El Número de Abreu

Martín Schnaiderman
Facultad de Ingeniería, Universidad de la República
Montevideo, Uruguay
martin.schnaiderman@fing.edu.uy

Jimena Mignaco
Facultad de Ingeniería, Universidad de la República
Montevideo, Uruguay
jimena.mignaco@fing.edu.uy

***Resumen*—El proyecto consiste en el armado de una base de datos de grafos de futbolistas y equipos, con el fin de aplicar distintas herramientas provistas por Neo4j y obtener así el camino más corto entre dos futbolistas dados. Luego de cargar los datos, se implementa un sitio web que los consume y se realizan distintas consultas desde el *Browser* de Neo4j Desktop. Finalmente, se observa y compara el tiempo requerido para realizar las distintas consultas en grafos de diferentes dimensiones.**

I. Introducción

Este informe es desarrollado con el fin de documentar el proyecto final del curso de Bases de Datos No Relacionales. El objetivo del proyecto es el de armar una base de datos de grafos que cuente con los futbolistas y directores técnicos de las principales ligas de fútbol, así como también los clubes y selecciones nacionales que estos defendían. Dado que la motivación de este trabajo surge del retiro de Sebastián Washington Abreu del fútbol profesional, se decide restringir la obtención de los datos a los años en que estuvo en actividad, es decir, los últimos 30 años. A su vez, el proyecto también incluye el desarrollo de un sitio web que consume datos de la base y experimentos sobre la misma.

II. Trabajos Relacionados

Existen distintas bases de datos de futbolistas, sin embargo, están enfocadas desde distintos puntos de vista. En algunas, el enfoque se centra en los partidos que tiene un equipo en una temporada, registrando los goles y tarjetas de cada uno. Este es el caso de LiveFutbol[1] o WhoScored[2]. Existen también otras que contemplan el punto de vista económico. Estas se enfocan en registrar el precio de los traspasos de cada jugador entre equipos o estimar el valor de mercado de cada futbolista a lo largo del tiempo. Algunos ejemplos son Goal[3] o Transfermarkt[4] entre otros. Desde el punto de vista más técnico no hay ningún indicio que estas bases de datos sean no relacionales ya que son privadas y no hay manera de saber su arquitectura interna.

Por otro lado existe una solución muy similar a la que se propone en este proyecto pero enfocada a actores y actrices de cine. Se trata de Oracle of Bacon[5], un sitio web donde se eligen dos actores y se busca el camino más corto entre ellos. Nuevamente, no se especifica el tipo de base de datos que utilizan pero mencionan que periódicamente se descarga la información de los actores de Wikipedia[6] en archivos JSON y a partir de la misma se arma la base.

[1]https://livefutbol.com/
[2]https://whoscored.com/
[3]https://goal.com/
[4]https://transfermarkt.es/
[5]https://oracleofbacon.org/

III. Desarrollo del proyecto

III-A. Obtención de los Datos

La obtención de los datos no es una tarea sencilla. En un principio se intenta obtener los datos de Dbpedia [7], ya que los mismos están estandarizados y enlazados. Sin embargo, están incompletos en la mayoría de los casos y la obtención masiva de los datos se dificulta.

Como segunda opción se busca una API pública para obtener los datos y se intenta el contacto con los desarrolladores de algunas de las paginas web (como Livefutbol o Transfermarkt) que manejan bases de datos de futbolistas.

Ante la falta de respuestas, se busca una tercer alternativa: hacer *scrapping* sobre Livefutbol, recorriendo cada una de los equipos de las ligas profesionales seleccionadas tanto para sus plantillas actuales como para sus plantillas históricas. Para una primer versión de la obtención de los datos se utiliza Python junto con las bibliotecas que se detallan a continuación.

- **Selenium** : Es una biblioteca utilizada para automatizar el navegador que se desee y de esta forma ir recorriendo iterativamente diferentes páginas web.
- **Beautiful Soup** : Es una biblioteca utilizada para obtener el código HTML de una página y descomponerla sintácticamente (*parsear*). Se utiliza con el fin de encontrar un elemento HTML puntual como por ejemplo el nombre de un jugador de fútbol.
- **re** : Es una biblioteca utilizada para realizar operaciones sobre expresiones regulares. En particular se utiliza para darle fin a la iteración sobre las plantillas históricas de un equipo. Mientras el año de la plantilla es posterior a 1993, se obtienen los datos. En caso contrario se detiene.
- **Pandas** : Es una biblioteca utilizada para el análisis y manipulación de datos. En particular se usa para guardar los datos obtenidos en archivos .csv.

Esta primer versión cuenta con un par de problemas. Para empezar, Selenium es muy lento (Alrededor de 8 horas para 10 ligas) ya que espera a que el navegador seleccionado interprete

[6]https://es.wikipedia.org/
[7]https://www.dbpedia.org/

el HTML completamente y lo muestre en pantalla. Por otro lado, surge otro problema asociado a la falta de identificadores únicos para los datos. Equipos de fútbol con el mismo nombre agrupan a jugadores que nunca habían jugado juntos y de igual forma sucede que jugadores con el mismo nombre son interpretados como el mismo nodo en el grafo. En una segunda versión del código se sustituye la biblioteca Selenium por Request la cual hace prácticamente lo mismo sin abrir un navegador y se reduce mucho el tiempo para la obtención de datos. Por otro lado, se le asocia una URI como identificador a cada equipo y a cada jugador. Esta URI proviene de la URL de cada página de Livefutbol. Por ejemplo, el de Sebastián Washington Abreu es "/ficha_jugador/sebastian-abreu/".

III-B. Diseño y Cargado de los datos

Para el diseño de la estructura de la base fueron manejadas varias opciones pero se optó por la que se considera más simple. Consiste en 2 tipos de nodos: Jugador y Equipo. Ambos cuentan con un identificador único, siendo importante destacar que los nodos Equipos (o plantillas como le llamaremos de aquí en adelante) están definido por el par (Equipo, Año). De esta forma se logra que un mismo equipo pueda tener asociada una lista distinta de jugadores dependiendo del año.

Finalmente se define un único tipo de relación, esta tiene como origen un jugador y como destino un equipo, el nombre de la misma es "JUGO_EN".
Existe un problema con este diseño y es que no respeta completamente la realidad, ya que por temporada hay dos períodos de pase. Puede llegar a suceder que dos jugadores hayan jugado en el mismo equipo el mismo año pero en distintos semestres, lo que hace que en nuestra representación figuren como compañeros, siendo que en la realidad esto no fue así. Sin embargo, se considera que se trata de casos límite poco comunes y que solucionar el inconveniente requeriría cambiar todo lo implementado, por lo que se opta por no hacerlo. Existen otros problemas con el diseño pero estos se detallarán más adelante en este documento.

Para la carga del .csv en Neo4j se ejecuta la siguiente consulta en Cypher:

Listado 1 Ejemplo de cargado de liga uruguaya Cypher

```
LOAD CSV WITH HEADERS FROM 'file:///ListaUruguay.csv'
AS line
MERGE (j:Jugador {
   id: line.Links,
   nombre: line.Jugadores,
   pais: line.PaisJugador
   })
MERGE (e:Equipo {
   id: line.linkCuadro,
   nombre: line.Cuadros,
   anio: line.Temporada,
   pais: line.PaisCuadro
   })
MERGE (j)-[r:JUGO_EN]->(e);
```

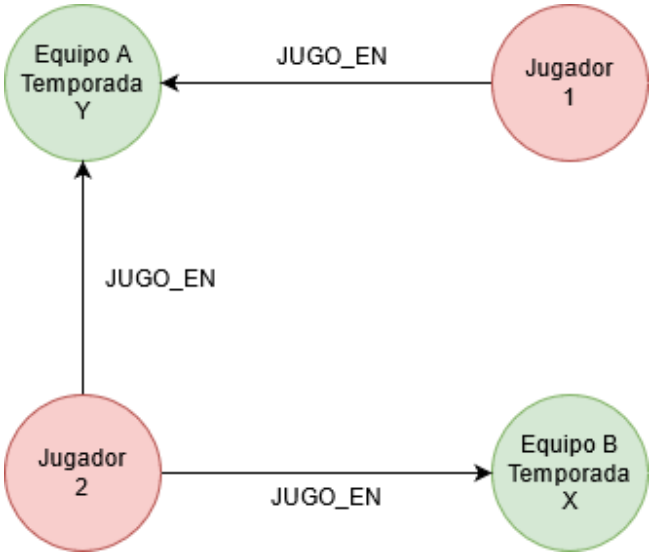

Figura 1: Ejemplo de la estructura del grafo.

Se utiliza MERGE para la creación de nodos y enlaces con el objetivo de verificar que este no exista previamente. Puede llegar a ser menos eficiente que CREATE pero como se sabe que los nodos se repiten, utilizar MERGE ahorra dicha verificación y depuración posterior.

III-C. Desarrollo e integración de sitio web

Se implementa un sitio web que devuelve el número de Abreu y cuál es el camino que conecta el par de jugadores. El objetivo es conectar una base de datos de Neo4j con una aplicación Web. Para llevar a cabo esta sección se utiliza Python con el *Framework* Flask. Además, se utiliza un *driver* provisto por Neo4j como nexo entre la base de datos y la aplicación.

Es importante destacar las estructuras que define dicho *driver*. En caso de tener como resultado un Jugador o un Equipo, dicha información se devuelve en forma de *Node* (Nodo), con sus atributos correspondientes. Si la consulta obtiene el camino más corto, el resultado corresponde a un *Path* (Camino), del cual se puede extraer el largo, los nodos y las relaciones involucradas. La estructura para representar una relación se denomina *Relationship*. Las tres presentan una serie de funciones y atributos que permiten la comparación y obtención de los datos.

Al momento de listar los jugadores, la primer dificultad que se presenta es que es imposible mostrarlos a todos. Es por esto que se opta por seleccionar los diez cuyo nombre sea el más próximo al indicado por el usuario.

La idea de proximidad se basa en el hecho de que posiblemente el usuario no escriba correctamente el nombre del jugador que desea. Por lo tanto, se utiliza la distancia de Jaro–Winkler de la biblioteca APOC[8] de Neo4j para calcular un índice de similitud entre el nombre y el texto introducido. Aquellos diez que tengan el índice más alto son los que se

[8]https://neo4j.com/developer/neo4j-apoc/

ponen a disposición del usuario para que elija el deseado. Como también puede ocurrir que más de un jugador tenga el mismo nombre (por ejemplo Luis Suárez), al lado del nombre se muestra también el país del jugador.

El sitio web implementa las siguientes consultas a la base de datos:

Listado 2 Obtención de los jugadores Cypher

```
MATCH (j:Jugador)
RETURN apoc.text.jaroWinklerDistance(
   toLower(j.nombre), toLower([nombre])
) as distancia,
j.pais as pais, j.nombre as nombre, j.id as id
ORDER BY distancia desc limit 10
```

Listado 3 Camino más corto entre ambos jugadores Cypher

```
MATCH p=shortestPath(
   (jugador1:Jugador {id:[desde]})
   -[*]-
   (jugador2:Jugador {id:[hasta]}))
RETURN p
```

Se presentan a continuación un par de imágenes del sitio web implementado:

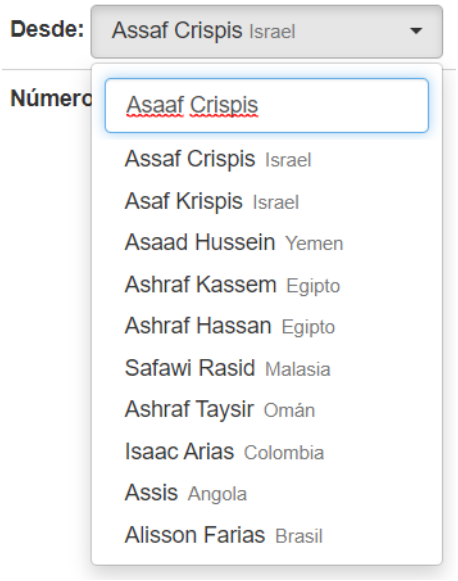

Figura 2: *Select* de Jugadores

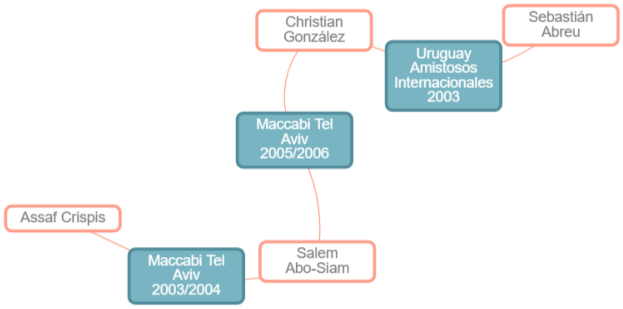

Figura 3: Camino más corto entre Sebastián Abreu y Asaaf Crispis

IV. Experimentación

IV-A. Algoritmo de detección de comunidades

Con el objetivo de obtener información de interés sobre el grafo se ejecutaron algoritmos de detección de comunidades. Estos se utilizan para evaluar cómo los grupos de nodos se agrupan o se dividen, así como su tendencia a fortalecerse o separarse. [Neo4j(2010e)]

Para ejecutar estos algoritmos se debe instalar la biblioteca "Graph Data Science Library". Cabe destacar que los mismos corren sobre un modelo que consiste en la proyección de un grafo de Neo4j. La proyección puede ser considerada como una vista del grafo y es almacenada en memoria usando estructuras optimizadas para las operaciones de búsqueda. [Neo4j(2010d)]

El catálogo de grafos es un concepto dentro de la biblioteca previamente mencionada que permite gestionar múltiples proyecciones por nombre.

Luego de usar, los grafos pueden ser quitados del catálogo para liberar la memoria.

Hay dos formas de proyectar un grafo desde la base de datos de Neo4j a la memoria principal:

Listado 4 Proyección Nativa

```
CALL gds.graph.create(
   'nombre-grafo',
   'Nodo',
   'RELACION'
)
```

Listado 5 Proyección con Cypher

```
CALL gds.graph.create.cypher(
   'nombre-grafo',
   'MATCH (n:Nodo) RETURN id(n) AS id',
   'MATCH (a:Nodo)-[:RELACION]->(b:Nodo)
   RETURN id(a) AS source, id(b) AS target'
)
```

En esta ocasión se opto por una proyección con Cypher, ya que la relación entre los jugadores no es directa, sino que pasa por un equipo. Por lo tanto, es necesario explicitar cual es el origen y cual el destino de la relación. La consulta resultante es la siguiente:

Listado 6 Proyección del Grafo

```
CALL gds.graph.create.cypher(
   'grafoAbreu',
   'MATCH (j:Jugador) RETURN id(j) AS id',
   'Match (j1)-[r1:JUGO_EN]->(e)<-[r2:JUGO_EN]-(j2)
   RETURN id(j1) as source, id(j2) as target'
)
```

Para este proyecto se corrieron 3 algoritmos sobre el grafo, que se detallan a continuación.

IV-A1. Método de Louvain: El método Louvain es un algoritmo para detectar comunidades en grandes redes. Maximiza una puntuación de modularidad para cada comunidad, donde la modularidad cuantifica la calidad de una asignación de nodos a las comunidades. Esto significa evaluar cuánto más densamente conectados están los nodos dentro

de una comunidad, en comparación con qué tan conectados estarían en una red aleatoria. [Neo4j(2010b)] En el grafo se detectaron 176 comunidades al correr el algoritmo. La más grande de estas cuenta con 6242 jugadores y se destaca por estar relacionada a Francia. Consiste en jugadores de la liga francesa, personas con nacionalidad francesa y jugadores de países africanos que antiguamente eran colonias francesas como Camerún o Argelia. La siguientes 2 comunidades más grandes que se detectaron son las asociadas a Brasil y Argentina. La comunidad a la que pertenece Abreu cuenta con 2808 jugadores y se caracteriza por incluir jugadores Uruguayos. Sin embargo, se pudo observar que existen jugadores Uruguayos que pertenecen a distintas comunidades. Esto se entiende que sucede en los casos en que estos jugaron la mayoría de su carrera en el extranjero.

IV-A2. Algoritmo de detección de componentes débilmente conexos (WWC): El algoritmo WCC encuentra conjuntos de nodos conectados en un grafo no dirigido, donde todos los nodos del mismo conjunto forman un componente conexo. WCC se utiliza a menudo al principio de un análisis para comprender la estructura de un grafo. Este uso permite ejecutar otros algoritmos de forma independiente en un grupo identificado. Como paso previo al procesamiento de grafos dirigidos, ayuda a identificar rápidamente los grupos desconectados.

En el grafo de futbolistas cuando se corrió el algoritmo se detectaron 99 componentes conexas. Lo curioso es que el 94.7 % de los jugadores pertenecen a una misma componente mientras que el resto son componentes conexas pequeñas de menos de 300 nodos. Esto sucede porque lo que tienen en común las componentes conexas de menor tamaño es que cuentan con jugadores que participaron de campeonatos con su selecciones, defendiendo a países poco competitivos futbolísticamente hablando como India o Vietnam. Esto se debe a que estos jugadores en su gran mayoría juegan en las ligas de sus respectivos países y no viajan al extranjero para jugar en otras ligas más competitivas, por lo tanto, al no haber obtenido datos de estas ligas "menores"no existe ningún jugador del componente conexo de mayor tamaño que haya jugado con alguno de ellos en la base de datos. También existe otro tipo de componente conexo y es el de aquellos equipos de los que se tiene poca información. Equipos que únicamente tienen 1 o 2 jugadores y los mismos jugaron únicamente en esos equipos o en otros que no se encuentran en la base de datos. [Neo4j(2010a)]

IV-A3. Algoritmo de recuento de triángulos: Se genera una base de datos con menos elementos para visualizar mejor el concepto de triángulo. Sin embargo, tal como se puede ver en la Figura 4, la existencia de equipos en medio de la relación dificulta la visualización. Debido a la definición del grafo, los triángulos considerados son los formados por los jugadores, por lo que serían los presentes en la Figura 6. En dicho ejemplo, Luis Suárez presenta un total de 8 triángulos.

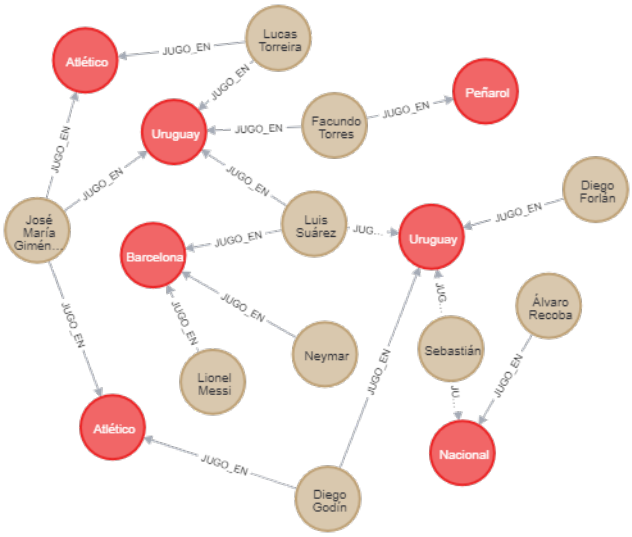

Figura 4: Grafo pequeño de futbolistas

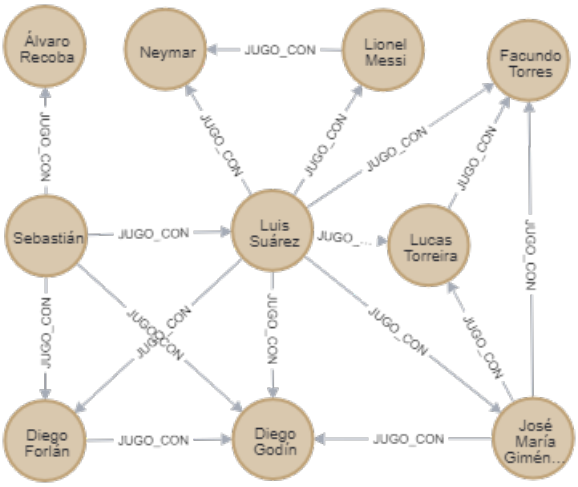

Figura 5: Triángulos considerados

El conteo de triángulos se utiliza para detectar comunidades y medir la cohesión de esas comunidades. [Neo4j(2010c)] El hecho de que un jugador forme parte de una gran cantidad de triángulos implica que jugó con muchas personas que también compartieron equipo. En el listado de jugadores ordenado de forma descendiente por la cantidad de triángulos se observan figuras conocidas a nivel mundial tales como Rafael Márquez, Gianluigi Buffon y Carlos Tévez. Todos ellos tienen en común haber jugado en cuadros europeos importantes, lo que tiene sentido ya que los mejores jugadores suelen rotar en los mejores cuadros dando lugar a esa alta cohesión. Un detalle importante es que para este proyecto se recolectaron datos de parte de los cuadros, por lo que posiblemente no esté completa la trayectoria de jugadores menos conocidos. Esta falta de datos lleva a que los últimos de la lista tengan pocos triángulos, ya que no se conoce la información de todos sus compañeros.

IV-B. Consultas de Interés

Si bien el objetivo del proyecto es el de encontrar el camino más corto entre 2 jugadores dados la forma en la que se diseñó la base permite otras consultas interesantes. Si se corre la siguiente consulta Cypher se puede obtener una lista descendente de los jugadores que tuvieron más compañeros en su carrera profesional.

Listado 7 Lista de jugadores con más compañeros de equipo

```
Match
   (j1:Jugador)-[r1:JUGO_EN]->(e:Equipo)
   <-[r2:JUGO_EN]-(j2:Jugador)
RETURN
   j1.id, count(distinct j2) as companeros
ORDER BY
   companeros desc
```

No es ninguna sorpresa que Sebastián Washington Abreu con 792 compañeros sea el jugador que jugó con más personas ya que al momento de seleccionar las ligas se decidió agregar todas aquellas en las que Abreu haya jugado. Sin embargo, si es interesante el patrón que se observa en los siguientes elementos de la lista. Todos son estrellas mundiales, jugadores que estuvieron en actividad hasta casi los 40 años y fueron convocados s su selección en múltiples ocasiones a lo largo de los años. Algunos de ellos son Rafael Márquez o Carlos Tevez. Tambien es interesante observar que Gianluigi Buffón es el octavo jugador con más compañeros ya que únicamente jugó en 3 equipos y 19 temporadas lo hizo en el mismo.

Averiguar cual es la plantilla que tuvo un plantel más diverso desde el punto de vista de las nacionalidades de sus jugadores fue también una de las consultas ejecutadas. Se realizó la siguiente consulta la cual lista de manera descendente las plantillas (equipo,temporada) según la cantidad de nacionalidades.

Listado 8 Equipo con más nacionalidades en su plantel

```
MATCH
 (j1:Jugador)-[r1:JUGO_EN]->(e:Equipo)
RETURN
 e.id, count(distinct j1.pais) as cantPaises
ORDER BY
 cantPaises desc
```

En este caso tampoco fue ninguna sorpresa que los primeros lugares de la lista sean de equipos Ingleses ya que se caracterizan por tener jugadores de distintos países. Lo sorprendente es que el puesto número 1 con 23 nacionalidades distintas es un equipo menor de la liga inglesa como es el West Ham United en la temporada 2011/2012.

La tercer consulta de interés que se realizó fue la dar una lista de los pares de plantillas que compartieron más jugadores. Para esto se realizó la siguiente consulta:

Listado 9 Par de plantillas con más jugadores en común

```
MATCH
   (e2:Equipo)<-[r2:JUGO_EN]-(j1:Jugador)-[r1:JUGO_EN]
   ->(e1:Equipo)
WHERE
   e1.nombre<>e2.nombre
RETURN
   e1.id,e2.id,e1.nombre,e2.nombre, count(distinct j1.id)
   as cantJugadores
ORDER BY
   cantJugadores desc
```

Para esta consulta los resultados no fueron los esperados. Al no tener identificadores únicos para cada equipo (ya que a cada plantilla la identifica el par equipo, año) la consulta no logró responder lo que se quería. Existen dos problemas con esta consulta, en primer lugar hay equipos que cambiaron de nombre o son conocidos por dos nombres distintos. Como consecuencia de esto los planteles del mismo equipo en temporadas consecutivas aparecen como resultado de la consulta. El siguiente problema que se tiene es que no se distingue entre un equipo y una selección. Lo que sucede es que hay selecciones que convocan principalmente jugadores de un mismo equipo en el mismo año y por lo tanto aparecen como resultado de la consulta cuando no deberían. De haber creado nodos distintos para las selecciones u otra relación para modelar los jugadores convocados por su selección se podría haber evitado este problema.

IV-C. Estudio de Rendimiento

En esta sección se compara el tiempo de ejecución de algunas de las consultas realizadas, sobre grafos de distintas dimensiones. En primer lugar se analiza el tiempo que lleva cargar distintos archivos .csv a la base de datos. Tal como se mencionó anteriormente, al momento de crear un nuevo nodo o relación se verifica que la misma no existe, lo que hace que no se trate de un proceso lineal. Como cada línea del archivo .csv representa una relación, se evalúa en función de ese valor. Cada archivo es cargado en un proyecto de Neo4j distinto para que no se afecten mutuamente.

Archivo	Cant. Relaciones	Tiempo(ms)
ListaArgentina	21236	26340
ListaEspaña	16948	19744
ListaUruguay	7509	5254
ListaPerú	2247	1171

Cuadro I: Tiempo de carga de archivos.

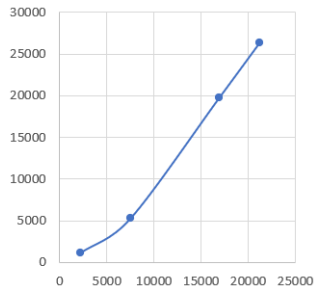

Figura 6: Tiempo en función de la cantidad de relaciones

La carga de todos los archivos (349950 relaciones, 47.4 MB) toma aproximadamente 12 horas.

Otra consulta interesante es la de realizar la proyección de la base de datos para cargarla en el catálogo de grafos. Se realiza la comparación entre la base de datos principal del proyecto, la que contiene solo los datos de Uruguay y la pequeña mencionada al analizar los triángulos.

Grafo	Cant. Relaciones	Cant. Nodos	Tiempo(ms)
Total	349950	85897	6274
Uruguay	7509	2795	3673
Muestra	18	17	2960

Como se puede ver, el tiempo aumenta en menor proporción que el tamaño del grafo.

Si bien se analizaron todas las consultas mencionadas en este informe, solo este par resulta interesante ya que el resto se realizan en un tiempo despreciable (menor a 10ms) para todos los grafos analizados.

Es importante destacar la diferencia entre el tiempo de ejecución de la consulta y el del despliegue de los datos, que suele ser significativamente mayor. Si bien las consultas se realizaron en el tiempo mencionado, el hecho de tener que mostrar grandes volúmenes de datos hace que se tarde más en ver los resultados.

V. Conclusiones y trabajo futuro

El trabajo en su conjunto es una interesante prueba de concepto de las principales herramientas que provee Neo4j. Se ejecutaron tareas de *scrapping*, diseño y cargado de base de datos, integración con sitio web, detección de comunidades y pruebas de *performance*.

Si bien la utilidad académica o económica del producto final es mínima, sirve como un primer acercamiento a las base de datos de grafos y es un éxito desde el punto de vista educativo ya que se pusieron a prueba distintos elementos relacionados con la asignatura. Es importante destacar que todos los programas implementados están disponibles en el repositorio[9].

Como trabajo futuro se podría publicar el sitio web de forma que cualquiera pueda experimentar con él. Por otro lado, se podría hacer un rediseño de la base para eliminar los problemas que se detectaron. Se podría crear un tercer tipo de nodo asociado a equipos y crear una nueva relación entre las distintas plantillas y el mismo. Por otro lado se puede crear otro tipo de nodo llamado selección para poder distinguir entre un equipo y una selección. Estos cambios requerirían un nuevo proceso de *scrapping* y un rediseño de la base así como de las consultas. En la Figura 7 se puede ver un nuevo diseño propuesto para la solución.

[9]https://gitlab.fing.edu.uy/jimena.mignaco/bdnr2021g18

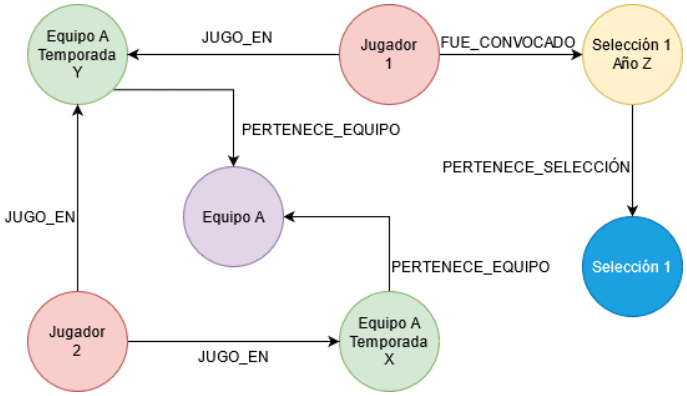

Figura 7: Posible diseño para nueva versión.

Referencias

[Neo4j(2010a)] Neo4j. Algoritmo de detección de componentes debilmente conexas, 2010a. URL https://neo4j.com/docs/graph-data-science/current/algorithms/wcc/#algorithms-wcc.

[Neo4j(2010b)] Neo4j. Algoritmo de louvain, 2010b. URL https://neo4j.com/docs/graph-data-science/current/algorithms/louvain/#algorithms-louvain.

[Neo4j(2010c)] Neo4j. Algoritmo de recuento de triángulos, 2010c. URL https://neo4j.com/docs/graph-data-science/current/algorithms/triangle-count/#algorithms-triangle-count.

[Neo4j(2010d)] Neo4j. Catálogo de grafos, 2010d. URL https://neo4j.com/docs/graph-data-science/current/management-ops/graph-catalog-ops/.

[Neo4j(2010e)] Neo4j. Documentación de neo4j, 2010e. URL https://neo4j.com/docs/graph-data-science/current/algorithms/community/.