UNIVERSIDAD NACIONAL DE CÓRDOBA

TRABAJO FINAL DE GRADO

Exploración y visualización de redes de músicos

En cumplimiento para la obtención del título de grado de Licenciado en Ciencias de la Computación de la Facultad de Matemática, Astronomía, Física y Computación

Tesista

Illak Zapata

Bajo la dirección de

Damián Barsotti

Mariano Barsotti

Esta obra está bajo una Licencia Creative Commons Atribución-CompartirIgual 4.0 Internacional .

Resumen

En la actualidad la forma más común de representar redes sociales es forma de grafos. De esta manera mediante la visualización de dicho grafo se puede comunicar información representando a personas como nodos y relaciones como aristas.
El problema surge cuando la cantidad de datos que se quiere representar es demasiada y por lo tanto nos encontramos con grafos que tienen millones de nodos y aristas. De esta manera resulta imposible extraer información esencial. Si la persona que visualiza este tipo de grafos no puede responder a preguntas básicas como ¿Qué nodos están en la red?, ¿Qué roles tienen?, ¿De qué manera se relacionan?, entonces mucho menos podrá detectar la información que queremos comunicar, es decir, información específica al tipo de red que se esté estudiando como por ejemplo: distancias más cortas entre nodos, relevancia o influencia de los mismos, etc.
Con el fin de responder a las preguntas previas, el trabajo se llevará a cabo en las siguientes fases, primero crearemos una red de músicos, luego analizaremos dicha red y finalmente procederemos con la visualización de la misma, siempre teniendo como objetivo final la creación de una visualización que resulte amigable y donde la información que queremos comunicar sea fácil de identificar por el usuario final.

Dedicatoria

A mi mamá María, por el constante apoyo, por sus valores, por su amor infinito.

A mi papá Pedro, por los ejemplos de lucha y perseverancia que me transmite día a día.

A mis hermanos Nahuel y Aimé, por los momentos compartidos y por que sé que siempre puedo contar con ellos, en las buenas y en las malas.

A mi director Damián Barsotti, por su apoyo, por su motivación y por su infinita paciencia, pero también por su sabiduría y conocimientos transmitidos.

A mi segundo director Mariano Barsotti, por su visión experta en el campo del jazz y por ayudar en la difusión de este trabajo.

Índice

1. Introducción y motivación

1.1. Problemática

En la actualidad, la abundancia y disponibilidad de datos contribuyen a solucionar diversos problemas en distintos ámbitos. Estos datos son recopilados y analizados en grandes volúmenes con el objetivo de interpretarlos y buscar patrones que resulten significativos. Gran parte del uso de estos datos se ha enfocado particularmente al análisis de redes, un área extensa de conocimiento cuyo objeto de estudio puede sintetizarse en la extracción de propiedades sobre un conjunto de entidades relacionadas entre sí. Las entidades junto con sus relaciones se denominan redes y abarcan objetos muy disímiles como redes sociales, de transporte, eléctricas, biológicas, epidemiológicas, etc.
Hoy en día gran parte del análisis de redes se ha enfocado particularmente al estudio de las redes sociales cuyos campos de aplicación van desde la sociología hasta el marketing empresarial.
La visualización de datos es un elemento crucial a la hora de interpretar los resultados obtenidos en el análisis de redes. La misma permite al usuario interactuar directamente con los datos facilitando su conocimiento y comprensión, y ayudando a la toma de decisiones. En este sentido, un problema ubicuo es el de seleccionar una gran cantidad de datos y filtrarlos con la finalidad de lograr una percepción directa y elocuente de ciertas características.

1.2. Motivación

El Festival de Jazz de la Provincia de Córdoba[1] es un evento organizado por la *Agencia Córdoba Cultura del Gobierno de la Provincia de Córdoba*[2] donde se desarrolla una programación musical de gran calidad. Es así que en las distintas ediciones del festival se contó con la presencia de músicos de jazz invitados de reconocida trayectoria internacional. La importancia de dichos músicos en el ámbito de la música y más específicamente del jazz, tornan al festival en uno de los eventos artísticos más importantes de la Provincia.
Si bien la propuesta cultural goza de gran prestigio entre el público entendido no siempre tiene una difusión masiva que permita evaluar la jerarquía de la programación del Festival. Por tal motivo en este trabajo se busca reflejar la importancia de los artistas invitados a través de una forma visual relacionándolos con artistas relevantes del ámbito del jazz, estableciendo caminos entre ellos y visualizando sus métricas en la red principal. Lo que se intenta zanjar, es la problemática de la proliferación diseminada de la información y así concentrar el análisis para volver relevante aquello inicialmente disperso. De este modo se intenta que el Festival de Jazz de la Provincia de Córdoba cuente con una herramienta que le permita acceder a un público mayor.

1.3. Propuesta

En el presente trabajo se propone la creación de un sistema de visualización de redes de músicos que permita representar datos biográficos o discográficos de los mismos, pero más importante aún que permita comunicar relaciones y afinidades entre ellos de una forma interactiva.

[1] "Córdoba Jazz Festival - CBA JAZZ | 2017 | Festival Internacional de ...." http://cordobajazzfestival.com.ar/2017/.
[2] "Agencia Córdoba Cultura | Gobierno de la ...." http://cultura.cba.gov.ar.

Como tarea inicial, se propone la creación de un flujo de datos provenientes de diversas fuentes con información sobre artistas musicales y discos. Parte importante del proceso consistirá en la limpieza de los datos y posteriormente la unificación de estos con el fin de generar una base de datos propia desde la cual se pueda obtener información relevante o realizar análisis sobre la misma. Una vez generada la base de datos el siguiente paso consistirá en la creación de un grafo que represente una estructura social, donde los actores principales serán artistas musicales y la relación que los une será la de "tocó con". Debido a la cantidad de datos (artistas y relaciones), el grafo resultante es tan grande que la extracción de información relevante del mismo se vuelve compleja o casi imposible. Es por esto, que el siguiente paso será la reducción del tamaño del mismo. Para ello, en una primera instancia, se obtendrán los caminos más cortos desde artistas invitados a una edición del festival hacía artistas relevantes del ámbito del jazz.
Debido a que la cantidad de caminos que se obtienen mediante este algoritmo pueden llegar a ser demasiados, el siguiente paso consistirá en filtrarlos mediante la asignación de rankings basados en la influencia colectiva de los artistas que conformen cada camino. Esta métrica da una noción de importancia del nodo en el sentido de que su eliminación de la red fragmenta eficientemente la misma en pequeños fragmentos. Para el cálculo de dicha métrica utilizaremos la fórmula propuesta en el trabajo de Morone Flaviano *et al* sobre la búsqueda de *influencers* en redes sociales masivas [1].
Además se propone una mejora del método de asignación de rankings mediante una modificación original del cálculo anterior que incluye la valoración del artista por parte de un experto.
Este filtro resultará fundamental para la reducción de dimensiones del grafo ya que al usar la influencia colectiva de los artistas nos aseguraremos de quedarnos con aquellos caminos que sean relevantes y que contengan artistas de importancia en la comunidad.
Finalmente con el grafo reducido se procederá con la visualización del mismo con el fin de poder comunicar al usuario final información como las distancias que separan a artistas invitados de artistas relevantes, otorgando de esta manera una visión de la importancia de cada artista invitado. El medio elegido para la visualización de los datos es el navegador, de esta manera los datos podrán ser visualizados desde una página web permitiendo que el usuario final pueda acceder fácilmente a los resultados.

1.4. Trabajo relacionado

A continuación se listan algunos de los trabajos relacionados al presente y que de alguna manera sirvieron de inspiración:

- Linked Jazz [2]: Es un proyecto de investigación en tecnologías de datos abiertos vinculados aplicados a materiales de patrimonio cultural digital. El objetivo es descubrir relaciones entre documentos y datos relacionados a la vida personal y profesional de artistas de jazz. Utiliza materiales históricos en formato digital para exponer relaciones entre músicos y revelar su red comunitaria.

- *Social Network Visualization: Can We Go Beyond the Graph?* [3] : Paper sobre visualización de redes sociales y sus limitaciones con respecto a dimensiones y límites.

- *Community Structure in Jazz* [4]: En este paper se estudia una red de colaboraciones entre músicos de jazz. Primero se estudia la red colaborativa entre individuos, donde dos músicos están conectados si tocaron en la misma banda. Luego se considera la

colaboración entre bandas, donde dos bandas están conectadas si tienen un músico en común. Estas construcciones capturan elementos esenciales en las interacciones sociales entre músicos de jazz.

1.5. Organización de tesis

A continuación se describe la organización del trabajo, comenzando por el marco teórico y luego haciendo una descripción detallada de las etapas realizadas desde la recolección de datos, procesamiento y análisis, hasta la visualización de los mismos:

- En el **capítulo 2** (marco teórico) se desarrollará el marco teórico necesario para la comprensión de la tesis. Se definen conceptos sobre los que se basa la tesis como *Grandes Volúmenes de Datos*, *Análisis de redes sociales y ETL.* Además se detallan aspectos técnicos y ventajas de la herramienta utilizada para acceder a los datos y analizarlos.
- En el **capítulo 3** (trabajo realizado) se describen las distintas etapas del desarrollo del sistema, desde la descripción del proceso *Extract, Transform, Load* (ETL) de datos y de cada etapa que lo compone, la creación de un grafo general que será usado para representar la estructura social que será objeto de estudio y finalmente la visualización de resultados.
- Finalmente en el **capítulo 4** (conclusión) se realiza una conclusión final del trabajo, seguida de una breve descripción de posibles mejoras al sistema y trabajo futuro.

1.6. Enlaces a repositorio y visualización

Todos los programas con documentación y explicación del flujo de trabajo se encuentran disponibles en el siguiente repositorio:

- https://bitbucket.org/bigdata_famaf/tesis_illak

Enlace a demo con las distintas visualizaciones:

- http://www.famaf.unc.edu.ar/∼ ilz0111/tesina/viz/

2. Marco Teórico

2.1. Grandes Volúmenes de Datos

En la actualidad los datos crecen más rápido de lo que lo hacen las velocidades de procesamiento. Para hacernos una idea de cuan grandes puede llegar a ser la cantidad de estos datos pensemos las siguientes analogías:

- 1 Petabyte = 1000 Terabytes = 20 millones de archivos de 4 cajones llenos de texto
- 5 Exabytes = 5000 Petabytes = todas las palabras habladas por la humanidad
- 1 Zettabyte = 1000 Exabytes (1 billón de Gigabytes)
- 1 Yottabyte = 1000 ZettaBytes = la totalidad de **internet**

Pero, ¿dónde se generan estos datos en la vida real? Algunos ejemplos que ilustran dónde se producen, son[3]:

- En 2012 el colisionador de partículas del CERN generó 40TB/seg.
- Un Airbus A380 genera 640 TB por vuelo.
- Twitter genera 12TB de datos por dia (o 360 TB por mes).
- La bolsa de New York genera 1 TB por dia.
- Una cosechadora genera 5.000 datos por hectárea en cada pasada. Un dron genera 50.000 datos por hectárea en cada pasada. En Argentina hay alrededor de 30.000.000 hectáreas cultivadas.

Además, estos datos se duplican cada dos años y si tenemos en cuenta que la mayoría de ellos son del año 2014 podemos hacer una suposición válida de que en la actualidad estas cantidades se han duplicado.
Este crecimiento es producido por diferentes factores, entre los cuales podemos destacar:

- El continuo crecimiento en el uso de Internet y las redes sociales.
- El smartphone como generador de datos personales (sensores, apps, etc).
- Migración de TV analógica a la TV digital.
- Imágenes generadas por máquinas como por ejemplo cámaras de seguridad.
- Crecimiento de la información sobre información (metadatos).

[3] "Big Data: ¡Qué Grande Sos! - Maestría en Data Mining." 15 may.. 2014, http://datamining.dc.uba.ar/datamining/files/Charlas_y_Paneles/p1_efeuerstein.pdf.

Se estima que en el año 2020 cada persona podría generar 1.7 megabytes de datos cada segundo de cada día, aportando a los 44 zettabytes (o 44 trillones de gigabytes) de datos existentes al universo digital para entonces[4].

Como podemos ver, estos conjuntos de datos son tan grandes que aplicaciones informáticas tradicionales de procesamiento de datos no son suficientes para tratar con ellos.
Entre las dificultades que encontramos en la gestión de grandes volúmenes de datos, podemos destacar la *recolección, almacenamiento, búsqueda, análisis y visualización*, siendo este último un tema de suma importancia en el presente trabajo final. Por tal motivo, desarrollaremos más adelante el mencionado aspecto.

2.2. Formas de procesamiento

A continuación se hará un repaso de posibles soluciones que dan respuesta al interrogante sobre cómo tratar el procesamiento de gran cantidad de datos.

- Uso de *supercomputadoras*: un ejemplo es SGI UV 3000[5]. No escalan debido a su capacidad de memoria RAM que puede llegar hasta 64TB (recordemos que Twitter genera 360TB por mes, el Airbus 640TB por vuelo, etc) además de su costo elevado (alrededor de los 4 millones de dólares).

- Scale-out NAS: son servidores de almacenamiento que pueden soportar arreglos de discos de 50PB. El acceso de los datos por red conlleva el inconveniente que la red es lenta y podemos suponer que en un futuro seguirá siendo así debido a las leyes de Kryder[6] y Nielsen[7] que básicamente establecen que las densidades de almacenamiento aumentan mucho más rápido que las velocidades de red.

- Discos locales en racks: con ello se aumenta la velocidad de acceso a los datos pero tampoco es útil debido nuevamente, a su capacidad limitada (en un servidor U2 con capacidad de 12 discos de 4TB cada uno, tenemos un total de 48TB).

- Clusters de computadoras: es la solución utilizada para paralelizar el procesamiento de grandes volúmenes de datos. El término cluster hace referencia a un conjunto de ordenadores unidos entre sí normalmente por una red de alta velocidad y que se comportan como si fueran una única computadora. En el contexto de datos, una arquitectura posible consiste en que cada *nodo* (o computadora) posea una parte de los datos y solamente procesa la información que tiene, esta es la arquitectura más común. De esta forma, el procesamiento es realizado de manera simultánea entre computadoras para lograr resultados lo más rápido posible y de manera escalable (esto debido al bajo costo de las computadoras y discos). A tal fin, fueron desarrollados distintos paradigmas para la llamada computación distribuida como por ejemplo *Network Programming (MPI)* y *Data flow programming* que veremos más

[4] "Executive Summary: Data Growth, Business Opportunities, and the IT ...." https://www.emc.com/leadership/digital-universe/2014iview/executive-summary.htm.
[5] "SGI UV 3000, UV 30: Big Brains for No-Limit Computing." https://www.risc.jku.at/projects/mach2/4555.pdf.
[6] "Mark Kryder - Wikipedia." https://en.wikipedia.org/wiki/Mark_Kryder.
[7] "Nielsen's Law of Internet Bandwidth - Nielsen Norman Group." 5 abr. 1998, https://www.nngroup.com/articles/law-of-bandwidth/.

adelante y que es el enfoque que sigue Spark, el framework elegido para realizar la mayor parte de nuestro trabajo.

2.3. Características de los datos masivos

Los datos masivos pueden describirse por las siguientes características:

- Volumen: la cantidad de datos generados
- Variedad: el tipo y naturaleza de los datos
- Velocidad: en que se generan y procesan los datos
- Veracidad: la calidad de los datos

Además, dichos datos pueden provenir de diversas fuentes:

- Internet y móviles
- Internet of Things
- Recopilados por empresas especializadas
- Datos experimentales
- Otras

Hay que destacar que la mayoría de los datos hoy en dia los fabricamos directa e indirectamente segundo tras segundo. La cantidad de datos generados por persona y en unidad de tiempo es muy grande. Dentro de la categoría Internet y móviles, contribuye el hecho de enviar correos electrónicos o mensajes por whatsApp, publicando nuestro estado en Facebook, relaciones laborales en Linkedin, tuiteando contenido y muchas otras acciones que realizamos a diario y que crean nuevos datos y metadatos[8].

A su vez dichos datos pueden venir en tres tipos:

- Datos estructurados: se refieren a la información con un alto grado de organización, de modo que la inclusión en una base de datos relacional es directa y fácil de extraer mediante algoritmos de búsqueda sencillos.
- Datos no estructurados: carecen de un formato específico y generalmentes son datos desorganizados. Si bien estos datos pueden tener una estructura interna, la misma no es producto de un esquema o modelo de datos predefinidos.

[8] "Metadatos - Wikipedia, la enciclopedia libre." https://es.wikipedia.org/wiki/Metadatos.

- Datos semiestructurados: describen los objetos y las relaciones que se pueden inferir de los mismos. Estos últimos son los tipos de datos mayormente manejados en este trabajo (*XML y JSON).*

Una vez fijada las fuentes de los datos, es posible que se dispongan de tablas que posiblemente no tengan relación alguna entre ellas, por lo tanto el siguiente objetivo es hacer que los datos se puedan recoger de un mismo lugar y darles un formato. Para lograr lo antes mencionado se realiza el proceso de *Extracción, Transformación y Carga* (ETL) el cual se describe detalladamente a continuación.

2.4. ETL

Del inglés *Extract, Transform and Load* (extraer, transformar y cargar), es el proceso que nos permite mover datos de diferentes fuentes, re-formatearlos, limpiarlos, unificarlos y cargarlos en otra base de datos para luego ser analizados.

La primera parte del proceso ETL, *extracción*, consiste en extraer los datos de los sistemas de origen. El principal objetivo de esta etapa es obtener todos los datos requeridos desde los sistemas de origen usando la menor cantidad de recursos posible. Debemos tener en cuenta que cada sistema puede usar una organización de los datos o formatos distintos. Algunos formatos provistos por las fuentes de datos incluyen bases de datos relacionales, XML, JSON, Parquet, etc., además de datos no estructurados.
Una extracción de datos correcta asienta las bases para el éxito de los procesos subsecuentes.

La etapa de *transformación* consiste en aplicar una serie de reglas o funciones (por ejemplo *map*) a los datos extraídos con el fin de prepararlos para su carga en el formato final deseado. Esta etapa asegura la calidad de los datos en el sistema final. Algunas de estas transformaciones consisten en:

- Selección de columnas para cargar (no considerar columnas con valores *null*).
- Estandarización (números de teléfono, convertir valores null a *No disponible/No provisto, etc.).*
- Unificar identificadores (categorías de sexo)
- Ordenar datos.
- Agregación (total de una columna de ventas).
- *Encoding* (generalmente para machine learning).

Este paso, además, requiere hacer *joins* con otros datos de diversas fuentes.

Por último, la fase de *carga* en la cual los datos son guardados en un repositorio final, el cual puede ser desde simplemente un *"flat file"* (texto plano o archivo binario) o un Parquet, hasta una base de datos en un *data warehouse* (almacén de datos).
En este punto se puede hacer un volcado directo de los datos manteniendo un historial de fechas de inserción o se pueden sobrescribir los datos con la nueva información.

A lo largo del proceso de ETL debería ser posible el reinicio de alguna de las fases independientemente de las otras. Por ejemplo si la etapa de *Transformación* falla, no debería ser necesario reiniciar la etapa de *Extracción.* Podemos asegurar este comportamiento implementando un *staging* apropiado. *Staging* significa que los datos son volcados en un lugar (llamado *Staging Area*) en el que puedan ser leídos por la siguiente fase de procesamiento. Este *Staging Area* es usado durante el proceso ETL también para almacenar resultados intermedios del procesamiento. Sin embargo el *Staging Area* no debería ser accesado por ningún proceso más que por el proceso de carga y no debería estar disponible para usuarios finales ya que puede poseer datos incompletos o a medio procesar. Es por esto que el diseño de un sistema ETL puede resultar complejo con el fin de evitar problemas operativos importantes.

En este trabajo se usó el formato *Parquet* del ecosistema Apache Hadoop para la etapa de carga el cual es un formato libre y de código abierto de almacenamiento de datos orientado en columnas. Provee compresión de datos y esquemas de *encoding* eficientes. El rendimiento en la consulta de estas tablas se ve incrementado particularmente en *datasets* de gran tamaño, debido a que salva el cuello de botella que produce el acceso a disco en entornos distribuidos.
Este tipo de formato puede optimizar de manera drástica las cargas de trabajo especialmente para herramientas que tienden a leer solo segmentos de registro en lugar de todo (algo que es común en *MapReduce*). Todos los datos dentro de una misma columna tienen el mismo tipo, lo cual es ideal para al momento de comprimir y así disminuir el costo de almacenamiento, además la compresión permite que operaciones columnares (SUM, COUNT, AVG, etc) se realicen rápidamente. Finalmente, al igual que otras bases de datos no relacionales, las bases de datos orientados en columnas fueron diseñadas para que escalen usando clusters distribuidos de hardware de bajo presupuesto para incrementar el rendimiento, haciéndolas ideales para *data warehousing y procesamiento de Big Data.* Se dice que escalan de manera "horizontal" en el sentido en que para agregar mayor capacidad, el administrador de la base de datos puede simplemente agregar servidores y la base de datos automáticamente distribuye los datos a través de los servidores. A diferencia de las bases de datos relacionales que almacenan los datos en formato orientado a filas y que escalan de manera "vertical" en el sentido de que un único servidor debe ser mejorado de forma incremental para soportar mayor demanda. Si bien es posible "repartir" una base de datos SQL en muchos servidores, esto requiere de mucha ingeniería y a veces tiene muchos inconvenientes y pérdida de beneficios inherentes al modelo relacional (la capacidad de ejecutar JOINS, integridad referencial, transacciones).
Por dichos beneficios, este formato es adoptado en herramientas de *Big Data* como Apache Spark y Hadoop, de los cuales se hablará más adelante.

2.5. Data flow programming vs network programming tradicional

El paradigma de programación de network programming (pasaje de mensajes o MPI [5]), trae consigo varios problemas que surgen a escala:

- El programador tiene que administrar la localidad de los datos y el código a través de los nodos (hay que escribir programas para cada máquina).
- Es necesario lidiar con las fallas de hardware lo cual puede generar pérdida de datos y la necesidad de resetear la computación (problemas de resiliencia).
- Hay que tener en cuenta nodos con hardware más lento que otros: "*Stragglers*"
- La velocidad de comunicación de datos a través de la red es lenta.
- Leer datos de un sistema de archivos distribuido es más lento que cargar datos de memoria

El primer modelo de data flow popular, MapReduce [6] , ofrecía una solución a la mayoría de estos problemas y fue la elección principal de los programadores hasta alrededor del 2012.
En el modelo MapReduce, el usuario provee las funciones *map* y *reduce*, las cuales son distribuidas de forma automática. A grandes rasgos, *map* debe generar pares *clave-valor* y como resultado se garantiza que la entrada de la función *reduce* es particionada por clave a través de las máquinas.

Para manejar fallas del hardware, los datos son replicados varias veces en diferentes máquinas. Para mejorar la performance, el código es transportado hacia donde están los datos.
Se puede resumir que los motores de dataflow son útiles debido a:

- Ahorro de tiempo y esfuerzo por parte del programador gracias a las abstracciones que proveen.
- Escalan bien.
- Varios algoritmos han sido rediseñados para adaptarse a dicho paradigma.
- Se han vuelto comunes en data centers comerciales.

A continuación se nombran algunos principios compartidos por lenguajes que utilizan dicho paradigma:

- Favorecen el flujo de datos sobre el flujo de control, por lo tanto estructuras de control se dejan de lado a no ser que estén dentro de funciones lambda.

- Son declarativos.
- Los programadores proveen una especificación de alto orden del procesamiento de los datos en vez de definir los pasos para calcular un valor.
- Se adopta un estilo de programación funcional que fomenta composicionalidad y abstracción por sobre el estado.
- Además provee ventajas al momento de (1) expresar una computación como un *pipeline* de operadores, y (2) al simplificar paralelismo, distribución y tolerancia a fallas.
- Finalmente ocultan la complejidad del *Framework* subyacente (ya sea llevando un control automático sobre las dependencias entre valores o agregando tolerancia a fallas)

2.6. Apache Spark

Spark [7] es un framework de programación en clusters de propósito general y orientado a lograr la mayor velocidad de procesamiento posible sobre grandes volúmenes de datos. Spark provee una interfaz para programar clusters enteros con paralelismo de datos implícito y tolerancia a fallos. Es open source y provee API's para los lenguajes Python, Java, R y Scala siendo este último el lenguaje sobre el que fue desarrollado. La base de su arquitectura es el llamado RDD (de las siglas en inglés: *Resilient Distributed Dataset*), el cual puede pensarse como un vector distribuido a lo largo del cluster.
Spark proporciona un motor optimizado que soporta grafos de ejecución en general. También soporta un conjunto extenso y rico de herramientas de alto nivel entre las que se incluyen Spark SQL (para procesar datos estructurados basado en SQL), MLlib para implementar machine learning de forma distribuida y escalable, GraphX para procesamiento de grafos y Spark Streaming para procesar flujos de datos en tiempo real.

Las aplicaciones en Spark se ejecutan como conjuntos de procesos independientes en un cluster coordinado por el objeto *SparkContext* que existe en el programa principal (o *driver program*).

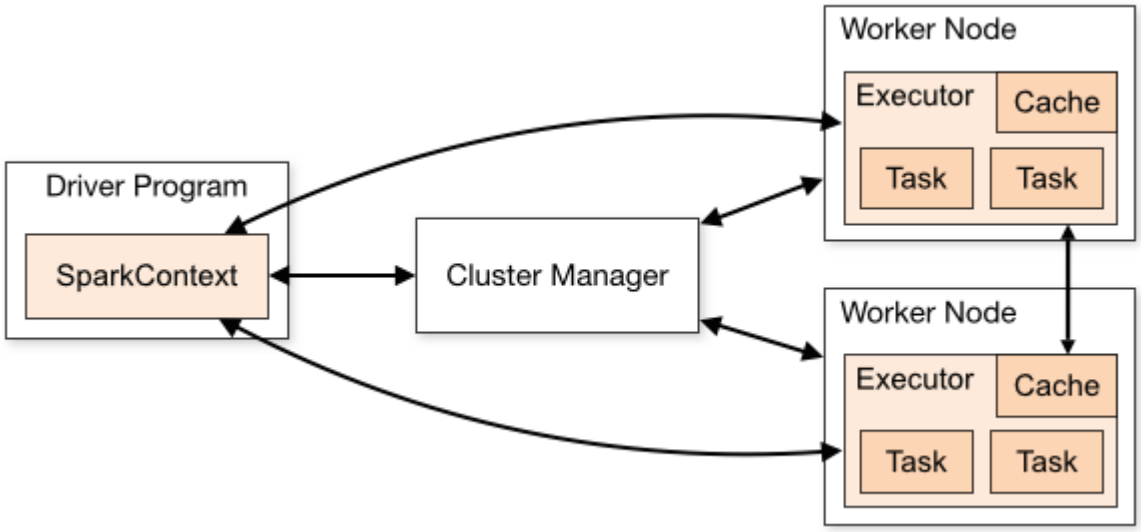

Figura 1: Arquitectura básica de Spark.

Específicamente, para correr en un cluster, el *SparkContext* puede conectarse a diferentes tipos de administradores de clusters los cuales son los encargados de asignar recursos entre aplicaciones.
Actualmente el sistema soporta los siguientes administradores:

- Standalone: el administrador más simple incluido en Spark que hace fácil levantar un cluster.
- Apache Mesos: un administrador de clusters general que permite correr Hadoop MapReduce y aplicaciones de servicio
- Hadoop YARN: el administrador de recursos en Hadoop 2.
- Kubernetes (experimental): Kubernetes es una plataforma *open-source* que provee infraestructura centrada en contenedores. El soporte es experimental y activamente desarrollado por una organización de Github.

Spark soporta un modo local, normalmente usado para pruebas o entornos de desarrollo donde el almacenamiento distribuido no es obligatorio y se puede usar el sistema de archivos local. Una ventaja fundamental es que cuando se necesite mayor capacidad de almacenamiento o cómputo (debido al incremento de los datos o al agregar nuevas fuentes de datos al sistema) el sistema desarrollado es escalable y no es necesario reformatear código sino que bastaría con cambiar la configuración de Spark para que corra en un cluster. Esta característica fue tenida en cuenta al momento de elegir una herramienta para el desarrollo de este trabajo

2.6.1. Apache Spark como mejora de MapReduce

Se decidió usar Apache Spark como herramienta principal debido a las siguientes falencias en el modelo MapReduce:

- Interfaz de programación muy restrictiva. Tenemos que enmarcar todo en términos de *mappers y reducers.*
- La existencia de múltiples operaciones I/O con discos distribuidos. Cuando hay múltiples operaciones *map y reduce*, la lectura y escritura en disco se vuelve un cuello de botella.

En relación a este último punto, debemos destacar que el problema principal de MapReduce se observa en problemas computacionales que requieren de varias iteraciones guardando estados intermedios. Consideremos un problema tal que en cada iteración se requiera una secuencia MapReduce. Como resultado, en cada iteración las máquinas deberían leer todos los datos de disco y escribir los resultados en disco, y por lo tanto la mayor parte del tiempo se gastaría en I/O. Sería cierto incluso si, en teoría, existiera una cantidad suficiente de máquinas para mantener los datos en memoria. Ello ocurre así porque MapReduce tiene como objetivo el almacenamiento permanente de datos en disco, principal motivo por el que sea una herramienta elegida para *Big Data,* pero también debido al método que usa para garantizar la tolerancia a fallas: persistencia de datos en disco, es decir, entre cada paso *map y reduce* se hace "data shuffling" y se escriben datos intermedios en disco.
En cambio en Spark, los datos son almacenados en memoria o transmitidos en pipelines de la ejecución del DAG, por lo que los tiempos de procesamiento se ven reducidos drásticamente debido a la rapidez con que estos son accesados en este tipo de memorias (RAM o Flash).
Un RDD (es la estructura de datos básica de Spark y se describe de forma más detallada en la siguiente sección) almacena sus valores en memoria, los datos que no entran en memoria son recalculados o el exceso de datos se envía a disco (*spill*). Cada vez que se quiere acceder a este RDD, este se puede extraer sin necesidad de ir al disco. Esto reduce la complejidad *espacio-tiempo* y el *overhead* de almacenamiento en disco.

Spark ejecuta programas hasta 100x veces más rápido *in-memory* que MapReduce Hadoop[9] (la implementación open-source de MapReduce) y hasta 10x veces más rápido en disco[10], esto último gracias a la optimización basada en DAGs.
Por tal motivo, la capacidad de Spark de trabajar *In-Memory* resulta conveniente para *aprendizaje automático* y para procesamiento en *micro-lotes (micro-batches)* ya que estas tareas requieren una ejecución iterativa de trabajos y de forma rápida.

Otra ventaja es la expresividad de Spark con respecto a Hadoop. MapReduce es complejo y el programador tiene que manipular APIs de bajo nivel para procesar los datos lo

[9] "Welcome to Apache™ Hadoop®!." http://hadoop.apache.org/.
[10] "Apache Spark™ - Lightning-Fast Cluster Computing." https://spark.apache.org/.

cual requiere mucho trabajo de escritura de código. Spark es fácil de usar, su abstracción (RDD) permite al usuario procesar datos usando operadores de alto nivel, en gran parte gracias a su implementación en Scala y su paradigma híbrido funcional-orientado a objetos, pero también gracias a que provee APIs en Java, Python y R.

2.6.2. Resilient Distributed Dataset (RDD)

Un RDD es una estructura de datos específico a Spark. A simple vista un RDD es un array distribuido implementado en Scala y con distintas APIs para lenguajes como python, java y R. Además, está implementado de forma tal que su uso en paralelo es tolerante a fallas. Un RDD es tipado, siendo su signatura de la forma: RDD[T]. Cada elemento de tipo T es almacenado localmente en una máquina y por lo tanto debe ser capaz de caber en la memoria.

Existen dos maneras de crear un RDD: paralelizando una colección existente en nuestro programa *driver* (programa que corre la función *main* definida por el usuario y ejecuta varias operaciones *paralelas* en un cluster) o referenciando un dataset en un sistema de almacenamiento externo (por ejemplo un sistema de archivos compartidos, HDFS, HBase o cualquier fuente de datos que ofrezca un formato de entrada Hadoop[11]).
Spark permite la opción de hacer un RDD persistente en la memoria (*caché*) en el caso de que los datos del mismo sean accedidos de manera frecuente.

Los RDDs soportan dos tipos de operaciones: *transformaciones*, las cuales crean un nuevo conjunto de datos a partir de un existente, y *acciones,* las cuales devuelven un valor al programa *driver* luego de correr una computación sobre el conjunto de datos. Por ejemplo, *map* es una transformación que pasa cada elemento del conjunto de datos a través de una función y devuelve un nuevo RDD representando los resultados. Por otra parte, *reduce* es una acción que agrega todos los elementos del RDD usando alguna función y devuelve el resultado final al programa *driver*.
Todas las transformaciones en Spark son *lazy* en el sentido de que no computan sus resultados de inmediato. En vez de eso, se recuerdan las transformaciones aplicadas al dataset y las transformaciones son computadas solamente cuando una acción requiere que el resultado sea devuelto al programa *driver*. Esto permite a Spark realizar optimizaciones sobre el flujo de trabajo.

2.6.3. DAG

Al momento de ejecutar código, Spark crea un DAG (*Directed Acyclic Graph* o *Grafo Acíclico Dirigido*) donde los vértices representan los RDDs y las aristas representan la operación a ser aplicada al RDD como se puede observar en la figura 2.
Cuando Spark crea un DAG de *stages* de computación consecutivos el plan de ejecución es optimizado, por ejemplo, minimizando el movimiento de datos entre nodos (shuffle).

[11] "RDD Programming Guide - Spark 2.3.0 Documentation - Apache Spark." https://spark.apache.org/docs/latest/rdd-programming-guide.html# resilient-distributed-datasets-rdds.

Al momento de realizar una acción sobre un RDD a alto nivel, Spark entrega el grafo al planificador de *DAG (DAG scheduler)* el cual se encarga de dividir las operaciones en *stages* para luego ser pasados al planificador de tareas (*Task scheduler*) y este lanza tareas a través del *cluster manager* a los nodos en el cluster (también llamados *worker nodes*). Finalmente son los *worker nodes* los que se encargan de ejecutar estas tareas mediante procesos denominados *executors* (o ejecutores) los cuales son lanzados al comienzo de la aplicación y generalmente tienen la misma duración de la misma (es decir, inician y terminan con la misma aplicación). Una vez que terminaron de ejecutar la tarea, envían los resultados de vuelta al programa *driver*. Además proveen de almacenamiento *in-memory* de RDDs cuya persistencia haya sido especificada a nivel de memoria .

Un DAG contiene información sobre dependencias entre RDDs, de esta manera Spark es tolerante a fallos ya que al momento en que falla un nodo, el DAG se vuelve a ejecutar desde el nodo más cercano para re computar el RDD.

Es necesario destacar algunas ventajas en comparación con MapReduce:

En el caso de MapReduce, el DAG consiste de únicamente dos vértices, uno para la tarea *map* y otro para la tarea *reduce* y por lo tanto una operación compleja consiste en una serie de map y reduce donde se dificulta saber qué operación es la que vendrá a continuación

En MapReduce, algunas veces para alguna iteración, es irrelevante leer y escribir de nuevo el resultado entre dos trabajos. Cuando se tienen múltiples iteraciones, todos los demás trabajos son bloqueados desde el comienzo hasta que finalice el trabajo anterior. Como resultado, una computación compleja puede requerir un tiempo largo con pocos datos.

En MapReduce la optimización se logra de forma manual y en cada paso *map* y *reduce*.

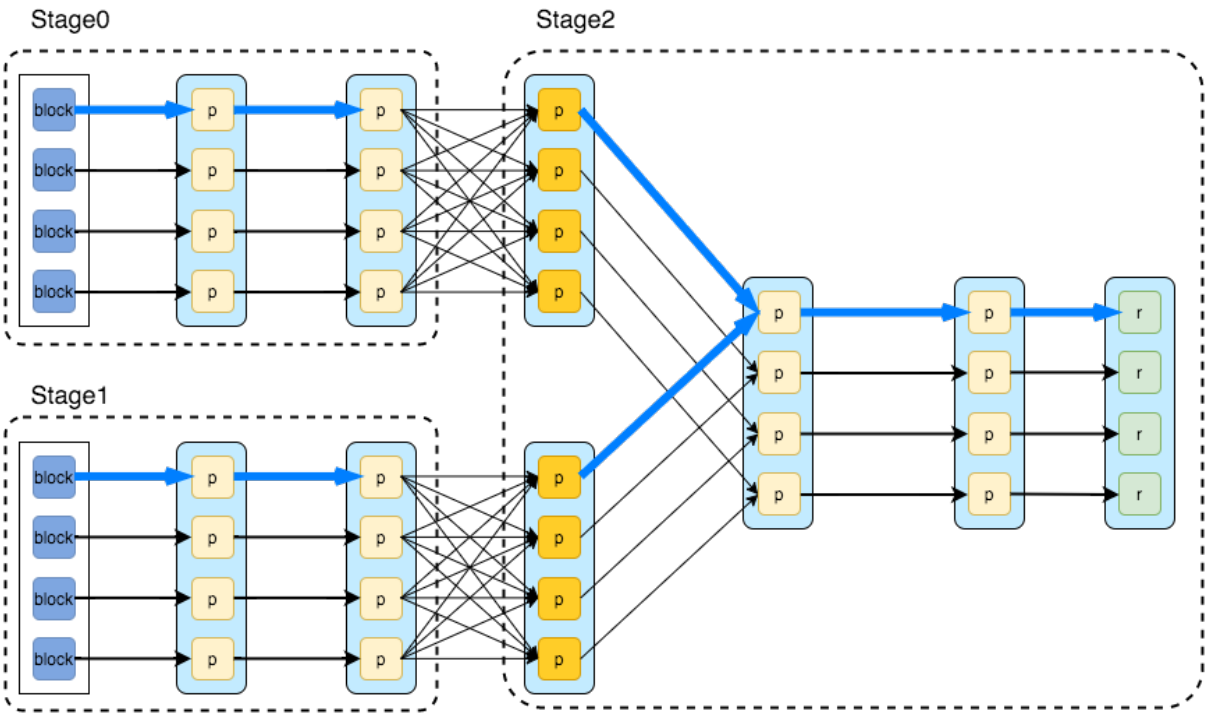

Figura 2: Ejemplo de DAG. Básicamente todo flujo de proceso de datos puede ser definido como la lectura de datos desde fuentes, aplicar un conjunto de transformaciones y materializar el resultado de distintas maneras.

2.6.4. Datasets y Dataframes

Un Dataset es una colección de datos distribuidos que proporciona una visión del esquema de los datos lo cual significa que los mismos son visualizados como columnas con nombre e información de tipos. Dataset es una interfaz nueva agregada en Spark 1.6 que provee los beneficios de los RDDs:

- Es fuertemente tipado.
- Transformaciones funcionales.
- Evaluación lazy.
- Tolerancia a fallos.

Un dataset puede ser construido a partir de un objeto JVM y luego ser manipulado usando transformaciones funcionales (map, flatMap, filter, etc).

Un DataFrame es un *Dataset* no tipado. Conceptualmente es equivalente a una tabla en una base de datos relacional o a un data frame en R/Python, pero con optimizaciones enriquecidas "under the hood". En Scala y Java, un DataFrame es representado por un Dataset de Rows. En la API de Scala un DataFrame es simplemente un *alias* de Dataset[Row].
Al igual que los RDD, los Datasets son "Lazy", es decir que las computaciones solamente son realizadas cuando se invoca una *acción* (collect, count, etc).
Internamente un Dataset representa un plan lógico que describe la computación requerida para producir los datos lo que se nota más cuando el Dataset es producto de filters, joins, groupBys, etc. Esto es producto de *laziness*. Las operaciones nombradas se van "encadenando" y no se ejecutan hasta realizar una acción (entonces tenemos un plan de ejecución lógico). Cuando una acción es invocada, el motor de ejecución optimizado de Spark SQL optimiza el plan lógico y genera un plan físico para la ejecución eficiente en una manera paralela y distribuida.

Es importante destacar que tanto DataFrame como Dataset internamente realizan la ejecución final únicamente con objetos RDD pero la diferencia es que los usuarios no escriben código para crear colecciones de RDD y no tienen control sobre los mismos. Los RDDs se crean en el *stage* final del plan de ejecución, después de atravesar por todas la optimizaciones que ofrecen los DataFrames y Datasets (*Project Tungsten* principalmente para ahorro de espacio en memoria *y Catalyst Optimizer* para evitar operaciones IO y de *shuffling* innecesarias, entre otras).

2.6.5. GraphFrames

GraphFrames es un paquete para Apache Spark que provee grafos basados en DataFrames[12]. Provee APIs de alto nivel en Scala, Java y Python. Apunta a proveer la funcionalidad de GraphX[13] (el cual es un API de Apache Spark para grafos y computación paralela de los mismos) y funcionalidad extendida sacando ventaja de los DataFrames de Spark. Esta funcionalidad extendida incluye *motif finding*, serialización basada en DataFrames y consultas de grafos altamente expresivas. Asi como GraphX está basado en RDD, GraphFrames se basan en los *DataFrames* de Spark.
Los GraphFrames representan grafos: vértices (usuarios) y aristas (relaciones entre usuarios). Es por esto que un GraphFrame se crea a partir de dos *DataFrames* que deben cumplir con ciertos requisitos:

- el *DataFrame* de vértices debe contener una columna especial con nombre "*id*" que especifique el ID único de cada vértice en el grafo.
- el *DataFrame* de aristas debe contener dos columnas especiales : *"src"* (ID del vértice de partida) y *"dst"* (ID del vértice de llegada o de destino).

Además ambos DataFrames pueden tener columnas extras que representan atributos de vértices y aristas, por ejemplo en un grafo que representa una red social los vértices

[12] "Overview - GraphFrames 0.5.0 Documentation." https://graphframes.github.io/.
[13] "GraphX | Apache Spark." https://spark.apache.org/graphx/.

pueden tener además de la columna de ID, una columna con el nombre de la persona y otra con la edad, mientras que las aristas pueden tener un atributo que destaque el tipo de relación que los une (amigo, familia, trabajo, etc).
GraphFrames además provee herramientas poderosas para ejecutar consultas (grados de salida/entrada) y algoritmos de grafos como por ejemplo: shortest path, PageRank, Breadth-first search (BFS), etc.
Al igual que GraphX, GraphFrames provee primitivas para crear nuevos algoritmos usando pasaje de mensajes, esto permitió implementar un algoritmo de influencia colectiva que se describe más adelante.
Al estar basados en DataFrames, los GraphFrames pueden hacer uso de las herramientas de lectura y escritura de los mismos, lo que permite leer y escribir grafos usando varios formatos como *Parquet, JSON y CSV.*
Actualmente no forma parte de Apache Spark y por ahora el plan es que se mantenga de esa manera ya que todavía se siguen haciendo ajustes a la API, pero una vez estable será considerada su incorporación a Spark.

2.7. Análisis de Redes Sociales

El Análisis de Redes Sociales consiste en el proceso de investigar estructuras sociales mediante el uso de *teoría de redes* y *teoría de grafos.*[14]
Por estructura social entendemos una formación, ya sea física o simbólica, que surge a partir del modo en que se relacionan los elementos vinculados a la sociedad en cuestión. Dichos elementos pueden ser un conjunto de individuos que comparten una cultura y conviven bajo determinadas normas como por ejemplo personas, grupos, organizaciones; pero también los productos que surgen como resultado de actividad humana o el conocimiento (como los sitios web, conceptos semánticos, etc.).
Dicho análisis está vinculado al estructuralismo en la *sociología* destacando la importancia de las relaciones entre actores sociales a fin de poder determinar comportamiento, opiniones y actitudes.
Existen dos perspectivas que dominan el *Análisis de Redes Sociales:*

- la perspectiva **sociocéntrica** que analiza la estructura de la red en general. Busca patrones de vínculos que denotan grupos sociales, actores centrales que pueden ser clave en la integración de la red social y asimetrías que pueden reflejar prestigio social o estratificaciones sociales.

- la perspectiva **egocéntrica** que se enfoca en la composición de estructuras de red locales. Determina la influencia de determinados actores sobre otros a través de sus vínculos en la red (modelo de influencia social) y/o determinar si los actores ajustan sus vínculos de acuerdo a las características de sus pares y de su vínculo con ellos (modelo de selección social).

Estas estructuras se caracterizan en términos de *nodos* (que representan personas, grupos, organizaciones o todo aquello que pueda ser considerado un actor social) y *aristas* (que

[14] "Social Network Analysis" https://en.wikipedia.org/wiki/Social_network_analysis .

representan relaciones o interacciones) que conectan estos nodos. Algunos ejemplos de estas estructuras sociales son las redes de medios sociales como por ejemplo *Facebook*, *Twitter, LinkedIn, YouTube, etc.*

El análisis de redes sociales tiene sus raíces en el aporte de diferentes ciencias (sociologia, psicologia, antropologia, matemáticas, etc.) y de la inquietud sobre cómo entender los fenómenos relacionales y cómo éstos condicionan la acción y comportamiento. Algunas figuras destacadas que realizaron aportes al análisis de redes sociales y que comenzaron a acuñar el concepto desde principios del siglo 20 en estas ciencias fueron:

- Jacob Moreno y sus sociogramas para destacar la influencia mutua que ocurría entre sus pacientes y las terapias [8] .
- El antropólogo John Barnes destacó la influencia de las conexiones sociales por encima de la cultura en cierto sector [9] .
- El sociólogo Harrison White señaló que había que mirar el mercado y la sociedad como una red de conexiones [10] .

Si bien el análisis de redes sociales emergió como una técnica fundamental en la sociología moderna, actualmente es usada en diversas disciplinas como antropología, biología, economía, geografía, historia, ciencias políticas, estudios organizacionales y ciencias de la computación.

De la teoría de grafos de la matemática se extrajeron diversos conceptos y procesos para analizar datos relacionales. Formalmente podemos conceptualizar una red social como un grafo, es decir, un conjunto de vértices (o nodos, unidades, puntos) representando actores sociales u objetos y un conjunto de líneas o aristas que presentan una o más relaciones sociales entre estos. Pero una red es más que sólo un grafo, debido a que contiene información adicional sobre los vértices y aristas. Características de un actor social como por ejemplo la edad, sexo, sueldo de una persona, son representados mediante atributos discretos o continuos de los vértices en la red y la intensidad, frecuencia o tipo de la relación social (amigo, familiar, etc.) son representados por peso, valor o tipo de las aristas.

A las métricas que podemos obtener haciendo análisis de redes sociales las podemos clasificar en tres niveles:

1) Nivel global

    a) **Coeficiente de agrupamiento:** Nivel de agrupamiento de los nodos, para saber cuan cohesionados están los actores/agentes.
    b) **Camino característico**: mide el grado de separación de los nodos, para determinar lo contrario a lo anterior, que tán alejados o separados están (uno o más caminos pueden ser calculados mediante un algoritmo de *shortest path*).

c) **Densidad**: Un grafo puede ser denso (cuando tiene muchas aristas) o disperso (muy pocas aristas). Es decir la conectividad.

d) **Diámetro**: Máximo de las distancias entre cualquier par de nodos.

e) **Grado medio**: número de vecinos medio que tiene un grafo. Indica la media de conexiones que tiene un nodo (popularidad).

f) **Centralidad**: Permite identificar nodos con mayor cantidad de relaciones y por ende los más influyentes dentro del grupo.

2) Nivel comunidad:

a) **Comunidades**: Agrupaciones de nodos por patrones de similitud.

b) **Puentes entre comunidades**: ¿Como se conectan las comunidades?¿Como de comunicables son dichas comunidades?.

c) **Centros locales vs Periferia**: Para detectar los nodos más centrales o críticos, frente a los que no lo son.

3) Nivel Nodo:

a) **Centralidad**: Es una métrica de "poder". Por lo general determina la influencia de un nodo en toda la red.

b) **Intermediación**: Control del flujo de comunicación. Importante para establecer estrategias de eliminación de propagación de un virus por ejemplo.

c) **PageRank**[15]: Algoritmo que permite dar un valor numérico (o ranking) a cada nodo del grafo que mide de alguna forma su conectividad.

d) **Closeness**: Que tan fácil es llegar a los otros vértices a partir de un nodo. Importante para establecer nodo difusor y de esta manera se logra que la difusión sea de la forma más rápida posible.

El uso del análisis de redes sociales en la actualidad varía de manera extensa en un amplio rango de aplicaciones y disciplinas. Algunas aplicaciones incluyen *modelado de redes, muestreo, análisis de comportamiento en usuarios, desarrollo de sistemas de recomendación, etc.* En el sector privado algunos negocios usan esta herramienta para mejorar la interacción con el cliente, desarrollar sistemas de información, marketing o la necesidad de *business intelligence*.
El análisis de redes sociales es usado también para inteligencia, contrainteligencia, y actividades donde se aplique la ley.

2.8. Influencia colectiva

El concepto de influencia está fuertemente relacionado al concepto de integridad de la red.[16] Más precisamente, los nodos de mayor influencia en la red forman un conjunto

[15] "PageRank - Wikipedia, la enciclopedia libre." https://es.wikipedia.org/wiki/PageRank .
[16] "Node Influence Metric". https://en.wikipedia.org/wiki/Node_influence_metric .

mínimo tal que si es removido podría desmantelar la red en varios componentes desconectados.
La siguiente fórmula calcula el valor de Influencia Colectiva del nodo i:

$$CI\,(i) \quad = \quad (k_i - 1) \sum_{j=i}^{N} a_{ij}\,(k_j - 1)$$

donde k_i es el grado del nodo i, a_{ij} es la matriz de adyacencia y N la cantidad de nodos. Si bien el algoritmo tiene como objetivo determinar los nodos que son "influencers" en la red, es decir aquellos con mayor influencia, en este trabajo solamente nos interesa conocer el valor inicial de *CI* de cada nodo, y usar estos valores para filtrar caminos por importancia (o influencia) de los nodos que lo componen. En el Capítulo 3 se detalla cómo se utiliza esta métrica y se describe una mejora al algoritmo mediante la inicialización previa de los valores de *CI* de cada nodo.

2.9. Visualización

La visualización de datos es un elemento esencial a la hora de interpretar los resultados obtenidos lo cual permite al usuario la interacción directa con los datos, facilitando la comprensión y el conocimiento de los mismos , ayudando a la toma de
decisiones y a la comunicación de los resultados obtenidos.

El principal objetivo de la visualización de datos es transformar los resultados en información comprensible para el usuario. Surge a raíz de la abundancia de datos y a la complejidad que esto genera en su búsqueda e interpretación lo que da lugar a la necesidad de un mecanismo que permita facilitar la comprensión y asimilación de la información.
La información resultante debe poder ser comunicada de tal forma que el usuario adquiera los conocimientos necesarios para comprender correctamente los datos facilitados. Para conseguirlo, el propio creador de la visualización debe tener un alto grado de conocimiento sobre la información que desea comunicar, para que le sea más sencillo transmitirla a otras personas.

Podemos pensar en la visualización como una abstracción de la información en alguna forma esquemática, codificando los datos en objetos visuales (puntos, líneas, barras, etc) contenidos en gráficas, y es motivo por lo cual a veces llega a ser considerada tanto *arte* como *ciencia.*

Actualmente contamos con numerosos métodos de visualización para los datos producidos en el análisis de redes sociales. Gran parte del software destinado al análisis de redes sociales provee de módulos destinados a la visualización. Los datos son explorados mediante la muestra de los nodos y sus relaciones en varios diseños (o *layouts*) y atribuyendo colores, tamaño y otras propiedades avanzadas a los nodos.
Por otra parte debemos tener en cuenta que, si bien la visualización de redes sociales es un método poderoso a la hora de interpretar y comunicar información compleja, no es la única manera de hacerlo. Y debemos tener cuidado de que la representación estructural de las propiedades no sean completamente capturadas por estas visualizaciones: probablemente

estas propiedades se capturen mejor a través de análisis cuantitativos.

2.9.1. Herramientas de visualización

Dentro de las herramientas utilizadas para la visualización de las redes sociales destacamos el uso de D3.js (o simplemente D3 por las siglas *Data-Driven Documents*) la cual es una librería *open-source* de JavaScript para producir visualizaciones de datos dinámicas e interactivas en navegadores web[17]. Hace uso de estándares como SVG, HTML5 y CSS. A diferencia de muchas otras librerías, D3 permite al usuario tener un gran control sobre el resultado final de la visualización. Actualmente es usado en gran cantidad de páginas ya sea para la creación de gráficos interactivos, tablas informativas para visualización de datos, producción de mapas a partir de datos, etc.
La creación de D3 es producto de varios intentos de llevar la visualización de datos a los navegadores. Algunos proyectos notables que sentaron las bases de D3 y que actualmente se consideran sus predecesores directos fueron *Prefuse, Bengala y Protovis,* siendo esta última herramienta reconocida entre creadores de infogramas y académicos.[18]
D3 permite ligar datos arbitrarios a un *Document Object Model* (DOM), y luego aplicar transformaciones basadas en dichos datos al documento. Por ejemplo, se puede crear una tabla HTML a partir de una *array* de números o usar los mismos datos para crear un gráfico de barras SVG interactivo con transiciones e interacciones.
Si bien existen otras herramientas (*Gephi, Graphviz, NetworkX, Cytoscape, etc*) y librerías de visualización (*chart.js, c3.js, Vis.js, etc*), se eligió trabajar con D3 ya que permite crear visualizaciones interactivas y dinámicas. Como se menciona al comienzo, D3.js le otorga al creador tener completo control sobre la visualización, lo cual se traduce en una mejor comunicación de la información. Cuenta con una comunidad grande de usuarios y a su vez podemos encontrar una enorme cantidad de ejemplos de casos de uso. Pero lo más importante es que la herramienta es puramente impulsada por datos (o *data-driven*) es decir, el "combustible" son datos estáticos o bien datos traídos de forma remota y con diferentes formatos (CSV, JSON, XML, etc).

2.9.2. Algoritmo para dibujo de grafo

Para la visualización de datos se usó un algoritmo de dibujo de grafos *Force-directed*[19], el mismo viene implementado en D3.
Estos algoritmos tienen como objetivo el dibujo de grafos de una manera lo más estéticamente posible. Esto se logra mediante el posicionamiento de los nodos de un grafo en un espacio bi-dimensional de forma tal que las aristas tengan aproximadamente la misma distancia y que haya la mínima cantidad posible de cruces entre caminos. El término *"Force"* se debe a que se asignan fuerzas entre el conjunto de aristas y vértices, basadas en sus posiciones relativas y luego estas fuerzas son usadas ya sea para simular el movimiento de las aristas y vértices o para minimizar superposición.

[17] "D3.js - Wikipedia." https://en.wikipedia.org/wiki/D3.js.
[18] "Antecedentes de D3.js." https://es.wikipedia.org/wiki/D3.js#Antecedentes .
[19] "Force-directed graph drawing - Wikipedia." https://en.wikipedia.org/wiki/Force-directed_graph_drawing .

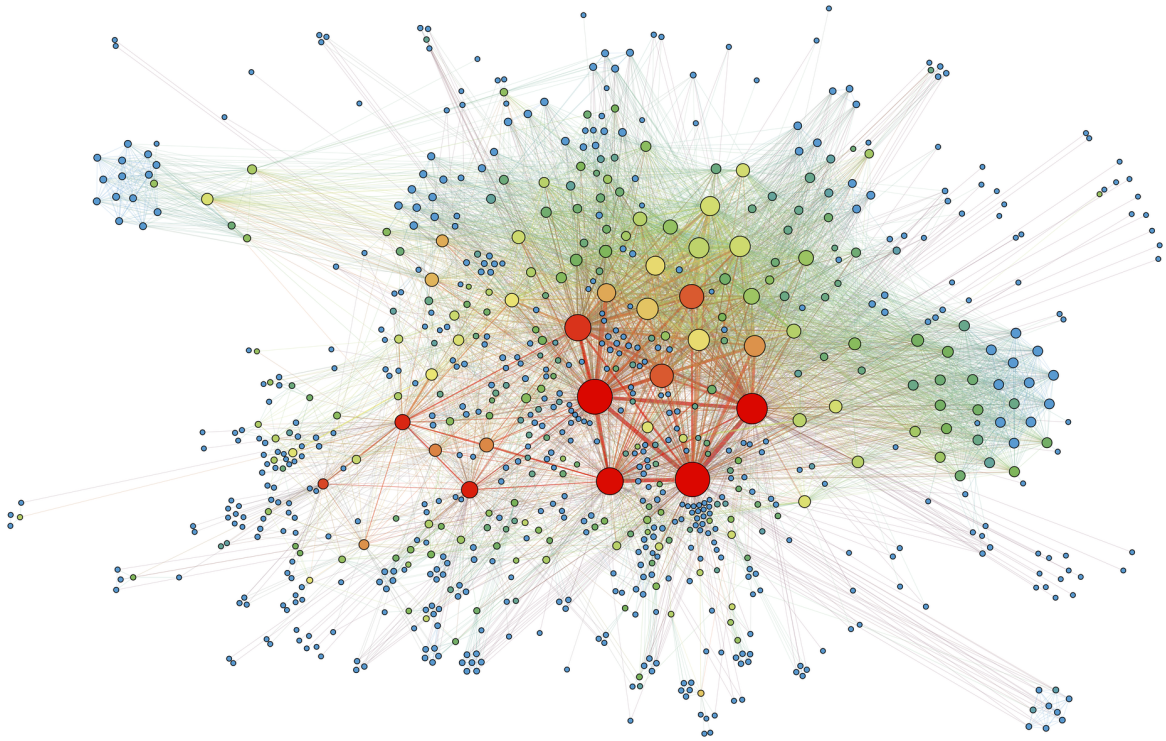

Figura 3: Visualización de una red social usando un layout Force-directed.

Estas fuerzas se basan típicamente en leyes de la física. Los pares de puntas de una arista se atraen mutuamente mediante fuerzas basadas en la ley de elasticidad de Hooke. De manera similar, los nodos son repelidos unos con otros usando fuerzas basadas en las cargas eléctricas de las partículas y la ley de Coulomb.

Un grafo *Force-directed* puede involucrar otras fuerzas además de las de resorte mecánico y repulsión eléctrica. Se puede usar una fuerza análoga a la gravitacional para mantener los nodos atraídos hacia un determinado punto del espacio, lo cual resulta útil a la hora de visualizar diferentes componentes conexas de un grafo no dirigido y también para juntar nodos con mayor centralidad en posiciones más centrales en el dibujo.
Se pueden agregar fuerzas repulsivas tanto en aristas como en nodos con el fin de evitar la superposición de los mismos en la visualización final

Una vez que las fuerzas de los nodos y las aristas han sido definidas, el comportamiento del grafo entero puede ser simulado como si fuera un sistema físico. En esta simulación las fuerzas son aplicadas a los nodos haciendo que estos se acerquen o sean repelidos. De la misma manera que son aplicadas en las aristas causando la compresión y expansión de estas. Esto se repite de forma iterativa hasta que el sistema alcanza un estado de equilibrio, esto es, sus posiciones relativas no cambian de una iteración a la otra.

3. Trabajo realizado

En este capítulo se describe el trabajo realizado y las distintas etapas que comprende. Además se describen los programas realizados para llevar a cabo cada tarea. El mismo consta de tres partes claramente diferenciadas. La primera, consiste en la extracción de datos, limpieza, unificación y carga de los mismos en nuestra base de datos. En la segunda parte se describe la creación del grafo de la estructura social estudiada a partir de la base de datos generada en la etapa anterior. Una vez creado el grafo, se procede con el cálculo de los caminos más cortos desde artistas invitados a artistas relevantes. A fin de reducir la cantidad de caminos calculados para una correcta visualización, se procederá a aplicar un filtro a los mismos basados en ranking. En esta etapa se explicarán las distintas alternativas para lograr esto. En la etapa final, se detallan los programas destinados a generar los archivos de las distintas visualizaciones de los resultados obtenidos en la etapa anterior y se describe cada diseño.
La arquitectura de todo el sistema está compuesta por pequeños programas modulares que realizan tareas simples sobre los datos que reciben. Esta modularización está basada en la filosofía Unix[20] la cual enfatiza la construcción de código simple, claro, modular y extensible que pueda ser fácilmente mantenido y modificado por otros desarrolladores. Esta filosofía favorece componibilidad, esto es la capacidad de poder relacionar los distintos componentes, así como también agregar nuevos componentes o eliminarlos. Esto resulta importante en nuestra arquitectura ya que nos permitirá diseñar un pipeline compuesto por los distintos módulos.

3.1. Datos

Para el trabajo se utilizaron tres fuentes de datos. La principal, corresponde a *Discogs*[21] (abreviado de *discografias*) de la cual se extrajeron los datos más importantes para más adelante generar un grafo representativo de la estructura social que nos interesa. Como complementos, además se usaron otras dos fuentes: *MusicBrainz*[22], que es una base de *datos abiertos* de música y *Wikidata*[23], una base de datos secundaria que brinda soporte a *Wikipedia*[24], enciclopedia de contenido libre.
Cabe mencionar que la arquitectura del sistema desarrollado permite agregar nuevas fuentes de datos de manera fácil, esto gracias a la división en módulos de los distintos programas que ingieren los datos.
A continuación se describirán en detalle cada una de las fuentes utilizadas y los esquemas de sus datos.

3.1.1. Discogs

Discogs es un sitio web y base de datos de colaboración abierta de información sobre discos comerciales, promocionales y discos no oficiales (también llamados *bootleg* o ediciones no autorizadas). Se trata de una de las mayores bases de datos de música en línea y la de mayor referencia de discos de vinilo aparte del formato CD. La información que se suministra a la página solo puede ser realizada por usuarios registrados en el sitio

[20] "Unix philosophy - Wikipedia." https://en.wikipedia.org/wiki/Unix_philosophy.
[21] "Discogs." https://www.discogs.com/.
[22] "MusicBrainz." https://musicbrainz.org/.
[23] "Wikidata." https://www.wikidata.org/wiki/Wikidata:Main_Page.
[24] "Wikipedia." https://www.wikipedia.org/.

y es evaluada por un grupo de moderadores con el fin de evitar errores, duplicados o vandalismo. En esta base de datos se puede encontrar información sobre discos y artistas o grupos asociados a ellos. Pero además, permite al usuario buscar artistas y ver los discos en los que participa: es una característica de las bases de datos de referencias cruzadas (en inglés *cross-references databases*) que además forman una estructura en red de relaciones existentes entre las diferentes partes de los datos. Por ejemplo, se pueden asociar las entidades "Herbie Hancock", "Roy Haynes", "Cecil McBee" y "Charles Tolliver" al disco "It's time" del artista "Jackie McClean", la información se puede extraer de los créditos los cuales presentan información detallada sobre las relaciones.
Mensualmente *Discogs* realiza un volcado de datos de artistas y discos. Estos datos se encuentran en formato XML y pueden ser descargados bajo licencia de dominio público. Actualmente la base de datos almacena información de alrededor de 9 millones de discos y 5 millones de artistas.
La base de datos consta de información asoociada a discos (o releases) y artistas. Los datos de discos además de contener información como título, identificador o ID, año de publicación y país, tienen información sobre todos los artistas asociados a un *release* (o créditos), estos datos existen en forma de array de estructuras que representan a cada artista y cada una de ellas contiene información como el nombre del artista, rol que tiene en el *release* (guitarrista, pianista, voz, etc.), pistas o *tracks*, nombres alternativos, identificadores, entre otros.
La tabla de artistas contiene información asociada a bandas y artistas solistas.
Más adelante se describe cómo se manejaron estas tablas y sus características.

3.1.2. MusicBrainz

MusicBrainz es un proyecto cuyo objetivo es el de la creación de una base de datos musical de contenido abierto. Fue creada en respuesta a las restricciones que traía la *"compact Disk Database"* (CDDB[25]) la cual es una base de datos destinada a aplicaciones de software para búsqueda de información de CDs de audio en internet. Musicbrainz captura información sobre artistas, sus trabajos y la relación entre estos. Los trabajos o grabaciones capturan como mínimo el título del álbum, títulos y longitudes de canciones. Estas entradas son mantenidas por editores voluntarios que se basan en guías escritas por la comunidad y que establecen qué formato debe usarse. Estas grabaciones pueden almacenar además información de la fecha y país de lanzamiento, número de identificación del CD, portada del disco y otros metadatos. En julio de 2017 la base de datos contaba con información de alrededor de 1.8 millones de discos y 1.2 millones de artistas.
MusicBrainz realiza mensualmente un volcado de datos y además ofrece un servicio web cuya arquitectura sigue los principios de diseño REST donde los datos pueden ser obtenidos en formato XML o JSON.
Esta base de datos se usa para reforzar los datos asociados a artistas que no provee *Discogs*, en particular links asociados a artistas para conectar con otras bases de datos y nacionalidades. Las tablas que se descargan y son usadas en este trabajo son las siguientes:

- tabla **artists**: contiene información de cada artista, identificador en las base de datos, identificador en MusicBrainz, nombre, identificador de Área.

[25] "CDDB - Wikipedia." https://en.wikipedia.org/wiki/CDDB.

- tabla **area**: contiene información de los países, pueblos, ciudades, estados, etc, asociados a artistas o trabajos discográficos. Identificador, nombre y tipo de área.
- tabla **url**: contiene información sobre las direcciones web o links pertenecientes a artistas o trabajos discográficos. URL, identificador del URL y tipo de URL (si es de *MusicBrainz, Discogs, Wikidata, YouTube, etc*).
- tabla **area_area**: asocia áreas, por ejemplo asocia el estado California con el país Estados Unidos.
- tabla **artist_url**: asocia artistas con URLs. Por ejemplo puede asociar al artista Miles Davis con los links externos que contienen información de todo tipo: como su página oficial, su perfil en *Discogs*, su perfil en *Wikidata,* canal de *YouTube,* perfil en *Wikipedia,* etc.

Existen también otras tablas pero aquellas que nos interesan y contienen información relevante para este trabajo son las que se listaron anteriormente.

3.1.3. Wikidata

Wikidata es una base de conocimientos libre,[26] y actúa como almacenamiento central de datos estructurados de su proyecto hermano *Wikimedia*[27], incluyendo *Wikipedia*[28], *Wikivoyage*[29], *Wikisource*[30] y otros.
Los datos en Wikidata son publicados bajo licencia de dominio público, permitiendo su reutilización en diversos escenarios. Se pueden copiar, modificar, distribuir e incluso usar para fines comerciales, sin necesidad de pedir autorización. El ingreso de datos puede ser realizado tanto por personas como por *bots* (también conocidos como *robots*) que son herramientas para realizar ediciones sin necesidad de que una persona tome decisiones. Tanto los *bots* como las entradas de datos son confeccionadas y supervisadas por editores de Wikidata que se encargan de establecer las reglas en la creación de contenido y administración. La edición en diferentes idiomas es posible y se alienta a que así sea. Además se registran las fuentes de los datos y conexiones a otras bases de datos lo cual refleja la diversidad de conocimiento disponible y agrega soporte a la noción de verificabilidad. Por último, la imposición de un alto grado de organización estructurada permite la reutilización de datos por proyectos de Wikimedia y terceros, pero también, que las computadoras los procesen y "entiendan".

[26] "Wiki". https://es.wikipedia.org/wiki/Wiki .
[27] "Fundación Wikimedia." https://es.wikipedia.org/wiki/Fundación_Wikimedia .
[28] "Wikipedia - La enciclopedia libre." https://www.wikipedia.org/ .
[29] "Wikivoyage - La guía turística libre." https://www.wikivoyage.org/ .
[30] "Wikisource - La biblioteca libre." https://es.wikisource.org/wiki/Portada .

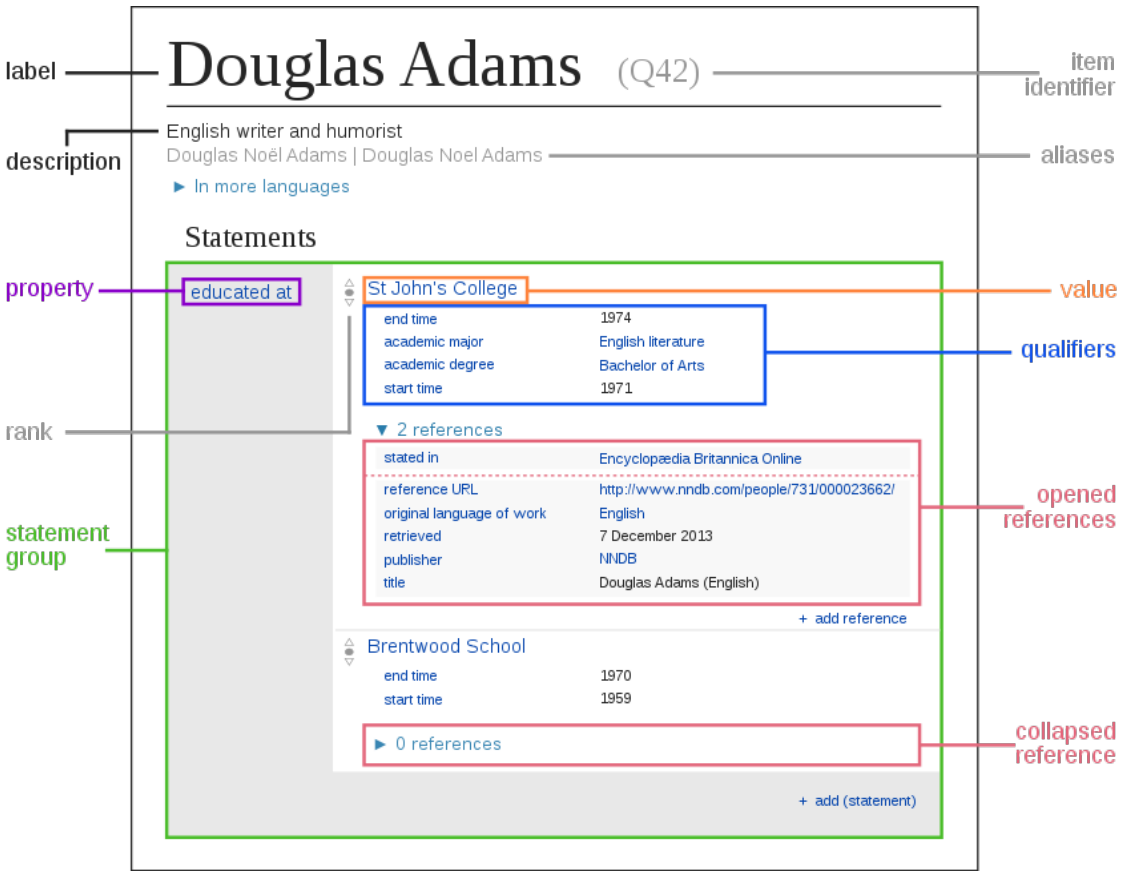

Figura 4: Diagrama de un ítem. Se muestran los términos más importantes de Wikidata.

El repositorio de Wikidata consiste principalmente de *Statements* (ver Cuadro 1) los cuales describen características detalladas de un *Ítem* (ver figura 4) y consisten de una propiedad y un valor. Estos ítems se identifican de manera única por una clave que comienza con la letra Q seguida de un número. Las propiedades tienen una P seguido de un número por ejemplo "país de ciudadanía" (P27). Para una persona se puede agregar una propiedad para especificar donde fue educado especificando un valor para un colegio. Para construcciones, se pueden asignar coordenadas geográficas como propiedades especificando valores de longitud y latitud. Además, estas propiedades pueden ser links a bases de datos externas (estas propiedades se las llama *identificadores).*

Item	Propiedad	Valor
Q93341	P136	Q8341
Miles Davis	Género	Jazz

Cuadro 1: Ejemplo de statements.

Wikidata ofrece copias del contenido disponible para descargar pero además existen otros métodos para acceder al contenido estructurado, que no requieren de un volcado de base de datos completo. Para este trabajo se decidió utilizar el volcado en formato JSON por ser el recomendado en la sección de descarga de base de datos de la página de Wikidata.[31] En este archivo cada objeto de entidad (elementos de datos o propiedad) se coloca en una línea separada permitiendo que el mismo sea leído línea por línea y a su vez cada línea se puede decodificar como un objeto JSON individual.
A continuación se muestra el esquema JSON:


```
{
    “id”: “Q60”,
    “type”: “item”,
    “labels”: { } ,
    “descriptions”: { } ,
    “aliases”: { } ,
    “claims”: { } ,
    “sitelinks”: { } ,
    “lastrevid”: 195301613,
    “modified”: “2015-02-10T12:42:02Z”
}
```


Descripción de campos:

- **id**: el ID canónico de la entidad.
- **type**: el identificador de tipos, “ítem” o “propiedad”.
- **labels**: contiene las etiquetas en diferentes idiomas.
- **descriptions**: contiene descripciones en distintos idiomas.
- **aliases**: contiene alias en distintos idiomas.
- **claims**: contiene cualquier número de *statements* agrupados por propiedad.
- **sitelinks**: contiene links a páginas en diferentes sitios que describen el ítem.
- **lastrevid**: la versión del documento JSON.
- **modified**: la fecha de publicación del documento JSON.

[31] “Wikidata:Database download.”. https://www.wikidata.org/wiki/Wikidata:Database_download/es .

Esta base de datos se usa como refuerzo para agregar información asociada a artistas musicales, como por ejemplo nacionalidades, instrumentos, IDs asociados a otras bases de datos (esto es importante para la etapa de unificación que se verá más adelante), géneros musicales y links de imágenes. Esta última información resulta importante para la etapa de visualización.

3.2. Arquitectura ETL de datos

Para el manejo de estos datos se propone dividir la primer etapa en dos procesos de ETL donde la salida del primero es la entrada del segundo.
Esta división separa el procesamiento de las distintas fuentes de datos de su unificación. También permite contar con tablas más pequeñas y con únicamente los datos que resulten relevantes para la unificación. Por último, esta división permite diferenciar las tareas que son (1) extracción y limpieza de los datos y (2) unificación de datos, y de esta manera facilitar el testeo de las mismas.
En la Figura 5 podemos observar el pipeline de ETLs diseñado para este trabajo. La primer etapa tiene como objetivo la lectura de datos de las fuentes, limpieza y generación de datasets con información relevante en cada una de ellas. La segunda etapa consiste en la unificación de las tablas generadas con el fin de generar una base de datos sobre la cual se realizará el análisis de redes y de grafo correspondientes.
Además hay que destacar que todos los datasets generados en el pipeline serán almacenados en formato *parquet*, esto es para aprovechar las optimizaciones que provee Spark al leer desde este tipo de formato (optimizaciones que se mencionan en el marco teórico)

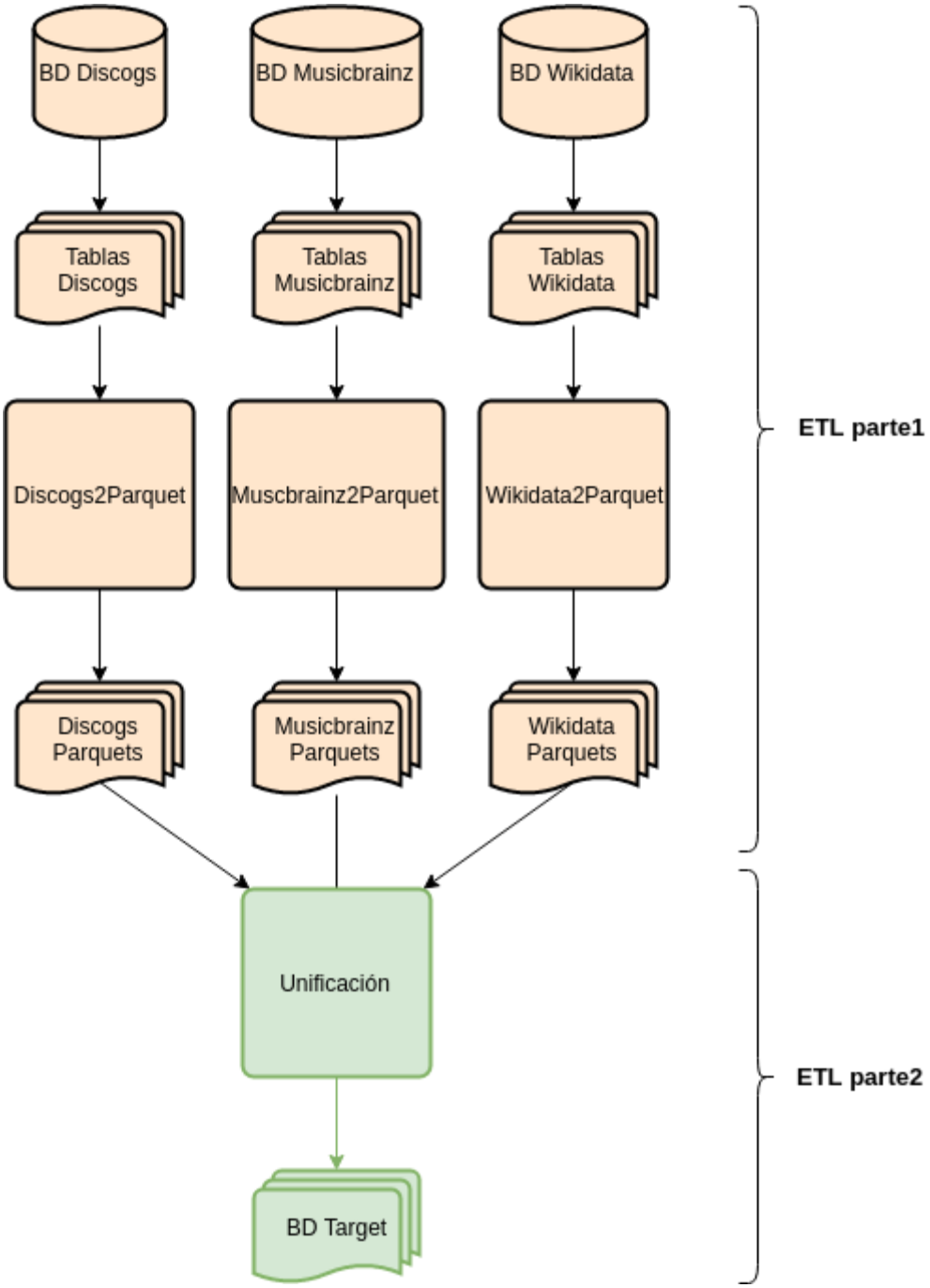

Figura 5: Representación visual de las etapas de ETL en forma de pipeline.

3.3. ETL parte 1

A cada una de estas fuentes se le aplicó un proceso ETL mediante el uso de programas específicos. Este primer ETL tiene como objetivo extraer los datos de las distintas fuentes, limpiarlos, filtrar datos necesarios y generar datasets en formato parquet para cada una de ellas.
A continuación se describen los programas, los datos que consumen, las transformaciones realizadas y el esquema de las tablas generadas por cada uno de ellos.

3.3.1. Discogs2Parquet

Este programa tiene como objetivo la generación de dos datasets, uno de *releases* y otro de artistas desde los respectivos archivos XMLs que provee Discogs.

Para obtener la tabla de *releases* se realizaron las siguientes tareas:

Extract:

- Se leen los datos de discos desde el archivo XML correspondiente descargado desde Discogs. El esquema es el siguiente:
    - artists
    - extraartists
    - tracks.artists
    - tracks.extraartists
    - tracks.subtracks.artists
    - tracks.subtracks.extrartists
- Cada una de estas columnas consiste en un arreglo de estructuras de artistas con información como:
    - nombre
    - rol
- Además de una columna especificando el tipo del release siendo esta información importante para filtrar en la siguiente etapa releases que son compilaciones.

Transform:

- Se concatenaron todos los arreglos de artistas que figuran asociados a un *release*. Estos arreglos son:
    - artists
    - extraartists
    - track.artists
    - track.extraartists
- Se filtraron los artistas que tienen un rol de músico, es decir se dejaron afuera roles como *Productor, Escritor, Ingeniero, Fotógrafo, Diseño, Lyrics, etc.*
- Se cambiaron las columnas que tenían un *null* por un arreglo vacío.
- Todas las columnas que contienen arrays son de tipos básicos (es decir hay arreglos de estructuras)

Load:

- Se guarda el DataFrame generado en formato parquet con el siguiente esquema:

root
| – **id_release**: long
| – **master_id**: long
| – **title**: string
| – **released**: string
| – **country**: string
| – **genres**: array
| | – element: string
| – **formats**: array
| | – element: string
| – **primary_artists**: array
| | – element: long
| – **id_artist**: long
| – **names_artist**: array
| | – element: string
| – **roles_artist**: array
| | – element: string

Para la creación de la tabla de artistas se realizaron las siguientes tareas:

Extract:

- Se leen los datos de artistas desde el archivo XML correspondiente descargado desde Discogs con el siguiente esquema:
  - members: esta columna sirve para discriminar bandas de artistas solistas, ya que si no es vacía, la lista members contiene los IDs y nombres de los artistas que conforman la banda.
  - namevariations: variaciones del nombre del artista o banda.
  - realname: nombre real del artista.
  - profiles: texto descriptivo del artista.
  - aliases: contiene sobrenombres.

Transform:

- Se optó por dejar todas las filas, es decir artistas y bandas.
- Se usan columnas separadas para *members.id y members.name.*

- Así todas las columnas *array* son de tipos básicos (no de estructuras)
- En las columnas con arrays se cambió *null* por arreglo vacío.

Load:

- Finalmente el DataFrame generado se guarda en formato parquet con el siguiente esquema:

root
| – i**d_artist**: long
| – **name**: string
| – **realname**: string
| – **aliases**: array
| | – element: string
| – **namevariations**: array
| | – element: string
| – **groups**: array
| | – element: string
| – **members_id**: array
| | – element: long
| – **members_name**: array
| | – element: string
| – **urls**: array
| | – element: string
| – **profile**: string

3.3.2. Musicbrainz2Parquet

Este programa tiene como argumento el directorio con los datasets del volcado de datos de MusicBrainz.

Extract:
Se cargan las tablas obtenidas de la página de Musicbrainz.
Estas tablas son cinco (en realidad son más pero solamente se utilizaron las que contienen datos relevantes a nuestro estudio):

- tabla **artist**
- tabla **area**
- tabla **url**

- tabla **l_area_area**
- tabla **l_artist_url**

Transform:
En esta etapa nos quedamos con las columnas que nos interesan.
De la tabla **artist** se extrae información de los artistas, se filtran aquellos que son del tipo *Person*, ya que existen seis tipos:

- Orchestra
- Choir
- Group
- Person
- Character
- Other

De la tabla **url** se se extraen: el tipo de url (discogs, allmusic, songkick, others) y el id asociado a cada uno.

De la tabla **area** se extraen: el ID del área, su nombre y tipo. El tipo de área corresponde a alguno de los siguientes (traducidas al español):

- País
- Subdivisión
- Condado
- Municipalidad
- Ciudad
- Distrito
- Isla

Las tablas **l_area_area** y **l_artist_url**, establecen relaciones entre áreas y artistas y urls respectivamente.

Load:
Finalmente se generan cinco datasets (uno por cada archivo de entrada) con los siguientes esquemas:

- artist.pqt:

  root
  | – **id**: long
  | – **mbID**: string
  | – **artist_name**: string
  | – **areaID**: long

- area.pqt:

  root
  | – **areaID**: long
  | – **area_name**: string
  | – **area_type**: integer

- url.pqt:

  root
  | – **urlID**: long
  | – **url**: string
  | – **type**: string ->(discogs, allmusic, songkick, others)
  | – **id_parsed**: string ->(el id extraído del url)

- area_area.pqt:

  root
  | – **areaID**: long
  | – **area2ID**: long

- artist_url.pqt:

  root
  | – **id**: long
  | – **urlID**: long

3.3.3. Wikidata2Parquet

Este programa crea dos datasets en formato parquet: el de las entidades que nos interesan filtradas y otro de las etiquetas (el concepto de etiquetas o *labels* se desarrollará más adelante). Se realizaron las siguientes tareas y suposiciones:

Extract:

- Se levanta el archivo de wikidata como textFile donde cada línea tiene formato JSON y es parseada con la librería *json4s* de Scala.

Transform:

- Se filtraron entidades usando la siguiente regla:

$$EsHumano \ \wedge \ (TieneID \vee TocaInstrumento)$$

- Se extraen Personas, no bandas.
- Un artista puede tener más de una ciudadanía.
- Se construyeron los links de Wikipedia (de todos los idiomas disponibles) y de las imágenes (de Wikimedia)
- Además se extrajeron los IDs correspondientes a otras fuentes de datos, en particular las que nos interesan son de *Musicbrainz y Discogs,* para la unificación de fuentes que se detalla más adelante.
- Se extraen los IDs de los géneros musicales a los cuales pertenece la entidad.
- Se extraen los IDs de las ciudadanas asociadas a la entidad.
- Se extraen los IDs de los instrumentos que tiene asociado la entidad.

Load:
Como resultado se generan dos datasets en formato *parquet*, el primero corresponde a las entidades que satisfacen la regla mencionada anteriormente y cuyo esquema es:

root
| – **wikidata_id**: string
| – **discogs_id**: string
| – **musicbrainz_id**: string
| – **allmusic_id**: string
| – **songkick_id**: string
| – **name**: string
| – **genre_ids**: array

| | – element: string
| – **instrument_ids**: array
| | – element: string
| – **citizenship_ids**: array
| | – element: string
| – **images**: array
| | – element: string
| – **links**: array
| | – element: string

El segundo *parquet* corresponde a las etiquetas (en inglés *labels*) asociados a cada entidad, en este contexto una etiqueta es el nombre que se le da a una entidad, por ejemplo la entidad Q93341 tiene como etiqueta "Miles Davis", la entidad Q8341 tiene como etiqueta "Jazz" y hace referencia al género o estilo musical, la entidad Q12377274 tiene como etiqueta "Trompetista" y hace referencia a la ocupación de la entidad. Además se agrega la descripción de la entidad como dato extra, un ejemplo de descripción de la entidad "Trompetista" es el siguiente:

"musician who plays the trumpet"

Por lo tanto el esquema de este segundo parquet queda de la siguiente manera:

root
| – **id**: string
| – **label**: string
| – **description**: string

3.4. ETL parte 2

El objetivo de este segundo procedimiento consiste en unificar todas las tablas generadas en el proceso ETL anterior y generar las tablas finales que serán usadas durante la etapa de análisis de grafo. Como se establece en la sección *Arquitectura de datos*, todos las datasets son guardadas en formato *parquet* con el fin de aprovechar las optimizaciones que provee *Spark* sobre este formato a la hora de leerlos y crear *DataFrames* a partir de los mismos.
Al conjunto final de datasets generados por esta etapa lo denominaremos *BD Target.*

3.4.1. TargetBuilder

Este programa es el encargado de realizar la unificación de tablas en una tabla final que será utilizada durante la etapa de análisis de grafos. Toma como argumentos todos los parquets generados en la etapa ETL anterior. La salida del programa consiste en tres parquets:

- un parquet de artistas de nombre: **dfArtists**
- un parquet de discos de nombre: **dfReleases**
- un parquet que establece relaciones entre artistas y discos de nombre: **dfArtistsRel**

Generación de dataset **dfArtists**:

Para la generación del dataset de artistas se dividió el procedimiento en dos etapas:

- la primer etapa consiste en la generación de tres datasets intermedias para los artistas, una por cada base de datos.
- la segunda etapa consiste en la unificación de las tres tablas anteriores con el fin de generar una tabla de artistas con datos de las distintas fuentes.

En la primer etapa se generan tres tablas intermedias*: dfArtists1, dfArtists2 y dfArtists3* correspondientes a las fuentes de *Discogs*, *Musicbrainz* y *Wikidata* respectivamente.
A continuación se muestran los esquemas de dichas tablas:

dfArtists1 (discogs):

root
| – **id_artist**: long
| – **name**: string
| – **namevariations**: array
| | – element: string
| – **urls**: array
| | – element: string
| – **genres**: array
| | – element: string
| – **roles**: array
| | – element: string
| – **url_discogs**: string

Notar que **id_artist** corresponde al id de Discogs, es decir, es el mismo que **discogs_id** de las tablas de las otras fuentes.

dfArtists2 (musicbrainz)

root
| – **musicbrainz_id**: string
| – **artist_name**: string
| – **discogs_id**: string

| – **wikidata_id**: string
| – **allmusic_id**: string
| – **discogs_url**: string
| – **wikidata_url**: string
| – **allmusic_url**: string
| – **country**: string

dfArtists3 (wikidata)

root
| – **wikidata_id**: string
| – **discogs_id**: long
| – **musicbrainz_id**: string
| – **name**: string
| – **images**: array
| | – element: string
| – **wikidata_links**: array
| | – element: string
| – **genres3**: array
| | – element: string
| – **instruments**: array
| | – element: string
| – **citizenships**: array
| | – element: string

Con estas tablas generadas se procede con la segunda etapa que consiste en la unificación de tablas, de la siguiente manera:

Primero se unifican solamente las tablas **dfArtists2** y **dfArtists3** correspondientes a *Musicbrainz* y *Wikidata* respectivamente. Para esto se hacen tres *joins*, uno por cada identificador de fuente, es decir:

- se realiza un join entre la tabla **dfArtists3** (*Wikidata*) y **dfArtists2** (*Musicbrainz*) usando como clave el id de *Musicbrainz* (*musicbrainz_id*).
- se realiza un join entre la tabla obtenida en el paso anterior y **dfArtists2** (*Musicbrainz*) usando como clave el id de *Discogs* (*discogs_id*).
- se realiza un join entre la tabla obtenida en el paso anterior y **dfArtists2** (*Musicbrainz*) usando como clave el id de *Wikidata* (*wikidata_id*).

Luego se procede a unificar columnas agregadas en cada *join* y se filtran aquellas que nos interesan:

- **discogs_id**: clave identificadora de Discogs.

- **wikidata_id**: clave identificadora de Wikidata.
- **wikidata_id_others**: lista de claves pertenecientes a distintos idiomas.
- **musicbrainz_id**: clave identificadora de Musicbrainz.
- **musicbrainz_id_others**: lista de claves pertenecientes a distintos perfiles generados para un artista.
- **wikidata_links**: lista de urls hacia las distintas versiones de Wikidata del artista.
- **musicbrainz_url**: lista de urls hacia distintas versiones de Musicbrainz del artista.
- **images**: lista de urls hacia páginas de contenido multimedia (imágenes).
- **instruments**: lista de instrumentos asociados al artista.
- **citizenships**: lista de nacionalidades o ciudadanías asociados al artista.
- **genres**: lista de géneros musicales a los que pertenece el artista.
- **name**: nombre del artista.
- **namevariations**: lista de variaciones del nombre del artista.
- **roles:** lista de roles en que se desempeña el artista.
- **profile**: texto describiendo el perfil del artista.

Finalmente se unifica la tabla obtenida anteriormente (denominada *dfArtists3_2)* con la tabla correspondiente a artistas de Discogs (*dfArtists1*), como se pudo observar en los esquemas presentados con anterioridad, esta última tabla únicamente posee una clave identificadora de Discogs. Para la unificación se procede a hacer *join* de las tablas *dfArtists1* con la tabla *dfArtists3_2* usando como clave el atributo *discogs_id* y se obtiene la tabla de artistas final *dfArtists1_2_3* con información de cada fuente.
Es importante destacar que si en la tabla *dfArtists3_2* hay entidades con valores *null* en la columna *discogs_id*, es decir, no poseen clave identificadora de la base de datos de *Discogs*, entonces estos datos no son agregados a la tabla final y por lo tanto se deja afuera información importante asociada al artista como por ejemplo ciudadania, instrumentos, enlaces a otras fuentes de datos, etc.

Generación de dataset de discos/releases **dfReleases**:

Para el dataset de *releases* básicamente se utilizó el dataset de entrada de *Discogs* pero sin información asociada a artistas.
El esquema del dataset de *releases* queda establecido de la siguiente forma:

root
| – **id_release**: long
| – **master_id**: long

| – **title**: string
| – **released**: string
| – **country**: string
| – **genres**: array
| | – element: string
| – **formats**: array
| | – element: string
| – **primary_artists**: array
| | – element: long
| – **cant_arts**: long
| – **cant_vers**: integer

Generación de dataset de relaciones **dfArtistsRel**:

Esta tabla refleja la relación: "*El artista A tocó con el artista B*" (de acá en más esta relación se denominará "*tocó con*"), la cual se da únicamente si ambos figuran en la lista de artistas que conforman los créditos de uno o varios releases.
Para generar el dataset que representa estas relaciones, se filtran discos con igual *master_id* dejando el menor *id_release*. Si *master_id* es *null* se pone el negativo del *release_id* (se hizo así para facilitar el *groupby*). Este filtrado no se mantuvo en los datos de artistas que salen de la tabla de releases.
La tabla *dfArtistsRel* es simétrica no reflexiva, es decir, se almacena sólo en un sentido, cumpliendo id_musico1 < id_musico2 y tiene el siguiente esquema:

- **id_artist_in**: id de discogs del músico 1.
- **id_artist_out**: id de discogs del músico 2.
- **id_releases**: lista de identificadores de discos que tienen en común, es decir, todos los discos tales que ambos artistas figuran en los créditos.

3.4.2. Programa artists_finder_app

Debido a que durante el proceso de unificación es posible que algunos datos no sean agregados (esto debido posiblemente a la ausencia del ID que haga referencia al perfil de Discogs de un artista) se optó por hacer un programa que permita la actualización de la base de datos unificada. Este programa hecho en Scala tiene como principal objetivo la búsqueda de artistas en la base de datos. La búsqueda de tales artistas es fundamental para generar un archivo CSV de invitados por edición, el cual será usado durante la siguiente etapa, la etapa de análisis.

La búsqueda de artistas se realiza de la siguiente manera:

1) El usuario ingresa el año de la edición del festival al que pertenecen los artistas invitados (esto es necesario para generar archivos de log por año).

2) El usuario ingresa los artistas invitados a la edición.

3) Para cada artista ingresado realiza una búsqueda principal en la BD de Discogs que se genera al final del primer ETL. Existen dos posibles resultados:

    a) En caso de no encontrarse en la misma entonces el artista tampoco estará en la BD Target.

    b) En caso de que se encuentren artistas cuyos nombres coincidan con el deseado, el usuario deberá elegir el artista que considere correcto. Para ayudar en esta selección el programa devuelve nombre de artista y link al perfil de Discogs. Una vez seleccionado el artista correcto se procede con la búsqueda del mismo en las BD complementarias (Musicbrainz y Wikidata).

Durante el proceso de búsqueda el programa va guardando información acerca de las decisiones del usuario, así como también las selecciones en las BD complementarias. De esta manera al finalizar la búsqueda se procede con la actualización de la BD Target, agregando información perteneciente a Musicbrainz y Wikidata en caso de que originalmente no la hubiera. De esta manera el programa ayuda a mejorar la información existente en la BD unificada actualizando atributos que hayan podido ser agregados durante la etapa de unificación

3.5. Análisis de red de músicos

Luego de haber generado la base de datos con información relevante asociada a *discos, artistas* y sus relaciones, debemos modelar la red social con un grafo de colaboraciones[32]. En este grafo, los vértices representan a los artistas. Dos artistas diferentes están unidos por una arista cada vez que existe una relación del tipo: *"tocó con"*, dicho de otra manera, dos artistas están relacionados si tocaron en el mismo disco. Este grafo será objeto de estudio para medir la cercanía de relaciones de colaboración entre los artistas invitados a cada edición del festival y artistas relevantes del ámbito del jazz. El análisis del grafo se hará por año y en base a los artistas invitados de esa edición. Es por esto que para cada año, además de la *BD Target* obtenida en la etapa anterior, son necesarios los siguientes datasets:

- Un archivo CSV que contiene información asociada a artistas invitados por año con las siguientes columnas (generado por el programa *artist_finder_app* el cual se describe en la sección anterior):

[32] "Collaboration graph - Wikipedia." https://en.wikipedia.org/wiki/Collaboration_graph.

    - **id**: identificador del artista en la BD de Discogs.
    - **nombre**: nombre del artista invitado

- Un archivo CSV que contiene información asociada a artistas relevantes. Este archivo es el mismo para cada año y contiene información importante para calcular la métrica de CII (*Collective Initialized Influence*, desarrollada más adelante). El mismo fue creado por un experto en el área y tiene las siguientes columnas:

    - **id**: identificador del artista en la BD de Discogs.
    - **nombre**: nombre del artista relevante.
    - **categoría**: alguna de las dos categorías (1 o 2) necesarias para calcular el valor de CII de cada vértice en el grafo.

Más adelante se detallarán las categorías asociadas a un artista relevante y qué función cumplen en el cálculo del valor de CII.
El pipeline propuesto para la etapa del análisis de grafo se muestra en la Figura 6 y a continuación se realizará una descripción detallada de cada etapa del mismo.

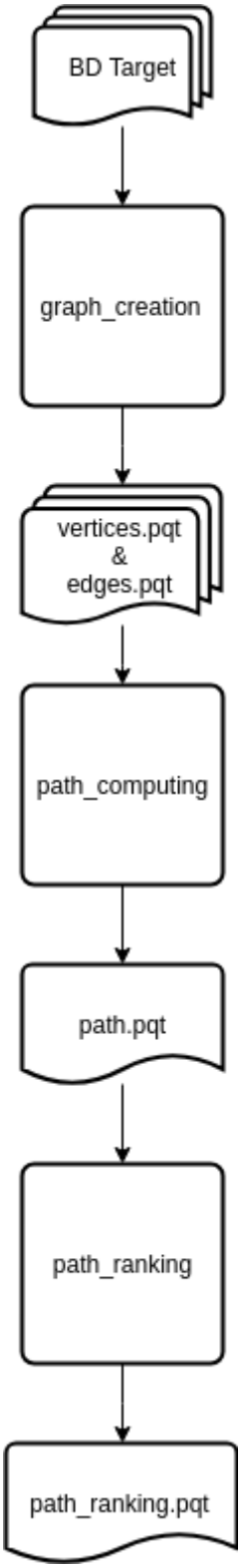

Figura 6: Pipeline para el análisis de grafo y creación de archivos para la visualización.

3.5.1. Creación de grafo (graph_creation)

El programa que genera el grafo de colaboraciones a partir de la *BD Target* se llama *graph_creation*. Los argumentos que recibe el programa son los siguientes:

- el directorio de la *BD Target* que contiene los *parquets* generados en el ETL 2*:*
  - *dfArtists.pqt*: Tabla de artistas.
  - *dfReleases.pqt*: Tabla de discos
  - *dfArtistsRel.pqt*: Tabla de relaciones entre artistas.
- y el directorio del CSV de artistas relevantes

El primer paso consiste en la creación de las tablas de vértices y aristas. Como se explicó en el marco teórico, un *GraphFrame* básicamente es un objeto formado por dos DataFrames, uno que contiene información acerca de los nodos o vértices y otro que contiene información acerca de las aristas.
La tabla de vértices es creada a partir de la tabla de artistas (*dfArtists*) y por lo tanto tiene el mismo esquema que esta pero con el cambio de nombre del identificador de Discogs **discogs_id** a **id** como lo exige *GraphFrame.* La tabla de aristas se crea a partir de las tablas de relaciones (*dfArtistsRel*) y de discos (*dfReleases*). Como se describió antes, la tabla de relaciones tiene el siguiente esquema:

- **id_artists_in:** columna con el id del artista 1.
- **id_artist_out:** columna con el id del artista 2.
- **id_releases:** lista de identificadores de discos que tienen en común el artista 1 y 2.

Con el fin de agregar información asociada a cada disco (o *release*) se usó la tabla de discos la cual posee información asociada a cada uno de ellos tal como título, año, país, entre otros. De esta manera se crea la tabla de aristas con el siguiente esquema:

- **src:** es la columna *id_artist_in* pero con el nombre requerido por GraphFrames.
- **dst**: es la columna *id_artist_out* pero con el nombre requerido por GraphFrames.
- **id_title_list:** es una lista de tuplas *(id_disco, título).*

Las dos primeras columnas son obligatorias para la creación del *GraphFrame* ya que indican conexión entre dos nodos y sentido. En este trabajo se hará uso de un grafo no dirigido, para esto se necesita que la aristas sean no orientadas. Debido a que GraphFrame agrega un sentido de direccionalidad (*src y dst*) una forma de generar indirección es que para cada par de vértices existan dos aristas, una en cada dirección. Es decir se agrega bidireccionalidad entre cada par de artistas. Si existe una arista que conecta al artista A con el artista B:

A ->B

entonces se debe agregar una arista que conecta al artista B con el artista A de la siguiente manera:

B ->A

Para tal fin se procedió a unir la tabla anterior con la misma tabla pero cambiando la columna **src** por la columna **dst**.
Una vez generado el *GraphFrame* se procedió con el cómputo del grado de cada nodo, necesario para calcular la métrica de CI (*Collective Influence*) y como veremos más adelante, la métrica de CII (*Collective Initialized Influence*) creada por nosotros. Para calcular ambas métricas se hizo uso de las primitivas que provee la herramienta para desarrollar algoritmos que utilizan pasaje de mensajes[33]. Como se explica en el Marco Teórico, el valor de CI o de CII de cada nodo será usado para filtrar caminos importantes y así dejar de lado aquellos caminos que no se consideren relevantes. En la siguiente sección veremos cómo se computan estos caminos.
Finalmente la salida de este programa consiste de dos parquets donde se listan los atributos más importantes de cada uno:

- **vertices.pqt** con todos los atributos de cada vértice y sus métricas.
  - todas los atributos mencionadas en dfArtists.pqt (BD Target)
  - **degree** (grado de cada vertice)
  - **ci** (valor de influencia colectiva)
  - **cii** (valor de influencia colectiva inicializada)
- **edges.pqt** que indica la relación "*tocó con*" entre artistas y la lista de discos en la que tocaron juntos.
  - **src** (nodo de partida)
  - **dst** (nodo de llegada)
  - **id_title_list** (lista de tuplas con identificador de disco y título del mismo)

El grafo generado por este programa representa a toda la base de datos (*BD Target*). El mismo tiene aproximadamente 14 millones de aristas y 1 millón de vértices. Estas son cantidades que herramientas específicas de análisis y visualización de grafos no pueden graficar. Dentro de las herramientas testadas se usó *Gephi*[34] que permite el análisis y exploración de grafos de forma visual y es una de las más conocidas en el ámbito. Con *Gephi* se pueden visualizar grafos de hasta 100.000 nodos y 1.000.000 de aristas. Dicho esto, la visualización de grafos de esta magnitud en el navegador queda claramente desestimada, esto debido a la capacidad limitada de los mismos y que con las escalas mencionadas resulta imposible la interpretación del grafo por parte de las personas. La enorme cantidad

[33] "User Guide - GraphFrames 0.5.0 Documentation - Message Passing via AggregateMessages." http://graphframes.github.io/user-guide.html# message-passing-via-aggregatemessages.
[34] "Gephi." https://gephi.org/.

de nodos y aristas (sobre todo estas últimas) forman conexiones tan densas que generan lo que se conoce como "hairball"[35].
Por lo tanto el objetivo será aplicar métodos que nos permita encontrar información relevante a determinados artistas y de esta manera reducir dimensiones para una mejor comprensión de los resultados.
En las secciones siguientes se describen las estrategias implementadas para conseguir este objetivo.

3.5.2. Cálculo de caminos más cortos (path_computing)

El programa encargado de calcular el conjunto de caminos más cortos (*shortest paths*) partiendo de un artista invitado hacia un artista relevante es el denominado *path_ computing.* Para esto, es necesario tener información de los artistas invitados a cada edición del festival y de artistas relevantes. Esta información se encuentra en formato de CSV y son los archivos que se describieron al comienzo.
Los argumentos de este programa son los siguientes:

- los parquets generados anteriormente, para poder reconstruir el *GraphFrame.*
- un directorio que contiene el CSV con información asociada a artistas invitados del año deseado.
- un directorio que contiene el CSV con información asociada a artistas relevantes.
- el número de discos mínimo considerado entre artistas. Por ejemplo si se quiere considerar todas las relaciones entre artistas que tocaron juntos en al menos dos discos este valor debería ser 2.

El número de discos entre artistas tiene como objetivo disminuir la cantidad de aristas y vértices del grafo de caminos. Esto es importante ya que en la visualización de grandes volúmenes de datos se quiere dejar la información relevante. En este caso son más importantes las relaciones con atributo NumRel $>= 2$ ya que indica que los artistas colaboraron de forma más continuada haciendo más relevante la relación. Sin embargo este filtro no es suficiente y para esto es que además se filtran caminos usando las métricas de CI y CII como se explicará en la siguiente etapa del pipeline.
Para cada artista invitado el programa computa los caminos más cortos hacia cada artista relevante, para esto se utiliza el algoritmo de *Breadth-first search (BFS)* que provee *GraphFrames*. Si el algoritmo encuentra un camino con tres aristas, es decir el camino más corto tiene distancia 3, cada fila del *DataFrame* devuelto (recordemos que el algoritmo encuentra todos los caminos con la mínima distancia) tendrá las siguientes columnas:

from | e0 | v1 | e1 | v2 | e2 | to

donde las columnas con nombre **from, v1, v2 , to** son los vértices partiendo desde el artista invitado (from) y con destino en un artista relevante (to) y las columnas **e0, e1,**

[35] "Graphs Beyond the Hairball - EagerEyes." 1 feb.. 2012, https://eagereyes.org/techniques/graphs-hairball.

e2 son las aristas que conectan a dichos artistas. En el caso de no encontrar un camino el algoritmo devuelve un *DataFrame* vacio.
Finalmente se guardan todos los caminos entre todos los artistas invitados y todos los artistas relevantes en un único dataset en formato *parquet*. Este dataset tiene el esquema del *DataFrame* de caminos de mayor distancia, de tal forma que los caminos más cortos tengan un valor nulo (o "*null*") en las columnas que no formen parte de los mismos.

3.5.3. Ranking de caminos (path_ranking)

Debido a que los caminos que se obtienen en el paso anterior entre un par de artistas invitado-relevante, pueden ser demasiados y por lo tanto dificultar la visualización, se decidió filtrar caminos usando un sistema de "rankings". Para esto se tiene en cuenta la relevancia de un camino usando las métricas CI (por *Collective Influence*) y CII (por *Collective Initialized Influence*). Ambas métricas tienen en cuenta la relevancia de los artistas que participan en un camino. Consideramos que un camino es más relevante que otro si lo conforman artistas con un nivel alto de relevancia en el grafo principal.
La métrica de *Collective Influence* se describe en el marco teórico, a continuación se detalla una mejora a *Collective Influence* que se denominó *Collective Initialized Influence* seguido de una explicación sobre el proceso de ranking de caminos.

3.5.4. Mejora a Collective Influence

Como ya se dijo anteriormente, objetivo de usar el valor CI en el filtrado de caminos relevantes es el de no saturar las visualizaciones. Sin embargo surge el problema de que solo se utiliza el grado de los nodos por lo que pueden aparecer músicos sesionistas (es decir músicos contratados para actuaciones en vivo o para sesiones de grabación en un estudio). Estos artistas tienen la particularidad de que se les asigna un alto CI aunque no son relevantes. Es por esto que se propone inicializar el cálculo de CI con un valor relativo a la influencia que tiene el músico según un experto en el área. Por este motivo los artistas relevantes se dividieron en dos categorías atendiendo a diferentes caracterizaciones: una división cronológica entre categorías por un lado; y por el otro, un posterior desarrollo del lenguaje *jazzístico*.

- Categoría 1: Músicos nacidos de la ruptura evidenciada a través del Be-Bop. Se refiere a hijos directos de esa ruptura. Son creadores de estilos o desarrolladores del lenguaje jazzistico (ya sea a nivel compositivo como improvisatorio). Algunos artistas que comprenden esta categoría son:
  - Miles Davis
  - John Coltrane
  - Bill Evans
- Categoría 2: Músicos surgidos de bandas lideradas por artistas de Categoría 1 que impulsaron al jazz durante las décadas posteriores a los años '60. También comprende músicos de una generación posterior caracterizados por desarrollar una escena (nuevas fusiones, espacios musicales de creación) a partir de la cual adquirieron peso propio dentro del género. Algunos artistas dentro de esta categoría son:

- Herbie Hancock
- Wayne Shorter
- John Zorn
- Ron Carter

Habiendo definido las categorías para artistas relevantes, se propone la siguiente fórmula para CII:

$$CII\,(i) \quad = \quad (r_i - 1) \sum_{j=1}^{N_i} a_{ij}\,(r_j - 1)$$

donde:

- a_{ij} es la matriz de adyacencia (= 0 o 1)
- N_i es la cantidad de nodos vecinos del nodo i
- r_i se define como la relevancia del nodo i y está dado por:
  - $r_i = k_i + \alpha * c_i * M$
  - k_i = grado del nodo i
  - c_i = 3 - categoría de relevante o 0 si no está categorizado (= 0 1 2)
  - M = máximo grado de todos los vértices del grafo
  - α = peso que se le da a la categorización manual.

Notar que a diferencia de la fórmula original de CI, acá se reemplaza k_i (el grado del nodo i) por el valor de r_i que definimos como "relevancia del nodo i". Esta relevancia está dada por el grado del nodo, su categoría (que tan relevante es el artista) y una constante α que define la importancia de la categorización manual. Notar además que si $\alpha = 0$ entonces la relevancia es igual al grado, es decir se obtiene la fórmula original.

3.5.5. Descripción de algoritmo de cálculo de ci/cii

Para calcular el valor de ci de cada nodo en el grafo se desarrolló el siguiente algoritmo aprovechando las primitivas provistas por GraphFrames que son:

- AggregateMessages: para el envío de mensajes entre vértices y luego aplicar una función de agregación a cada uno.
- joins: para luego juntar los mensajes computados o "agregados" con el grafo original.

En una primera instancia cada nodo j le envía a sus vecinos (de destino y de origen) el siguiente mensaje:

$$k_j \quad - \quad 1$$

donde k_j es el grado del nodo j.
Una vez que cada nodo recibió el mensaje de cada uno de sus vecinos, se procede con la suma de estos valores mediante la API de *AggregateMessages.*
En esta instancia cada nodo tiene el siguiente valor computado:

$$\sum_{j=1}^{N_i} a_{ij} \, (k_j - 1)$$

Como paso final se procede a agregar este valor al grafo original para luego multiplicarlo por el siguiente valor:

$$k_i \quad - \quad 1$$

donde k_i es el grado de cada nodo del grafo. Es decir cada nodo multiplica el valor de la sumatoria de todos los mensajes recibidos de sus vecinos por su grado menos uno. De esta forma se obtiene el valor de influencia colectiva tal como se plantea en la fórmula.

Para el cálculo de cii (*collective initialized influence*) el algoritmo es similar, lo único que cambia es el mensaje enviado por cada nodo. Ahora el mensaje que recibe cada nodo de sus vecinos es el siguiente:

$$r_j \quad - \quad 1$$

donde r_j es el valor de relevancia del nodo como se definió anteriormente con los siguientes valores calculados previamente:

- un valor α que denota el peso que se le desea dar a la categorización manual.
- un valor M que denota el valor de grado máximo en el grafo.
- la categoría que se le asigna a cada nodo en base a lo explicado en la sección anterior.

con el mensaje recibido de todos los vecinos, cada nodo procede a sumarlos para obtener:

$$\sum_{j=1}^{N} a_{ij} \, (r_j - 1)$$

Finalmente se procede con la unión de estos valores y el grafo original, donde cada nodo computa el siguiente producto:

$$(r_i - 1) \sum_{j=1}^{N} a_{ij} \, (r_j - 1)$$

donde r_i es el valor de relevancia de cada nodo como se definió antes.

Hay que destacar que gracias al uso de las primitivas provistas por GraphFrames que otorgan una abstracción sobre el mecanismo de envío y recepción de mensajes entre nodos, la generación de algoritmos como el de ci y cii, resultan fáciles de implementar y no requieren de una enorme cantidad de líneas de código, de hecho lo único que se modifica en este caso es el mensaje transmitido.

3.5.6. Asignación de rankings a caminos

Recordemos que estas métricas son calculadas durante la etapa de creación de grafo y el programa en esta etapa es el encargado de asignar valores de importancia o ranking a los caminos en base a las mismas.
Para esto una vez generada la tabla de caminos en el paso anterior, se agrupan dichos caminos por par **invitado-relevante** y se procede de la siguiente manera:

- Primero se suman los valores de CI/CII de cada vértice que conforman un camino agregando una columna para cada resultado de la suma: *sum_ ci* y *sum_ cii* respectivamente. Estas sumas se consideran como el valor de relevancia de un camino.
- Luego se define una ventana[36] sobre el *DataFrame* por cada par de artistas **invitado-relevante**, esto nos permite usar funciones de agregación sobre cada ventana.
- FInalmente se aplica la función de agregación *dense_ rank* de Spark a los valores de la suma. Esta función asigna valores de ranking (1, 2, 3 , etc) siendo 1 el ranking más alto. En el caso de haber empate se asigna el mismo valor a cada fila. Esto es, si usamos *dense_ rank* para clasificar caminos y se tiene que dos caminos empatan para el segundo lugar, entonces se asigna el valor de ranking 2 a los dos caminos quedando la clasificación de la siguiente manera: 1, 2, 2, 3, etc.

Finalmente el *DataFrame* generado queda con el siguiente esquema:

from | e0 | v1 | e1 | . . . | to | rank_ci | rank _cii

de esta manera cada camino tiene asociado un valor de ranking el cual usaremos en la siguiente etapa al momento de filtrar caminos para la generación de los archivos necesarios para la visualización de los mismos.

3.6. Visualización

En esta etapa se describirán los distintos programas y diseños utilizados para la visualización de los caminos obtenidos y de esta manera lograr una mejor comprensión de los mismos. La relación entre artistas siempre será la misma: "tocó con", sin embargo la forma de visualizar los caminos que se generan mediante estas relaciones serán distintas.

[36] "Window - Apache Spark.",
https://spark.apache.org/docs/1.6.2/api/java/org/apache/spark/sql/expressions/Window.html.

Como se explica en el marco teórico, esta etapa es la más importante a la hora de comunicar la información deseada al usuario final. Como se dijo anteriormente la visualización del grafo que representa a toda la base de datos resulta en lo que se conoce como "hairball" (ver figura 3) esto es debido a la gran densidad de relaciones que hay en el mismo. Cabe destacar que el grafo que se está visualizando representa a un **subconjunto** de artistas pertenecientes al género Jazz y sus relaciones, esto debido a los límites de la herramienta utilizada. Aunque se trata de un subconjunto de relaciones podemos notar que la extracción de información a simple vista resulta imposible. Es por esto que en las etapas anteriores se aplicaron varios filtros sobre los datos para que la visualización final sea lo más amigable posible.
Con el objetivo de resaltar las distancias entre artistas invitados y relevantes y transmitirlos de distintas maneras se utilizaron cuatro diseños.

Como se explica en el marco teórico, los diseños descritos en esta sección fueron construidos usando como base el algoritmo de dibujo de grafos que provee D3.js denominado *Force Layout*. Los diseños utilizados para comunicar la información al usuario final son los siguientes:

- Concentric Circle Layout
- Concentric "Single" Circle Layout
- Path Layout
- Tree Layout

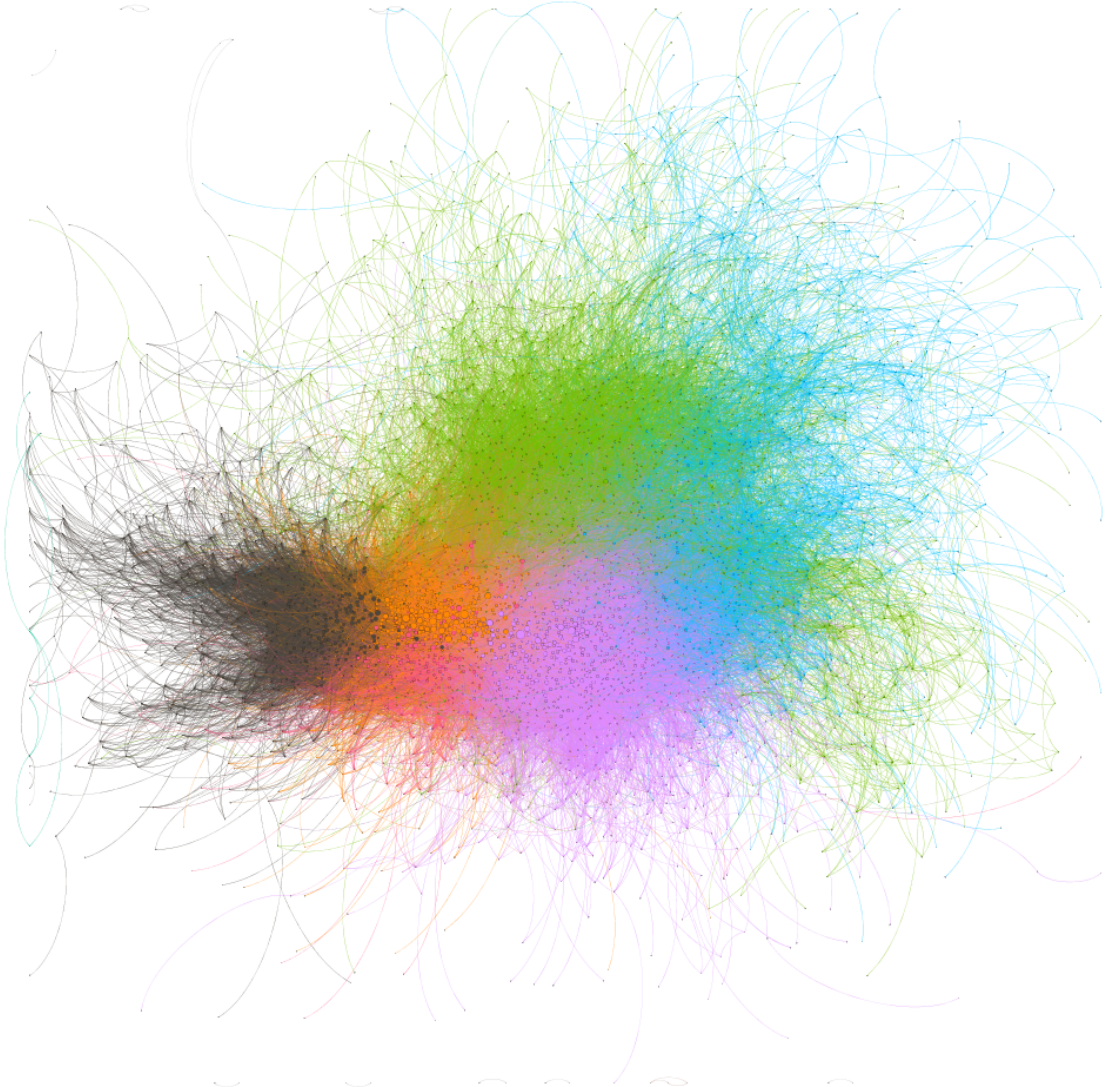

Figura 7: Visualización de un subconjunto de artistas pertenecientes al género de Jazz. En color se pueden apreciar comunidades obtenidas por la herramienta.

A continuación se describen cada uno de estos layouts o diseños y los programas que generan los archivos necesarios para la visualización.

3.6.1. Concentric Circle Layout

En este diseño se representan las distancias con respecto a los artistas relevantes mediante círculos concéntricos. A medida que los círculos se alejan del centro, las distancias aumentan. De esta forma los artistas que se encuentran en el segundo círculo tocaron directamente con un artista relevante, es decir, tienen una "relación directa", mientras

que los artistas que se encuentran en círculos más alejados tienen una "relación indirecta" con el artista relevante a través de una cadena de relaciones "tocó con".
Las posiciones de los artistas invitados en los círculos se calcula en base a sus distancias a artistas relevantes. Pueden existir varios caminos de distintas distancias para un par invitado-relevante, en este caso se posiciona al nodo entre los círculos que representan la menor y mayor distancia.
Además se intenta mostrar todos los caminos que pasan por cada nodo, de esta manera se pueden detectar artistas que "conecten" a varios invitados con varios relevantes. Por último cada nodo comunica visualmente su importancia en la red usando el tamaño del mismo como indicador. Este diseño está inspirado por un ejemplo[37] de uso de la librería D3.js en donde los círculos más internos representan años recientes y los círculos de más afuera son años anteriores.

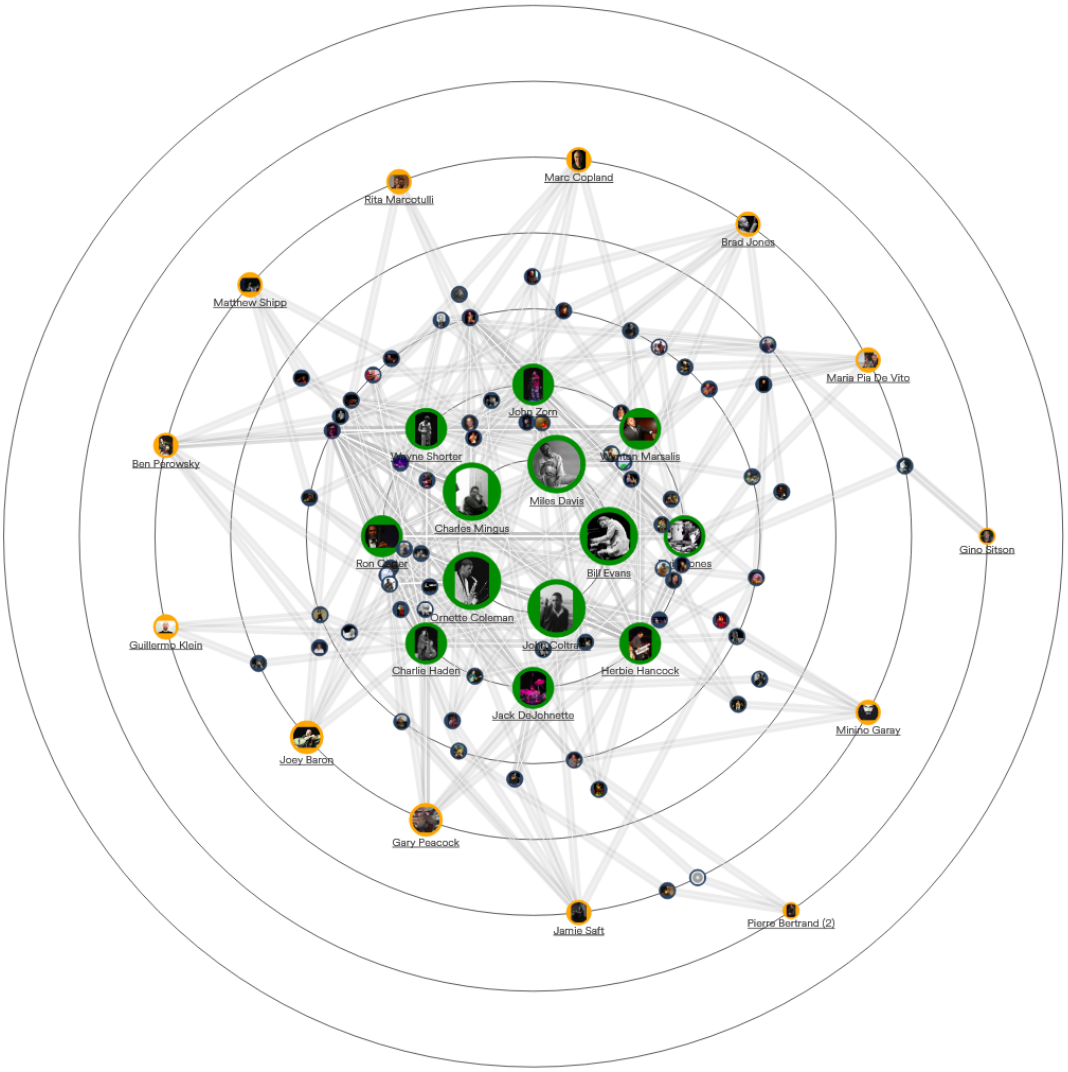

Figura 8: Concentric Circle Layout

[37] "Concentric circle force layout graph in D3.v4 - bl.ocks.org." 12 sept.. 2017, https://bl.ocks.org/davidcdupuis/3f9db940e27e07961fdbaba9f20c79ec.

El programa encargado de generar los archivos necesarios para la visualización es el denominado *graph_popcha*. El nombre de programa está basado en un ejemplo de visualización de un grafo de similitud entre películas[38] cuya relación está dada por la selección de las mismas por parte de los usuarios de la aplicación *PopCha!*.

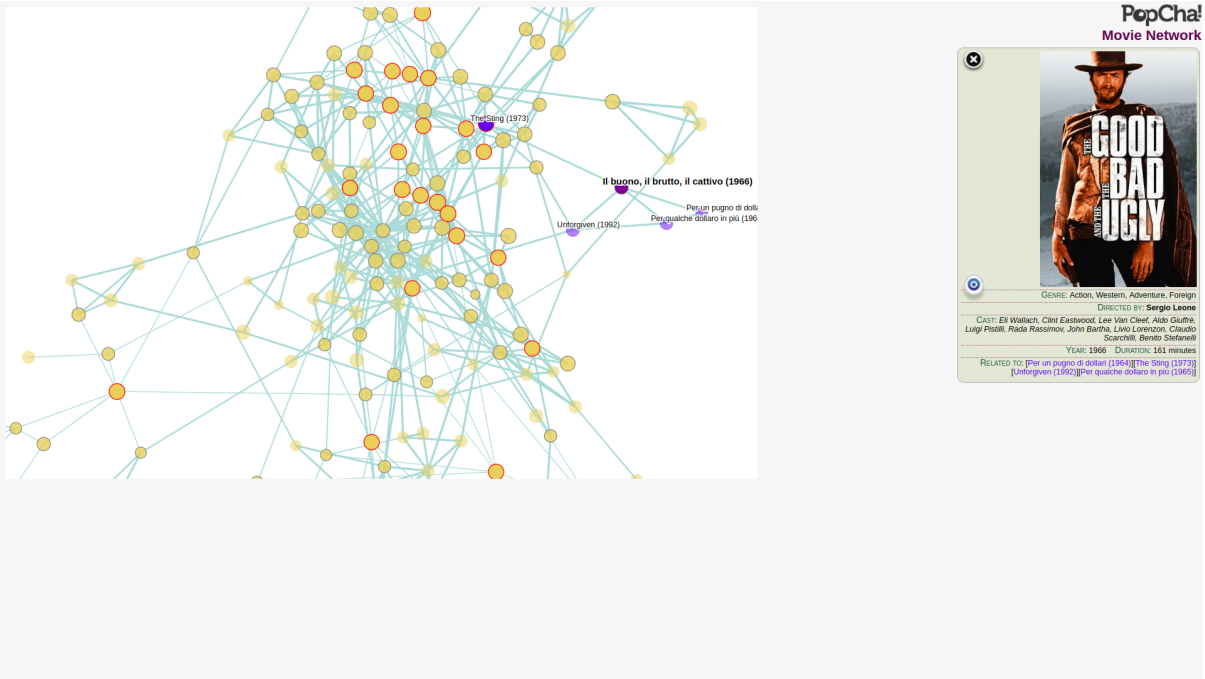

Figura 9: Visualización de similitud entre películas usando un grafo. Datos obtenidos a partir del motor de recomendación PopCha!.

El flujo de trabajo en este programa es el siguiente:

carga de parquet de caminos → extracción de vértices y aristas → JSON

Es importante destacar que este flujo de trabajo también se aplica en el resto de los programas.

Luego de cargar el DataFrame de caminos con valores de ranking asignados, se crean los DataFrames de vértices y aristas a partir del mismo. Debido a que en su definición más simple un grafo es un conjunto de vértices y de aristas. Estas tablas harán más fácil la generación de los JSON ya que los mismos requieren como dato principal tales datos de forma separada. Estos archivos JSON serán cargados posteriormente por código Javascript para la visualización de los mismos.
El esquema de los JSON generados por estos programas es el siguiente:

[38] "9686202 - bl.ocks.org." 5 may.. 2016, http://bl.ocks.org/paulovn/9686202.

```
{
    “nodes” : [
    {
        “index” :
        “id” :
        “class” :
        “name” :
        “image” :
        “degree” :
        “degree_origin” :
        “mbLink” :
        “wikiLink” :
        “ci” :
        “cii” :
        “category” :
        “cat1Distance” :
        “citizenships” :
        “instruments:
        “links1” :
        “links2” :
        ...
        “linksn” :
    } ,
    {
    ...
    }
    ],
    “links” :[
        {
            “source” :
            “target” :
            “weight” :
            “releases” :
            “titles” :
            “images” :
            “path_id” :
            “rank_ci” /” rank_cii” :
        } ,
        {
        ...
        }
```

]
}

Es decir, cada JSON contiene información en formato de lista de objetos nodos y lista de objetos aristas. Recordemos que cada nodo representa a un artista y cada arista la relación "tocó con" que une a dos artistas. Cada nodo tiene los siguientes atributos:

- **index:** id requerido por D3 (debe partir de 0).
- **id:** el id de *Discogs*.
- **class:** la clase del artista (relevante, invitado, otro).
- **name:** nombre del artista.
- **image:** link de la imagen asociada al artista.
- **degree:** grado del nodo en el subgrafo.
- **degree_origin:** grado del nodo en el grafo original.
- **mbLink:** link del artista en *Musicbrainz*.
- **wikiLink:** link del artista en *Wikidata*.
- **ci:** valor de CI del nodo.
- **cii**: valor de CII del nodo.
- **category:** categoría del artista (0, 1 o 2).
- **cat1Distance:** arreglo de distancias a artistas de categoría 1.
- **citizenships:** arreglo de ciudadanías asociadas al artista obtenidas de las distintas fuentes.
- **instruments:** arreglo de instrumentos asociados al artista obtenidos de las distintas fuentes.
- **linksn:** arreglo de ids de vértices de ranking n que conforman los caminos que atraviesan al nodo.

A su vez cada arista tiene los siguientes atributos:

- **source:** id del nodo de origen.
- **target:** id del nodo de llegada.
- **weight:** peso calculado en base a la cantidad de discos que tienen en común los artistas que relaciona esta arista.

- **releases:** arreglo de ids de los discos que tienen en común los artistas que relaciona esta arista.
- **titles:** arreglo de títulos de discos que tienen en común los artistas que relaciona esta arista.
- **images:** arreglo de links de portadas de discos.
- **path_id:** id del artista invitado para identificar a qué camino pertenece esta arista.
- **rank_ci/rank_cii:** el valor del ranking del camino al cual pertenece esta arista.

La salida del programa es un JSON con el esquema que se describe arriba.

3.6.2. Concentric "Single" Circle Layout

Este diseño tiene el mismo concepto de distancias que "Concentric Circular Layout", pero en este caso el usuario puede enfocarse en las relaciones de artistas invitados con un único artista relevante a la vez.

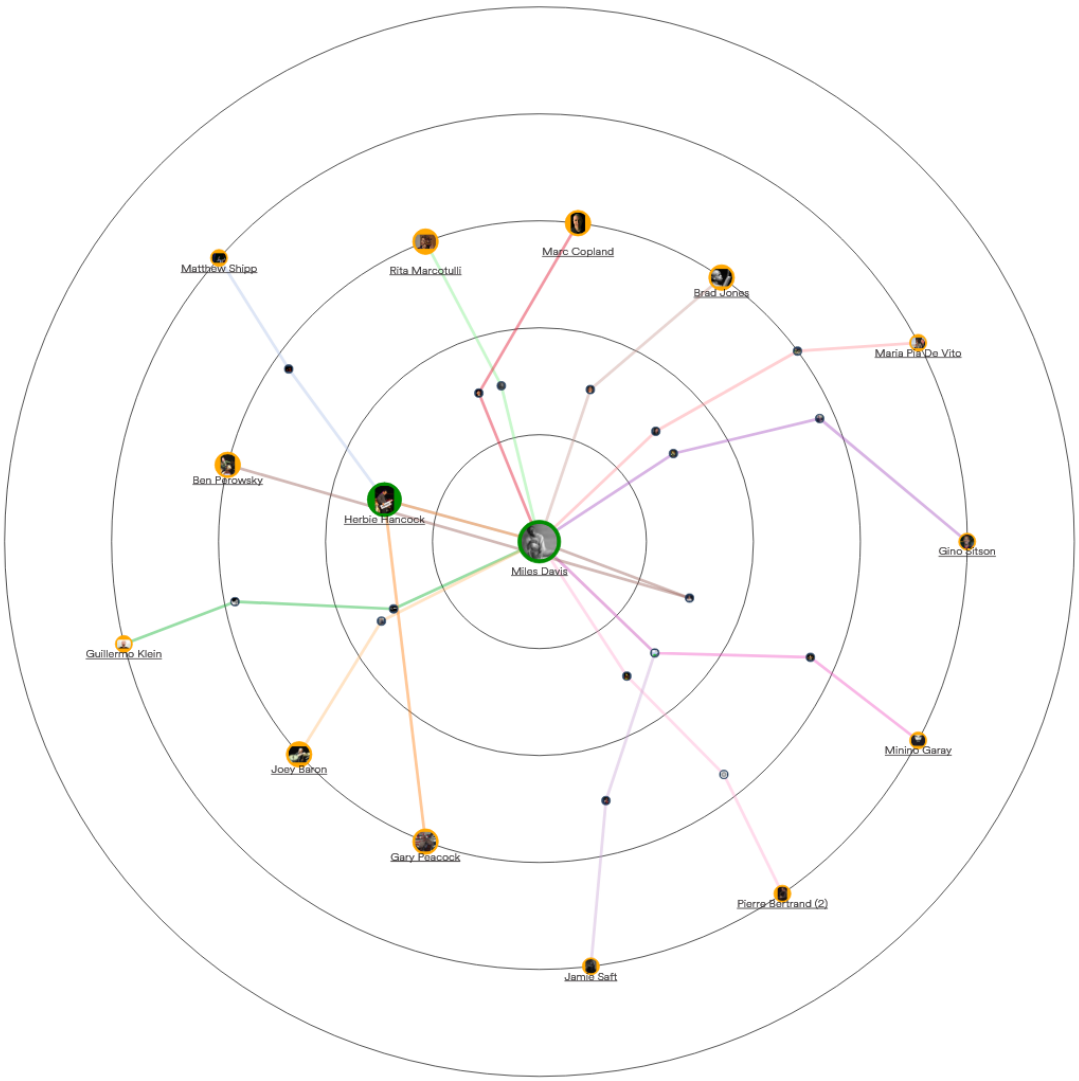

Figura 10: Concentric Single Circle Layout.

El programa encargado de generar los archivos correspondientes para la visualización es el denominado *graph_popcha_individual* que, al igual que *graph_popcha*, genera un archivo JSON para cada métrica (CI y CII) con los caminos a cada artista relevante, sin embargo este JSON tiene la particularidad de que los caminos están agrupados por artista relevante. Además genera un archivo que contiene la lista de artistas relevantes para automatizar la selección de los mismos en la visualización.

3.6.3. Path Layout

En este diseño se intenta mejorar la visualización de caminos del diseño anterior en un formato horizontal.
El posicionamiento de artistas invitados se basa en la mayor distancia a artistas de categoría 1 los cuales son visualizados en el extremo derecho. Esto es así para evitar que artistas intermedios (artistas que forman parte del camino hacia artistas relevantes) no sean posicionados detrás de un artista invitado ya que lo que se quiere es dar una idea de avance de izquierda a derecha evitando retroceder en el camino.
El usuario tiene la posibilidad de expandir los nodos para una correcta visualización de caminos (por ejemplo para evitar superposición de caminos).

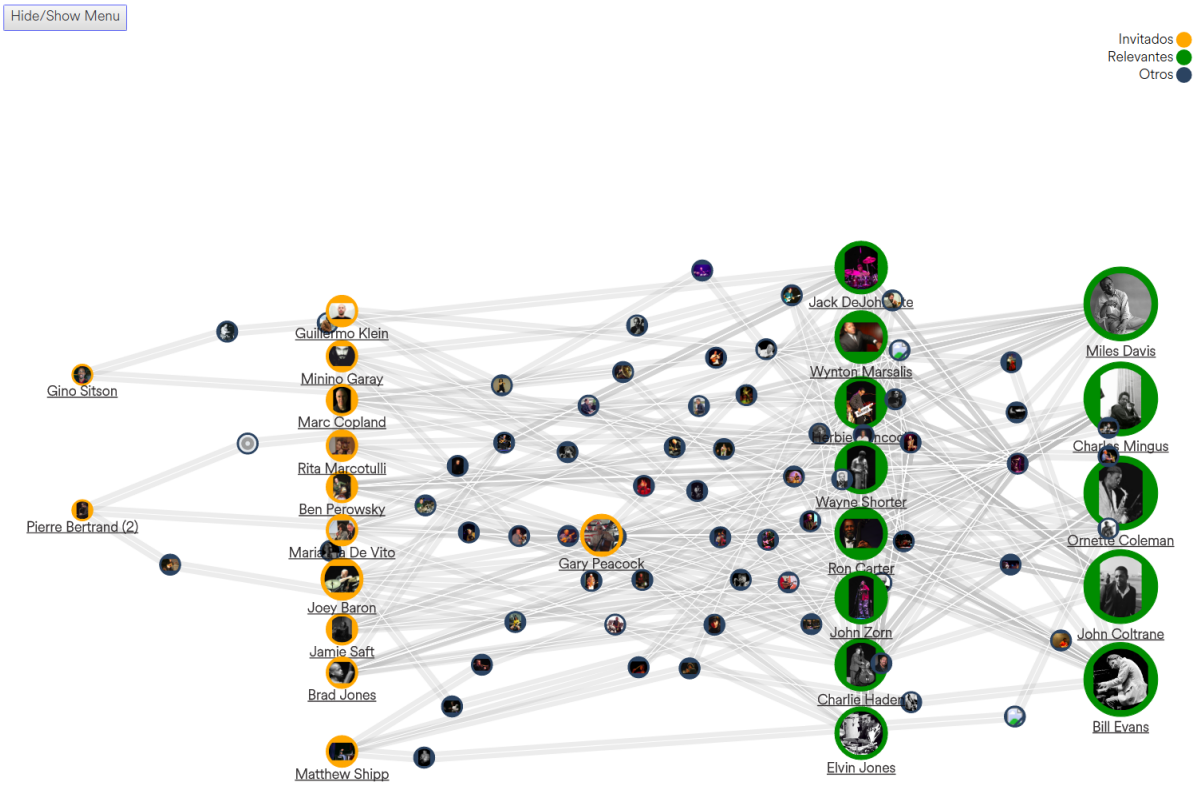

Figura 11: Path Layout.

El programa que genera el JSON correspondiente a esta visualización es el mismo que se utiliza para el diseño *Concentric Circle Layout*, es decir *graph_popcha*.

3.6.4. Tree Layout

Este diseño es el único que no fue construido usando *force layout*. la visualización utiliza el layout d3.tree[39] de D3.js el cual implementa el algoritmo Reingold-Tilford[11] para una disposición ordenada y eficiente de los nodos en capas. La profundidad de los nodos se calcula a partir de la distancia desde el nodo raíz, lo cual da lugar a una apariencia irregular.
El mismo, como se dijo anteriormente, es utilizado en la app de Spotify "Artist Explorer". Este diseño tiene como objetivo que el usuario pueda explorar las relaciones entre artistas invitados y relevantes.

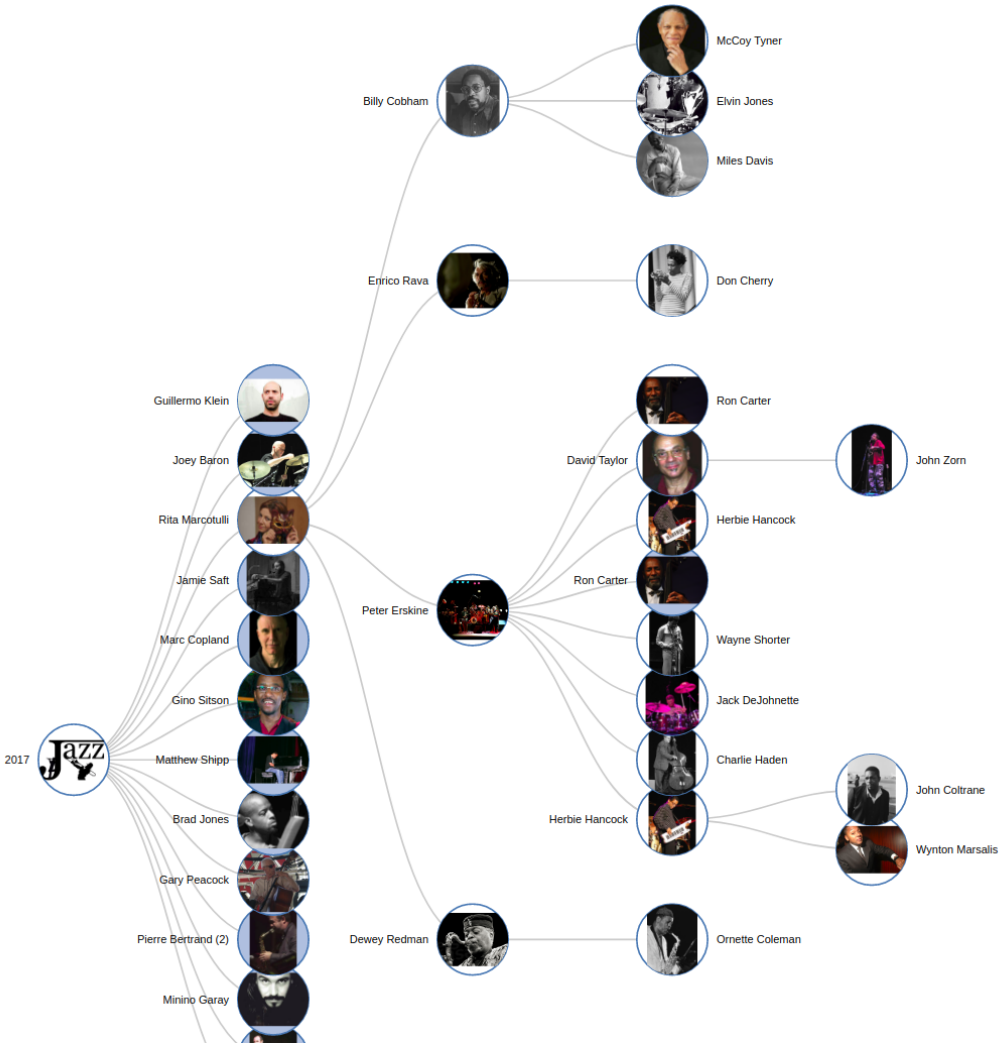

Figura 12: Tree Layout.

[39] https://github.com/d3/d3-hierarchy/blob/master/README.mdtree

El programa que genera los archivos JSON para esta visualización es el denominado *graph_path_tree*. Este programa genera un archivo JSON para la visualización de caminos en formato similar al *Artist Explorer*[40] de Spotify, el cual muestra datos de artistas obtenidos de la Web API[41] de Spotify donde la relación entre artistas es de similaridad, en base al análisis del historial de reproducciones de los usuarios.

El esquema del JSON generado por el programa es el siguiente:


```
{
        "name" :
        "image" :
        "children" :[
                {
                        "name" :
                        "image" :
                        "children" : [{ ...} ]
                } ,
                {
                        "name" :
                        "image" :
                        "children" : [{ ...} ]
                } ,
                ...
        ]
}
```


El esquema es similar al de un árbol con un nodo principal o nodo raíz representando al artista invitado y una determinada cantidad de nodos "hijos". A su vez cada nodo "hijo" posee una determinada cantidad de nodos "hijos". Es así que cada nodo tiene 3 atributos: *node, image* y *children,* a excepción de los nodos "hoja" o nodos finales que representan a los artistas relevantes los cuales únicamente tienen los dos primeros atributos: *name* e *image.* A continuación se describen cada uno de estos atributos:

- **name**: nombre del artista.
- **image**: link de la imagen asociada al artista.
- **children**: lista de nodos "hijos" o de artistas del siguiente nivel, cada uno de estos artistas conforma un "salto" en el camino hacia artistas relevantes, es decir a medida que expandimos estos nodos nos vamos acercando a artistas relevantes.

[40] "Artist Explorer." https://artist-explorer.glitch.me/.
[41] "Web API | Spotify for Developers." https://developer.spotify.com/documentation/web-api/.

4. Conclusión y trabajo futuro

4.1. Conclusión

La representación de información en grafos es ampliamente utilizada en aplicaciones como las redes sociales y, en el caso de este trabajo, en la representación de una red social de músicos. La manera más simple de representar esta información es mediante diagrama de nodos y aristas. Estas imágenes son fáciles de entender incluso por personas que nunca han visto un diagrama de este tipo. La mayoría puede responder fácilmente preguntas básicas usando dicho diagrama como por ejemplo quien es el artista relevante más cercano. El problema surge cuando el número de nodos y aristas aumenta considerablemente debido a la enorme cantidad de datos provenientes de diversas fuentes.
Resulta importante, entonces, el uso de herramientas capaces de procesar estas cantidades de datos de manera rápida y eficiente. Pero también la reducción de dimensionalidad del grafo de tal manera que la información que se quiere transmitir o comunicar no se vea opacada. Con el fin de reducir dimensionalidad, se usaron métricas para filtrar caminos mediante la sumatoria de importancia de artistas en los mismos. Esta métrica se calculó en base a la influencia colectiva de los artistas pero no fué suficiente debido a que los caminos filtrados aún eran demasiados y la visualización, por lo tanto, se tornaba densa. Es por esto que se elaboró una métrica de importancia de músicos (*collective initialized influence*) que agrega información de un experto, a diferencia de la métrica anterior que usaba solo la información del grado de los nodos. Con estos nuevos valores de importancia se logró reducir dimensionalidad filtrando los caminos que son de mayor relevancia.
Finalmente, la selección de diseños específicos de visualización y la modificación de algunos de estos permitió una mejor representación de las relaciones entre artistas y sus cercanías a artistas relevantes del género.
Con el fin de obtener *feedback* sobre las visualizaciones producidas, se formularon las siguientes preguntas a la parte experta:

¿Reflejan correctamente la lejanía o cercanía de los artistas invitados a los artistas relevantes?

"Se clasificó a los artistas relevantes pretendiendo agruparlos en grados de importancia (de acuerdo a la importancia del artista con relación a su contribución al desarrollo del campo musical, en particular al **jazz moderno**). También se tuvieron en cuenta las variantes revisionistas de las generaciones mas cercanas a la actualidad.
De ese modo se logró reducir el prolifico mapa de músicos del género hasta llegar a un puñado de nombres representativos: por su relevancia y reconocimiento.
La visualización de la cercanía, pone de relieve la jerarquía de los músicos que visitan nuestra provincia durante el Festival Internacional de Jazz de la Provincia de Córdoba. De todas las alternativas producidas, solo algunas cumplen con la totalidad de los siguientes requisitos, considerados importantes a la hora de comunicar algo visualmente: elocuencia, claridad y usabilidad. Con el objetivo de mejorar estas características, resulta importante el trabajo posterior en conjunto con el diseñador de la página web del festival."

¿Se entiende lo que la visualización intenta comunicar?

"En mi carácter de experto, las visualizaciones cumplen con el objetivo de comunicar las relaciones de los músicos visitantes con los relevantes. Las mismas han sido testeadas con personas no expertas y si bien se entiende la común intención comunicativa de las visualizaciones, algunas de ellas han demostrado ser más usables. Es este sentido, se propone la continuación en el trabajo sobre estas visualizaciones con los encargados de comunicación del Festival a fin de mejorar aún más el mencionado aspecto."

4.2. Trabajo futuro

A continuación se proponen diversas tareas que pueden mejorar el sistema creado en este trabajo y que por motivos de tiempo y dificultad no fueron incorporados.

- Realizar un filtrado más agresivo en el primer ETL, con el fin de achicar el input en la etapa de unificación del segundo ETL.
- Mejorar la unificación de tablas en el segundo ETL, usando los nombre de los artistas. Para lo cual se hace necesario la definición de reglas de modificación de nombres.
- Generar layouts en base a atributos de artistas, así como nacionalidad, instrumentos, roles, etc. Otra alternativa puede ser modificar los diseños actuales para agregar esta información.
- Considerar variantes en las relaciones, es decir, agregar otros tipos de relaciones aparte de la que se utiliza en este trabajo.
- Asignar ranking a los caminos usando *machine learning*, en particular se propone usar *embeddings* de nodos[12] para aprendizaje de features o *feature learning*.

Referencias

[1] Morone Flaviano y col. “Collective Influence Algorithm to find influencers via optimal percolation in massively large social media”. En: *Scientific Reports* (2016).

[2] M. Miller, J. Walloch y M. C. Pattuelli. “Visualizing Linked Jazz: A web-based tool for social network analysis and exploration”. En: (2012).

[3] Fernanda B. Viégas y Judith Donath. “Social Network Visualization: Can We Go Beyond the Graph?” En: *Workshop on Social Networks for Design and Analysis: Using Network Information in CSCW* (2004).

[4] Pablo Gleiser y Leon Danon. “Community Structure in Jazz”. En: *Advances in Complex Systems, Vol. 6, No. 4 (2003) 565-573* (2003).

[5] W. Gropp y E. Lusk. “The MPI communication library: its design and a portable implementation”. En: (1993).

[6] Jeffrey Dean y Sanjay Ghemawat. “MapReduce: Simplified Data Processing on Large Clusters”. En: (2004).

[7] Matei Zaharia y col. “Spark: Cluster Computing with Working Sets”. En: (2010).

[8] Jacob Moreno. “The Theatre of Spontaneity”. En: (1947).

[9] John Barnes. “"Class and Committees in a Norwegian Island Parish.” En: (1954).

[10] Harrison C. White. “Markets from Networks: Socioeconomic Models of Production”. En: (2002).

[11] Edward M. Reingold y John S. Tilford. “Tidier Drawings of Trees”. En: *IEEE Trans. Software Engineering 7* (1981).

[12] Aditya Grover y Jure Leskovec. “node2vec: Scalable Feature Learning for Networks”. En: (2016).

Los abajo firmantes, miembros del Tribunal de Evaluación de tesis, damos Fe que el presente ejemplar impreso, se corresponde con el aprobado por éste Tribunal