Big Data Analytics

Attività 4 – 21/12/2020

Graph Analytics – Football Transfers

Pietro Orlandi – matricola 161052

Indice

Dati

Schema iniziale

Inizialmente sono state eseguite tutte le query mostrate usando il comando Cypher *:PLAY football_transfers*. È stato quindi ottenuto lo schema mostrato in figura 1.1.

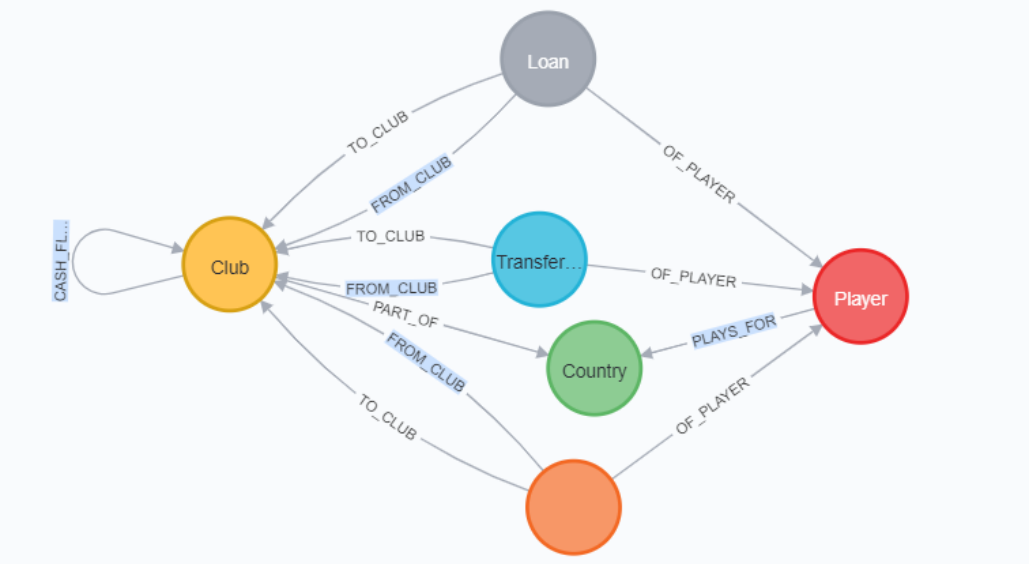

Figura 1.1

L'analisi verterà sui club e sui *CASH_FLOW* che avranno tra loro. Inizialmente, nel dataset offerto utilizzando tutti i comandi descritti usando :*PLAY football_transfers*, avremo 26084 club collegati da 25638 relazioni *CASH_FLOW*.
Per prima cosa faccio un'analisi esplorativa e guardo quali sono le prime e le ultime date dei trasferimenti a nostra disposzione.

```
MATCH (t:Transfer)
RETURN t.fee,t.season,t.timestamp
ORDER BY t.timestamp DESC
LIMIT 50
```

Come si nota dai risultati il dataset è aggiornato alla stagione *2017/2018,* più precisamente, possiamo vedere che il timestamp corrisponde al giorno *28/8/2017, 01:00:00*. E ci sono solamente i trasferimenti dal *1 gennaio 2010*. Tutte le analisi fatte da qui in avanti rappresenteranno la situazione dopo il *2010.* I trasferimenti effetuati prima di tale data, non vengono quindi presi in considerazione

Modifiche ai dati

Decido di creare una nuova relazione *TRANSFERED_IN*, in modo da fare riferimento direttamente al trasferimento di un player a un club. Questa relazione mi serve per accedere in modo più semplice ai dati quando successivamente farò delle analisi ad esempio per vedere quali sono i calciatori che si sono trasferiti in un determinato club e quanto il loro trasferimento abbia influenzato il flusso di cassa di un club.

```
MATCH (c:Club)<-[:TO_CLUB]-(t:Transfer)-[:OF_PLAYER]->(p)
MERGE (p)-[r:TRANSFERED_IN {fee:t.numericFee, season:t.season}]->(c)
```

Per prima cosa iniziamo a analizzare possibili anomalie nei dati.

Analizzando i trasferimenti fatti dalla squadra *Juventus* ad esempio possiamo notare che esistono due squadre di nome *Juventus*. Esse potrebbero essere solamente omonimi e rappresentare squadre diverse, tuttavia analizzando i dati su Transfermark, si può notare come la squadra sia la stessa e i trasferimenti rappresentino quindi la stessa squadra. La differenza principale in questo caso è il fatto che la property *id* in un caso rappresenti il valore ***/juventus-turin****/startseite/verein/506*, mentre in un altro caso il valore ***/juventus****/startseite/verein/506.*

Tenendo i dati in questo modo, essi influenzerebbero il risultato dei vari algoritmi che applicheremo sul grafo. Nel caso specifico della *Juventus* notiamo che tutti i trasferimenti di un determinato nodo si riferiscono a season successive al 2016/2017. Si è notato che questo fenomeno succede spesso, quindi si è scelto, di unificare le squadre e i relativi trasferimenti secondo la seguente logica: se esistono più di due squadre che appartengono allo stesso paese esse vengono unificate in una sola squadra.

Questo approccio da una parte ci consente di unificare l'informazione, ma dall'altro ci potrebbe far commettere un errore sui dati: potrebbero esistere squadre che anche essendo dello stesso paese e avendo lo stesso nome rappresentano istanze diverse. Tuttavia, cercando di campionare le squadre con la coppia di name e country più frequenti come mostrato dalla query in figura 1.2 non si è trovata nessuna squadra che avesse stesso paese e stesso nome ma rappresentasse squadre diverse.

```
MATCH (n:Club)-[:PART_OF]-(country:Country)
WITH n.name as club_name,country.name as
country_name,count(*) as freq
WHERE freq>1
RETURN club_name,country_name,freq
ORDER By freq DESC
```

Figura 1.2

Unifico quindi i nodi con questa query:

```
MATCH (c:Club)-[:PART_OF]-(country:Country)
WITH c.name AS club, country.name as country,collect(c) as
list_nodes,count(*) as freq
WHERE freq > 1
CALL apoc.refactor.mergeNodes(list_nodes,{mergeRels:true}) YIELD node
RETURN node
```

Con questa modifica, passo da avere un grafo composta da 26084 club e 25638 relazioni a un grafo composto da 20692 club e 25594 relazioni.

Graph-Analytics

Obiettivi

Gli obiettivi principali di questa analisi sono di trovare i club più importanti nel calciomercato, quelli che hanno mosso in generale più soldi, quelli che hanno generato più utili e quelli che hanno generato più perdite. Inoltre, un obiettivo di questa analisi è quello di analizzare se esistono delle comunità tra i club.

Nell'ultima fase verrà invece analizzato anche il cambiamento nel tempo dei vari club, analizzando la variazione dell'importanza di essi nella rete, quindi magari riuscendo a capire eventuali fallimenti di determinati club.

Scelta delle label e delle relationship

È stato scelto di svolgere un'analisi sui club collegati tramite la relazione *CASH_FLOW*. Questa relazione è contraddistinta dalle property *total*, che rappresenta il totale che un club A trasferisce a un club B, dalla property *season*, che contraddistingue la stagione presa in considerazione, e dalla property *player_count*, che rappresenta il numero di giocatori trasferiti dalla squadra A alla squadra B in quella season. Da sottolineare il fatto che la relazione *CASH_FLOW* è una relazione orientata: in una stessa stagione posso avere una relazione *CASH_FLOW* da A verso B e da B verso A, in quanto rappresentano due flussi di cassa diversi.

Creazione in-memory graph

Decido di creare l'in-memory graph che tiene conto dei tipi di nodo *Club*, della relazione *CASH_FLOW* con le relative proprietà. Siccome in questo caso non ho esigenze particolari, posso utilizzare una native projection, essendo quest'ultima più performante rispetto alla Cypher projection. In questo caso scelgo di rappresentare la relazione *CASH_FLOW* come orientata, mettendo l'attributo orientation a *NATURAL*.

```
CALL gds.graph.create('my-graph','Club',
{CASH_FLOW:{
type:'CASH_FLOW',orientation:'NATURAL',
properties:{playerCount:
{property:'playerCount'},total:{property:'total'}
}
}}) YIELD graphName,nodeCount,relationshipCount
```

	graphName	nodeCount	relationshipCount
1	"my-graph"	20692	12797

Figura 2.1

Viene quindi creato un in-memory graph di 20692 nodi di tipo *Club* e 12797 relazioni *CASH_FLOW* che li collegano.

Statistiche sugli archi e i nodi

Essendo un grafo orientato, faccio statistiche sia su *in-degree* che su *out-degree*.

IN-DEGREE

```
MATCH (c1:Club)
WITH c1.name as club,size((c1)<-[:CASH_FLOW]-(:Club)) as in_degree
RETURN min(in_degree),max(in_degree),avg(in_degree),stdev(in_degree)
```

	min(in_degree)	max(in_degree)	avg(in_degree)	stdev(in_degree)
1	0	75	0.6184515754881126	3.407246888749746

Figura 2.2

Utilizzando questa query posso vedere le varie statistiche sull'in-degree dei nodi, ovvero sul numero degli archi entranti di ciascun nodo. Facendo un'analisi ulteriore possiamo vedere che il club con il numero di *in-degree* massimo (75) è la *Juventus.*

OUT-DEGREE

```
MATCH (c1:Club)
WITH c1.name as club,size((c1)-[:CASH_FLOW]->(:Club)) as out_degree
RETURN min(out_degree),max(out_degree),avg(out_degree),stdev(out_degree)
```

	min(out_degree)	max(out_degree)	avg(out_degree)	stdev(out_degree)
1	0	98	0.6184515754881112	4.162569684700053

Figura 2.3

Guardo quindi le statistiche sull'*out-degree* dei nodi, ovvero sul numero degli archi uscenti di ciascun nodo. Analizzando più accuratamente posso vedere come la squadra con più archi uscenti sia *AS Roma* con 98 archi, seguita a distanza dalla *Juventus* e altre squadre italiane con circa 80 archi.

Diametro del grafo tra il flussi di cassa dei Club

Utilizzando la seguente query calcolo il *diametro* del grafo, andando a usare *allShortestPath* e andando a selezionare quello con distanza maggiore. In figura 2.4 sono mostrati i risultati in ordine di distanza crescente.

```
CALL gds.alpha.allShortestPaths.stream('my-graph',{concurrency:4})
YIELD sourceNodeId, targetNodeId, distance
WHERE gds.util.isFinite(distance) = true
RETURN gds.util.asNode(sourceNodeId).name AS source,
gds.util.asNode(targetNodeId).name AS name, distance
ORDER BY distance DESC
```

	source	name	distance
1	"Miedz Legnica"	"QFC"	13.0
2	"Miedz Legnica"	"Dingli Swallows"	13.0
3	"Dubai CSC"	"QFC"	12.0
4	"Dubai CSC"	"Yifang Reserves"	12.0
5	"Dubai CSC"	"Al-Arabi SC"	12.0
6	"Dubai CSC"	"JX Liansheng"	12.0

Figura 2.4

Il diametro di questo grafo orientato è quindi 13, tuttavia avrebbe più senso calcolare il diametro per il grafo non orientato, considerando il fatto che se due club hanno trattato tra loro non ci interessa il fatto della relazione orientata, ma il fatto che essi siano entrati in contatto.

Porre attenzione alla squadra Club Miedz Legnica, che ha degli shortest-path molto alti

Densità del grafo

Siccome il grafo è orientato andiamo a usare la seguente formula per calcolare la densità:

$$Density = \frac{TotalEdges}{TotalPossibleEdged} = \frac{|E|}{|V|(|V|-1)}$$

Considerando che il numero di nodi nella rete sono 20692, utilizzo la seguente query per calcolare la densità del grafo orientato.

```
MATCH (c1:Club)-[r:CASH_FLOW]->(c2:Club)
RETURN (size(collect(id(r)))*1.0)/((20692)*(20692-1))
```

La densità del grafo è pari a *0.00002988988330617715*, quindi si può concludere che questo grafo è estremamente sparso.

Weakly Connected Component

Analizziamo ora le comunità presenti nel grafo. Utilizziamo l'algoritmo *Weakly Connected Components* in cui vengono estratte le comunità del grafo andando a vedere tutti i vertici che possono essere raggiunti da altri vertici attraverso un cammino considerando gli archi come non orientati.

Prima di tutto eseguo la seguente query per stimare la memoria che mi occorre per eseguire l'algoritmo.

```
CALL gds.wcc.write.estimate('my-graph', { writeProperty: 'wccComponentId' })
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory
```

Ottengo quindi il risultato mostrato in figura 2.5

	nodeCount	relationshipCount	bytesMin	bytesMax	requiredMemory
1	20692	12797	165664	165664	"161 KiB"

Figura 2.5

Non sussistono quindi problemi di memoria, quindi è possibile eseguire l'algoritmo in modalità *stream* con la seguente query.

```
CALL gds.wcc.stream('my-graph') YIELD componentId
RETURN (COUNT(DISTINCT componentId))
```

Risultano *18109 Weakly Connected Component*. Analizzando meglio usando la seguente query possiamo vedere statistiche su di esse come mostrato in figura 2.6

```
CALL gds.wcc.stats('my-graph')
YIELD componentCount,componentDistribution
```

Figura 2.6

Si può vedere infatti che mediamente in ogni componente siano presenti 1.14 nodi, mentre una componente riesce a contiene 2451 nodi su un totale di 20692, la media quindi viene profondamente influenzata dalla componente più grande.
Analizziamo più a fondo queste *WCC* andando a vedere per ogni componente il numero di nodi che ha usando la seguente query:

```
CALL gds.wcc.stream('my-graph') YIELD nodeId,componentId RETURN
componentId,COUNT(nodeId) as tot
ORDER BY tot DESC
```

Il risultato di tale query è il fatto che quasi la totalità delle componenti ha 1 solo nodo, una componente ha 2451 nodi e altre 100 componenti hanno dai 6 ai 2 nodi. Se andiamo ad analizzare la seconda componente più grande si può notare come siano tutti club appartenenti al settore giovanile delle squadre, con un cash flow molto basso.
Andiamo a scrivere direttamente sui nodi i *componentId* e etichettarli.

```
CALL gds.wcc.write('my-graph', { writeProperty: 'wccComponentId' })
YIELD nodePropertiesWritten, componentCount;
```

Utilizziamo il tool Neovis.js [1] per visualizzare il grafo, in modo che le diverse componenti trovate da *WCC* siano visualizzate in colore diverso. Il grafo descritto è mostrato in figura 2.7.

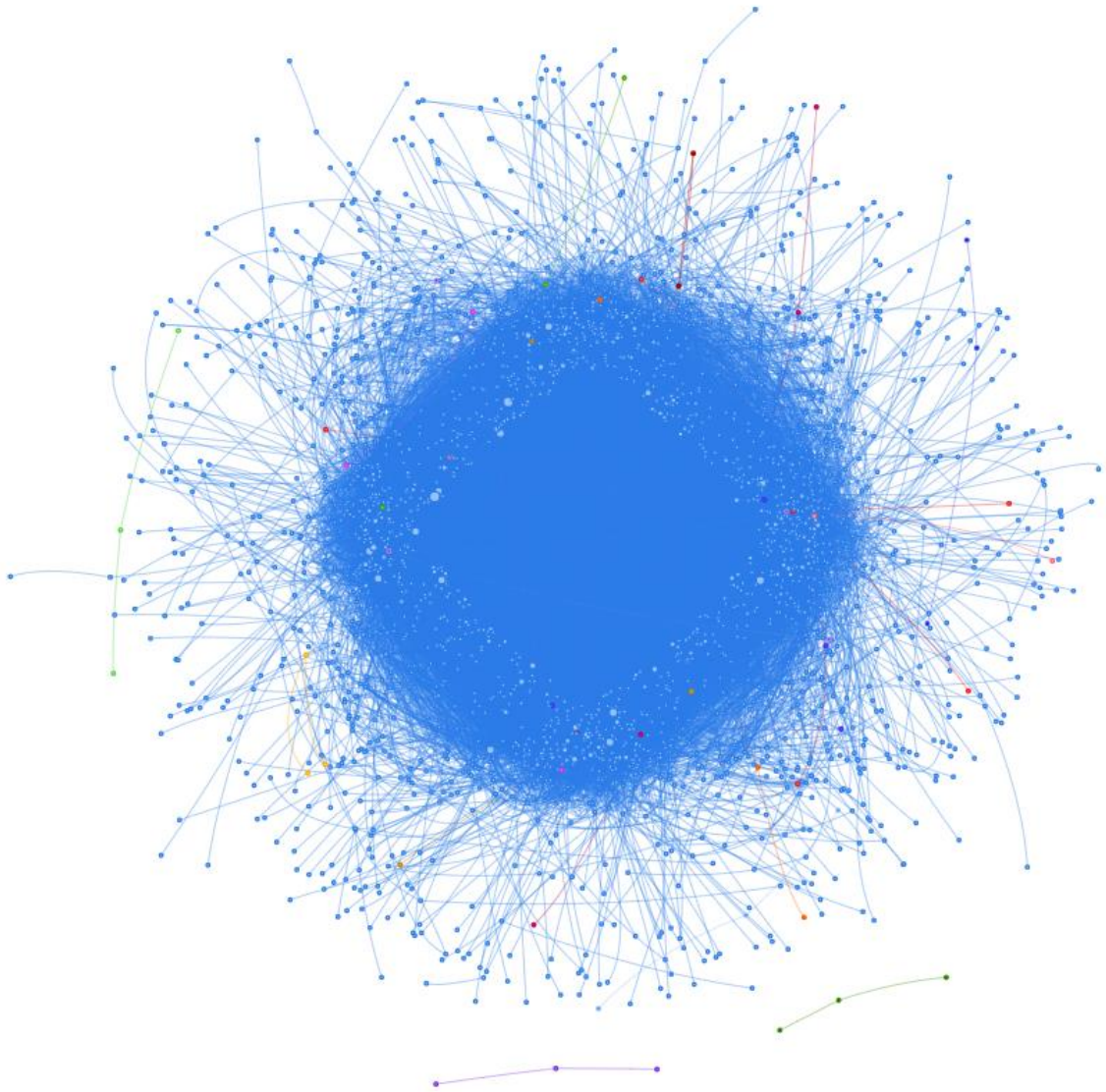

Figura 2.7

Nella figura 2.7 è possibile notare che oltre alla grossa componente principale mostrata in blu, siano presenti altre piccolissime componenti.

Utilizzando la seguente query andiamo a vedere il flusso di cassa presenti nelle componenti diversi da quella principale.

```
MATCH (c1:Club)
WITH c1.wccComponentId as componentId,count(*) as freq
ORDER BY freq DESC
WHERE componentId<>133
MATCH (c1:Club {wccComponentId:componentId})-[r:CASH_FLOW]-
(c2:Club {wccComponentId:componentId})
RETURN sum(r.total)
```

sum(r.total)
40625620.0

Figura 2.8

Il risultato di tale query, come si vede nella figura 2.8, è di circa *40 milioni di euro,* mentre se vediamo il flusso di cassa su tutta la componente principale è di *60 miliardi di euro*. Notiamo quindi che il flusso di cassa presente nelle varie componenti è molto inferiore rispetto a quello della componente più grande, in particolare esso è solamente il 0.06% rispetto al cash flow della componente principale.

Le analisi successive saranno eseguite solamente sulla componente più grande, essendo quella più collegata, quella in cui risiedono tutti i principali club, e quella con i maggiori flussi di cassa.

Analisi sulla componente più grande

In-Degree della componente più grande

```
MATCH (c1:Club {wccComponentId:133})
WITH c1.name as club,size((c1)<-[:CASH_FLOW]-(:Club
{wccComponentId:133})) as in_degree
RETURN min(in_degree),max(in_degree),avg(in_degree),stdev(in_degree)
```

min(in_degree)	max(in_degree)	avg(in_degree)	stdev(in_degree)
0	75	5.1660546715626205	8.632341081681176

Figura 2.9

Out-Degree della componente più grande

```
MATCH (c1:Club {wccComponentId:133})
WITH c1.name as club,size((c1)-[:CASH_FLOW]->(:Club
{wccComponentId:133})) as out_degree
RETURN min(out_degree),max(out_degree),avg(out_degree),stdev(out_degree)
```

min(out_degree)	max(out_degree)	avg(out_degree)	stdev(out_degree)
0	98	5.166054671562626	11.080916040904718

Figura 2.10

Se si confrontano le statistiche mostrate in figura 2.9 e 2.10 con quelle del grafo completo mostrate in figura 2.2. e 2.3, si può notare che in questa componente il grado dei nodi è molto più alto. Basta infatti vedere la media (qui di circa 5 sia per l'out-degree che per l'in-degree), mentre nel grafo completo era di circa 0.6.

Densità della componente più grande

Analizziamo la densità del nostro grafo dei club con componentId uguale a 133 (la componente più grande).

```
MATCH (c1:Club)-[r:CASH_FLOW]->(c2:Club)
WHERE c1.wccComponentId=133 AND c2.wccComponentId=133
RETURN (size(collect(id(r)))*1.0)/((2451)*(2451-1))
```

Densità di questa componente: *0.0021*

La componente 133 risulta ancora un grafo sparso.

Creo in-memory graph non orientato della componente 133

In questo caso utilizzo una Cypher projection perché una native projection non è abbastanza espressiva per fare al mio caso. In particolare, siccome voglio solamente i club della componente più grande, ovvero quella con *wccComponentId* a 133, devo selezionarli usando una query Cypher.
Carico anche le property *total* e *player_count* della relationship *CASH_FLOW* perché mi serviranno dopo. Scelgo di usare un in-memory graph perché mi servirà in vari algoritmi.

```
CALL gds.graph.create.cypher(
    'component133_u',
    'MATCH (n:Club {wccComponentId:133}) RETURN id(n) AS id',
    'MATCH (n:Club {wccComponentId:133})-[r:CASH_FLOW]-(m:Club
{wccComponentId:133}) RETURN id(n) AS source, id(m) AS
target,r.total as total,r.player_count as player_count,r.season as
season')
```

Creo in-memory graph orientato della componente 133

Come nel caso precedente, scelgo di usare un in-memory graph poiché esso verrà usato in vari algoritmi. Questo grafo differisce rispetto al precedente per il fatto che qui, essendo orientato, considererò anche la direzione del flusso di cassa, riuscendo ad esempio a cercare le squadre che hanno avuto più flussi di cassa entranti rispetto a quelli uscenti.

```
CALL gds.graph.create.cypher(
    'component133_o',
    'MATCH (n:Club {wccComponentId:133}) RETURN id(n) AS id',
    'MATCH (n:Club {wccComponentId:133})-[r:CASH_FLOW]->(m:Club
{wccComponentId:133}) RETURN id(n) AS source, id(m) AS
target,r.total as total,r.player_count as player_count,r.season as
season')
```

Coefficienti di clustering locale

Su questa componente voglio vedere i coefficienti di clustering locale per tutti i nodi. Questo dato ci dice la probabilità che dato un nodo *N*, i vicini di *N* siano anche tra di loro connessi.

Inoltre, per ciascun nodo vogliamo associarci anche il degree (ricordiamo che stiamo considerando un grafo *UNDIRECTED*), per analizzare dopo se c'è una relazione tra il coefficiente locale di clustering e il degree del nodo.

Utilizzando la modalità *estimate* otteniamo i risultati in figura 2.11, capendo quindi che non ci saranno problemi di memoria.

```
CALL gds.localClusteringCoefficient.write.estimate('component133', {
  writeProperty: 'localClusteringCoefficient'
})
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory
```

nodeCount	relationshipCount	bytesMin	bytesMax	requiredMemory
2451	25324	39408	39408	"38 KiB"

Figura 2.11

Utilizziamo poi la modalità *stream*.

```
CALL gds.localClusteringCoefficient.stream('component133')
YIELD nodeId, localClusteringCoefficient
WITH gds.util.asNode(nodeId) AS club, localClusteringCoefficient
RETURN club.name,size((club)-[:CASH_FLOW]-(:Club)) AS
degree,localClusteringCoefficient
ORDER BY localClusteringCoefficient DESC
```

Il *coefficiente di clustering locale medio è di 0.082*, che sta ad indicare che mediamente, preso un nodo, l'8.2% di tutte le coppie di club che hanno avuto transazioni col nodo hanno avuto transazioni anche tra di esse.

Notiamo che tutti i nodi con coefficiente locale di clustering alti hanno valori molto bassi degree.

Andiamo quindi a plottare la relazione tra coefficiente di clustering locale e grado dei nodi, ottenendo il risultato mostrato in figura 2.12.

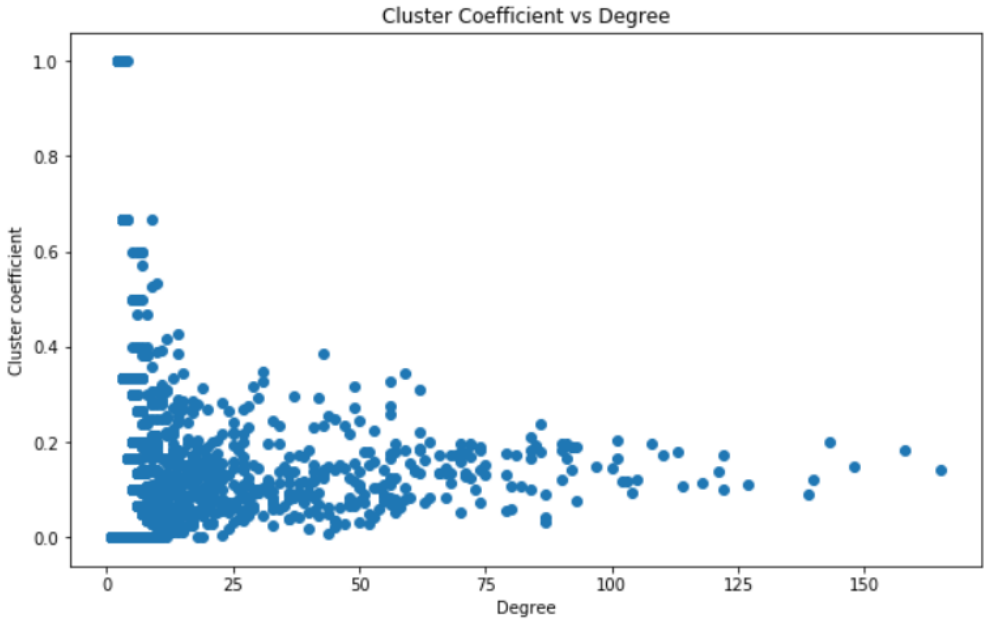

Figura 2.12 – Relazione tra coefficiente di clustering locale e il degree dei nodi

Dal grafico in figura 2.12 possiamo notare che i nodi con coefficiente di clustering più elevato sono i nodi con un degree molto basso. Si può vedere che i nodi con un degree maggiore di 20 non superano mai un coefficiente di clustering di 0.4.

Centralità

Degree Centrality pesandolo sul total

```
CALL gds.alpha.degree.stream('component133_u',{relationshipWeightProperty:
'total'})
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).name AS name, score AS degree_centrality
ORDER BY degree_centrality DESC
```

Utilizzo questo algoritmo sul grafo non orientato, misurando la *Degree Centrality* e pesandola in base alla property *total* di *CASH_FLOW*. L'esecuzione della query mi ritorna i risultati mostrati in figura 2.13.

	name	score
1	"Chelsea"	1391251000.0
2	"Man City"	1354451000.0
3	"Juventus"	1200029000.0
4	"FC Barcelona"	1173900000.0
5	"Man Utd"	1074398000.0
6	"Paris SG"	1000260000.0

Figura 2.13

Questi sono i club che hanno fatto girare più soldi nel mercato compreso tra il 2010 e il 2017, ovvero l'arco di tempo dei nostri dati.

Andiamo a scrivere il valore di degree centrality nel grafo attraverso la seguente query.

```
CALL gds.alpha.degree.write('component133_u',{relationshipWeightProperty:
'total',writeProperty: 'degree_centrality'})
YIELD nodes, createMillis, computeMillis, writeMillis, writeProperty
```

Attraverso il tool di visualizzazione *Neovis.js* [1] andiamo a vedere il grafo, con la dimensione dei nodi proporzionale al *degree centrality score* restituito dall'algoritmo e lo spessore dell'arco *CASH_FLOW* proporzionale alla property *total* del *CASH_FLOW* stesso.

Il grafo è stato generato dalla seguente query, che considera solamente gli archi dei club con degree centrality più alto al fine di una migliore visualizzazione.

```
MATCH (c1:Club)
WHERE NOT c1.degree_centrality is NULL
WITH c1
ORDER BY c1.degree_centrality DESC LIMIT 10
MATCH p=(c1)-[:CASH_FLOW]-(:Club)
RETURN p
```

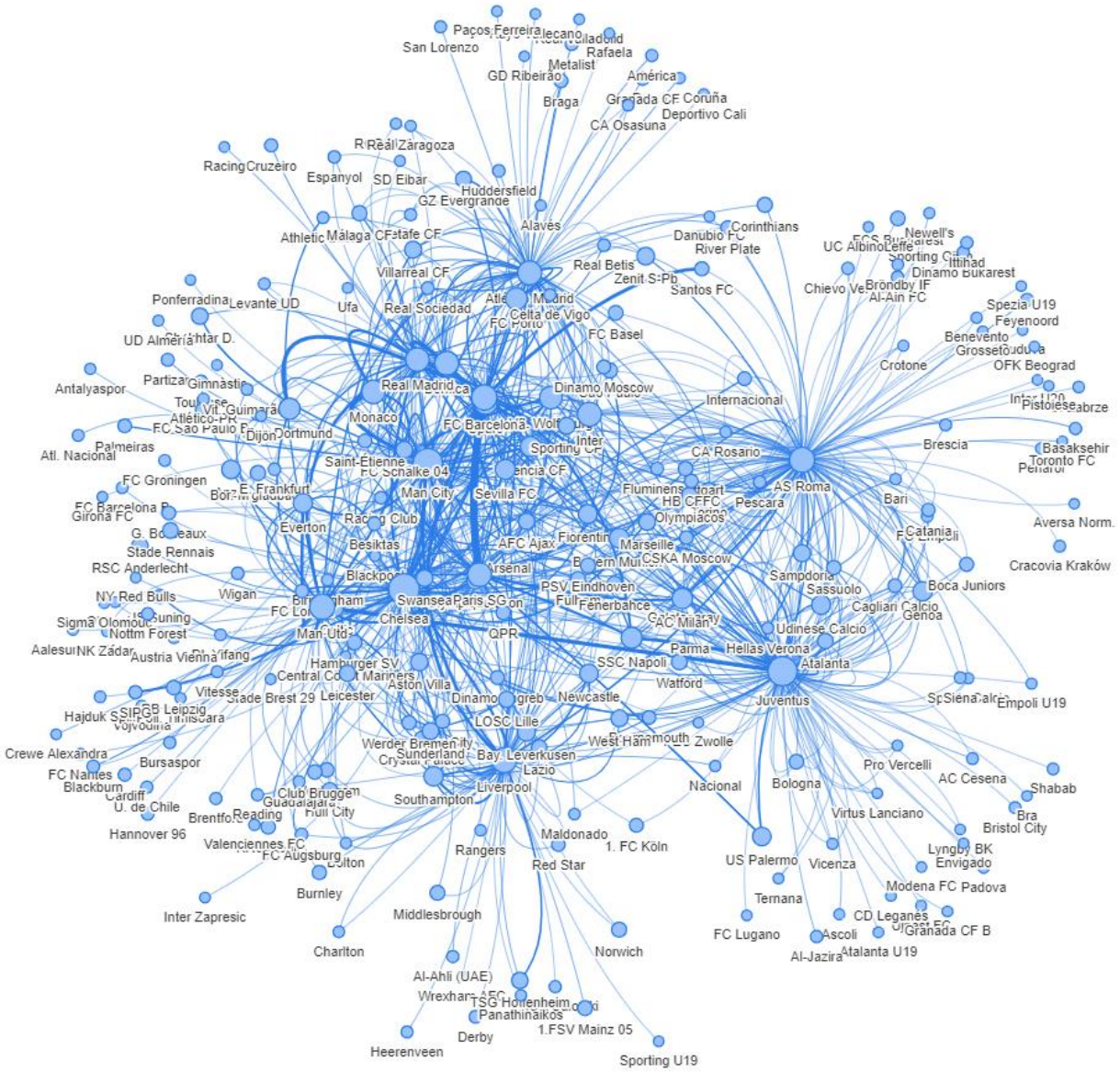

Figura 2.14 - Grafo con dimensione dei nodi proporzionale alla degree centrality (generato con Neovis.js)

Possiamo notare che in figura 2.14, al centro ci sono molti archi, e i nodi delle squadre sono mediamente più grandi rispetto all'esterno, ovvero sono presenti squadre con degree centrality score più alto. Queste squadre possono essere identificate come le squadre che hanno speso più soldi, e rispecchiano quelle che sono nella realtà le squadre più blasonate.

Analizzando più accuratamente, possiamo notare che ad esempio il *Man City* è la squadra che ha speso più soldi, seguita dal *Man Utd* e dal *Chelsea*, mentre quella che ne ha guadagnati di più è il *Benfica*, seguita dal *Chelsea*. In figura 2.13 si vede che alla posizione 6 è presente il *Paris SG*. Si può vedere come questo valore sia influenzato dal mercato dell'ultima stagione disponibile nei dati ovvero la *season 2017/2018* e in particolare dall'investimento più costoso nella storia del calcio, ovvero quello di Neymar dal Barcellona al Paris SG per circa 200 milioni di euro.

Proviamo quindi a rifare il calcolo della *degree centrality* pesata sul totale escludendo tutte le relazioni *CASH_FLOW* avvenute nella stagione *2017/2018*. Per fare ciò è stato usato un *anonymous graph* generato tramite Cyper projection.

```
CALL gds.alpha.degree.stream({nodeQuery:'MATCH (n:Club
{wccComponentId:133}) RETURN id(n) AS id',
                              relationshipQuery:'MATCH (n:Club
{wccComponentId:133})-[r:CASH_FLOW]-(m:Club {wccComponentId:133}) WHERE
NOT r.season="2017/2018"RETURN id(n) AS source, id(m) AS target,r.total
as total,r.player_count as
player_count',relationshipWeightProperty:'total'})
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).name AS name, score
ORDER BY score DESC
```

	name	score
1	"Chelsea"	1154731000.0
2	"Man City"	1080041000.0
3	"Juventus"	958829000.0
4	"Liverpool"	942028000.0
5	"Man Utd"	918788000.0
6	"Atlético Madrid"	873380000.0

Figura 2.15

Come si nota dalla figura 2.15, sia il *Barcellona*, sia il *Paris SG,* ovvero i due club che hanno realizzato il trasferimento più costoso della storia, non sono più presenti nella TOP 6 dei nodi con la maggior *degree centrality* se non consideriamo la stagione 2017/2018. Il *Paris SG* passa infatti all'undicesima posizione.

PageRank su grafo orientato pesandolo sul total

In questo caso utilizzo l'in-memory graph orientato che avevo creato in precedenza e denominato *component133_o*. In questo caso, essendo il grafo orientato e pesato in base alla property *total*, mi ritornerà le squadre che vengono puntate di più dagli altri club e con un più alto costo entrante della relazione *CASH_FLOW,* quindi ci ritornerebbe l'informazione delle squadre che hanno fatto molto utile in questi anni.
Per prima cosa controlliamo l'utilizzo di memoria necessario all'algoritmo, come mostrato in figura 2.16.

nodeCount	relationshipCount	bytesMin	bytesMax	requiredMemory
2451	12662	79832	79832	"77 KiB"

Figura 2.16

Una volta accertati, possiamo eseguire *PageRank* in modalità *stream.*

```
CALL gds.pageRank.stream('component133_o',{relationshipWeightProperty: 'total'})
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).name AS name, score
ORDER BY score DESC, name ASC
```

Il risultato ottenuto dall'esecuzione dell'algoritmo *PageRank* è quello mostrato in figura 2.17

	name	score
1	"Benfica"	3.715347731276415
2	"Juventus"	3.577822758385446
3	"Inter"	3.320029747113585
4	"Boca Juniors"	2.949361767782829
5	"CSKA Sofia"	2.858524476547609
6	"Udinese Calcio"	2.7974654783844017

Figura 2.17

Possiamo notare che i club più centrali in questo caso si differenziano molto rispetto ai club più centrali ottenuti utilizzando *degree centrality*. Questo fatto avviene sia perché l'algoritmo *PageRank* considera nel calcolo dello score anche la rilevanza dei vicini di un determinato nodo, sia perché qui abbiamo calcolato lo score in un grafo orientato, quindi avranno uno score più alto le squadre che hanno effettuato più utili.
Andiamo a fare un'analisi successiva, andando a etichettare i nodi con lo score restituito dall'algoritmo.

```
CALL gds.pageRank.write('component133',{relationshipWeightProperty:
'total',writeProperty: 'pagerank_component133'})
YIELD nodePropertiesWritten, ranIterations
```

Ora attraverso il tool di visualizzazione *Neovis.js* [1] andiamo a vedere il grafo, con la dimensione dei nodi proporzionale allo score restituito da *PageRank* e lo spessore dell'arco *CASH_FLOW* proporzionale alla property *total* del *CASH_FLOW* stesso.

In particolare, vengono rappresentati gli archi dei primi 15 club con *pagerank score* più elevato. Otteniamo quindi il grafo in figura 2.18.

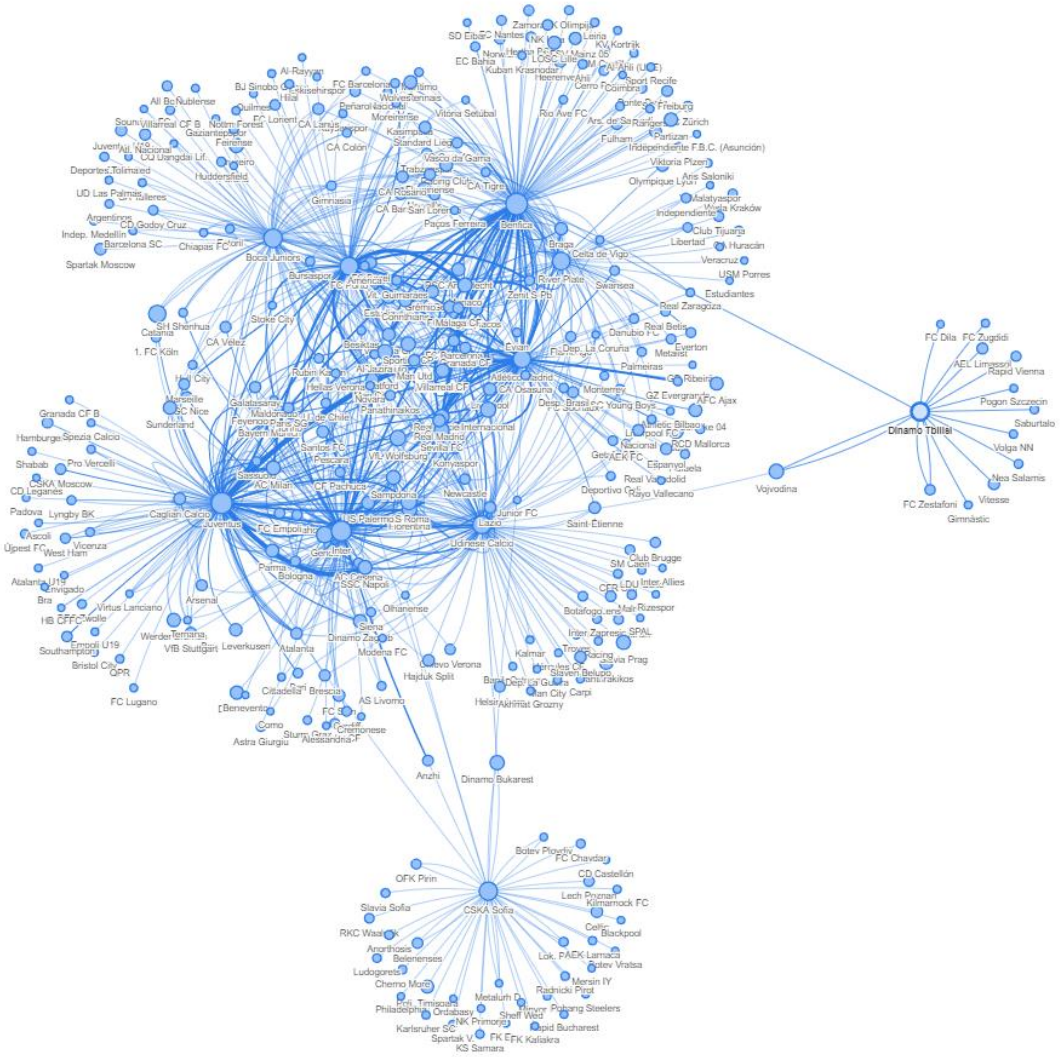

Figura 2.18 – Grafo con dimensioni dei nodi proporzionali al pagerank score (generato con Neovis.js)

Anche se il grafo è grande, quindi una sua visualizzazione con la label associata risulta complicata da leggere, ingrandendo l'immagine è possibile vedere il comportamento restituito dagli score mostrati in figura 2.17. Possiamo vedere infatti *'abbastanza bene'* i primi sei club con pagerank più alto. Notiamo ad esempio a destra che il club *Dinamo Tbilisi* sia abbastanza centrale. Con un'analisi più approfondita, si nota che molto del suo pagerank score sia determinato dal pagerank score del club *Vojvodina*. Esso infatti è puntato da molti club, tra cui il club *Udinese*, uno dei club col pagerank score più alto, e punta solamente a un nodo, ovvero il nodo *Dinamo Tbilisi*, quindi traferirà gran parte del suo pagerank score a quest'ultimo club.

Analizziamo invece il perché il *Benfica* sia la squadra col pagerank più alto. Ricordiamoci che l'algoritmo PageRank è stato pesato in base al valore totale del *CASH_FLOW* di una determinata stagione. Il valore dello score sarà più elevato quindi non solo se i nodi da cui si viene puntati hanno un pagerank score alto, ma inciderà nel calcolo anche il totale del CASH_FLOW in entrata e in uscita. Considerando quindi il grafo orientato, andranno ad avere in generale uno score più alto le squadre che hanno effettuato utili più grandi. Ad esempio notiamo che squadre come *Man City* e *Real Madrid*, che erano risultate molto importanti per la degree centrality, qui risultano rispettivamente alla posizione 49 e alla posizione 65, perché sono club che hanno speso molto, ma non guadagnato altrettanto. Si può quindi affermare che le squadre mostrate nella figura 2.17 siano quelle che abbiano avuto un buon utile, e probabilmente abbiano trattato con altre squadre che hanno effettuato un buon utile. Nel calcio quando sussistono fatti di questo genere spesso è perché la squadra decide di puntare su giocatori giovani, solitamente comprati a pochissimo, che grazie a una possibile crescita tecnica acquisteranno valore di mercato e verranno venduti a cifre esorbitanti.

Possiamo analizzare questo comportamento appunto nel *Benfica*. Controllando l'età a cui sono stati ceduti i dieci giocatori che sono stati venduti a di più da questo club, si può vedere che mediamente essi siano stati venduti a 22.6 anni di età a una cifra media di 30 milioni.

	p.name	r2.age	t.numericFee	t.season
1	"Axel Witsel"	"23"	36000000.0	"2012/2013"
2	"Ederson"	"23"	36000000.0	"2017/2018"
3	"Renato Sanches"	"18"	31500000.0	"2016/2017"
4	"Victor Lindelöf"	"22"	31500000.0	"2017/2018"
5	"Ángel Di María"	"22"	29700000.0	"2010/2011"
6	"Nélson Semedo"	"23"	27450000.0	"2017/2018"

Figura 2.19

PageRank orientato senza peso

Proviamo quindi ad eseguire PageRank senza utilizzare alcun peso, utilizzando l'in-memory graph orientato *component133_o*.

```
CALL gds.pageRank.stream('component133_o')
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).name AS name, score
ORDER BY score DESC, name ASC
```

L'esecuzione di tale query mi ritorna come primo risultato il *CSKA Sofia*. Molte squadre infatti puntano verso di lei, mentre essa punta verso pochissime squadre. Il fatto risulta un po' anomalo dato che il club necessita di un numero minimo di giocatori. Analizziamo quindi il grafo principale eseguendo questa query:

```
MATCH (c1:Club {name:'CSKA Sofia'})<-[r1:TO_CLUB]-(t:Transfer)-
[r2:FROM_CLUB]-(c2:Club)
RETURN c2.name,t.numericFee
ORDER BY t.numericFee DESC
```

Noto dai risultati che solamente 18 trasferimenti su 97 hanno valore diverso da 0, mentre tutti gli altri trasferimenti hanno importo zero. I trasferimenti di importo zero non generano flusso di cassa, quindi non sarà presente nemmeno la relazione *CASH_FLOW*. Ricapitolando, il *CSKA Sofia* risulta una squadra che compra molti giocatori a parametro zero.

PageRank orientato in modo inverso pesandolo su total

Come scritto negli obiettivi dell'attività, voglio vedere anche i club che hanno speso di più a discapito dei guadagni. Eseguo quindi questa query utilizzando un anonymous graph generato da questa Cypher projection in cui inverto i nodi target e source in modo da generare una relazione inversa.

```
CALL gds.pageRank.stream({nodeQuery:'MATCH (n:Club {wccComponentId:133}) RETURN id(n) AS
id',
    relationshipQuery:'MATCH (n:Club {wccComponentId:133})-[r:CASH_FLOW]->(m:Club
{wccComponentId:133}) RETURN id(m) AS source, id(n) AS target,r.total as
total',relationshipWeightProperty:'total'})
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).name AS name, score
ORDER BY score DESC, name ASC
```

I risultati della query precedente sono mostrati in figura 2.20.

	name	score
1	"Paris SG"	39.334230485558514
2	"Man City"	37.67340357154609
3	"Chelsea"	37.6065881088376
4	"Man Utd"	35.46737304627895
5	"FC Barcelona"	29.849232849478717
6	"Juventus"	28.49363184049725

Figura 2.20

Il *Paris SG* e il *Man City*, rispettivamente prima e seconda, in effetti sono state acquistate da due gruppi molto ricchi del Medio Oriente e questo ne ha permesso le grosse spese effettuate negli ultimi anni.

Betweeness Centrality

Usando la *Betweeness Centrality* mi calcolo per ogni nodo uno score in base agli *shortest path* che passano per tale nodo.

In questo caso si utilizza l'in-memory graph come non orientato, per rappresentare il fatto che due club (e quindi i dirigenti di due club) si conoscano e quindi probabilmente siano in buoni rapporti indifferentemente dal verso del flusso di cassa.

Prima di tutto andiamo a eseguire l'algoritmo in modalità *estimate*:

nodeCount	relationshipCount	bytesMin	bytesMax	requiredMemory
2451	25324	1314912	1314912	"1284 KiB"

Figura 2.21

Anche in questo caso, siccome non ci dà particolari problemi di memoria, possiamo eseguire l'algoritmo in modalità *stream*, ottenendo i risultati mostrati in figura 2.22.

```
CALL gds.betweenness.stream('component133_u')
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).name AS name, score
ORDER BY score DESC
```

	name	score
1	"US Palermo"	178693.6729659699
2	"Legia Warszawa"	175623.6110460943
3	"Sampdoria"	162970.43799034273
4	"Sporting CP"	155505.39651495757
5	"AS Roma"	152661.18843431797
6	"FCS Bucharest"	147919.858404513

Figura 2.22

Come si può notare questa misura di centralità, dato che non abbiamo tenuto conto del totale, non ci dà una misura dei club che hanno speso di più, ma di quanto siano centrali rispetto la topologia della rete. Infatti, essendo che questa misura produce uno score basato sul numero di shortest path che passano per un determinato nodo, ci darà l'informazione di quanto questi club siano *centrali* dal punto di vista della conoscenza tra i vari club.

Path-Finding

Provo ora a eseguire un algoritmo di *path-finding*. Possiamo ad esempio pensare a un dirigente di una piccola squadra, ad esempio il *Cittadella*, che vuole mettersi in contatto con il dirigente del Real Madrid, e per fare ciò voglia sfruttare la rete di affari del club.

Si decide di usare l'in-memory graph non orientato, in quanto un dirigente di un club *A* conosce un dirigente del club *B* indifferentemente dalla direzione del flusso di cassa. Andiamo quindi ad eseguire la seguente query:

```
MATCH (start:Club {name:'Cittadella'}), (end:Club {name: 'Man City'})
CALL gds.alpha.shortestPath.stream('component133_u',{startNode:start,endNode:
end})
YIELD nodeId, cost
RETURN gds.util.asNode(nodeId).name AS name, cost
```

Essa ci ritorna il seguente risultato:

Label

name	cost
"Cittadella"	0.0
"Inter"	1.0
"Man City"	2.0

Propagation

Concentro la mia analisi cercando di trovare ulteriori comunità presenti nella community principale su cui stiamo lavorando. Applico quindi l'algoritmo *Label Propagation* al grafo non orientato e divido la mia componente in ulteriori sotto-componenti.

Anche in questo caso, l'esecuzione tramite *estimate* non evidenzia criticità nella gestione della memoria, quindi è possibile eseguire in modalità *stream*.

```
CALL gds.labelPropagation.stream('component133_u')
YIELD nodeId, communityId AS Community
RETURN Community,count(*) as freq
ORDER BY freq DESC
```

Risultano quindi 31 componenti, con le prime componenti più numerose mostrate in figura 2.23.

	Community	freq
1	92605	2239
2	94483	40
3	115629	39
4	94842	18
5	96900	10
6	117477	7

Figura 2.23

Analizzando più accuratamente, si può notare come tutte le community differenti dalla prima hanno qualche proprietà che le contraddistinguono. Ad esempio, nella community 94483 si è visto come le squadre fossero squadre africane, o ad esempio nella community 94842 le squadre fossero tutte squadre giovanili.

Analisi del cash flow dei club al variare degli anni

L'idea è quella di considerare sempre la relazione *CASH_FLOW* tra i club, ma questa volta di fare un'analisi basata sulla stagione. Si potranno capire quindi interessanti informazioni. È abbastanza risaputo che negli anni i soldi investiti dalle squadre per il calciomercato siano stati sempre di più. Con questa analisi potremo vedere se vi sono stati effettivamente cambiamenti particolari.

Ricordando che i dati a nostra disposizione riguardavano i trasferimenti dal 01/01/2010 al 27/08/2017 compresi, possiamo scartare la stagione 2009/2010 dalla nostra analisi, poiché è composta solamente da 304 trasferimenti dato che non tiene conto della sessioni di mercato estiva. Inoltre il calciomercato della stagione 2017/2018 è finito il 31/08/2017 [2], quindi mancano i dati degli ultimi quattro giorni di mercato. Solitamente gli ultimi giorni di mercato si spende di più rispetto agli altri giorni della sessione di mercato, quindi la stagione 2017/2018 sarà un po' sottostimata.

Si dividono quindi le stagioni in due fasce: la prima fascia comprende le stagioni *2010/2011, 2011/2012, 2012/2013, 2013/2014*, mentre la seconda fascia comprende le stagioni *2014/2015, 2015/2016, 2016/2017, 2017/2018*.

Creazione in-memory graph

Si andranno a creare in entrambi casi due in-memory graph orientati che tengono conto di tutti i club e di tutti flussi di cassa tra essi nelle stagioni scelte. Si è dovuta fare una Cypher projection per il fatto che bisogna filtrare in base alla stagione.

Creazione dell'in-memory graph dal 2010/2011 al 2013/2014

```
CALL gds.graph.create.cypher(
    'first_season',
    'MATCH (n:Club) RETURN id(n) AS id',
```

```
    'MATCH (n:Club)-[r:CASH_FLOW]->(m:Club) WHERE r.season IN
["2010/2011","2011/2012","2012/2013",”2013/2014”] RETURN id(n) AS source,
id(m) AS target,r.total as total,r.season as season')
```

Creazione dell'in-memory graph dal 2014/2015 al 2017/2018

```
CALL gds.graph.create.cypher(
    'last_season',
    'MATCH (n:Club) RETURN id(n) AS id',
    'MATCH (n:Club)-[r:CASH_FLOW]->(m:Club) WHERE r.season IN
["2014/2015","2015/2016","2016/2017","2017/2018"] RETURN id(n) AS
source, id(m) AS target,r.total as total, r.season as season')
```

Densità

Densità del grafo dal 2010/2011 a 2013/2014

```
MATCH (c1:Club)-[r:CASH_FLOW]->(c2:Club)
WHERE r.season IN ["2010/2011","2011/2012","2012/2013","2013/2014"]
RETURN (size(collect(id(r)))*1.0)/((20692)*(20692-1))
```

Risulta una densità di $15 * 10^{-6}$

Densità del grafo dal 2014/2015 a 2017/2018

```
MATCH (c1:Club)-[r:CASH_FLOW]->(c2:Club)
WHERE r.season IN ["2014/2015","2015/2016","2016/2017","2017/2018"]
RETURN (size(collect(id(r)))*1.0)/((20692)*(20692-1))
```

Risulta una densità di $15 * 10^{-6}$

Quindi, la densità del grafo negli ultimi quattro anni è rimasta pressochè uguale.

Flussi di cassa

Flussi di cassa totali dal 2010/2011 a 2013/2014

Utilizzo la seguente query per ritornare il totale in milioni di flussi di cassa generati in quegli anni e il flusso di cassa medio.

```
MATCH (c1:Club)-[r:CASH_FLOW]->(c2:Club)
WHERE r.season IN ["2010/2011","2011/2012","2012/2013","2013/2014"]
RETURN sum(r.total)/10^6 as totale_in_mln,
sum(r.total)/(size(collect(id(r)))*1.0)/10^6 as avg_cash_flow_in_mln
```

Riesco così ad analizzare quanti sono i soldi mossi in totale in questi quattro anni e quanto è il cash flow medio.

totale_in_mln	avg_cash_flow_in_mln
11879.549405	1.8538622666978775

Figura 3.1

Flussi di cassa totali dal 2014/2015 a 2017/2018

```
MATCH (c1:Club)-[r:CASH_FLOW]->(c2:Club)
WHERE r.season IN ["2014/2015","2015/2016","2016/2017","2017/2018"]
RETURN sum(r.total)/10^6 as totale_in_mln,
sum(r.total)/(size(collect(id(r)))*1.0)/10^6 as avg_cash_flow_in_mln
```

	totale_in_mln	avg_cash_flow_in_mln
1	18327.319791	2.9384832116402118

Figura 3.2

Negli ultimi quattro anni sono aumentati di molto i flussi di cassa e il flusso di cassa medio è quasi raddoppiato rispetto ai 4 anni precedenti.

PageRank

PageRank su grafo orientato da 2010/2011 a 2013/2014

Applico PageRank nel grafo orientato riferito alle prime 4 stagioni e ottengo i risultati mostrati in figura 3.3. Come sempre controllo prima con estimate che la memoria sia abbastanza per eseguire l'algoritmo.

```
CALL gds.pageRank.stream('first_season',{relationshipWeightProperty:
'total'})
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).name AS name, score
ORDER BY score DESC, name ASC
```

	name	score
1	"Genoa"	2.403400606219657
2	"River Plate"	2.365333950304194
3	"Red Star"	2.3316498662607046
4	"Benfica"	2.252785449742805
5	"Boca Juniors"	2.164083406579448
6	"Juventus"	2.0358170772378803

Figura 3.3

Come già analizzato in precedenza, questo risultato mi dà l'informazione sulle squadre che hanno realizzato più utile, ovvero che hanno guadagnato di più a discapito di quanto hanno speso. Vediamo quindi che il Genoa è quella che risulta con uno score più alto. Essa quindi è una delle squadre che si è mossa meglio nel mercato generando più utili. Il responsabile del mercato, ovvero il direttore sportivo, in quegli anni era Andrea Berta, che guardando da questa pagina di Transfermarkt [3]è andato successivamente all'Atletico Madrid. In questa classifica, come mostrato dalla figura 3.4, l'Atletico Madrid si trova alla ventisettesima posizione.

27	"Atlético Madrid"	1.5350296662887557

Figura 3.4

Andiamo quindi ad analizzare il PageRank orientato delle stagione successive.

PageRank su grafo orientato da 2014/2015 a 2017/2018

Applichiamo la seguente query, per vedere i risultati di PageRank sulle ultime quattro stagioni.

```
CALL gds.pageRank.stream('last_season',{relationshipWeightProperty:
'total'})
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).name AS name, score
ORDER BY score DESC, name ASC
```

	name	score
1	"Zalgiris"	3.5860979030840094
2	"Inter"	3.329493651888332
3	"Atlantas"	3.3164079651236533
4	"Juventus"	3.1524986793170675
5	"Benfica"	2.905544014484622
6	"Atlético Madrid"	2.359969192539575

Figura 3.5

Si può notare come nella top 6 siano presenti club abbastanza sconosciuti come *Zalgiris* e *Atlantas*. Lo score così alto per questi due club è dovuto al fatto che *Zalgiris* viene puntato da molti club e punta solamente al club *Atlantas*, trasferendo buona parte del suo pagerank score anche ad esso. Tuttavia, se analizziamo i vari trasferimenti di denaro non risultano flussi di cassa molto elevati. Il dato che più ci interessa in questo caso è il fatto che alla sesta posizione ci sia *l'Atletico Madrid*, che dal 2010 al 2013 occupava la ventisettesima posizione. Il Genoa invece che dal 2010 al 2013 risultava primo, ha compiuto il percorso inverso rispetto all'*Atletico Madrid*, andando a piazzarsi in questo caso cinquanteseiesimo.

56	"Genoa"	1.1530894219613406

Figura 3.6

Abbiamo visto prima infatti che il direttore tecnico *Andrea Berta* sia passato all'Atletico Madrid proprio nel 2013 e qui sia rimasto fino ad ora. Probabilmente quindi c'è una correlazione tra il fatto che una volta che *Andrea Berta* sia passato all'Atletico Madrid essa abbia fatto buone operazioni di mercato, mentre il Genoa non abbia fatto altrettanto, evidenziando quindi la bravura di questo dirigente a comprare poco spendendo poco e vendendo molto generando grossi flussi di cassa attivi. La bravura del dirigente è confermata in questi articoli [4] [5].

Degree Centrality

Degree Centrality dal 2010/2011 al 2013/2014

Si esegue questo algoritmo per capire le squadre con i maggiori flussi di cassa nei primi quattro anni.

```
CALL gds.alpha.degree.stream('first_season',{relationshipWeightProperty:
'total'})
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).name AS name, score AS degree_centrality
ORDER BY degree_centrality DESC
```

Ottenendo i risultati mostrati in figura 3.7

	name	degree_centrality
1	"Chelsea"	400060000.0
2	"Man City"	398220000.0
3	"Paris SG"	352750000.0
4	"Real Madrid"	321300000.0
5	"Liverpool"	246853000.0
6	"FC Barcelona"	240030000.0
7	"Juventus"	238472000.0
8	"Man Utd"	220670000.0
9	"Anzhi"	216180000.0
10	"SSC Napoli"	206065450.0

Figura 3.7

Notiamo che i risultati corrispondo più o meno a delle squadre abbastanza conosciute. Tuttavia, alla nona posizione notiamo che è presente il club *Anzhi*. Andiamo quindi a vedere la degree centrality delle ultime quattro stagioni e vediamo se *l'Anzhi* ha mantenuto l'andamento iniziale.

Degree Centrality dal 2014/2015 al 2017/2018

I risultati sono mostrati in figura 3.8

	name	degree_centrality
1	"Man City"	672256000.0
2	"Man Utd"	630680000.0
3	"Paris SG"	493380000.0
4	"Juventus"	481995000.0
5	"FC Barcelona"	479680000.0
6	"Chelsea"	450900000.0
7	"Liverpool"	366960000.0
8	"Atlético Madrid"	334855000.0
9	"AS Roma"	311941000.0
10	"Inter"	309880000.0

Figura 3.8

Come si nota dalla figura 3.8, club in top 10 sono abbastanza simili, con l'aggiunta dell'*Atletico Madrid* (probabilmente anche per i motivi che abbiamo analizzato al punto precedente) e *l'Inter.* Vediamo inoltre che il *Napoli* è passata dalla decima alla venticinquesima posizione. Tuttavia il fatto più eclatante è il risultato dell'*Anzhi*. Nei primi 4 anni infatti aveva fatto girare tra entrate e uscite quasi 220 milioni di euro, mentre negli ultimi 4 anni ha fatto girare meno di 2 milioni di euro. Questo è dovuto al fatto che i proprietari della società hanno avuto dei grossi problemi finanziare nel 2013 che hanno fatto sì che gli investimenti diminuiti enormemente. [6]

406	"Anzhi"	1683000.0

Figura 3.9

Conclusioni

Ho quindi analizzato il grafo, focalizzandomi sull'informazione presente tra la relazione *CASH_FLOW* dei club, riuscendo a capire quali siano i club più ricchi (*Paris SG* e *Man City*), i club più centrali, e i club che hanno generati più utili e più spese nel grafo, capendo che ad esempio i club che hanno generato più utili investissero molto nei giocatori giovani con una grossa crescita potenziale di valore di mercato.

Successivamente sono passato ad analizzare i flussi di cassa dividendo le stagioni in due fasce di tempo. In questo caso, ad esempio, si è capito che probabilmente il successo sul mercato del Genoa potesse dipendere da un dirigente, che poi, in seguito al suo passaggio all'Atletico Madrid, ha determinato il successo di questa squadra.

Oltre a tutte le analisi statistiche sui grafi rappresentanti le due fasce di tempo, analizzando la degree-centrality si è riuscito a vedere l'anomalo comportamento del club *Anzhi*, che nella prima fascia di tempo era un nodo molto importante, mentre nella seconda fascia di tempo ha perso quasi tutta la sua importanza.

Bibliografia

[1] Neovis.js– Tool visualizzazione - https://github.com/neo4j-contrib/neovis.js/
[2] Fine calciomercato 2017 - https://www.goal.com/it/notizie/calciomercato-estivo-2017-quando-inizia-e-quando-finisce/2rl8c3kjtzaf1u9aofoc49mws#:~:text=Ma%20quali%20sono%20le%20date,22%3A59%20del%2031%20agosto
[3] Carriera di Andrea Berta - https://www.transfermarkt.it/andrea-berta/profil/trainer/63428
[4] Articolo su Andrea Berta - https://www.globesoccer.com/winners/andrea-berta-best-sporting-director-of-the-year/
[5] Articolo su Andrea Berta - https://www.giornaledibrescia.it/sport/andrea-berta-direttore-sportivo-dell-anno-ai-globe-soccer-awards-1.3443060
[6] Articolo Anzhi- https://www.eastjournal.net/archives/107585

Appendice

Per questa attività è stato usato *Neo4j Desktop 1.3.11* e sono stati eseguiti tutte le indicazioni ottenute eseguendo il seguente comando:

:play football_transfers

Tutte le indicazioni presenti sono riportate qui sotto.

```
CREATE CONSTRAINT ON (player:Player)
ASSERT player.id IS UNIQUE

CREATE CONSTRAINT ON (club:Club)
ASSERT club.id IS UNIQUE

CREATE CONSTRAINT ON (transfer:Transfer)
ASSERT transfer.id IS UNIQUE

CREATE CONSTRAINT ON (country:Country)
ASSERT country.name IS UNIQUE

:auto USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM 'https://s3-eu-west-1.amazonaws.com/football-
transfers.neo4j.com/transfers-all.csv' AS row
MERGE (player:Player {id: row.playerUri})
ON CREATE SET player.name =  row.playerName, player.position =
row.playerPosition
```

Importo i paesi

```
WITH 'https://s3-eu-west-1.amazonaws.com/football-transfers.neo4j.com/transfers-
all.csv' AS url
LOAD CSV WITH HEADERS FROM url AS row
```

```
WITH row WHERE row.playerNationality <> ''
WITH DISTINCT row.playerNationality AS nationality
MERGE (country:Country {name: nationality })
```
(ci mette 60 secondi creando circa 200 nodi)

Importo le relazioni tra calciatori e paesi

```
:auto USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM 'https://s3-eu-west-1.amazonaws.com/football-
transfers.neo4j.com/transfers-all.csv' AS row
WITH row WHERE row.playerNationality <> ''
MATCH (player:Player {id: row.playerUri})
MATCH (country:Country {name: row.playerNationality })
MERGE (player)-[:PLAYS_FOR]->(country)
```

Importo i dati dei club

```
WITH 'https://s3-eu-west-1.amazonaws.com/football-transfers.neo4j.com/transf
ers-all.csv' AS url
LOAD CSV WITH HEADERS FROM url AS row

WITH row where row.sellerClubUri <> ''
MERGE (club:Club {id: row.sellerClubUri})
ON CREATE SET club.name = row.sellerClubName
MERGE (country:Country {name: row.sellerClubCountry})
MERGE (club)-[:PART_OF]->(country)
```

Importo le relazioni tra club e country

```
WITH 'https://s3-eu-west-1.amazonaws.com/football-transfers.neo4j.com/transf
ers-all.csv' AS url
LOAD CSV WITH HEADERS FROM url AS row
WITH row where row.buyerClubUri <> ''
MERGE (club:Club {id: row.buyerClubUri})
ON CREATE SET club.name = row.buyerClubName
MERGE (country:Country {name: row.buyerClubCountry})
MERGE (club)-[:PART_OF]->(country)
```

```
WITH 'https://s3-eu-west-1.amazonaws.com/football-transfers.neo4j.com/transf
ers-all.csv' AS url
LOAD CSV WITH HEADERS FROM url AS row
UNWIND [
  {uri: row.sellerClubUri, name: row.sellerClubName, country: row.sellerClub
Country},
  {uri: row.buyerClubUri,  name: row.buyerClubName,  country: row.buyerClubC
ountry}
] AS club
WITH club WHERE club.uri <> ''
```

```
WITH DISTINCT club
MERGE (c:Club {id: club.uri})
ON CREATE SET c.name = club.name
MERGE (country:Country {name: club.country })
MERGE (c)-[:PART_OF]->(country)
```

Importo i dati sui trasferimenti dei giocatori

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM 'https://s3-eu-west-1.amazonaws.com/football-tran
sfers.neo4j.com/transfers-all.csv' AS row
MATCH (player:Player {id: row.playerUri})
MATCH (source:Club {id: row.sellerClubUri})
MATCH (destination:Club {id: row.buyerClubUri})
MERGE (t:Transfer {id: row.transferUri})
ON CREATE SET t.season = row.season,
              t.fee = row.transferFee,
              t.timestamp = toInteger(row.timestamp)
MERGE (t)-[ofPlayer:OF_PLAYER]->(player) SET ofPlayer.age = row.playerAge
MERGE (t)-[:FROM_CLUB]->(source)
MERGE (t)-[:TO_CLUB]->(destination);
```

Puliamo i dati

```
MATCH (t:Transfer)
WHERE t.fee contains "?" or t.fee  contains "-"
REMOVE t:Transfer
SET t:TransferWithoutFee
```

Settiamo i trasferimenti in prestito (Loan)

```
MATCH (t:Transfer)
WHERE t.fee STARTS WITH 'Loan'
SET t:Loan
```

Settiamo i trasferimenti gratuiti

```
MATCH (t:Transfer)
WITH t, replace(replace(replace(replace(t.fee, "k", ""), "m", ""), "Loan fee
:", ""), "£", "") AS rawNumeric
WITH t,
CASE
 WHEN t.fee ENDS WITH "k" THEN toFloat(apoc.number.exact.mul(trim(rawNumeric
),"1000"))
 WHEN trim(t.fee) IN ["Free transfer", "ablösefrei ", "gratuito", "free", "f
ree transfer",
                      "Ablösefrei", "transfervrij", "ablöserei", "Free Trans
fer", "Libre",
```

```
                    "gratutito", "ablsöefrei", "ablösefrei", "ablösefei",
"abösefrei", "Loan",
                "draft", "Swap deal", "trade", "ablösefrei", "ablösefreei", "F
ree",
                    "ablosefrei", "Draft", "Trade", "Libre para traspaso",
"bez odstępnego",
                    "ablossefrei", "Bez odstępnego", "Gratuito", "ablödefre
i", "Bonservissiz",
                    "ablösfrei", "ablõsefrei", "ablösefre", "custo zero", "
ablösefrei!",
                    "ablösefreo", "svincolato", "Ablösfrei", "livre", "libr
e", "Leihe",
                "abolsfrei", "ablösefrai", "ablösefreil", "abllösefrei", "aböl
sefrei",
                "ablöserfrei", "abklösefrei", "ablöaefrei", "Ablosefrei", "Nes
suno",
                    "ablösesfrei", "Free Tranfer", "abblösefrei", "Spielert
ausch", "ablösebrei",
                    "abslösefrei", "spielertausch", "a", "ablöseferi", "abl
öserfei", "Tausch"] THEN 0
 WHEN NOT(exists(t.fee)) THEN 0
 WHEN rawNumeric = '' THEN 0
 WHEN t.fee ENDS WITH "m" THEN toFloat(apoc.number.exact.mul(trim(rawNumeric
),"1000000"))
 ELSE toFloat(trim(rawNumeric))
END AS numericFee
SET t.numericFee = numericFee
```

Escludiamo ulteriori trasferimenti anomali

```
MATCH (t:Transfer)
WHERE not exists(t.numericFee)
REMOVE t:Transfer
SET t:TransferWithoutFee
```

Creo la relazione NEXT

```
MATCH (p:Player)<-[:OF_PLAYER]-(transfer)
WHERE transfer.numericFee > 0


WITH p, transfer
ORDER BY p.name, transfer.timestamp


WITH p, collect(transfer) AS transfers
WHERE size(transfers) > 1
UNWIND range(0, size(transfers)-2) AS idx


WITH transfers[idx] AS t1, transfers[idx+1] AS t2
```

```
MERGE (t1)-[:NEXT]->(t2)
```

Creo la relazione CASH_FLOW

```
MATCH (t:Transfer)
WITH DISTINCT t.season AS season
MATCH (seller)<-[:FROM_CLUB]-(t:Transfer)-[:TO_CLUB]->(buyer)
WHERE t.season = season AND t.numericFee > 0
WITH season, seller, buyer, sum(t.numericFee) AS cash_flow, count(t) AS play
er_count
MERGE (buyer)-[:CASH_FLOW {total: cash_flow, playerCount: player_count, seas
on: season}]->(seller)
```