Open Data Project

Anass Benali
FIB - UPC Student
Barcelona, Spain
anass.benali@est.fib.upc.edu

Oriol Borrell
FIB - UPC Student
Barcelona, Spain
oriol.borrell.roig@est.fib.upc.edu

Giovanni Mauro
FIB - UPC Student
Barcelona, Spain
giovanni.mauro@est.fib.upc.edu

June 18, 2020

1 Business idea and identification of the data sources

1.1 Base purpose statement

In this project we will focus on the soccer player exchanges. We have found two datasets that contain information about the player transfers all over the world. As we will explain in following sections, we have integrated both datasets in order to create a property graph and exploit it. The graph will be exploited with queries, graph algorithms and a recommender will be built. The recommenender will be focused on looking for bargain transfers to find promising young players for a club to buy. This document pretends to explain the project in a more theoretical way, while the proof of concept can be found at the public repository: `https://github.com/oriolborrellroig/OD-FinalProject`

1.2 Relevant data of each source

The first dataset[1] was extracted from a Neo4j repository. The dataset contains transfers of players from 2009 to 2018 and contains the following atributes: season, *playerUri, playerName, playerPosition, playerAge, sellerClubUri, sellerClubName, sellerClubCountry, buyerClubUri, buyerClubName, buyerClubCountry, transferUri, transferFee, playerImage, playerNationality* and *timestamp*. The *transfer_fee* arrtibute stores the price of the transfer and the type of transfer (Loan or Purchase), so the type of transfer can be inferred from this attribute.

The second dataset[2] we found contains all the football player transactions between teams, from season 2000 to 2019. The dataset was found in the Kaggle website. The dataset has the following attribute: *name, position, age, team_from, league_from, team_to, league_to, season, market_value* and *transfer_fee*.

In the integration phase needed to join both datasets, we decided to keep the *ID, Name, Team_from, Team_to, Position, Age, League_from, League_to, Season* and *Transfer_fee*, information present in both datasets, so we will need to ensure that two values are the same for the same instance, or deal with those type of problems. We also wanted to keep the *playerImage, playerNationality* and the inferred attribute *type* that are only present in the first dataset, and the *market_value*, witch is only contained in the second one.

[1] First dataset: `https://s3-eu-west-1.amazonaws.com/football-transfers.neo4j.com/transfers-all.csv`

[2] Second dataset: `https://www.kaggle.com/vardan95ghazaryan/top-250-football-transfers-from-2000-to-2018`

1.3 Graph family justification

We want to exploit the data, specially the relationships. Moreover, we want to use graph algorithms to find interesting characteristics in the data. Hence, property graphs are fit for this type of graph analysis. Furthermore, since we are not specially interested in the data sharing (given competitiveness nature of the sport) we think that knowledge graphs are not really suitable in this case.

2 Creation of the integrated repository

2.1 Design of the integration schema

As we mentioned previously, the attributes that we have stored are the following: *ID, Name, Team_from, Team_to, Position, Age, League_from,League_to, Season, Transfer_fee, type, playerImage, playerNationality* and *market_value.*

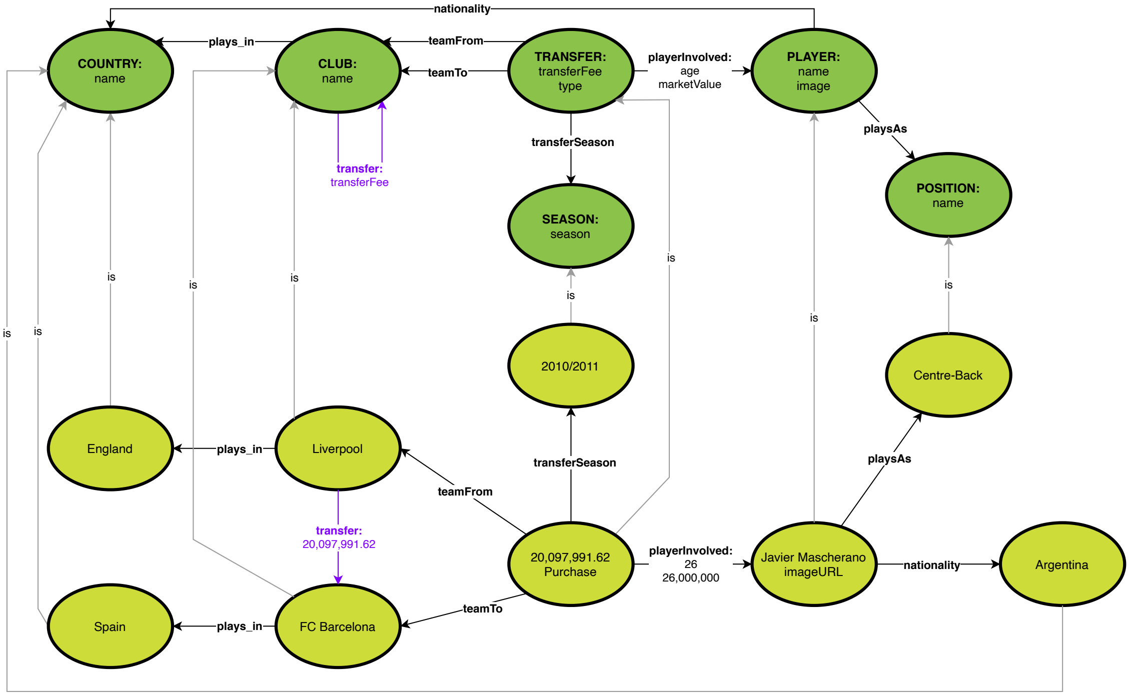


Figure 1: Target schema, Integration graph

We decided to store the mentioned information as it is shown in Figure 1. There are some modeling decisions that we want to mention:

- *Position* is stored in a different node, an not as a attribute of *Player*. This is because we expect to have in the future players that can play in multiple positions.
- *Season* is stored in a different node and not as a attribute of *Transfer* because we think is a node that can be used when performing queries over the graph. The same happens with *Country* & *Player*, and *League* & *Club*.

- We created a node *Transfer* that contains all types of transfers, and its type is stored in the attribute *type.* We don't expect the queries to differ from different types, and if we create different nodes, we can loose centrality.
- We added the *transfer* edge from the origin to the destiny *Club* in order to execute some Community detection or Centrality algorithms. This edge will be derived from current information present in the model. In this edge we will only store the weight (transferFee) as for the moment is the only information needed.

2.2 Design of data flows

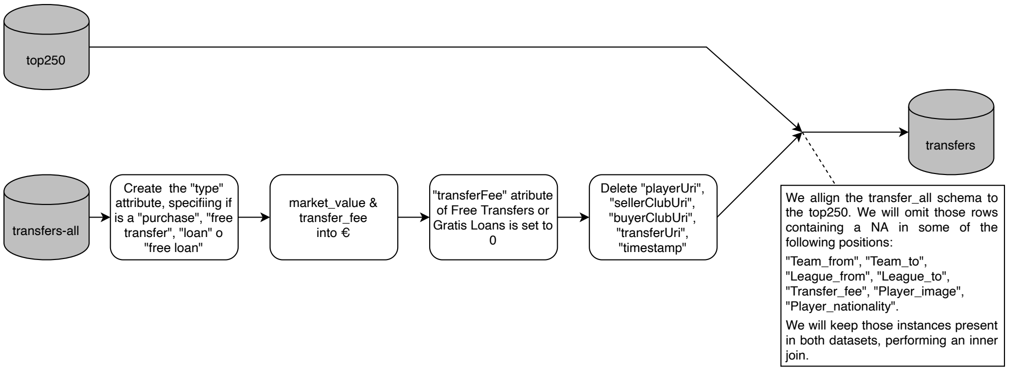
In this section we will describe the steps applied in order to integrate both data sources. There are three major steps that we will describe: The pre-process applied in the *top250* dataset, the pre-process applied in the *transfers-all* dataset and how we integrated both results in order to generate the new one.

The pre-process applied in the *top250* is the easiest one: We analyzed how the values are stored, and in the big majority of attributes the data is as we expected, so we did not perform any change. However, we realized that we don't want to store the name of the league but the country were the team plays. We will obtain this information from the other data set.

Regarding the *transfers-all* dataset, there are more steps to take into account: First of all we extracted the *type* attribute from the *transfer_fee* one, leaving tho this last attribute only the number cost of the transfer. Afterwords we converted the value from pounds to euros. Finally, we deleted the columns that we do not want to store in order to prepare the *transfers-all* dataset to perform the merge.

Once pre-processed both data sources, we omitted in both datasets the rows containing a NA value in some of the following attributes: *Team_from, Team_to, League_from, League_to, Transfer_fee, Player_image* and *Player_nationality.* With the remaining rows we performed a inner join, so we will keep the rows that exist in both datasets. We defined as primary key the union of the following attributes: *Name, Team_from, Team_to* and *Season.* We will store the resulting data in a dataset called *transfers.*

Figure 2 describes a summary of the integration process applied to each data source, and how both datasets are joined.


top250
transfers-all
Create the "type" attribute, specifiing if is a "purchase", "free transfer", "loan" o "free loan"
market_value & transfer_fee into €
"transferFee" atribute of Free Transfers or Gratis Loans is set to 0
Delete "playerUri", "sellerClubUri", "buyerClubUri", "transferUri", "timestamp"
transfers

We allign the transfer_all schema to the top250. We will omit those rows containing a NA in some of the following positions:

"Team_from", "Team_to", "League_from", "League_to", "Transfer_fee", "Player_image", "Player_nationality".

We will keep those instances present in both datasets, performing an inner join.

Figure 2: Integration data flow

2.3 Metadata needed in order to automate the exploitation process

From the target schema a new edge is derived which is labeled `transfer`. This new edge will be materialized on the graph in order to apply graph algorithms over it. The edge is reflexive, going from a club to another club. The origin node of the edge is the club from which the player was leaving and the destination node is the club to where the player was transferred. This relationship can be seen in purple in the target schema on 1. The edge has as attribute the *transferFee*, that can be used as a weight when performing some algorithms. Optionally, this could be extended so the weight is a custom metric which could take into account for example the transfer fee, the market value, a ratio of the aforementioned, or user feedback. The adaptive weight would allow to enrich the results of the graph algorithms.

3 Exploitation idea. Goals and algorithms

Once we had the model of of our data we want to exploit it to obtain knowledge from it. In the following sections we will describe the Queries and Algorithms applied, and the result obtained and the knowledge that we can extract from them. If you want to see the running version of the algorithm calls and of the query application, you can look at the GitHub `https://github.com/oriolborrellroig/OD-FinalProject`. In this repository, you can find the results too.

3.1 Queries

Here there are some interesting queries for our analysis.

- **Q1-Top Bargains:** We gave the best bargains on our graph. We understand a bargain as a transfer of a player at a lower price than its market value. To compare different bargains, we will compute $\frac{transferFee}{marketValue}$. The smaller the result, the better is the bargain. We discard the free transfers.

- **Q2-Top3 buyer clubs per season:** We selected the 3 clubs that performed more transfers, and printed the number of transfers. The result is given grouped by season. This query can be easily transformed to another interesting query.

- **Q3-Top transferred players:** We printed the players that were transferred more times.

- **Q4-Most expensive positions:** We for each position we computed it's average cost, in order to see witch positions are the more expensive ones.

3.2 Graph Algorithms

The graph algorithms will be executed over the *Club* nodes and the reflexive edge *transfer*. This is the edge represented in purple in the target schema on Figure 1. Before starting to applying our algorithms, we decide to focus our analysis over the transfer in general and over the selling actions. Thus, we consider or *both* edge directions or *out* edge direction. Specifically, we only use out direction when calculating centrality measures (betweenness and closeness), since it made more

sense to us to understand paths following only one direction (i.e. the flow of selling actions). We use *both* directions in all the other cases, since we focus our attention (and the datasets are focused over it too) over the act of a transferring a player.
For the remaining parameters of the graph algorithms the default values are enough. For instance, for PageRank the damping factor will be 0.85 and with 20 maximum iterations.
We applied both centrality measures and algorithms and community discovery algorithms, in order to capture various aspect of the graph and exploit and understand its relationships.

3.2.1 Centrality algorithms

- **A1-Page Rank:** With PageRank algorithm we are able to detect the most important clubs in the graph, in terms of transfers with other important clubs.

- **A2-Betweenness Centrality:** We can see betweenness as a measure of how many shortest paths pass through the node in analysis (i.e., target node). The more is this measure high, the more the influence of this node in the flow information. On our graph, we can understand which nodes serve as bridge for connecting clubs to clubs in transferring players.

- **A3-Closeness Centrality:** Closeness is a measure about how much the target node is closer (i.e., directly connected) to all other nodes. This way we are able to detect nodes that are able to spread information very efficiently through our graph. On our graph, we can understand how much clubs are directly connected to other clubs in their connected component.

3.2.2 Community detection algorithms

- **A4-Triangle Counting / Clustering Coefficient:** Triangle Counting is simply the number of triangles a node form with its neighborhood. We can exploit this information in the calculus of the clustering coefficient, in order to better characterize the embeddedness of a node into its neighborhood. Clustering Coefficient is the fraction of the neighbors of a node i that is connected together. It is used to detect clubs which form triads and cliques, and it measures their local cohesiveness.

- **A5-Connected Components:** A Connected Component is a graph in which any two vertices can be joined by a path. In general, and also in our case, we have mainly only one giant connected component and other small components composed by few nodes, that is, a huge subgraph of connected teams and three small other ones.

- **A6-Strongly Connected Components:** In the specific case of directed graphs, a strongly connected component is a graph in which we have a path from each node to every node and vice versa.

- **A7-Louvain modularity:** Louvain is a bottom-up and hierarchical approach based on modularity optimization. Modularity measures the strength of the division of a graph into set of well-separated clusters or modules, and it is calculated as the sum of the differences between the fraction of edges that actually fall within a given community and the expected fraction if edges were randomly distributed. The communities (clusters) are obtained by considering the best improvement in terms of modularity. In our dataset, clusters indicate clubs which densely transfer to each other while poorly transfer with clubs in other communities.

3.3 Recommender system

Finally have built our recommender system. Buying football players is a difficult and expensive choice that's why we proposed a recommender in order to try to facilitate the decision.

Our recommender will be based on the ratio between the transfer price and the real value of the player in the market, lets call it price ratio. We will suppose that the present is in 2018, the last year we have information about. The system will recommend those players whose price ratio is less than 1, their age is smaller than x years old, and the transaction was made within the last 3 years. We propose to set the x to 22, as we will obtain players that are already young in the present. This recommender can be built with a query that uses *Player, Transfer* and *Season* nodes (and the corresponding edges).

Using this recommender we expect to find players that in the year of the transfer were starting to became famous. We will also find players that are too young and inexperienced to play in big clubs, and that have been loaned to smaller teams in order to gain playing experience.

4 Comments about the Proof of Concept

We provide a solid proof of the concept explained in the GitHub repository above linked. As we expected, when executing *PageRank*, we pick out, as hubs, european top teams that are not used to grow their player since they are young, like Chelsea, Manchester United, Manchester City, Juventus, Liverpool etc. As we expected, similar results are obtained for the *beetweennes* centrality query, while we denote slightly different results in the *closeness* centrality results. In fact, we have a value of 1 for the clubs that have been discovered to belong to the same connected subgraph as the one discovered by the *strongly connected components* query. The other *top-closeness* teams are mostly the same as above, but this time with the presence of teams like Atlético Madrid, Valencia or Barcelona that are very used to send their player on loan to other minor teams for making them grow and then rebuy them immediately, creating very short paths. *Louvain* algorithm gives us 12 communities, corresponding, more or less, to club belonging to the same country, mixed with some small random noise or foreign clubs that are used to buy from the specified country. The obtained modularity is 0.36, so we are quite satisfied with these results. Finally, applying the clustering techniques, we see that small clubs tend to be very embedded in triangles with their neighbours, since they are used to exchange players in a very inner circle. A quite high score triangle coefficient score, is obtained by the kind of big clubs like Bayern München, who are used to make huge exchanges with his kind of satellite teams.