Curso 2025

Proyecto Final

Bases de Datos No Relacionales

Rocío Bernadet
5.147.049-2
Bruno Lombardo
5.136.781-5

Grupo 10

24 de junio de 2025

Índice

I. Introducción

El propósito principal de este trabajo, consiste en aplicar los conceptos adquiridos en la materia Bases de Datos No Relacionales, con énfasis en arquitecturas de grafos, transformación de datos y formulación de consultas.

El resultado esperado es obtener un grafo, a partir de datos en formato CSV, que permita realizar consultas de interés. Para ello utilizaremos un conjunto de datos procedente de Kaggle [1], que contiene información sobre partidos de la NBA entre 2004 y 2022. Sobre este conjunto de datos se plantean dos objetivos principales.

El primer objetivo radica en calcular la distancia entre dos jugadores cualesquiera. Esta distancia se define de manera específica: es uno si ambos participaron en el mismo partido, dos si no compartieron cancha pero existe un tercer jugador que sí lo hizo con ambos, y así sucesivamente. Esta métrica se determinará para tres casos distintos: en el caso general, se consideran conexiones entre jugadores que hayan participado en un mismo partido, sin importar si fueron compañeros o rivales; en el caso rival, solo se tiene en cuenta que hayan estado en equipos contrarios durante un partido; y en el caso de aliados, se considera que hayan sido compañeros de equipo. A partir de estas definiciones, crearemos un subgrafo que permite calcular las diferentes distancias entre cualquier par de jugadores.

El segundo objetivo se centra en la aplicación de algoritmos de análisis disponibles en la biblioteca GDS [2] (Graph Data Science) de Neo4j. Usaremos algoritmos de centralidad, con el fin de identificar a los jugadores más influyentes dentro de la red según diferentes criterios, y algoritmos de detección de comunidades, para analizar la estructura del grafo y los agrupamientos entre jugadores.

II. Antecedentes

II-A. Trabajos relacionados

Este trabajo toma como referencia dos proyectos previos. El primero [3] de ellos, realizado en 2022, generó el primer modelado de esta realidad y desarrolló un conjunto de consultas sobre el mismo. Se enfocó principalmente en analizar la distancia general, tal como fue definida anteriormente, entre Esteban Batista y otras estrellas de la NBA.

El segundo [4], presentado en 2023, amplió y mejoró el modelo anterior mediante la incorporación de nuevas entidades. Además, profundizó el análisis de distancias entre jugadores, introduciendo las nociones de distancia rival y aliada, que retomamos y adaptamos en este trabajo.

Nuestro proyecto introduce el cálculo de las distancias general, rival y aliada entre cualquier par de jugadores, junto con la construcción de un subgrafo específico que represente estas conexiones. Además, realizamos un análisis semántico de estas distancias, explorando, por ejemplo, qué significa que un jugador sea un nodo central dentro de esta red. También identificamos las comunidades en el subgrafo, permitiéndonos conocer mejor su estructura.

II-B. Conceptos claves

Para el desarrollo de este informe, es necesario comprender ciertos conceptos y algoritmos que permiten realizar análisis de grafos. A continuación, se define qué es una proyección de grafos en Cypher y los algoritmos de centralidad y detección de comunidades que aplicaremos sobre dichas proyecciones.

Algoritmo de camino más corto (Shortest Path)[5]: identifica la ruta con la menor cantidad de saltos (o aristas) entre dos nodos dentro de un grafo, utilizando internamente una búsqueda BFS (Breadth-First Search) [6]. Esta lógica se aplica únicamente a grafos no ponderados; si las aristas poseen peso, es necesario emplear los procedimientos de GDS, con algoritmos como Dijkstra [7] para obtener la ruta más corta ponderada.

Proyección en Cypher[8]: permite crear un grafo en memoria a partir del contexto de una consulta Cypher. Esta funcionalidad permite la ejecución eficiente de algoritmos avanzados sobre el grafo proyectado, gracias al acceso directo en memoria.

Algoritmos de centralidad:

- Centralidad de intermediación (Betweenness Centrality)[9]: evalúa la influencia de un nodo en el flujo de información dentro del grafo, detecta nodos que actúan como puentes entre distintas partes de la red. El algoritmo calcula las rutas más cortas entre todos los pares de nodos y asigna una puntuación a cada nodo basada en la frecuencia con la que aparece en dichas rutas. GDS implementa este algoritmo sobre grafos ponderados y no ponderados. En el caso no ponderado, utiliza una versión aproximada del algoritmo de Brandes [10], mientras que en grafos ponderados recurre a múltiples ejecuciones concurrentes del algoritmo de Dijkstra.
- Centralidad de cercanía (Closeness Centrality)[11]: permite identificar nodos que pueden difundir información de manera eficiente a través del grafo. Esta métrica se basa en la inversa de la distancia promedio entre un nodo y todos los demás. Cuanto menor es la distancia media, mayor es la centralidad de cercanía. En esta implementación no se consideran pesos, solo el número de aristas entre nodos.
- Centralidad de grado (Degree Centrality)[12]: permite identificar nodos populares en la red. Se basa en el conteo de relaciones entrantes, salientes o ambas, según la orientación definida en la proyección. Este algoritmo se puede aplicar tanto a grafos ponderados como no ponderados. En grafos ponderados, la centralidad se calcula como la suma de los pesos positivos de las relaciones adyacentes a cada nodo, ignorando aquellos que sean negativos.

Algoritmos de detección de comunidades:

- Louvain [13]: detecta comunidades a través de un proceso jerárquico de dos fases. En la primera fase, cada nodo comienza como su propia comunidad; luego, se evalúa el aumento en la modularidad que resultaría de moverlo a una comunidad vecina, realizando el movimiento más beneficioso. Este proceso se repite hasta que no se logran mejoras significativas. En la segunda fase, las comunidades encontradas se agrupan en un nuevo grafo condensado (meta-grafo), y el procedimiento se repite hasta la convergencia. Este enfoque jerárquico permite descubrir estructuras comunitarias a múltiples niveles en redes extensas. Una característica importante de Louvain es su sensibilidad al orden y a la inicialización: la asignación inicial de comunidades puede ser aleatoria o basada en el primer nodo elegido, lo que puede producir resultados no deterministas.
- Leiden [14]: hereda la estructura jerárquica de Louvain, pero incorpora una fase intermedia de refinamiento tras la fase de optimización local (la primera fase). En esta etapa, las comunidades encontradas se subdividen para garantizar que cada una sea internamente conexa, evitando la formación de comunidades con componentes desconectados. Luego, el grafo se condensa como en Louvain y se repiten las fases hasta alcanzar la estabilidad.

III. Implementación

El subgrafo para las distancias representa a los jugadores como nodos y define tres tipos de aristas según el tipo de relación entre ellos: `SHARE_GAME_WITH` para la distancia general, `ALLY` para la distancia aliada, y `RIVAL` para la distancia rival.

Cada arista tiene dos atributos: uno que indica la cantidad de partidos en los que los jugadores coincidieron bajo la condición correspondiente, y otro que representa el peso de la arista, definido como el inverso de esa cantidad. Este peso será útil para aplicar algoritmos de centralidad donde se pueden pasar ponderaciones en las aristas. En la Figura 1 se muestra un ejemplo del grafo original junto con estas nuevas aristas.

Decidimos implementar este enfoque con aristas diferenciadas en lugar de una única relación con múltiples atributos por varias razones. Primero, semánticamente cada distancia representa algo distinto, por lo que resulta más claro modelarlas como relaciones independientes. Segundo, porque la consulta para crear una única arista con todos los pesos era muy compleja. Al dividirlo en tres casos, podemos tratar cada distancia como un subproblema independiente y escribir una consulta distinta más simple para cada uno.

Además, los algoritmos de centralidad utilizados (que internamente emplean variantes del algoritmo de Dijkstra) presentan dificultades al trabajar con pesos nulos o iguales a cero. Si dos jugadores no tienen una relación de un tipo determinado, pero sí tienen alguna de las otras, deberíamos asignar un valor `NULL` o 0. Sin embargo, un `NULL` en un peso hace que el algoritmo devuelva `NaN` como resultado, y un peso 0 provoca que Dijkstra priorice esa arista, cuando en realidad, semánticamente no debería existir. Al separar cada tipo de distancia en su propio subgrafo, nos aseguramos de que todas las aristas tengan un peso mayor que cero y se evitan estos problemas.

Por lo tanto, solo se agregan aristas entre jugadores que hayan compartido al menos un partido, según el criterio correspondiente. La distancia entre dos jugadores se define entonces como la cantidad de saltos del camino más corto entre ambos dentro del subgrafo adecuado.

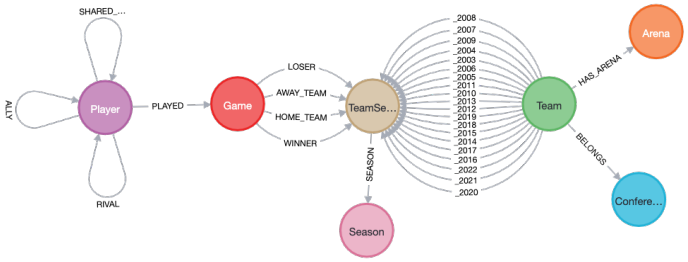

Figura 1: Ejemplo del grafo original con las nuevas aristas de tipo general, aliada y rival entre jugadores.

La construcción de cada uno de los tipos de aristas se realizó con las siguientes consultas:

```
MATCH (p1:Player)-[pl1:PLAYED]->(g:Game)<-[pl2:PLAYED]-(p2:Player)
WHERE elementId(p1) < elementId(p2)
WITH p1, p2, count(g) AS shared_games
MERGE (p1)-[r:SHARED_GAME_WITH]->(p2)
SET r.weight = 1.0/shared_games, r.shared_games = shared_games
```

Creación de aristas para el caso general.

```
MATCH (p1:Player)-[pl1:PLAYED]->(g:Game)<-[pl2:PLAYED]-(p2:Player)
WHERE elementId(p1) < elementId(p2) AND pl1.team = pl2.team
WITH p1, p2, count(g) AS shared_games_as_ally
MERGE (p1)-[r:ALLY]->(p2)
SET r.weight = 1.0/shared_games_as_ally, r.shared_games_as_ally = shared_games_as_ally
```

Creación de aristas para el caso aliado.

```
MATCH (p1:Player)-[pl1:PLAYED]->(g:Game)<-[pl2:PLAYED]-(p2:Player)
WHERE elementId(p1) < elementId(p2) AND pl1.team <> pl2.team
WITH p1, p2, count(g) AS shared_games_as_rival
MERGE (p1)-[r:RIVAL]->(p2)
SET r.weight = 1.0/shared_games_as_rival, r.shared_games_as_rival = shared_games_as_rival
```

Creación de aristas para el caso rival.

Una vez agregadas las aristas, ya podemos calcular cualquiera de las distancias entre cualquier par de jugadores. Para eso, utilizaremos la primitiva `ShortestPath` para obtener el camino más corto entre ellos en el subgrafo correspondiente, y por lo tanto la distancia.

```
MATCH p = shortestPath((p1:Player {name:"LeBron James"})-[:SHARED_GAME_WITH*]-(p2:Player {name: "Kyrie Irving"}))
RETURN p
```

Consulta para obtener la distancia general entre LeBron James y Kyrie Irving.

Figura 2: Resultado de la consulta: camino más corto entre LeBron James y Kyrie Irving para la distancia general, con una distancia resultante de 1.

Figura 3: Resultado de la consulta: camino más corto entre Stephen Curry y Kyle Korver para la distancia aliada. Como no jugaron juntos, el camino incluye un jugador intermedio que compartió equipo con ambos, resultando en una distancia de 2.

Para poder aplicar los algoritmos de la biblioteca GDS, primero es necesario construir un grafo proyectado. Como ejemplo, en esta sección vamos a crear ese grafo y aplicar los algoritmos sobre la distancia general. Para las otras dos distancias (aliada y rival), el procedimiento es exactamente el mismo, simplemente cambiando el tipo de relación usada en la proyección.

```
CALL gds.graph.project(
    'distanceGraph',
    'Player',
    {
        SHARED_GAME_WITH: {
                    orientation: 'UNDIRECTED',
                    properties: 'weight'
        }
    }
)
YIELD graphName, nodeCount, relationshipCount;
```

Creación del grafo proyectado para la distancia general.

Como en el grafo original solo agregamos una arista dirigida por par de jugadores para evitar redundancia, debemos declarar la relación como no dirigida en la proyección. De esta forma, GDS la trata como una relación simétrica y calcula las distancias correctamente.

Sobre este grafo proyectado aplicaremos tanto algoritmos de centralidad como de detección de comunidades.

Los resultados de las centralidades se presentan en el cuadro I.

Cuadro I: Comparación de los algoritmos de centralidad sobre la distancia general.

Betweenness Centrality

Jugador	Puntaje
LeBron James	74949.88
Jamal Crawford	72305.65
Kyle Korver	66967.08
Jeff Teague	62555.33
LaMarcus Aldridge	61787.76
Dirk Nowitzki	61424.14
Udonis Haslem	60960.33
James Harden	58319.28
Marvin Williams	57050.13
JJ Redick	56404.37

Degree Centrality

Jugador	Puntaje
Udonis Haslem	487.71
Kyle Korver	483.91
Tyson Chandler	474.43
Dwight Howard	469.17
Dwyane Wade	458.88
Pau Gasol	457.96
Vince Carter	454.61
Trevor Ariza	442.55
Dirk Nowitzki	439.64
Joe Johnson	432.12

Closeness Centrality

Jugador	Puntaje
Udonis Haslem	0.7768
Kyle Korver	0.7751
Vince Carter	0.7736
LeBron James	0.7603
Tyson Chandler	0.7595
Pau Gasol	0.7590
Dwyane Wade	0.7562
Zaza Pachulia	0.7541
Nene	0.7541
Dirk Nowitzki	0.7536

Como comentario general, podemos observar que varios jugadores aparecen en más de una de las tres listas, lo cual tiene sentido, ya que las distintas centralidades capturan aspectos relacionados en el grafo. En particular, los jugadores con alta participación a lo largo de muchas temporadas o que han pasado por varios equipos tienden a ocupar posiciones centrales bajo las diferentes definiciones.

Si observamos el caso de Betweenness Centrality, el top 1 es LeBron James. Esto es coherente con su trayectoria: jugó durante todas las temporadas incluidas en el dataset, con una alta cantidad de minutos y participación en múltiples equipos. Su rol como conector entre distintas "generaciones" de jugadores lo ubica naturalmente como un puente en la red, lo que se refleja en su alto puntaje en esta métrica.

Sin embargo, llama la atención que LeBron no aparece entre los diez primeros en Degree Centrality, que mide cuántos jugadores distintos están conectados directamente con cada nodo. Esto puede explicarse porque esta métrica solo considera la cantidad de conexiones inmediatas, sin tener en cuenta el rol del jugador en el resto del grafo. Jugadores como Udonis Haslem, Kyle Korver o Vince Carter aparecen en los primeros puestos probablemente porque compartieron equipo con muchos jugadores diferentes a lo largo de su carrera, aunque no hayan tenido un protagonismo tan alto. Para confirmar esta hipótesis, ejecutamos la siguiente consulta en Cypher, que permite contar cuántos jugadores distintos están conectados directamente con cada uno de estos nombres:

```
UNWIND ["Udonis Haslem", "Kyle Korver", "Vince Carter", "LeBron James"] AS playerName
MATCH (p:Player {name: playerName})--(other:Player)
RETURN playerName AS nombre, count(DISTINCT other) AS conexiones
ORDER BY conexiones DESC;
```

El resultado se presenta en la siguiente tabla:

Cuadro II: Cantidad de conexiones directas de algunos jugadores destacados.

Nombre	**Conexiones**
Udonis Haslem	1510
Kyle Korver	1504
Vince Carter	1499
LeBron James	1451

Estos datos confirman nuestra interpretación: aunque LeBron tiene un rol central en otras métricas como Betweenness Centrality, su cantidad de conexiones directas es ligeramente menor, lo que explica por qué no lidera en Degree Centrality.

Por otro lado, en Closeness Centrality, que mide qué tan cerca está un jugador del resto del grafo en promedio, LeBron sí vuelve a aparecer, aunque no en la primera posición. Esto refleja que, aunque no tenga el mayor

número de conexiones directas, sí está bien posicionado dentro de la red general, con caminos cortos hacia la mayoría de los jugadores.

Para analizar las comunidades, consideramos los tres grafos proyectados, uno por cada tipo de distancia. Observamos en el cuadro III, que los grafos correspondientes a la distancia general y a la distancia rival, presentan muchas menos comunidades que el de la distancia aliada. Esto era esperable, ya que la condición para que exista una arista entre dos jugadores es más débil en el ultimo caso. En el grafo de distancia rival basta con que los jugadores se hayan enfrentado en un mismo partido, algo que puede darse con frecuencia. En la distancia general, alcanza con haber coincidido en cancha, sin importar si fueron aliados o rivales. En cambio, en el grafo de distancia aliada, los jugadores deben haber sido compañeros de equipo, una condición más estricta y menos común.

Esta diferencia en las condiciones genera grafos con estructuras distintas: los de distancia general y rival tienden a estar más densamente conectados, lo que lleva a una menor cantidad de comunidades detectadas. En cambio, el grafo de aliados presenta una estructura más dispersa y segmentada.

Antes de analizar los resultados de la distancia aliada, esperábamos encontrar alrededor de 30 comunidades, una por cada equipo de la liga. Sin embargo, no habíamos considerado que los equipos cambian sus planteles constantemente: los jugadores se transfieren de un equipo a otro, lo cual genera nuevas conexiones entre equipos a lo largo del tiempo. Por eso, tiene sentido que el número de comunidades detectadas sea menor a 30, ya que esas conexiones entre jugadores que pasaron por distintos equipos tienden a unir comunidades que originalmente habrían estado separadas.

Cuadro III: Cantidad de comunidades detectadas por los algoritmos Leiden y Louvain.

Tipo de distancia	**Leiden**	**Louvain**
General	3	3
Aliada	20	19
Rival	3	3

En el siguiente repositorio de GitHub [15] se puede ver el contenido completo del proyecto, incluyendo los scripts de construcción.

IV. Experimentación

Durante el desarrollo del trabajo, enfrentamos diversas restricciones técnicas que condicionaron el proceso experimental. En primer lugar, utilizamos Neo4j Aura en su versión gratuita, la cual impone límites de hasta 200.000 nodos y 400.000 relaciones por clúster. Esta restricción nos obligó a reducir el rango temporal de los datos originales para poder ejecutar consultas básicas. Sin embargo, esta reducción comprometía la integridad del análisis, por lo que optamos por instalar y utilizar Neo4j de forma local, lo cual nos permitió trabajar con el conjunto completo de datos sin restricciones de espacio.

Además del límite en la cantidad de nodos y relaciones, la versión gratuita de Aura no incluye el módulo de Graph Data Science (GDS), por lo que no era posible ejecutar los algoritmos directamente en dicha plataforma. Esta fue otra razón determinante para migrar el entorno de trabajo a una instalación local.

Una vez superadas estas limitaciones, comenzamos la experimentación con algoritmos de detección de comunidades. En particular, probamos el algoritmo Louvain, pero observamos que presentaba resultados inconsistentes al ejecutarse múltiples veces. Esta inconsistencia se explica por su naturaleza estocástica, ya que el algoritmo depende de la elección aleatoria del nodo inicial. Por ejemplo, al aplicarlo sobre el grafo de jugadores aliados, los resultados variaban entre 11 y 20 comunidades en distintas ejecuciones, lo que evidencia una falta de estabilidad en la detección.

Con el objetivo de obtener resultados más robustos, exploramos el uso del algoritmo Leiden, una versión mejorada de Louvain que garantiza que cada comunidad identificada sea conexa. Esta mejora metodológica permite mitigar algunos de los inconvenientes del algoritmo original, como se explica en la Sección II-B. En nuestros experimentos, Leiden mostró una mayor estabilidad, arrojando entre 18 y 21 comunidades en sucesivas ejecuciones, con un rango de variación considerablemente menor que Louvain.

V. Conclusiones y trabajo futuro

El objetivo principal de este proyecto, que consistía en transformar archivos CSV a una estructura de grafos e implementar consultas Cypher sobre ella, se cumplió satisfactoriamente. No solo se ejecutaron consultas básicas, sino que también se exploraron y aplicaron bibliotecas avanzadas como Graph Data Science (GDS), lo que permitió la aplicación de algoritmos complejos sobre el grafo desarrollado.

Uno de los aportes más significativos fue la definición de distintas nociones de distancia (general, aliada y rival) y la construcción de subgrafos proyectados específicos para cada una. Esta metodología posibilitó la aplicación de algoritmos de centralidad y de detección de comunidades de manera más precisa y semánticamente coherente.

Durante el desarrollo del proyecto, se enfrentaron diversas dificultades técnicas. Inicialmente, el uso de Neo4j Aura se vio limitado por sus restricciones de capacidad y funcionalidad. Esta situación llevó a la decisión de migrar a una instalación local de Neo4j, lo cual fue fundamental para poder trabajar con el conjunto de datos completo y aplicar los algoritmos de la biblioteca GDS.

Respecto al trabajo futuro, una posible línea sería analizar las centralidades en las dos distancias restantes (Rival y Aliada), que no fueron incluidas en este informe. Incluso, se podría comparar los distintos tipos de centralidad entre las tres variantes de distancia para detectar patrones o diferencias significativas.

Además, sería interesante profundizar en el análisis temporal del grafo, estudiando cómo evolucionan las conexiones entre jugadores a lo largo de las temporadas. También se podrían incorporar atributos adicionales a los nodos o relaciones —como la posición del jugador, estadísticas por minuto o por cuarto— y explorar visualizaciones más ricas y dinámicas que faciliten una comprensión más profunda de la estructura y dinámica de la red.

En resumen, se lograron alcanzar los objetivos establecidos para el proyecto, se aportaron mejoras en el modelado y análisis del grafo.

Referencias

[1] Nathan Lauga. *NBA Games Dataset*. Accedido en junio de 2025. 2021. URL: https://www.kaggle.com/datasets/nathanlauga/nba-games.

[2] Neo4j. *Graph Data Science Documentation*. Accedido en junio de 2025. 2024. URL: https://neo4j.com/docs/graph-data-science/current/.

[3] FING - Universidad de la República. *NBA Neo4j - Esteban Batista*. Material de curso, accedido en junio de 2025. 2024. URL: https://eva.fing.edu.uy/pluginfile.php/463701/mod_folder/content/0/NBA%20neoj4%20Esteban%20Batista.pdf?forcedownload=1.

[4] FING - Universidad de la República. *NBA Estadísticas - BDNR4*. Material de curso, accedido en junio de 2025. 2025. URL: https://eva.fing.edu.uy/pluginfile.php/548193/mod_folder/content/0/NBA%20Estadisticas%20BDNR4.pdf?forcedownload=1.

[5] Neo4j. *Shortest Path Patterns - Cypher Manual*. Accedido en junio de 2025. 2024. URL: https://neo4j.com/docs/cypher-manual/current/patterns/shortest-paths/.

[6] Neo4j. *Breadth-First Search (BFS) - Graph Data Science*. Accedido en junio de 2025. 2024. URL: https://neo4j.com/docs/graph-data-science/current/algorithms/bfs/.

[7] Neo4j. *Dijkstra's Algorithm - Single Source Shortest Path*. Accedido en junio de 2025. 2024. URL: https://neo4j.com/docs/graph-data-science/current/algorithms/dijkstra-single-source/.

[8] Neo4j. *Cypher Projection - Graph Creation*. Accedido en junio de 2025. 2024. URL: https://neo4j.com/docs/graph-data-science/current/management-ops/graph-creation/graph-project-cypher-projection/.

[9] Neo4j. *Betweenness Centrality - Graph Data Science*. Accedido en junio de 2025. 2024. URL: https://neo4j.com/docs/graph-data-science/current/algorithms/betweenness-centrality/.

[10] Ulrik Brandes y Patrick Pich. *Centrality Estimation in Large Networks*. Accedido en junio de 2025. 2007. URL: https://www.uni-konstanz.de/mmsp/pubsys/publishedFiles/BrPi07.pdf.

[11] Neo4j. *Closeness Centrality - Graph Data Science*. Accedido en junio de 2025. 2024. URL: https://neo4j.com/docs/graph-data-science/current/algorithms/closeness-centrality.

[12] Neo4j. *Degree Centrality - Graph Data Science*. Accedido en junio de 2025. 2024. URL: https://neo4j.com/docs/graph-data-science/current/algorithms/degree-centrality/.

[13] Neo4j. *Louvain Modularity Optimization - Graph Data Science*. Accedido en junio de 2025. 2024. URL: https://neo4j.com/docs/graph-data-science/current/algorithms/louvain/.

[14] Neo4j. *Leiden Community Detection - Graph Data Science*. Accedido en junio de 2025. 2024. URL: https://neo4j.com/docs/graph-data-science/current/algorithms/leiden/.

[15] Rocío Bernadet y Bruno Lombardo. *Repositorio del proyecto BDNR Grupo 10 (2025)*. Accedido en junio de 2025. 2025. URL: https://github.com/robernadet/BDNR_Grupo10_2025.git.