra algorithm, unlike the average distance function, allows for connections to be weighted and allows for further analysis of “competitiveness”

among the drivers.

Main (main.rs):

The main module also contains the struct that defines the parts of the data set. Next, it includes a few tests to ensure the code's functionality. The first test ensures that the code can add in a new pilot to the data set and -- given the additional pilot -- adjusts the graph size and its node count accordingly. Should another developer seek to combine another dataset with the given one through Rust, this test gives them confidence that the new data will be incorporated. The second test checks the average distance calculation by adding more nodes to the graph using the previously used *add_pilot()* function and then ensures the new edges made by *add_race_edge()* are accounted for in the average distance calculation. *test_dijkstra()* does the same thing. These tests, like the first, ensure that the algorithm can account for new data as it is implemented. The main function then does the following. We open the CSV file and then begin counting the shared *race_ID* values between pilots. It then uses an iterator function to add connections between nodes. It then prints the graph by converting node indices to be of type u32, and then finds the weight between two given nodes. After the code prints the graph itself, it prints the DOT output and shows the list of pilots and their associated indices.

Running the code and output

By running cargo build and then cargo run, the output (a sample of which is attached on the next page) includes the graph itself, which includes the weight of the connections between nodes on the graph. This allows you to see which connections are weighted the most (i.e. those that have the highest connections, indicating pilots that have competed with each other a lot). It also includes a DOT graph, which shows the index of node labels, filled by the *driver_Ref* data. By looking at which two indices have the highest number of connections, you can find out which two pilots have competed with each other the most in this data set.

Reflection

This project was incredibly difficult, but I believe I learned a lot about how Rust operates syntactically and structurally. This project took an extremely long amount of time (I would guess around 25 or so hours from the start of downloading and cleaning to submission), but it was somewhat beneficial to teaching the intricacies of such a program and the importance of perseverance when projects don't proceed in the way that you expect. If I were to redo this project, I would like to have created a way to sort the graph and better see which nodes had the highest weight; however, I did not find a way to implement this system.

Sample output of the graph:

924 -> 109 [weight: 2]

750 -> 717 [weight: 2]

951 -> 485 [weight: 2]

38 -> 889 [weight: 6]

375 -> 834 [weight: 20]

458 -> 604 [weight: 2]

470 -> 25 [weight: 2]

647 -> 617 [weight: 6]

263 -> 312 [weight: 2]

300 -> 238 [weight: 8]

898 -> 636 [weight: 2]

992 -> 464 [weight: 2]

397 -> 424 [weight: 2]

283 -> 316 [weight: 8]

Sample DOT Output:

0 [label = "\"jerry_hoyt\""]

1 [label = "\"basil_van_rooyen\""]

2 [label = "\"emilio_de_villota\""]

3 [label = "\"jordan_king\""]

4 [label = "\"mark_donohue\""]

5 [label = "\"mike_parkes\""]

6 [label = "\"brian_hart\""]

7 [label = "\"keith_greene\""]

8 [label = "\"carlo_franchi\""]

9 [label = "\"luigi_piotti\""]

10 [label = "\"patrick_friesacher\""]

11 [label = "\"robert_kubica\""]

12 [label = "\"scott_dixon\""]

13 [label = "\"sebastien_buemi\""]

14 [label = "\"alessandro_pesenti-rossi\""]

15 [label = "\"michael_schumacher\""]

16 [label = "\"francois_hesnault\""]

17 [label = "\"lirim_zendeli\""]

18 [label = "\"felice_bonetto\""]

19 [label = "\"raymond_sommer\""]