Minería de Medios Sociales - 2024-2025

Análisis de Redes

Universidad de Granada

Análisis de Redes con Gephi

Miguel García López

77149865E

migue8gl@correo.ugr.es

Índice

Índice de figuras

Índice de cuadros

1. Introducción

A lo largo de esta práctica, se estudiarán diversas medidas fundamentales en el análisis de redes, permitiendo comprender las características estructurales de una red social y la influencia de sus actores principales.

En esta práctica, se trabajará con datos provenientes de una red que codifica la estructura de un programa de *Java* (*S.Heymann, J.Palmier, 2008.*) (fig 1). Se trabajará esta red en el *software* de *Gephi* y se aplicarán herramientas de análisis y visualización para extraer información relevante sobre su estructura y dinámicas internas.

A través de este proceso, se aplicarán conceptos de análisis de redes sociales, incluyendo medidas como el coeficiente de agrupamiento, la distribución de grado y la centralidad, esenciales para comprender la estructura global y local de la red. Además, se explorará la detección de comunidades dentro de la red, proporcionando una visión detallada de sus interacciones y sub-estructuras.

2. Análisis básico

Medida	**Valor**
Número de nodos N	724
Número de enlaces L	1025
Número máximo de enlaces $L_{\text{máx}}$	524176
Densidad del grafo $\frac{L}{L_{\text{máx}}}$	0.002
Grado medio $\langle k \rangle$	1.416
Distancia media d	1.985
Diámetro $d_{\text{máx}}$	5
Coeficiente medio de clustering $\langle C \rangle$	0.079
Número de componentes conexas	24
Número de nodos componente gigante (y %)	667 (92.13 %)
Número de aristas componente gigante (y %)	981 (95.71 %)

Cuadro 1: Métricas del grafo

Como puede verse en la tabla 1, la densidad es de tan solo el 1,9 %, lo que corresponde con un valor muy bajo dada la cantidad máxima de enlaces que la red puede tener, es una red poco densa.

En cuanto al grado medio, este tiene un valor de 1,41, es decir, cada nodo tiene de media solo 1 nodo referenciado. Siendo un programa de *Java*, pese a que cada nodo podría tener potencialmente $724 \cdot 724$ referencias, ya que es un grafo dirigido cíclico (N^2), es normal que no todas las clases, métodos, funciones se referencien a todos ellas.

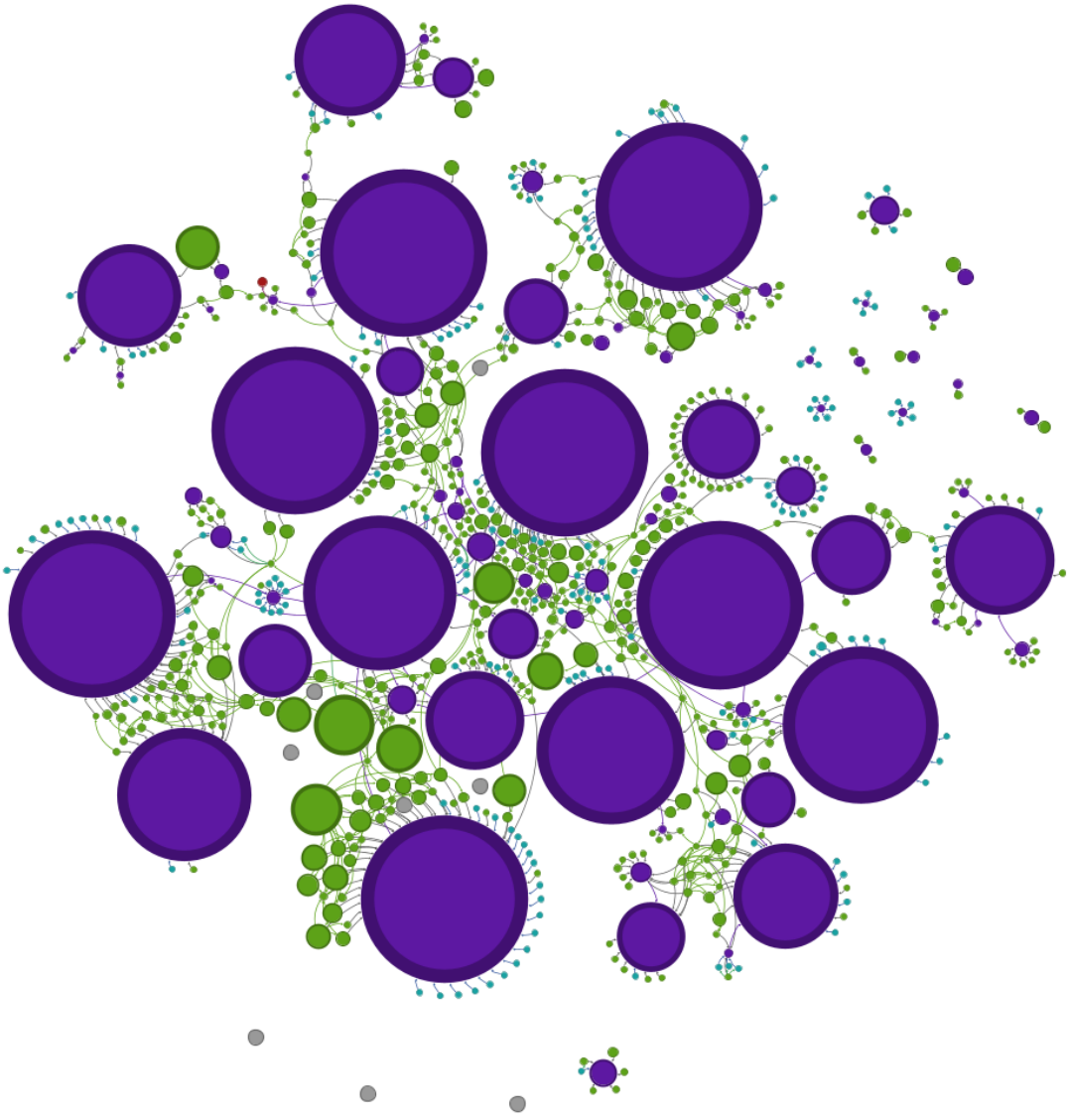


Figura 1: Red original de un programa de *Java*

Figura 2: Componente gigante del grafo

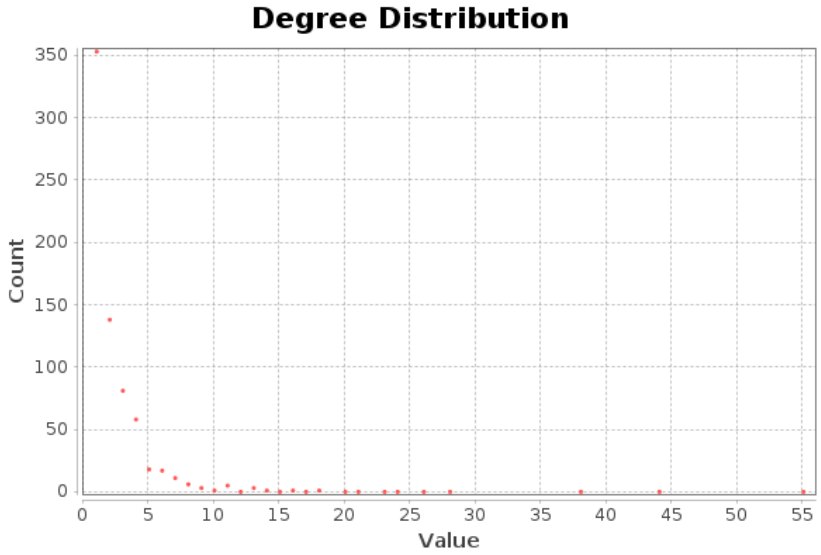


Figura 3: Distribución de grados de los nodos

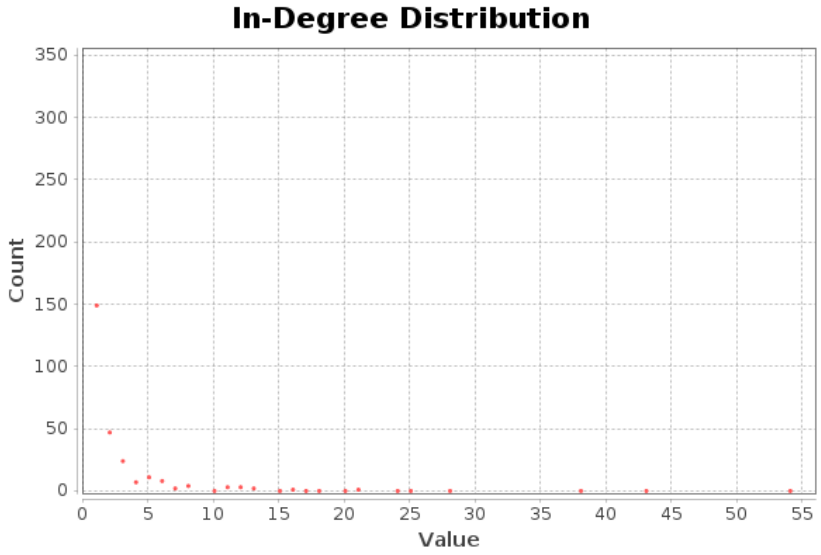


Figura 4: Distribución de grados (enlaces entrantes) de los nodos

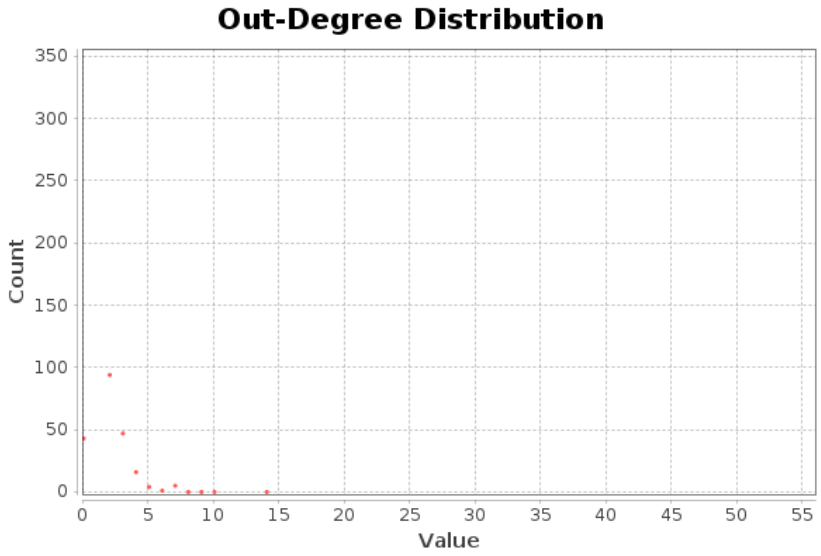


Figura 5: Distribución de grados (enlaces salientes) de los nodos

Como puede observarse en la figura 3, la mayoría de los nodos, concretamente el 90 % tienen menos de 5 referencias. Como cursiosidad, en la figura 6 se ven el 90 % de los nodos con grado menor a 5.

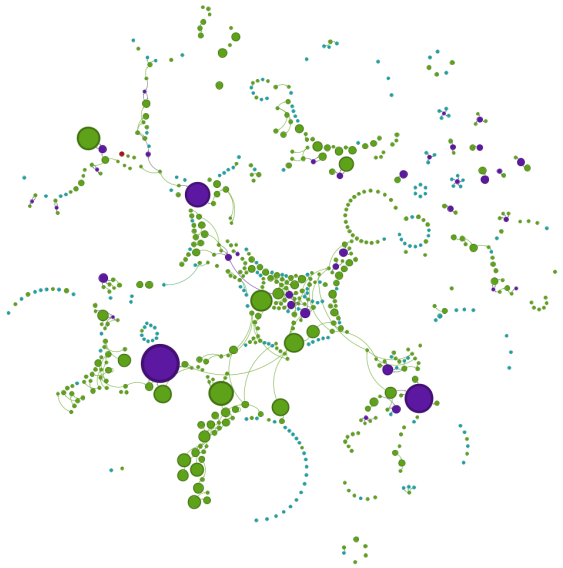


Figura 6: Grafo con solo nodos de $k \leq 5$ (90 %)

En esta red se da la propiedad de libertad de escala, es decir, su distribución de grados sigue una ley de potencias. Esto quiere decir que la gran mayoría de nodos apenas tienen conexiones, solo unos pocos nodos abarcan la mayoría de enlaces.

El 4,7 % de los nodos tienen una cantidad de enlaces superior a 9 (fig 7), y solo el 0,55 % superan los 28. Estos son los *hubs* de esta red. Concretamente son clases como `CodeData`,

`GEXFExporter` o `StdPage`, que sin tener conocimiento experto sobre su funcionamiento, son funcionalidades muy importantes y tiene sentido que tengan tantos enlaces. Esas clases además tienen su mayoría de enlaces entrantes (*in-degree*), también tiene sentido ya que seguramente, al ser clases tan importantes, realicen una carga importante de trabajo y sean llamadas desde muchos archivos de código.

Por el contrario, nodos como `add` o `build` son métodos con mayoría de enlaces salientes y ningún entrante. Esto es así ya que referencian clases como las ya mencionadas en sus argumentos.

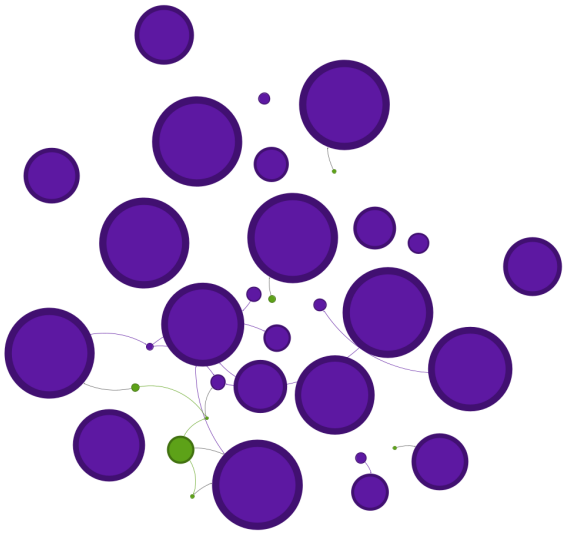


Figura 7: Grafo con solo nodos de $k > 9$ (4,7 %)

La distancia media es de 1,9, es decir, la cantidad de enlaces entre dos nodos es casi 2. La distancia máxima o diámetro es 5, por lo que se puede recorrer cualquier camino del grafo con tan solo cinco saltos de nodo a nodo. Son valores bastante bajos.

La red presenta 24 componentes conexas, pero son agrupaciones de tan solo unos pocos nodos cada una de las componentes detectadas. Se tiene que el 92,13 de los nodos están agrupados en una componente gigante. Esto es una señal de estructura robusta en términos de conectividad.

El valor de coeficiente de *clustering* medio es de 0,079, es decir, es un valor bastante bajo que indica que solo el 7,9 % de las conexiones posibles entre vecinos de un nodo están realmente presentes, lo que indica poca cohesión local. Se puede ver que la distribución de estos coeficientes es dispersa (fig 8), ya que la mayoría tienen valores cercanos a cero, mientras que existen algunos nodos atípicos que tiene un coeficiente mayor a 0,4. También hay unos pocos en mitad de la distribución de valores de coeficiente, pero la mayoría se agrupan en cero.

La densidad del grafo es muy baja, de tan solo el 2 % (pocos enlaces en la red) y

el coeficiente medio de *clustering* es también bastante bajo. Esto sugiere una estructura laxa, descentralizada y con pocos grupos fuertemente conectados. En la figura 9 se observa como el coeficiente de *clustering* es mayor en nodos poco conectados, aunque la tendencia sigue siendo que la mayoría de nodos tengan un coeficiente bajo, esto indica que los *hubs* suelen formar estructuras más dispersas y los nodos con pocos enlaces se agrupan en zonas locales más densas (aunque no la mayoría).

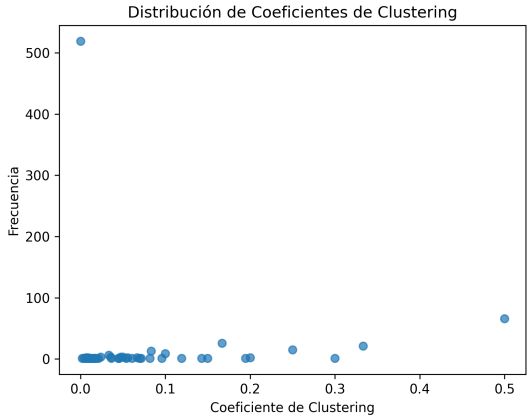


Figura 8: Distribución de los coeficientes de *clustering*

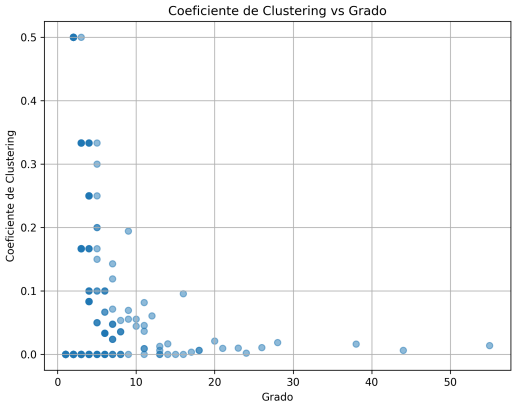


Figura 9: Distribución de los coeficientes de *clustering* por grado

3. Identificación de actores principales

Los actores principales quedan identificados en las tablas 2 y 3. A continuación de analizan todos ellos por métrica.

Cercanía	**Intermediación**
`IExporter`: 1	`AbstractExporter`: 0,00066
`ICodeData`: 1	`GEXFExporter`: 0,00048
`AbstractScenario`: 1	`CodeData`: 0,00038
`ICodeMinerParser`: 1	`ICodeData`: 0,0003
`CodeminerResourceVisitor`: 1	`IExporter`: 0,0002

Cuadro 2: Medidas de Cercanía e Intermediación en la red (parte 1).

Grado	**Vector propio**
`GEXFExporter`: 55	`AbstractExporter`: 1
`CodeData`: 44	`GEXFExporter`: 0,84
`StdPage`: 38	`IInfoElement<T>`: 0,73
`ExtendedMethodDependenceDetector`: 28	`CodeData`: 0,61
`StdAnalyse`: 26	`CodeDataChangesListener`: 0,59

Cuadro 3: Medidas de Grado y Vector Propio en la red (parte 2).

3.1. Centralidad de cercanía

Todos los nodos, e incluso muchos más fuera del top 5, tienen una valor de cercanía máximo. Dado que la red tiene una distancia media de 1,9 y un diámetro de 5, por lo que es normal que existan muchísimos nodos con un valor tan alto en cuanto a cercanía con el resto de nodos. Dados estos resultados en los que hay tantísimos nodos con el mismo valor, se considera que no tiene sentido la detección de actores en esta métrica, al menos de forma aislada.

3.2. Centralidad de intermediación

Los actores con valores más altos en esta métrica actúan como "puentes" entre comunidades. `AbstractExporter` es una clase abstracta que centraliza la lógica de exportación, que podría estar conectando implementaciones concretas (`GEXFExporter`) con otros módulos. `CodeData` podría estar almacenando datos esenciales del programa, siendo un punto de paso obligado para operaciones de análisis.

Lo mismo se aplica a las interfaces `ICodeData` y `IExplorer`, las cuales, tratándose de *Java*, que es un lenguaje con paradigma de programación orientada a objetos, se utilizan en múltiples definiciones de clases, lo cual cuadra con que estén relacionadas con las clases más importantes y con su naturaleza de interfaz.

3.3. Centralidad de grado

Estos actores son *hubs* con numerosas conexiones directas, concretamente conexiones *in-degree* o entrantes. Posiblemente `GEXFExporter` sea una implementación concreta de exportación a formato *GEXF*, referenciada por múltiples clases que necesitan generar salidas, y `CodeData` una clase que centraliza datos críticos, siendo accedida por numerosos componentes (métodos de análisis, visualización, etc.). Su alto grado refleja su rol como núcleos funcionales del programa.

3.4. Centralidad de vector propio

Estos actores están conectados a otros nodos influyentes, formando una élite estructural, ya que un nodo es considerado importante, si está conectado a otros nodos importantes. La clase abstracta `AbstractExporter` está conectada con `GEXFExporter` y otras clases de exportación, lo cual le otorga influencia global. `IInfoElement<T>` es una interfaz genérica que probablemente define elementos de datos utilizados transversalmente en el sistema, lo cual hace que esté conectada a (seguramente) casi todas las clases importantes.

3.5. Información extraída del anásis de actores

Existen dependencias críticas en clases como `CodeData` y `GEXFExporter`, ya que son puntos únicos de fallo y cambios en estos nodos afectarían a gran parte del sistema. También parece existir un bajo acoplamiento, debido a la baja densidad general (0,002) que indica que la mayoría de nodos tienen responsabilidades especializadas, pero los *hubs* aseguran la integración global.

Se puede ver en la figura 10 una visualización de aquellos nodos con mayor valor en la centralidad de vector propio representado por el tamaño. Los nodos más oscuros son aquellos con mayor centralidad de cercanía. Como se puede observar hay actores, como por ejemplo `AbstractExporter`, que mantienen valores grandes en ambas métricas. Es interesante ver como la mayor parte de nodos con altos valores de centralidad de cercanía son nodos con un tamaño muy pequeño, es decir, con poca importancia en cuanto a enlaces a otros nodos importantes.

Esto sugiere que estos nodos, aunque pueden acceder rápidamente a otros componentes del sistema (alta cercanía), no están necesariamente conectados a los actores más influyentes de la red, lo que se comentaba anteriormente del bajo acoplamiento.

Esta distribución podría indicar un diseño modular donde ciertos componentes están bien posicionados para comunicarse eficientemente con el resto del sistema, pero no dependen exclusivamente de los nodos centrales. Tal estructura podría beneficiar la man-

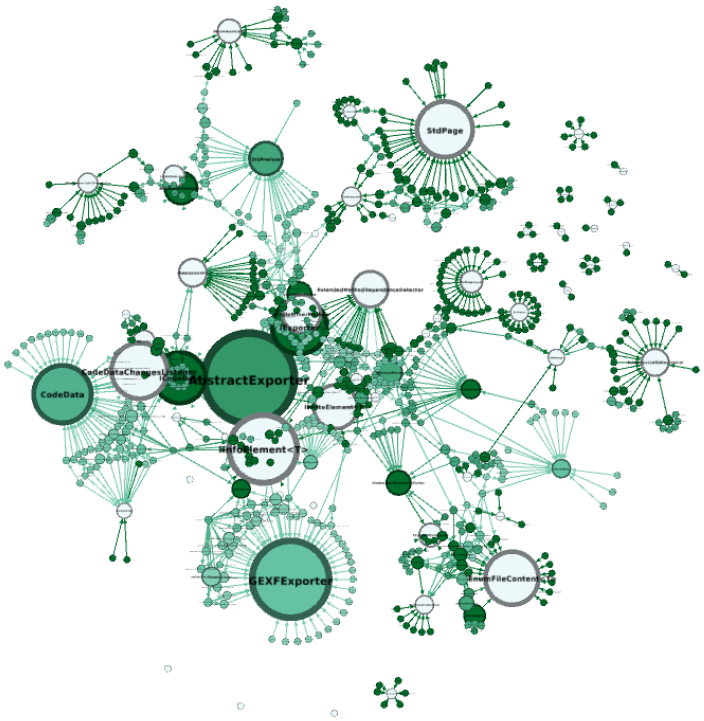


Figura 10: Nodos más oscuros representa mayor centralidad cercanía. Nodos más grande mayor centralidad de vector propio

tenibilidad y la escalabilidad del *software*, permitiendo cambios localizados sin afectar significativamente a los componentes principales.

En cambio, aquellos nodos con mayor valor de intermediación también son aquellos que mayor centralidad de vector propio tienen (fig 11). Esta correlación sugiere la existencia de un "núcleo arquitectónico" formado por clases como `AbstractExporter` y `CodeData`, que no solo controlan el flujo de información entre diferentes módulos del sistema, sino que también están integrados con otros componentes estructuralmente importantes. Esto podría ser una ventaja, por la mantenibilidad, y un riesgo crítico, al depender el sistema completo en estos componentes centrales.

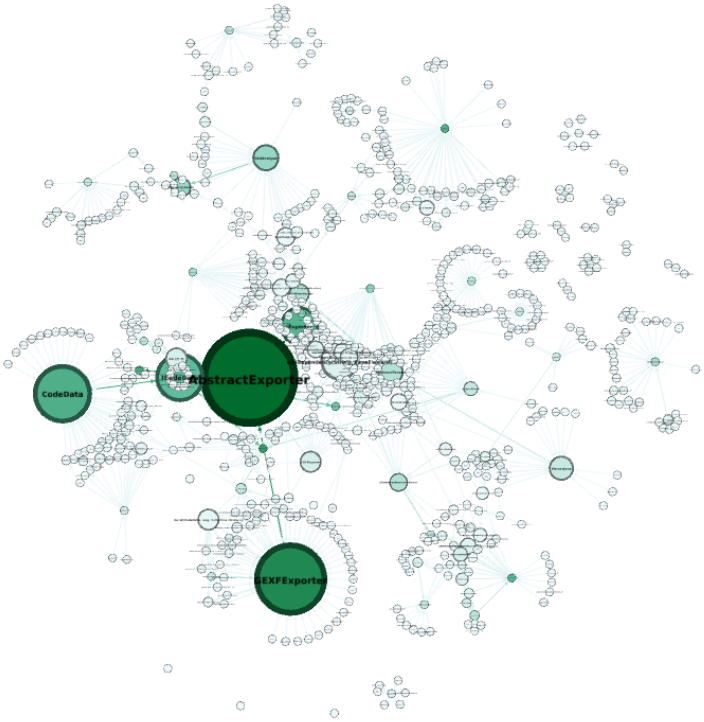


Figura 11: Nodos más oscuros representa mayor centralidad de vector propio. Nodos más grande mayor centralidad intermediación

4. Detección de comunidades

Para la detección de comunidades de han utilizado los algoritmos de *Lovaina* y *Statistical Inference*. Este último no tiene hyperparámetros. Mientras *Lovaina* optimiza la modularidad mediante un enfoque jerárquico ascendente, *Statistical Inference* utiliza un modelo probabilístico que puede capturar estructuras comunitarias distintas basadas en patrones estadísticos de conexión.

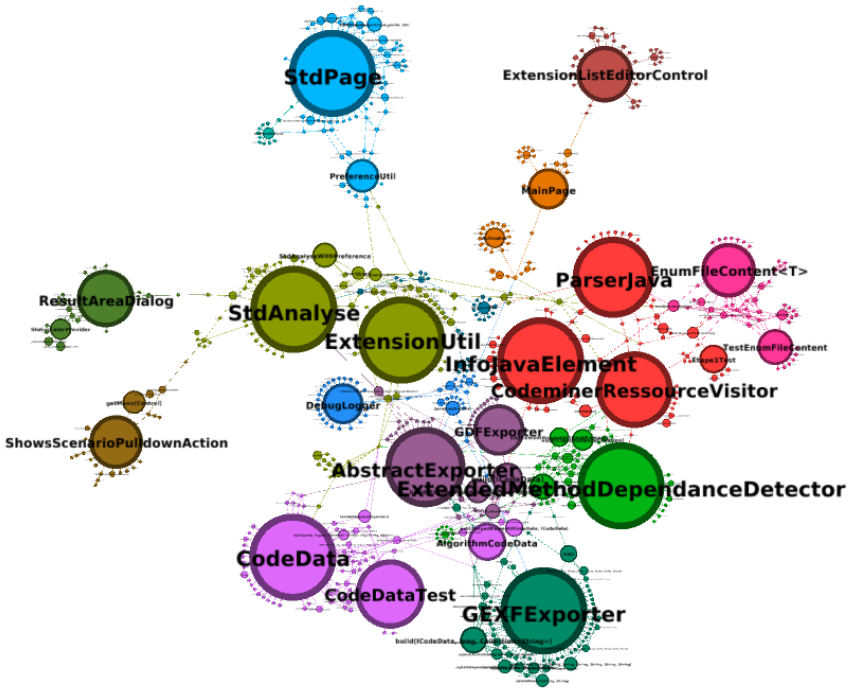


Figura 12: Comunidades por *Lovaina*

Los resultados obtenidos variando el valor de resolución son muy similares, entre $15 - 10$ resultados dan todas las ejecuciones. En *Lovaina*, el mejor valor de modularidad siempre era obtenido por valores cercanos a 1, donde la red llegaba a 15 comunidades. Sin embargo, ningún valor de los probados bajaba de 0,7 en modularidad, por lo que la red tiene unas comunidades claramente definidas. Además, el número de redes variaba entre rangos de valores muy cercanos y parecidos, alcanzando su máximo valor de modularidad en en valor de resolución 1, aumentando de uno en uno por mucho que se aumentara en el tamaño de esta variable, por lo que como mínimo el algoritmo no parece encontrar (sin forzar demasiado este parámetro) más de $11 - 10$ comunidades. En la figura 12 puede verse el grafo filtrado por comunidad usando los colores.

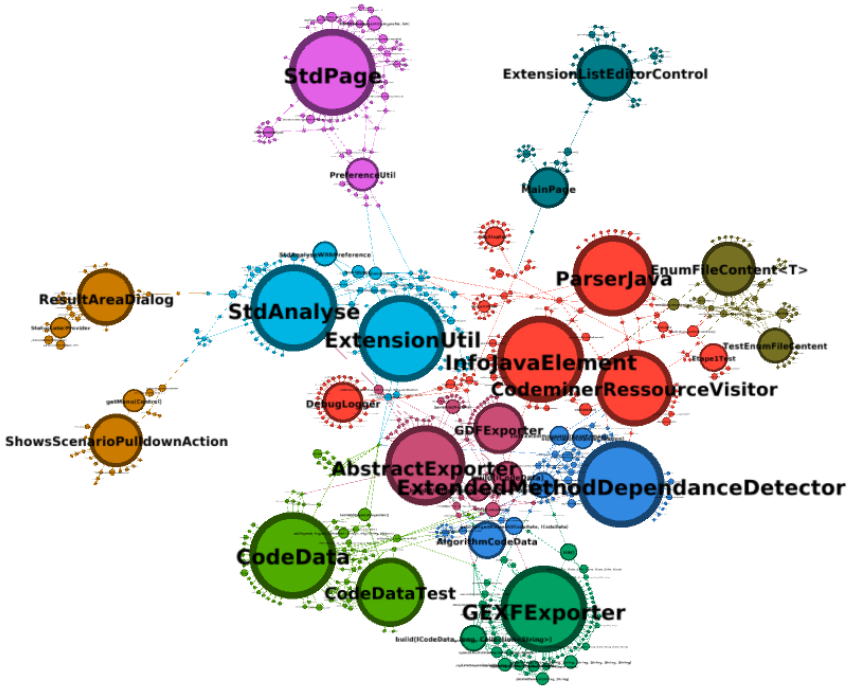
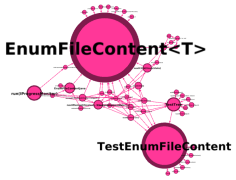


Figura 13: Comunidades por *Statistical Inference*

El algoritmo de *Statistical Inference* devuelve un total de 10 comunidades. En la figura 13 puede verse el grafo filtrado por comunidad usando los colores. Muchas de sus comunidades coinciden en la agrupación de varios nodos clave si se compara con los resultados obtenidos por *Lovaina*.

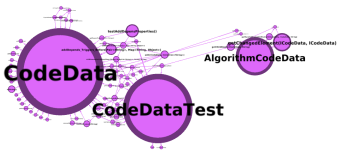
Se escoge el grafo con valor de resolución 1, valor de comunidad 0,85 y 15 *clusters* encontrados, ya que el valor de número de comunidades ha sido muy repetido y parece bastante robusto entre ejecuciones, además alcanza el mayor valor de modularidad de todos.

En las figuras 14 se observan algunas de las más cohesionadas semánticamente y más



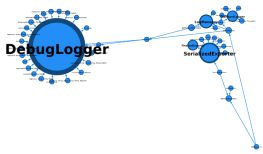
(a) Comunidad 1: Análisis de elementos Java (InfoJavaElement, ParserJava). Manejo de estructuras de código y relaciones entre elementos.

(b) Comunidad 2: Exportación de datos (GEXFExporter, AbstractExporter). Conversión de resultados a formatos gráficos.



(c) Comunidad 3: Interfaz de usuario (StdPage, ResultAreaDialog). Componentes para configuración y visualización.

(d) Comunidad 4: Gestión de extensiones (ExtensionUtil). Coordinación de módulos y plugins.



(e) Comunidad 12: Registro y logging (LogManager, DebugLogger). Monitorización del proceso de análisis.

Figura 14: Análisis de comunidades en la estructura del software

interesantes.

La Comunidad 1 está centrada en elementos de enumeración y contenido de archivos, con nodos clave como `InfoJavaElement` y `ParserJava`. Se encarga de la representación y manejo de contenido de archivos, creando estructuras internas para el procesamiento de la información.

La Comunidad 2 abarca la interfaz de usuario, con nodos centrales como `StdPage` y `ResultAreaDialog`. Esta comunidad agrupa componentes de la capa de presentación, incluyendo páginas, diálogos y elementos de visualización. Su alta centralidad de grado (especialmente `StdPage` con grado muy alto) indica que estos componentes se comunican con muchas otras partes del sistema.

La Comunidad 3 se enfoca en `CodeData`, principalmente *tests*. Esta comunidad maneja los componentes relacionados con la estructura de datos del código y, más significativamente, los casos de prueba que validan el funcionamiento de estas componentes.

La Comunidad 4 se dedica a la exportación de datos, con `GEXFExporter` y `AbstractExporter` como nodos principales. Esta comunidad implementa el patrón de diseño *Strategy* para la exportación de datos, con `AbstractExporter` como clase base y especializaciones como `GEXFExporter`. Los valores altos de centralidad de intermediación en estos nodos confirman su papel como puente entre el análisis y la representación externa de datos.

La Comunidad 12 implementa el registro y *logging*, con `LogManager` y `DebugLogger` como nodos centrales. Esta comunidad proporciona la infraestructura de *logging*, crucial para la depuración y el seguimiento de la ejecución. Su estructura relativamente aislada pero con conexiones a múltiples comunidades refleja su naturaleza transversal en el sistema.

El análisis de las conexiones entre comunidades revela patrones arquitectónicos importantes. Existen dependencias funcionales entre las comunidades 1 (Enumeración y contenido) y 4 (Exportación), donde el contenido procesado es posteriormente exportado en distintos formatos. La comunidad 3 (`CodeData`) mantiene conexiones con prácticamente todas las demás comunidades. Las comunidades 2 (Interfaz) y 4 (Exportación) muestran una relación donde los datos exportados son visualizados a través de la interfaz de usuario.

La comunidad 12 (*Logging*) funciona como servicio transversal, con conexiones hacia prácticamente todas las demás comunidades, indicando su naturaleza de servicio común utilizado por múltiples componentes.

La estructura modular identificada sugiere un diseño orientado a componentes con una arquitectura que separa claramente la representación de datos, la interfaz de usuario, las pruebas y la exportación. El bajo coeficiente de clustering general (0,079) indica un diseño donde los componentes tienen responsabilidades bien definidas y limitadas. La comunidad 12 (*Logging*) proporciona servicios utilizados transversalmente a lo largo de toda la arquitectura.