UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

TRABAJO FIN DE GRADO

Análisis de comunidades científicas basadas en fuentes de datos online

Miguel Ángel Jiménez Zarzuelo

JULIO 2014

Análisis de comunidades científicas basadas en fuentes de datos online

AUTOR: Miguel Ángel Jiménez Zarzuelo
TUTOR: David Camacho Fernández

Dpto. de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid

RESUMEN

La formación de comunidades es un tema apasionante que puede ser enfocado desde múltiples ámbitos (formación, agrupamiento, cohesión, …)
En este caso se van a detectar comunidades científicas, grupos de trabajo de investigadores. Esto permitirá entre otras cosas poder identificar autores importantes de una determinada materia, o por el contrario, fraude científico (autores con fama pero sin especial relevancia).

Para ello se ha desarrollado una herramienta que permite extraer esa información de una fuente de datos online (Google Scholar), interpretarla y generar un grafo para su posterior análisis. El grafo generado se puede visualizar con cualquier herramienta de visualización de grafos como Gephi. Una vez que se tienen los grafo, se han probado algunos algoritmos de detección de comunidades sobre ellos.

Además se ha realizado un breve estudio sobre la detección de cadenas de caracteres con ligeras diferencias entre ellas que refieren al mismo elemento (record linkage), ya que es un problema que aparece a la hora de analizar el trabajo de los autores en base a su firma de artículos.

Palabras clave:

Minería de datos, grafos, clustering, Google Scholar**,** gephi, chinese whispers, record linkage

ABSTRACT

The formation of communities is an amazing subject that can be approached in several fields (creation, development, clustering, nodes cohesion,...)

Scientific communities (working groups of researchers) will be detected in this project. This will permit, among other things, to identify distinguished authors in a certain subject or, on the contrary, detect scientific fraud (famous authors with limited relevance).

To do so, it has been developed a tool which permit to extract information from an online database (Google Scholar), structure it and generate a graph to analise it later. The generated graph can be visualised with any tool of graph visualisation, such as Gephi or Cytoscape. Some community detection algorithms, like Chinese Whispers, has been tested over those graphs.

Furthermore a brief research about string comparison and record linkages were carried out, since this is a problem that appears when it comes to analise the work of the authors based on their articles signatures.

Index Words

data mining, graphs, clustering, Google Scholar**, G**ephi, chinese whispers, record linkage

AGRADECIMIENTOS

Primero dar las gracias a David, por su paciencia, y por haber sacado tiempo de donde no tenía para ayudarme en esto.
También quiero agradecer a mi familia y amigos, por ese apoyo que siempre he tenido durante estos años.
Y por último, dar las gracias a Ángela, por ayudar a que estos meses me hayan sido mucho más llevaderos.

Glosario

Cadenas de Markov: proceso estocástico en el que la probabilidad de avanzar a otro estado solo depende del estado actual, y no de sucesos anteriores.

Captcha: Completely Automated Public Turing test to tell Computers and Humans Apart. Prueba para diferenciar humanos de máquinas. En particular un campo donde hay que interpretar una imagen y escribir el resultado.

Embeber: encerrar dentro de sí a otra cosa.

Grafo: Conjunto de objetos (nodos) unidos por ramas (aristas) que permiten establecer relaciones entre ellos.

HTML: HyperText Markup Language. Lenguaje de marcas usado en la elaboración de páginas web.

Layout: Esquema de distribución de los elementos dentro de un diseño.

Nodo: unidad fundamental de un grafo y que contiene una cantidad discreta de información.

Parsear: Analizar una secuencia de caracteres a fin de encontrar su patrón gramatical.

Rama: Relación entre los nodos de un grafo.

String: cadena de caracteres. Conjunto de 1 o varios tokens.

Token: cadena de caracteres que tiene un significado coherente. Palabra.

Índice

Índice de tablas

Índice de ilustraciones

1 Introducción

1.1 Motivación

La motivación de este proyecto es detectar y analizar las comunidades de autores científicos.

Para ello se desarrollará una API que pueda extraer información automatizadamente de alguna fuente online.

Posteriormente, se emplearán los datos extraídos para la detección de comunidades científicas, para ello se emplearán algoritmos y métodos de agrupamiento.

Las técnicas de clustering de documentos se ha aplicado intensamente durante los últimos años para organizar automáticamente un corpus de documentos en grupos o categorias similares. Estas técnicas se aplicarán sobre un conjunto de investigadores para detectar, por ejemplo, autores importantes dentro de una comunidad.

Además se analizarán el estado del arte de herramientas y sistemas utilizables, y se utilizará alguna herramienta de visualización y análisis de grafos.

1.2 Objetivos

El objetivo principal de este TFG es encontrar comunidades de autores científicos y observar su evolución temporal.

Para ello primero es necesario tener datos con los que trabajar, con lo que el primer corresponde con la parte de minería de datos, y es extraer datos del sitio de forma automatizada mediante algún programa, ya sea usando alguno existente o desarrollando uno nuevo.

El segundo objetivo es clasificar esos datos y representarlos en forma de estructura visible, como por ejemplo un grafo cuyos nodos sean los autores y las aristas sean las relaciones entre ellos. Para ello es necesario encontrar algún programa de visualización de grafos.

Por último, el análisis de los grafos, detectando comunidades en los mismos y encontrando los nodos fuertes del mismo mediante el uso de algunos algoritmos existentes.

Además es necesario encontrar un criterio de comparación de cadenas de caracteres que permita saber cuando dos cadenas parecidas refieren al mismo elemento y así evitar innecesariamente duplicar autores (nodos) en el grafo.

1.3 Organización de la memoria

En la sección 2 de esta memoria se describirá el estado actual de los elementos importantes que ha motivado este proyecto y que serán necesarios para entender los siguientes apartados. En la sección 3 se describe el diseño y arquitectura de la aplicación desarrollada para extraer información automatizada de Google Scholar y devolver un fichero con un grafo. En la sección 4 se detallan los resultados de las pruebas realizadas sobre los grafos obtenidos, con diversos algoritmos. Y de los algoritmos de comparación de cadenas. En la sección 5 se explican las conclusiones obtenidas de la realización de este trabajo, así como unos apuntes de como continuar en el futuro estas investigaciones.Por último está la sección de referencias, en la que se indican las fuentes de información utilizadas en la realización de este trabajo.

2 Estado del arte

Esta sección introduce de forma general las técnicas y elementos aplicados en el desarrollo del trabajo. Esto engloba tanto los algoritmos de clustering y de comparación de cadenas de caracteres usados, como un análisis de las posibles fuentes de datos y los programas existentes para obtener dichos datos o visualizar grafos, o los investigadores sobre los que se realizarán las pruebas.

2.1 Bases de datos online

Actualmente existen muchas bases de datos científicas online. Se entiende po base de datos científica aquella que almacena artículos de investigación, papers o citas, ya sea físicamente o simplemente los enlaces a los mismos. A continuación se explican algunas de las más importantes.

2.1.1 Google Scholar

Google Scholar [1] es una base de datos online que almacena enlaces a publicaciones científicas de cualquier disciplina y formato.

Esta base de datos es gestionada por la multinacional Google [2], usando algoritmos propios ya vistos en su buscador tanto para indexación como para búsquedas de artículos.

Pero Google Scholar también funciona como una red social, puesto que cada investigador se puede crear su perfil, aportando la información personal que él considere oportuna, como su foto, entidades académicas con las que trabaja, campos en los que investiga o su propia web personal. Pero lo más interesante es que él añada enlaces sus trabajos de investigación para darlos a conocer.

Cada perfil de usuario tiene un campo con enlaces a los coautores que han trabajado con él y tienen perfil en Scholar. Este campo se rellena automáticamente bajo los criterios de Google (con lo cual es información útil, pero no siempre es correcta. A veces aquí aparecen autores con los que el investigador en realidad no ha trabajado nunca. En otros muchos casos, directamente aquí no aparece nadie, está vacio).

A los artículos se puede acceder mediante el buscador (introduciendo una palabra clave), o mediante la página de perfil de uno de sus autores.

Los campos principales de un artículo de una página de perfil se pueden ver en ilustración 1. Son el título (hipervínculo al artículo), nombre de los coautores, año de publicación, medio de publicación y número de citas (en la tabla 1 se ve la información relativa a la ilustración 1).

Título	Coautores	Año	Medio	# de citas
Programmins robosoccer agents by modeling human behavior	R Aler, D Camacho, JM Valls, A Lopez	2009	Expert Systems with Applications 36 (2), 1850-1859	26

Tabla 1: Artículo Google Scholar

Ilustración 1: Artículo Google Scholar

La mayoría de investigadores tienen perfil en esta base de datos, con lo cual la cantidad de información que almacena es altísima.

2.1.2 DBLP

DBLP [14] (Digital Bibliography and Library Project) es una base de datos científica online más orientada hacia las ciencias de la computación. Está gestionada por la Universidad de Trier, en Alemania.

Esta base de datos la rellenan los administradores del sitio (en vez de los investigadores, como en Google Scholar), añadiendo ellos los links a los artículos y referenciándolos.

Las búsquedas solo se pueden realizar a través del nombre del autor (aunque ultimamente han abierto las búsquedas por conferencias), listándose todos sus artículos encontrados en la base de datos, ordenados por fecha.

Cada artículo tiene los siguientes datos: año de publicación, enlace visual a la url donde el artículo en cuestión está alojado, autores que han trabajado en el artículo, título del mismo y medio donde fue publicado (ilustración 2 con sus datos en tabla 2).

Título	Coautores	Año	Medio
Decomposition of speech into voiced and unvoiced components based os state space signal model	Mark Thomson, Simon Boland, Julen Eppi, Michel Smithers	2003	ICASSP (I) 2003 160-163

Tabla 2: Artículo DBLP

Ilustración 2: Artículo DBLP

2.1.3 Cite SeerX

Cite SeerX [15] (anteriormente conocido como Cite Seer) es una base de datos centrada principalmente en artículos del campo de las matemáticas, estadística y ciencias de la computación, aunque poco a poco se va volviendo multidisciplinar.

Está gestionada por la Universidad Estatal de Pennsilvania. Con Isaac Councill y Lee Gilles como cabezas visibles del proyecto.

Además de las búsquedas comunes por autor o título del documento, esta base de datos destaca porque su buscador incorpora un motor para realizar búsquedas de citas que estén contenidas en los artículos (de ahí el nombre). Otra característica poco frecuente es que los resultados de las búsquedas se pueden ordenar por año o por relevancia.

Cite SeerX no indexa artículos por autor. Al realizar una búsqueda por autor, devuelve todos los artículos que contengan esa palabra, esté en el campo del autor o en el texto del documento.

En este caso, la información de los artículos encontrados es escueta [E]. Solamente tiene el link al artículo en cuestión, los nombres de los autores y el año de publicación (ilustración 3, con datos en tabla 3).

Título	Coautores	Año
Of Labor Sample Attrition in the Canadian Survey of Labor and Income Dynamics	Brahim Boubardat, Lee Grenon	2013

Tabla 3: Artículo CiteSeerX

Ilustración 3: Artículo CiteSeerX

2.1.4 Mendeley

Mendeley es una base de datos distinta a las anteriores. Requiere registro y posterior autenticación para poder acceder y realizar búsquedas. El motivo de tener que registrarse es que permite crear una biblioteca digital donde almacenar los artículos que uno quiera, además de poder contribuir con aportes propios.

Es multidisciplinar, en ella se encuentran artículos de cualquier rama.

Además, tiene un cliente de escritorio desde el que mucho más sencillo e intuitivo trabajar.

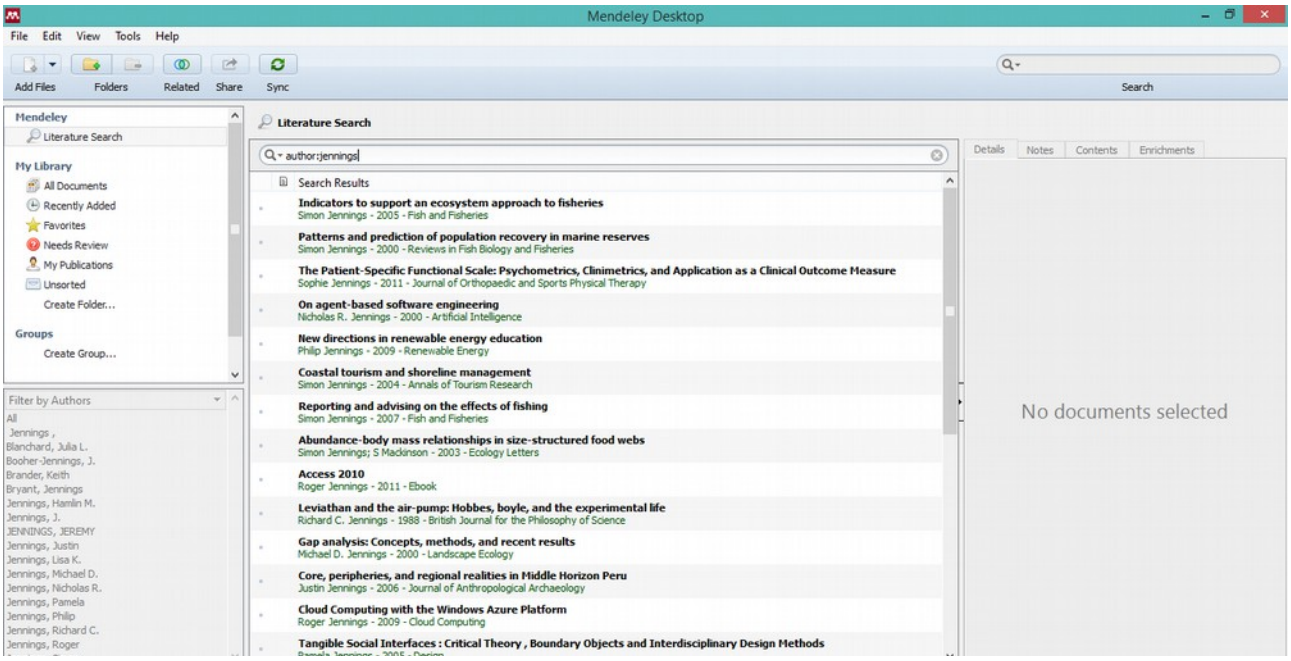

Ilustración 4: Búsqueda Mendeley Desktop

Permite realizar fácilmente búsqueda por título, autores, palabras clave o año y devuelve los resultados de una forma mucho más estructurada que su cliente web.

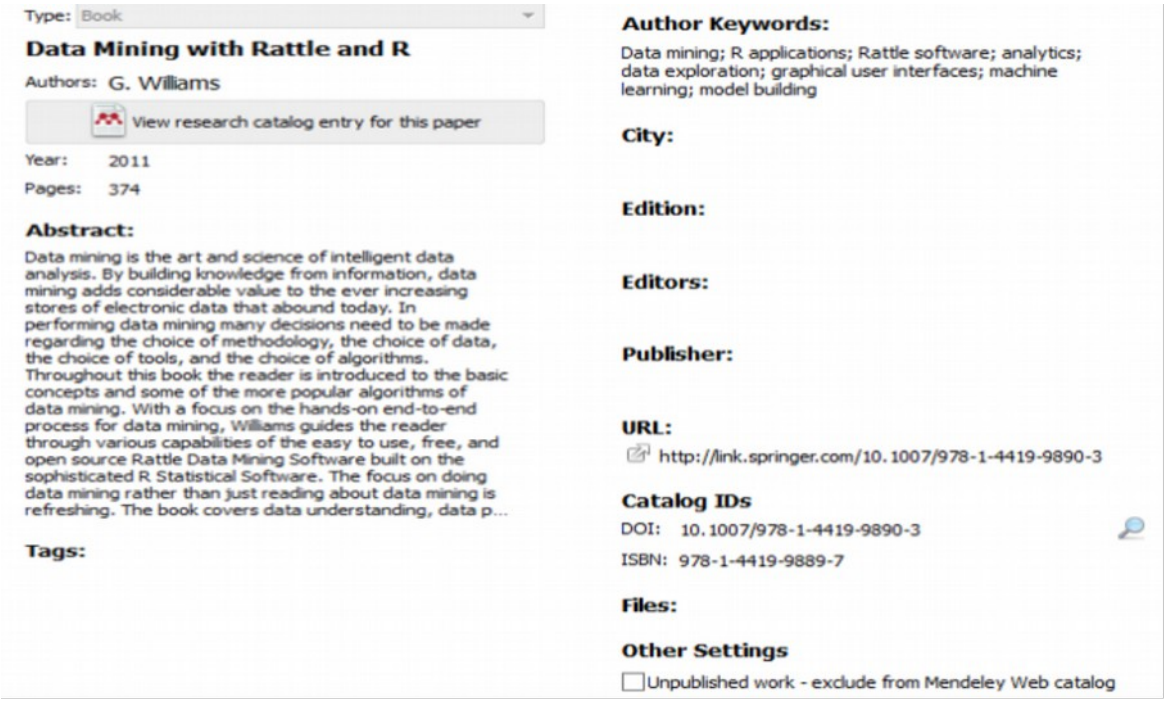

Ilustración 5: Ejemplo Mendeley Desktop Client

Esta base de datos, además, tiene funcionalidad de *red social*, se pueda dar *follow* a otros investigadores para estar al tanto de sus últimas publicaciones (esta funcionalidad actualmente solo está disponible desde su cliente web).

Pero como para hacer búsquedas es necesario estar autenticado, esto dificulta mucho realizar búsquedas automatizadas por medio de un *robot*.

2.2 Extracción de información

Se ha decidido usar como fuente de información Google Scholar por tener los artículos claramente ordenados por autor, por ser su base de datos muy extensa y por la simpleza de su código html.

A mayo de 2014, Google no ha desarrollado ninguna API oficial abierta para analizar Google Scholar.

En la red se encuentran algunos programas desarrollados por *aficionados* en JAVA, Python o otros lenguajes que parsean la página oficial y extraen diversos campos de información.

Como suele ser normal, una parte importante de los programas existentes no funcionan como deberían, ya

sea por estar obsoletos o por no estar suficientemente bien elaborados.

Teniendo en cuenta solo los programas que funcionan correctamente, básicamente estos se dividen en dos tipos, los que realizan las búsquedas (por autor o palabras clave) sobre el buscador del sitio [1], o los que extraen información sobre un perfil de usuario [3].

2.2.1 Por buscador

Muchos programas extraen la información directamente desde el buscador, y todos tienen un comportamiento similar (almacenar el HTML, extraer los campos que consideran útiles y devolverlos por pantalla). El mejor de los existentes es el siguiente (es el que más datos extae y el más sencillo de comprender):

Scholar.py [4] está en desarrollo por un investigador de la Universidad de Berkeley (Christian Keibrich [19]), aunque invita a la gente a colaborar. Está desarrollado en Python y realiza la búsqueda del autor introducido por parámetro sobre el propio buscador del sitio. Obtiene el HTML del sitio, ayudándose de la librería BeautifulSoup 4.0, y extrae toda la información posible de cada artículo que aparece (título, enlace al artículo, año de publicación o número de citaciones) excepto el nombre de los coautores.

El nombre de los coautores no lo extrae porque en el HTML no aparece de forma explícita en un campo. En el campo que aparecen los coautores, hay más información, y no sigue un formato estándar entre todos los artículos, lo que imposibilita un parseo automático.

2.2.2 Por página de perfil

La otra forma de obtener la información con Google Scholar es a través de los perfiles de usuario. Realizando la búsqueda así, además de información sobre los artículos, se puede obtener información sobre cada autor. Un ejemplo de programa que realiza búsquedas así es el siguiente:

Scholar.java [5] está desarrollado sobre JAVA, y es de autoría anónima. Este programa realiza la búsqueda sobre un perfil de usuario, pidiendo el HTML al servidor y extrayendo información de él. Los campos que extrae son sus datos personales y los autores con los que ha colaborado.

Pero no se dedica a explorar las publicaciones del autor.

2.3 Visualización de grafos

Para poder analizar una red social es necesario representarla en forma de grafo. Existen varias aplicaciones gratuitas que se pueden conseguir fácilmente en internet que se dedican a ello.

Pero para poder visualizar el grafo, primero hay que tenerlo representado en un formato que los programas entiendan. Uno de los formatos más usados es **graphml** [8].

Graphml tiene una semántica muy simple, basadas en etiquetas HTML, pero ofrece muchas posibilidades, pudiendo determinar no ya solo los nodos, las aristas o los pesos de las mismas, si no se pueden seleccionar opciones predeterminadas de representación. Hay que tener en cuenta que antes de declarar una arista debe estar declarado el nodo origen y el destino.

Para poder visualizar y entender un grafo es necesario establecer un criterio para colocar los nodos en la pantalla. Para eso existen los layouts.

Como lo que se busca en este trabajo es detectar comunidades en grafos, la mejor forma de representarlos es usando un layout *force directed*. Este layout se caracteriza porque intenta representar el grafo de forma que se cruzen el menor número de aristas posibles. Para conseguir se basa en un idea similar a los movimientos de partículas debido las cargas magnéticas. Todos los nodos se repelen entre sí, pero los que tienen aristas en común se atraen. Esto provoca que en el resultado final los nodos aparezcan agrupados en comunidades en la mayoría de los posible. Este layout no produce un único resultado de salida, aunque todos suelen ser muy similares. El resultado final depende del orden de exploración de los nodos. Puede tener en cuenta o no el valor de las aristas para hacer los cálculos.

Entre las herramientas de visualización, hay dos que destacan por encima del resto: Gephi [7] y Cytoscape [6]

2.3.1 Cytoscape

Cytoscape es un software de código abierto orientado especialmente hacia la rama de la biología. Se usa para visualizar redes de moléculas (ADN y proteínas sobre todo), y para probar algoritmos genéticos. Tiene el problema de que resulta bastante complicado y poco intuitivo la personalización visual de los grafos, por lo que se usa sobre todo para análisis, que en ese campo sí que es muy potente.

Con este programa se ha usado un layout forcedirected estándar, que viene de serie con la herramienta.

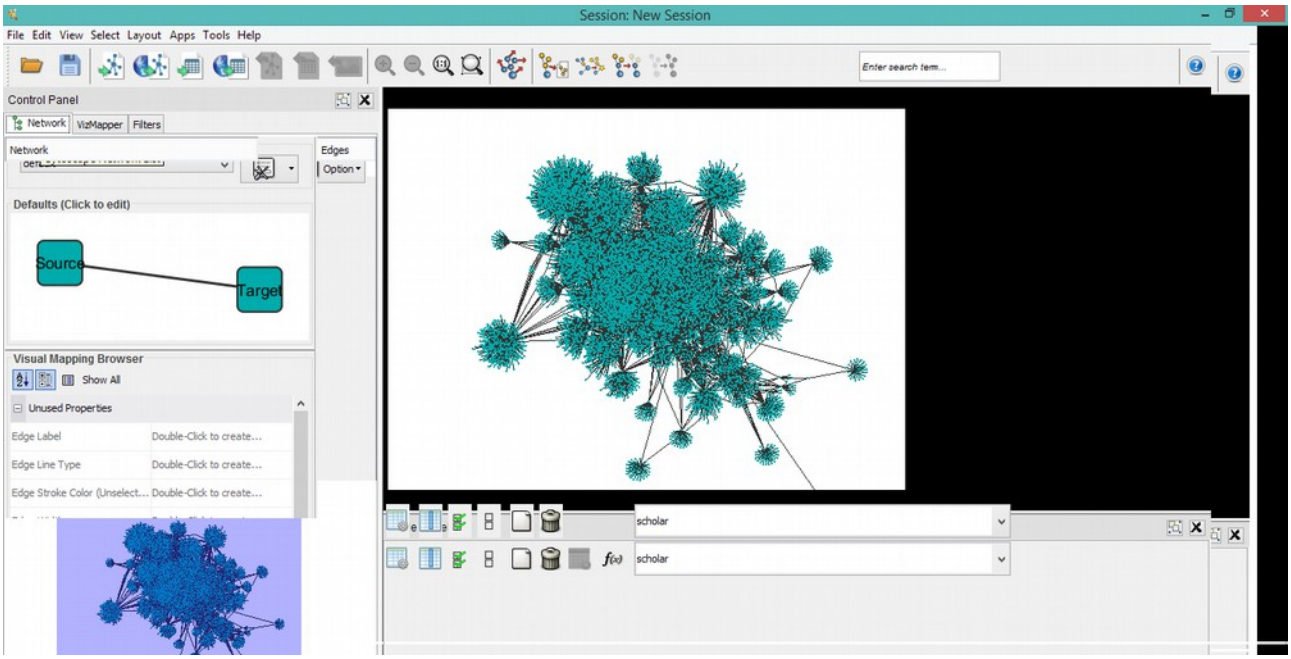

Ilustración 6: Ejemplo de Cytoscape

2.3.2 Gephi

Gephi es un software de código abierto destinado a la visualización y análisis de grafos. Pero no está pensado en inguna rama en especial. Tiene incorporadas varias métricas de grafos (grado medio, page rank, densidad,...), y una potente herramienta de visualización con la que se puede personalizar el grafo a gusto del usuario fácilmente (colores, tamaños, etiquetas...). Además, incorpora un *market*, del que uno puede bajarse plugins creados por aficionados que añaden nuevas funcionalidades a la herramienta.

Para visualizar los grafos, viene implementado un algoritmo llamado *Force Atlas 2* [24]. Este algoritmo es de tipo force directed, pero con la peculiaridad de que no tiene condición de parada. El algoritmo sigue analizando cada nodo y colocándolo donde crea oportuno hasta que el usuario pare su ejecución manualmente. Esto tiene la ventaja de que así el usuario es quien decide el momento en que el grafo presenta un mejor aspecto visual, y no está sujeto a una condición de parada que puede dejar grafos ininteligibles. Cuando tiempo esté ejecutándose el algoritmo, más se van expandiendo y alejando las comunidades de nodos unas de otras.

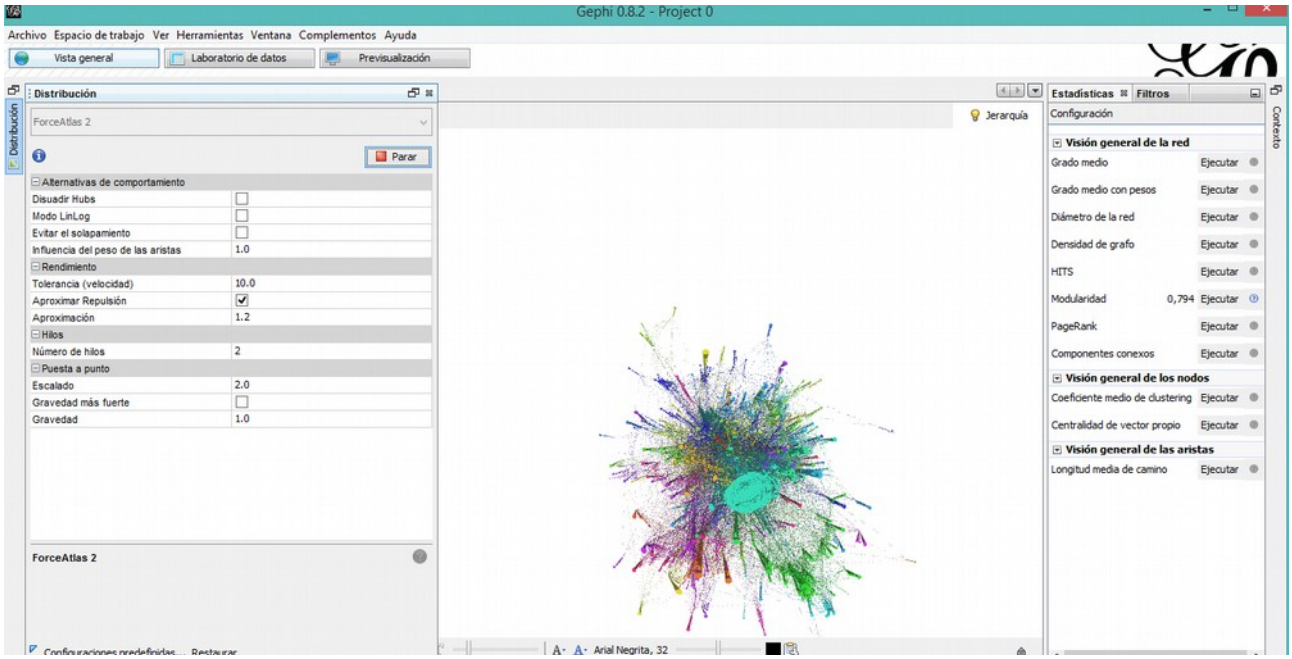

Ilustración 7: Ejemplo de Gephi

2.4 Análisis de datos

En este proyecto se pretende detectar comunidades científicas. Se entiende por una comunidad científica a un grupo de investigadores que investigan juntos en un determinado campo.

La información se representa en forma de grafo. La detección de comunidades en teoría de grafos se entiende como un conjunto de nodos fuertemente conectados entre sí, y con pocas aristas hacia el exterior de la comunidad. Lo que se entiende por “fuertemente conectados” y “pocas aristas”, son las variables que determinan el número de comunidades que aparecen en un determinado grafo. Este número puede ser solo 1 (todo el grafo es una única comunidad) si todos los nodos están homogéneamente conectados entre sí.

Algunos de los algoritmos más usados para detectar comunidades en grafos son:

- **Mínimo corte** [27]: este algoritmo no busca el número de comunidades de un grafo, se introduce el número de comunidades que se quiere encontrar, y el algoritmo parte el grafo en esas partes y redistribuye los nodos hasta encontrar la mejor manera de agrupar todos los nodos en ese número de comunidades.

- **Markov Clustering Algorithm (MCL)** [28] **:** Este algoritmo basa su idea en el funcionamiento de las cadenas de Markov. Estando en un determinado analiza las probabilidades de que viajando por una arista al azar (el peso de la arista determina la probabilidad de viajar a través de ella), se continúe en la misma comunidad o por el contrario se salga de ella. En base a estos resultados agrupa los nodos en comunidades. Es un algoritmo iterativo que converge siempre. Este algoritmo es muy lento para grafos grandes (decenas de

miles de nodos).

- **Chinese Whispers** [26]: Este algoritmo está basada en la idea del MCL, pero es una implementación mucho más rápida. Solo es válido para grafos no dirigidos.

Al empezar, todos los nodos pertenecen a grupos separados, y en cada iteración, el el grupo de un nodo se asigna por mayoria de nodos adyacentes. En caso de empate, se asigna uno al azar. El número de clusters converge al llegar a un determinado número de iteraciones, pero la distribución de los mismos puede que no. Hay casos en que un nodo cambia de grupo con cada iteración. Por ejemplo, si hay el mismo número de nodos adyacentes de dos grupos distintos y estos no varían.

- **Girvan Newman** [27]: este algoritmo se basa en buscar las aristas que unen comunidades, estas aristas se supone que son las que tienen tráfico (muchos caminos mínimos entre nodos pasan por ellas). Identificando y eliminando estas aristas, se tiene una visión clara de los nodos agrupados por comunidades. Este algoritmo puede servir para agrupar todos los nodos en un número fijo de comunidades, como para detectar el número de comunidades existentes en el grafo, indicando los valores límite de tráfico de las aristas. El número de iteraciones viene determinado por la anchura del grafo (camino mínimo entre nodos más largo).

- **Fast Unfolding** [25]: este algoritmo se basa en el concepto de modularidad. La modularidad es la fuerza de cohesión entre i,j. Este valor se puede representar numéricamente

$$Q = \frac{1}{2m}\sum_{i,j}\left[A_{ij} - \frac{k_i k_j}{2m}\right]\delta(c_i, c_j),$$

Ilustración 8: Fórmula cohesión

Donde Aij es el peso de la arista entre i ,j. Ki es el sumatorio de los pesos de las ramas que salen de i. ci es la comunidad a la que i está asignado. La función delta es 1 si ci y cj son iguales, y 0 en otro caso. M es $\frac{1}{2}\sum_{ij} A_{ij}$

Este algoritmo tiene dos fases, en la primera todos los nodos empiezan perteneciendo a comunidades distintas. Cada nodo i se prueba con cada nodo adyacente j introduciéndolo en su comunidad, si la fórmula da valor positivo, se asigna a la comunidad que de un valor mayor. Después de varias iteraciones los valores convergen.

La segunda fase consiste en crear un nuevo grafo en el que cada nodo es una comunidad encontrada anteriormente, y aplicar de nuevo el paso anterior para intentar *unir* comunidades. Nuevamente, los valores convergen tras varias iteraciones. Este algoritmo es muy rápido computacionalmente, y es muy eficiente para grafos grandes (decenas de miles de nodos).

2.5 Grado de similitud entre cadenas de caracteres

Uno de los problemas que existen actualmente a nivel de computación es como detectar la similitud entre dos cadenas de caracteres. En este caso, lo que hay que detectar es cuando dos cadenas de caracteres se corresponden con el mismo autor, puesto que a veces el mismo autor puede firmar con su nombre completo, solo sus iniciales, con uno o los dos apellidos...

Hay multitud de algoritmos que intentan resolver este problema. Existe una librería para JAVA que contiene muchos de estos algoritmos. Se llama **Simmetrics** [9].

Existen varios algoritmos implementados en esa librería que pueden ayudar a resolver el problema:

-**Similitudes fonéticas (Soundex) [16]**: se basa en determinar la similitud entre dos cadenas en base a la pronunciación de la misma. Agrupa los caracteres en conjuntos en base a su fonética similar, sustituye cada carácter por el valor del conjunto, y compara el resultado de ambas cadenas.

-**Distancia de Levenshtein** [11] es un algoritmo que se basa en contar el número mínimo de operaciones necesarias para cambiar una cadena de caracteres en otra. Esto tiene en cuenta el orden de los caracteres que coinciden.

-**Monge-Elkan** [12]. Este algoritmo solo sirve para comparar strings que contengan varios tokens. Se basa en aplicar una función de similitud a cada token, y luego calcular la media entre todos.

Una de las peculiaridades de este algoritmo es que si los strings no tienen el mismo número de tokens, el valor de similitud entre ambas varía mucho según la que se coja como base.

-**Jaro-Wilken** [10]. Este algoritmo se basa en calcular los caracteres comunes que tienen los strings, calcular las permutaciones entre caracteres que existen, y hallar un valor promedio.

- NeedlemanWunsch [17]: Este algoritmo se usa principalmente en bioinformática para encontrar alineamientos (secuencias de caracteres con una similitud muy alta, contenidas en otra más grande) entre

cadenas de proteínas. Es un algoritmo iterativo que en cada pasada compara los caracteres que se encuentran en la misma posición en ambas cadenas. En cada nueva pasada realiza un desplazamiento, y devuelve el valor mayor de similitud encontrado entre todas las pasadas.

-Smith Waterman [12]: Este algoritmo está pensado inicialmente para comparar secuencias de ADN. Se basa en buscar caracteres seguidos idénticos entre ambas cadenas para determinar el valor de la similitud. Es más conocido como *Gotoh.*

- Jaccard [18]: Este algoritmo en realidad es una métrica de similitud de conjuntos, su valor es simplemente una división entre el número de elementos coincidentes en ambos conjuntos (interesección), entre el número de elementos totales de ambos conjuntos (unión).

No tiene en cuenta ni orden, ni posición de los caracteres en la cadena.

3 Diseño y arquitectura

El diseño de la aplicación para analizar las comunidades de autores científicos (en adelante ParserScholar.Java) está representado en el siguiente diagrama (Ilustración 9) para mayor claridad.

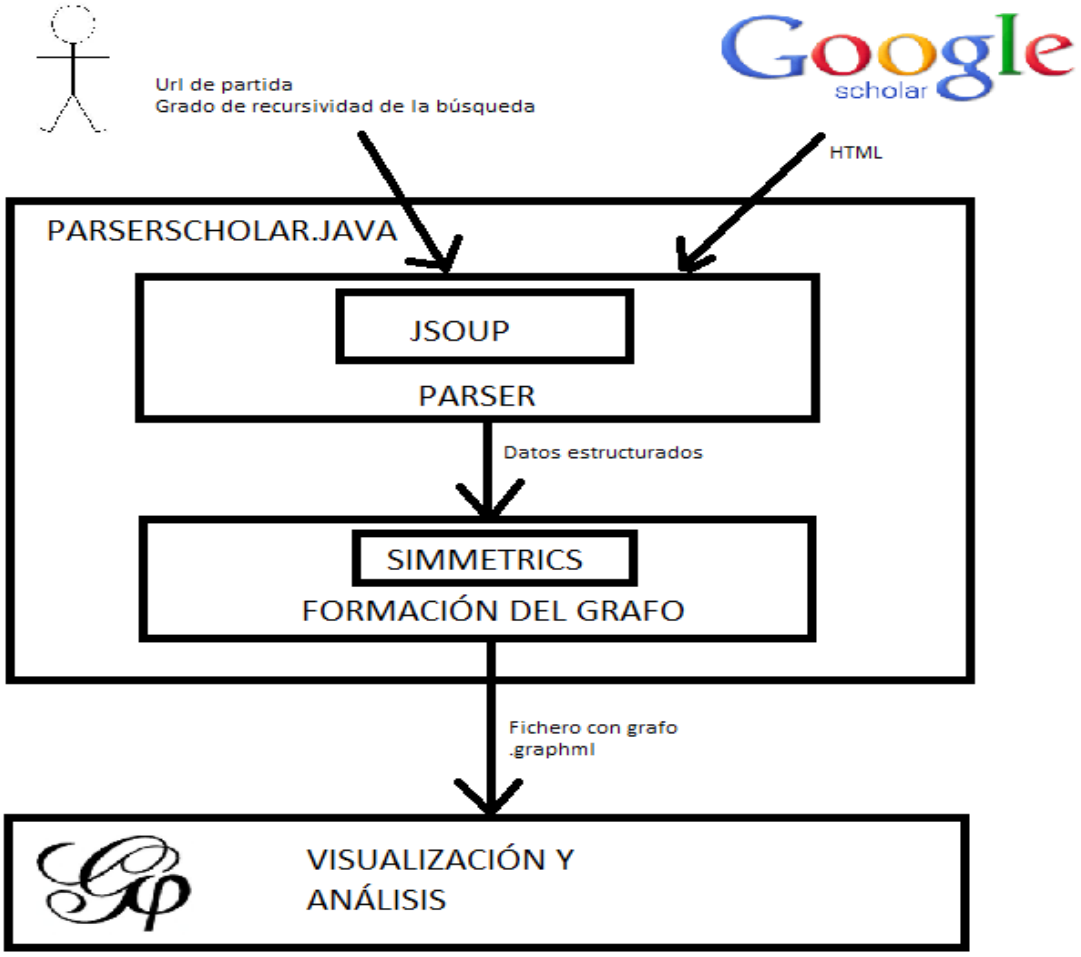

Ilustración 9: Diseño de la API

En él se aprecia como la API desarrollada se compone de dos módulos. Uno (parser) extrae la información de Google Scholar y devuelve la información estructurada para poder trabajar con ella, mientras que la otra (formación del grafo), recibe los datos obtenidos por el parser, y los interpreta para generar un fichero con el grafo resultante. Estos dos módulos están desarrollados en JAVA. La justificación de dicha elección es porque este lenguaje tiene librerias como JSOUP o SIMMETRICS que ayudan con el análisis de los datos.

Este fichero con el grafo, se introduce en alguna herramienta de visualización de grafos (en este caso GEPHI), y se aplican sobre él diversos algoritmos para detectar las comunidades en el mismo.

3.1 API

En el anexo 6.2 hay un manual para ejecutar la API. A continuación se detalla su diseño y arquitectura.

3.1.1 Datos de entrada

Fuente de datos

Para poder analizar una comunidad científica de autores, primero es necesario determinar la fuente de información. Y esta ha de ser online y estar bien estructurada, para poder procesarla automáticamente.

Además, la base de datos usada como fuente debe ser suficientemente amplia y precisa (fiel a la realidad), para que los resultados obtenidos a partir de ella sean correctos.

La mejor forma de representar una comunidad de autores para su posterior análisis es en forma de grafo. Cogiendo como nodos los autores, y como aristas algún tipo de relación entre ellos (como por ejemplo, unir los que alguna vez han colaborado juntos en alguna publicación científica).

Para ello es necesario poder acceder directamente desde un autor, a los datos de otros autores (búsquedas por nombre, link a la página de perfil, etc) y así poder rellenar el grafo mediante algún proceso automatizado.

De entre todas las bases de datos online analizadas, se ha decidido que la mejor para trabajar es Google Scholar [1], es muy amplia, intuitiva, aporta bastante información, y lo más importante, cada usuario tiene una página de perfil con sus investigaciones y links a los perfiles de sus coautores para poder automatizar la búsqueda. Además, por supuesto, de que es Google quien está detrás de la base de datos, eso es sinónimo de calidad en todos los aspectos.

Aunque a los artículos también se puede acceder a través del buscador, como en este caso lo que más interesa son los nombres de los autores, como base de la extracción de información se van a coger las páginas de perfil de los mismos, que tienen mucha información.

La información útil que tiene una página de perfil se puede ver en la siguiente imagen (Ilustración 10).

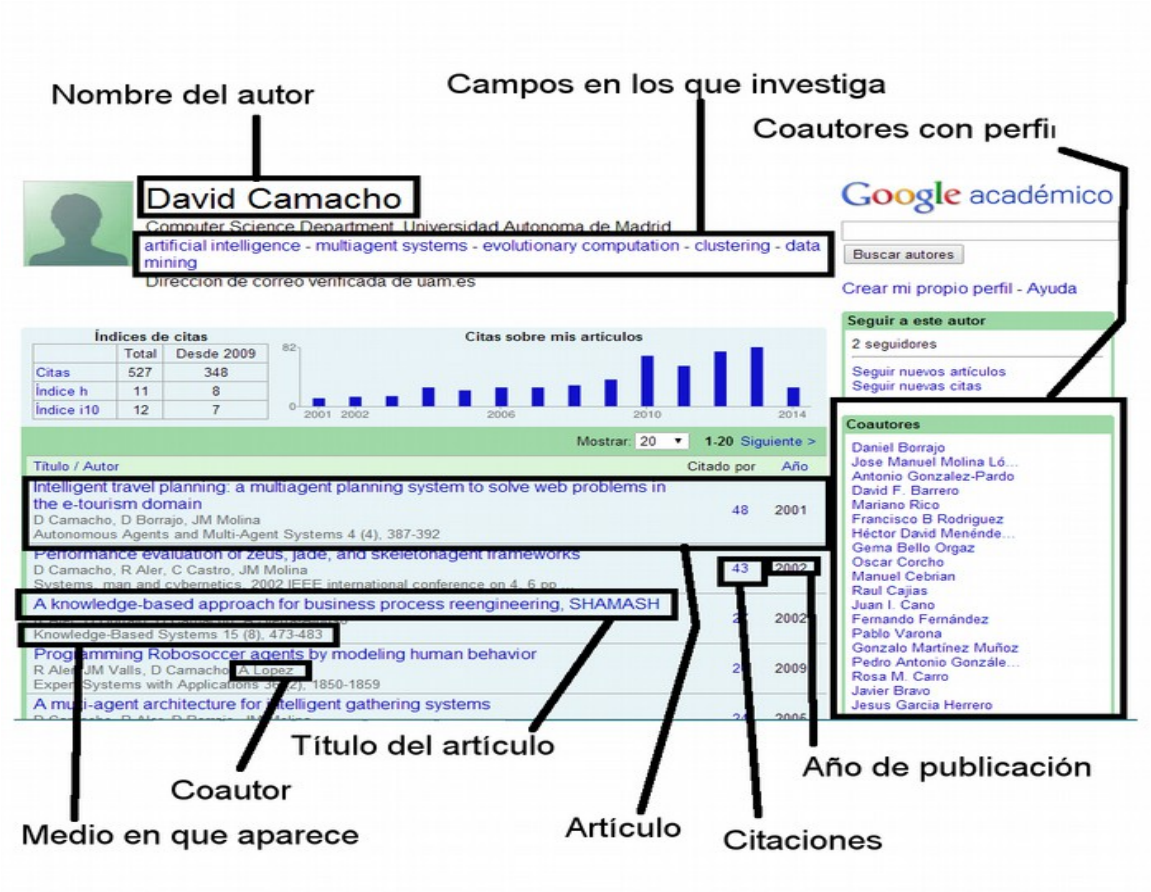

Ilustración 10: Esquema datos Google Scholar

Pero como es obvio, Google Scholar, a pesar de parecer la más adecuada para realizar los análisis que nos interesan, tiene algunos inconvenientes, por ejemplo:

- La información no es del todo completa, solo se tiene la de los autores que tienen perfil, los que no tienen perfil solo aparecen espontáneamente como autores de obras de estos. Puede haber publicaciones que solo estén realizadas por autores sin perfil, con lo que no parecerían reflejadas en el análisis.

- La lista de coautores de la derecha de la página de perfil se genera automáticamente, con lo que a veces tiene fallos, como algunos autores que aparecen ahí con los que el autor principal no ha colaborado nunca.

Parámetros de entrada

El programa necesita que el usuario le indique los siguientes datos:

- URL del investigador sobre el que empezar la búsqueda.

- Nivel de profundidad del recorrido en anchura con el que extraer datos para formar el grafo.

3.1.2 Procesamiento

La API está compuesto de dos módulos diferenciados. Uno de ellos extrae la información de Google Scholar (Parser), y la devuelve de forma estructurada. El otro recibe los datos del parser, y los interpreta para formar el grafo.

3.1.2.1 Parser

Datos de entrada

Los datos de entrada del parser son los que entran directamente a la API (sección 3.1.1)

Procesamiento

Fase previa

Una vez que se ha decidido que la fuente de los datos va a ser Google Scholar, es necesario realizar un estudio de herramientas o APIs ya desarrolladas que puedan servir..

Después de valorar todos los programas que extraen información de Google Scholar encontrados. El que mejor aspecto presenta (entre otras características, porque funciona correctamente y tiene los comentarios adecuados para poder comprender el código sin ser un experto en Python) es **scholar.py** [4].

Pero tiene el problema de que no extrae los coautores de los artículos, con lo que no hay nada sobre lo que realizar una nueva búsqueda para obtener nuevos autores y explorar y rellenar el grafo automáticamente.

Además, este programa obtiene información sobre el buscador del sitio, con lo que los resultados obtenidos no tienen por qué corresponder con las publicaciones del autor que se tiene en mente (hay autores con el mismo nombre), por lo que era necesario hacer modificaciones para adaptarlo a lo que se necesita.

Se intentaron realizar las modificaciones necesarias (cambiar la URL de base, y cambiar las estructuras para recoger exactamente los campos que se necesitan), embebiendo el código sobre JAVA, un lenguaje sobre el que es más fácil trabajar. Pero después de varios intentos sin obtener el resultado adecuado, se abortó la idea.

Como segunda opción, se intentó trabajar sobre algún programa ya hecho sobre JAVA que se centrara en la extracción de información directamente sobre el perfil del investigador para así evitar el problema explicado anteriormente. Se analizó [5], y se vio que los campos que extraía no eran suficientes, ni siquiera relevantes para construir un grafo.

Se decidió que lo mejor era empezar de cero.

Codificación

El programa está desarrollado en JAVA. Una de las principales razones de esta decisión (aparte de la familiaridad que tiene el autor con él) es que este lenguaje tiene librerías que hacen muy sencillo la extracción del html, y la búsqueda de determinados campos en el mismo.

En concreto, la librería para explorar un HTML es Jsoup [13].

JSOUP[13] es una librería externa de JAVA que permite analizar fácilmente textos HTML (extraídos de páginas web). Tiene métodos que interpretan el texto, y devuelve los campos que precisamos.

Esta librería está integrada en el parser para facilitar dicha tarea.

Lo primero es definir la URL de la que se quiere conseguir el HTML.

Todas las URLS de las páginas de perfil siguen la misma sintaxis, y solo cambia en el identificador único del autor. La forma más sencilla de averiguar este valor es sacarlo de la propia URL, es el parámetro *user*.

Por ejemplo, si la página de un autor está en la dirección

http://scholar.google.es/citations?hl=es&user=fpf6EDAAAAAJ

su id es fpf6EDAAAAAJ

Por defecto, se muestran los 20 primeros artículos de cada autor, pero para ir más rápido en la extracción de información (y tener que hacer menos consultas contra el servidor de Google Scholar), es preferible buscar los artículos de 100 en 100. Para ello simplemente hay que añadir al final de la url: &view_op=list_works&pagesize=100.

Con lo que la URL inicial para realizar la primera búsqueda sería así, donde id es el identificador único del autor.

"http://scholar.google.es/citations?hl=es&user=" + id + "&view_op=list_works&pagesize=100";

Si el artículo que se quiere buscar no está entre los 100 primeros, simplemente hay que añadir al final de la url: &view_op=list_works&cstart=100. Donde el número *cstart* (en este caso 100), es el número del primer artículo que muestra la página. Si se quiere empezar por un número de artículo mayor de los que tiene el autor, el servidor atenderá la petición correctamente, pero la tabla de los artículos estará vacía.

Una vez que se conoce la url de partida, hay que extraer el HTML de la página haciendo peticiones al servidor de Google Scholar. Eso se puede conseguir mediante un código similar al siguiente.

```
URL url = new URL(URLbase);
InputStream is = (InputStream) url.getContent();
BufferedReader br = new BufferedReader(new InputStreamReader(is));
String line = null;
StringBuffer sb = new StringBuffer();

while ((line = br.readLine()) != null) {
   sb.append(line);
}
String htmlContent = sb.toString();
```

Primero se transforma el string que contiene la url en un objeto JAVA del tipo URL. Este objeto permite mediante una llamada a *getContent* hacer una petición al servidor que devuelve el HTML. Posteriormente se lee todo el código mediante un bucle *while* y se almacena en un buffer, para, por último convertir el buffer en una cadena de caracteres (String) y poder trabajar cómodamente con ello.

Es importante saber que el HTML devuelto no está en el formato estándar UTF-8, por lo que los caracteres acentuados se corrompen y se pierden.

Una vez que se tiene el HTML de la página, es el momento de parsearlo para buscar y extraer de él la información que se necesita. Aquí es donde entra en juego la librería Jsoup, encargada de interpretar un HTML y organizar internamente sus atributos y devolver la información que se precisa.

Para ello hay que convertir el String con el HTML en un objeto Document de Jsoup:

```
org.jsoup.nodes.Document doc = Jsoup.parse(htmlContent);
```

HTML es un lenguaje basado en etiquetas, la mejor forma de parsearlo es identificar dentro de que etiqueta está el campo que queremos extraer. Una vez que se sabe, se extraen del html todos los elementos que estén dentro de esa etiqueta mediante select, convirtiéndolo en un tipo Elements de Jsoup.

Por ejemplo, si queremos recorrer una tabla (td), la seleccionamos así:

```
Elements fichas = doc.select("td");
```

Y ahora se recorre cada elemento (Element) de la tabla mediante un bucle for.

```
for (Element a : fichas) {
```

La mejor forma de extraer un campo concreto de un Element es mediante su id. En este caso, el título de cada artículo lleva asociado el id “col-title”

```
a.id().equals("col-title")
```

Y una vez que se identificado inequívocamente el campo, se extrae mediante text() (en el caso de que se encuentre como texto).

```
String titulo = a.select("a").text();
```

El ejemplo anterior muestra como se extraen los títulos de los artículos, todos los campos se extraen mediante un procedimiento similar, solo varían los identificadores de las etiquetas.

Todos los campos extraídos se almacenan directamente, menos uno, que se interpreta, y ese campo es el número de los artículos mostrados. Si el número del artículo mayor es un múltiplo de 100, significa que muy posiblemente el autor tenga más artículos que no han sido parseados y almacenados, por lo que hay que realizar una nueva petición al servidor cambiando el valor del campo *cstart* de la url por un valor incrementado en 100, y repetir todo el proceso. Así hasta que el valor del último artículo no sea un múltiplo de 100, o hasta que no se devuelvan artículos.

Los links de los coautores se pueden extraer directamente de la tabla de la derecha, pero si el número es superior a 50 no se muestran todos. Para ver todos hay que pinchar en el link de “Ver todos los coautores”, por tanto la obtención de esta información se hace directamente desde ese link, que sigue un formato estándar similar al de las páginas de perfil. Solo hay que añadir el id de usuario al siguiente texto.

```
http://scholar.google.es/citations?view_op=list_colleagues&hl=es&user=
```

Como consideraciones adicionales hay que comentar que no todo es tan bonito como se pinta, y para trabajar con los datos, además de extraerlos, hay que adaptarlos y limpiarlos. El tratamiento que se ha hecho sobre los datos es:

-Pasar todos los nombres de personas a letras mayúsculas, y eliminar los caracteres corruptos (los que previamente eran 'ñ', caracteres acentuados u otros valores extraños que no han sido devueltos correctamente), para así poder realizar más adelante una comparación de cadenas más fiable.

-El primer artículo devuelto, en realidad no es un artículo y es el nombre de la tabla. Esto se detecta fácilmente porque como título tiene “Título / Autor”. Se filtra.

- Si el autor tiene página web personal (un caso no demasiado frecuente), el campo que habitualmente corresponde al número de lso artículos mostrados, es sustituido por este. Situándose el valor buscado en el registro siguiente. Por ello directamente se toma el último valor.

- Los caracteres chinos o árabes (que siguen el estandar UTF-8) se interpretan correctamente, no dan ningún tipo de problema.

El programa codificado tiene dos modos de funcionamiento, uno más simple (obtiene poca información), y otro desarrollado más completo (extrae más información y sigue una lógica más coherente). En la parte del parseado de la información, solo se diferencian en que uno extrae más campos que el otro, pero siguen un comportamiento idéntico.

La función de parseado básico es

public void parseaAutor(String id, int number, int rec) throws MalformedURLException, IOException.

Junto con

public void parseaCoautor(String id, int number) throws MalformedURLException, IOException

Donde id es el identificador único del autor, y los otros campos corresponden a parámetros de formación del grafo.

Mientras que la función de parseado más avanzada es

public AutorScholar extraeAutor(String id) throws MalformedURLException, IOException

Parámetros de salida

Como se ha comentado antes, coexisten dos versiones de la aplicación.

En la versión inicial los datos obtenidos tras procesar el perfil de autor se guardaban en un fichero de formato txt que contenía el nombre del autor, seguido de una sucesión de títulos de obras y en la línea siguiente el número de citas y el año de publicación.

Mediante un hashmap se llevaba la relación del autor con su fichero.

La versión *interesante* es la segunda, por ser más completa.

En esta versión, una vez que se ha parseado un autor, se tienen los siguientes datos (correspondencia en [A]) guardados en una clase:

-Coautores con perfil que proporciona el site.

-Coautores con los que tienes alguna publicación en común.

-Campos en los que investiga.

-Publicaciones. (Otra clase) Las cuales se componen de:

- Título

-Año de publicación

-Autores

-Medio en el que aparece.

La idea de esta estructura es tener todos los datos interesantes del autor, para posteriormente clasificarlos y analizarlos según se crea conveniente.

Además, si se parsean varios autores a la vez, el programa guarda información útil como una lista de los nombres de todos los autores procesados en diversas estrucutras, que se pueden recuperar llamando a sus métodos get correspondientes.

3.1.2.2 Formación del grafo

A continuación se explica el segundo módulo de la API.

Datos de entrada

Los datos de entrada son las estructuras con los datos de autores devueltas por el parser.

Procesamiento

Para entender y analizar las comunidades de Google Scholar, es necesario representarlas en forma de grafo. Una vez que se ha parseado un autor, hay que determinar por donde continuar la búsqueda de los demás autores. En este caso es sencillo, la búsqueda se continúa mediante los links de las páginas de perfil de sus coautores, y se continúa extrayendo información mediante un recorrido en anchura (explorando coautores de coautores) hasta una determinada profundidad.

Los nodos del grafo representan los autores, y las aristas las coautorías de artículos existentes entre ellos.

En esta parte es en la que más se diferencian las dos versiones.

La **versión inicial** recorre recursivamente los perfiles de autores llamando a *parseaAutor*, hasta llegar al último nivel de profundidad, que lo analiza con *parseaCoautor y* frena la recursión. Los datos de todos los autores se guardan en ficheros temporales (posteriormente serán borrados). La decisión de guardarlo en ficheros temporales es para posiblitar crear un grafo muy grande, sin preocuparse por los límites de la memoria RAM del ordenador (los ficheros tienen un tamaño del orden de KB, por lo que el espacio libre en disco no supone ningún problema).

Una vez que se han explorado todos los autores, se procede a crear el grafo, para ello se comparan los autores uno contra uno, leyendo sus ficheros y contando cuantos títulos de publicaciones tienen en común. Ese es el valor asignado a la arista. Si no tienen ninguna obra en común, no se establece arista entre ellos.

Con los autores (nodos) y las relaciones entre ellos (aristas), se vuelca el grafo a fichero.

Una vez que se ha acabado de comparar todos, se borran los ficheros con los datos.

La **segunda versión** es más eficiente, explora el grafo solo con una función que sabe cuando pararse, y crea las ramas en tiempo de exploración.

Al acabar de guardar todos los datos de los artículos de un autor, se busca cuantas veces aparece cada coautor entre los artículos, y ese es el número de colaboraciones conjuntas que tienen, y por tanto el valor de la arista.

Con este método se pueden representar coautores que no tengan perfil en la red social, simplemente debe

aparecer su nombre como autor de alguna publicación.

Para un autor que ya se ha explorado su perfil, se ignoran las relaciones con otros autores que se encuentren a posteriori, puesto que ya estarán reflejadas en el grafo y sería redundante.

Pero este método presenta el problema de como determinar cuando dos cadenas de caracteres corresponden con el mismo autor (record linkage), porque no siempre escriben igual su nombre, a veces lo hacen con el nombre completo, otras con un solo apellido, otras con sus iniciales, otras sin acentos. Es un problema complejo.

Para ello se ha utilizado Simmetrics [9]. Es una librería externa de JAVA que proporciona métodos para analizar el grado de similitud entre cadenas de caracteres en base a diversos algoritmos.

Ahora no se va a entrar en muchos detalles (en la sección 1 de los anexos de este documento están explicadas las pruebas realizadas para llegar a esta conclusión), pero para determinar igualdad de cadenas, se exige un valor de similitud superior a 0.926 para el algoritmo Jaro-Winkler.

En la tabla 4 se muestran ejemplos de strings diferentes detectadas como pertenecientes al mismo autor, y por tanto sus datos se agrupan en el mismo nodo haciendo más preciso y real el grafo.

DS VAN FOSSEN	DS VANFOSSEN
DH BARRET	DH BARRETT
BP O'CONNOR	BP OCONNOR
C GREENâ€□PEDERSEN	C GREEN-PEDERSEN

Tabla 4: Ejemplo de record linkage

El segundo problema resuelto con la ayuda de Simmetrics es el de detectar los coautores erróneos (que no han participado en ningún artículo con el autor base), en el anexo 1 también se detallan las pruebas. Se ha resuelto mediante una combinación de los algoritmos de Levenhstein y Smith Waterman.

Todas las cadenas de caracteres de autores diferentes se almacenan en un fichero temporal correspondiente a los nodos, y todas las relaciones (aristas) entre ellos en otro. Una vez que se han acabado de explorar todos los autores, se juntan ambos ficheros en uno solo, añadiendo las cabeceras y los cierres. El propósito de ambos ficheros temporales es poder declarar primero todos los nodos y después todas las aristas, para evitar problemas, ya que estos se van descubriendo en tiempo de exploración.

Datos de salida

Este módulo devuelve un fichero con el grafo en formato graphml. Este fichero es el que devuelve la API, está explicado en detalle en la sección 3.1.3

3.1.3 Datos de salida

La API devuelve un fichero en formato graphml, conteniendo el grafo obtenido.

Graphml es un formato de fichero usado en la representación de grafos.

La justificación de elegir este formato es que es muy usado en la representación de grafos, por lo que la mayoría de los programas de visualización de los mismos son capaces de interpretarlo correctamente.

En este caso los nodos del grafo representan los autores mientras que las aristas representan las relaciones entre ellos (una arista significa que ambos autores han trabajado juntos en alguna publicación).

Es un lenguaje muy simple, en esencia es un sistema de etiquetas anidadas basado en XML. En la ilustración 10 se puede ver un ejemplo de grafo.

Al comienzo del fichero, se declara la *cabecera*, que suele ser fija.

En la primera etiqueta se declara la versión de XML en que está el texto, así como el lenguaje de caracterización del mismo.

```
<?xml version="1.0" encoding="UTF-8"?>
```

Posteriormente ya se declara el inicio del grafo en sí (etiqueta graphml), junto con unas declaraciones de espacios de nombres que son fijas.

```
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
     http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">
```

Si se quiere declarar algún atributo que lleve asociado el grafo (como en este caso el peso de las ramas) se declara a continuación.
Debe llevar la etiqueta *key*, el identificador único del atributo (id), el elemento del grafo al que va asociado (node o edge), el nombre del atributo (attr.name), y el tipo del mismo (en este caso int, pero puede ser boolean, float, etc).

```
  <key id="enlaces" for="edge" attr.name="weight" attr.type="int"/>
```

Además, es necesario definir un nombre para el grafo. Y si se quiere, se puede indicar la orientación de las ramas (por defecto es indirecto).

```
<graph id="scholar" edgedefault="undirected">
```

Una vez que se ha declarado la cabecera del fichero, se puede empezar a declarar los nodos (nodes). Es importante declararlos antes que las ramas, porque haber problemas debido que algún nodo no esté declarado.

La declaración de nodos en este caso es muy simple, simplemente tiene el nombre del nodo. Obviamente, debe ser único para evitar conflictos. El nombre del nodo no puede llevar comillas (“), porque crea conflictos con las comillas del nombre, y algunos caracteres poco frecuentes como '&' también dan problemas, y no es recomendable usarlos.

```
<node id="A"/>
```

Una vez que se han declarado todos los nodos, se procede a declarar las ramas que los conectan.

La declaración de las ramas lleva un identificador único de la misma (en este caso como no se necesita identificar las ramas individualmente para su analisis, el identificador es una cadena de caracteres con un contador), el nombre del nodo origen, y el origen del nodo destino.
No hay ningún problema en declarar varias ramas con el mismo origen y destino.

```
<edge id="d4" source="A" target="D">
```

En este caso, como además hay un valor asociado a cada rama, se indica a continuación, diciendo el identificador del atributo y el valor.

```
<data key="enlaces">1</data>
```

Hay que cerrar la etiqueta de cada rama antes de empezar a declarar la siguiente.

```
</edge>
```

Una vez que están todas las ramas declaradas, se procede a cerrar todas las etiquetas del grafo que aún queden abiertas.

```
  </graph>
</graphml>
```

Y con esto ya estaría el fichero .graphml perfectamente especificado para proceder a su visualización y análisis mediante cualquier herramienta.

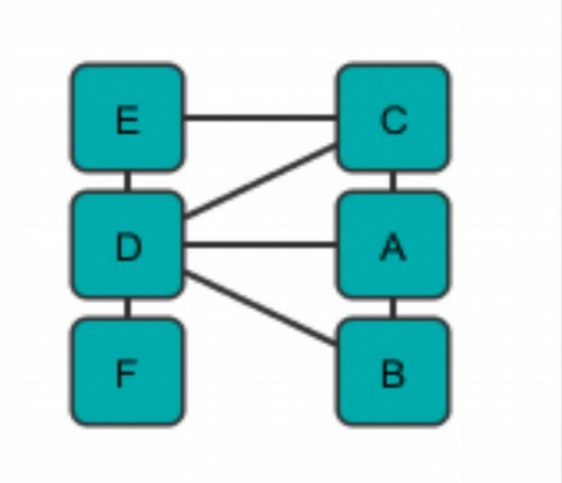

Ilustración 11: Ejemplo de grafo final

3.2 Visualización y análisis

El grafo devuelto en formato puede ser leído por cualquier herramienta de visualización de grafos. En este caso se va a usar Gephi [7], porque con esta herramienta es muy sencillo representar grafos de una forma que se puedan entender (el algoritmo de layout Force Atlas 2 [24] es muy útil para agrupar nodos en comunidades), y porque su market oficial proporciona plugins de algoritmos de detección de comunidades.

En este proyecto se van a probar dos algoritmos: Fast Unfolding [25] y Chinese Whispers [26].

Fast Unfolding viene por defecto con Gephi, mientras que Chinese Whispers hay que obtenerlo mediante un plugin de su market [29].

Se han elegido estos dos algoritmos principalmente por su rapidez y eficiencia, ya que se quiere trabajar con grafos de decenas de miles de nodos y se deben poder ejecutar las pruebas en un tiempo razonable. Además, que al ser dos algoritmos con una base totalmente diferente, se supone que no saldrán unos resultados idénticos, lo que permitirá extraer conclusiones interesantes sobre la detección de comunidades científicas.

En la sección 6.3 de los anexos se incluye un manual sobre como ejecutar los algoritmos y personalizar el grafo para que se pueda visualizar correctamente y entender.

4 Integración, pruebas y resultados

En esta sección se explican las pruebas realizadas.

4.1 Detección de comunidades

A continuación se vana introducir los 4 autores sobre los que se vana realizar posteriormente las pruebas. Tanto de la API, como de los algoritmos.

4.1.1 Investigadores

Como se ha explicado anteriormente, no vale cualquier autor para el análisis. Debe tener perfil en Google Scholar, y además, para poder crear un grafo a partir de él, es necesario que tenga algún coautor en su lista de coautores.

Teniendo en cuenta estos requísitos, se han seleccionado 4 autores con gran reputación a nivel internacional, para realizar las pruebas partiendo de ellos, así es más probable obtener grafos grandes con gran cantidad de información. Aunque los 4 pertenecen al mismo ámbito (inteligencia artificial), este ámbito es tan amplio que deberían ser independientes, y no aparecer unos en el grafo de otros hasta no alcanzar un alto nivel de profundidad en la exploración del mismo.

Francisco Herrera

Francisco Herrera Trigueros es un investigador español y catedrático de la Universidad de Granada.

Destaca por sus investigaciones en *soft computing*, más concretamente por sus algoritmos de toma de decisiones y de minería de datos.

Actualmente participa en un gran número de investigaciones y tesis, todas ellas relacionadas con la inteligencia artificial.

Marco Dorigo

Marco Dorigo [21] es un investigador italiano que actualmente trabaja para la Universidad de Bruselas.

Pertenece al campo de la inteligencia artificial y es mundialmente conocido por sus investigaciones en algoritmos y de enjambre y de colonias de hormigas para resolver problemas. Tiene varios premios internacionales por sus contribuciones, entre el prestigioso Marie Curie, otorgado en 2003.

Nick Jennings

Nick Jennings es un investigador británico, trabaja para la universidad de Southampton y para el gobierno inglés en el área de seguridad nacional.

Destaca en las áreas de computación multiagente y de inteligencia de sistemas.

Es la segunda persona más importante en el ámbito de la inteligencia artificial para Google Scholar.

Frans Coenen

Frans Coenen es profesor de la Universidad de Liverpool, y un gran experto en data mining. Tiene investigaciones en minería web, minería en grafos, minería en retina, de imágenes por satélite...

Es menos importante que los anteriores (tiene menos artículos y menos coautores).

4.1.2 Pruebas

Todas los datos para las pruebas se han obtenido con la versión *avanzada*, puesto que proporciona más información (los autores sin perfil en Google Scholar), y así tener datos más fiables a la hora de buscar comunidades.

Mediante la API se han obtenido los grafos con profundidad de 1 a 6 para los cuatro autores mencionados en el apartado 4.2.1. Se ha considerado que los grafos de profundidad ya eran suficientemente grandes, pasan los 250.000 nodos. Tanto datos son dificiles de manejar e interpretar, y la representación visual de los grafos resulta un poco caóitica.

El *cuello de* botella de la API se produce en las peticiones al servidor, que tiene que esperar a recibir los datos. De media, analiza 43 autores diferentes por minuto.

Los tiempos promedio en obtener los grafos son los siguientes:

En profundidad 1 tarda 5 segundos.

En profundidad 2 tarda minuto y medio

En profundidad 3 tarda 6 minutos.

En profundidad 4 tarda alrededor de 13 minutos.

En profundidad 5 tarda una media hora.

En profundidad 6 sobre hora y media.

La tabla 5 muestra el número de nodos y de aristas de los grafos. (nodos/aristas)

Recursividad	1	2	3	4	5	6
Francisco Herrera	236/235	2688/3867	9788/13448	52250/76672	89086/133988	248131/420461
Marco Dorigo	332/331	3250/4609	15176/19456	51134/70740	129002/192244	323296/560489
Nick Jennings	345/344	5990/8480	23812/33071	60922/90686	150875/275718	289101/597741
Frans Coenen	168/167	1323/1425	15592/20292	53106/83511	156002/290083	332373/747110

Tabla 5: Nodos/Aristas

Se aprecia como los grafos crecen exponencialmente con cada nuevo nivel de profundidad. Como media, se puede decir que el grafo de un nivel es 3 veces mayor que el de su predecesor (sin tener en cuenta el paso de nivel 1 a 2). Aunque en ocasiones se pueden producir variaciones muy significativas, como en el caso de Francisco Herrera, que del nivel 3 al 4 el grafo creció casi 6 veces. Esto es debido a que en ese nivel han coincidido muchos autores importantes (más de 40 coautores con perfil) a los que aplicar la recursividad.

La relación de nodos y aristas se suele mantener constante, siendo aproximadamente 1,5 veces mayor el número de aristas que el de nodos (hay que tener en cuenta que el grafo es no dirigido).

Los grafos con profundidad 1 no tienen ningún tipo de interés en cuanto a la detección de comunidades. Solo tienen una comunidad, son una estrella con el autor inicial en el medio (Ilustración 12)

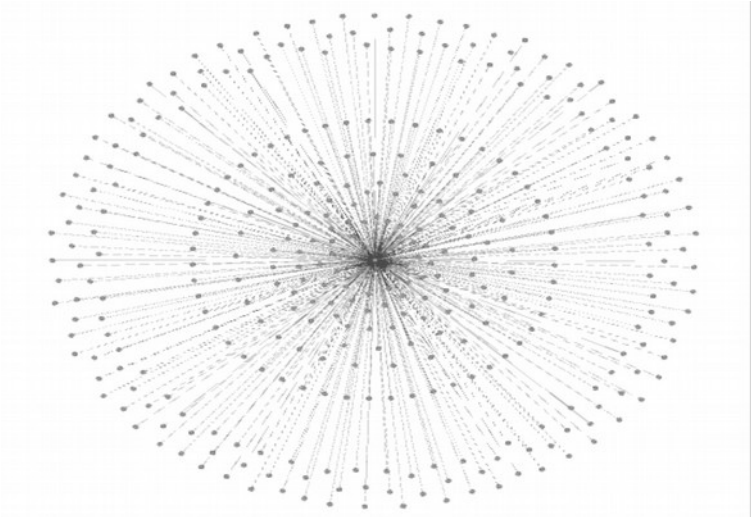

Ilustración 12: Grafo de Nick Jennings con profundidad 1

La detección de comunidades de Fast Unfolding se ha probado dando al algoritmo los valores de ratio 0'2, 1 y 2 para todos los grafos. Según el orden de exploración de los nodos, el número de comunidades encontradas puede variar, pero siempre devuelve valores similares.

Para los niveles 2, 3 y 4 la prueba tarda menos de 5 segundo. A partir de profundidad 5, el algoritmo empieza a sufrir, tardando minuto y medio en devolver resultados. Para profundidad 6, tarda unos 10 minutos en ejecutarse con cada ratio.

En la tablas 6, 7, 8 y 9 se muestra el número de comunidades detectadas con cada profundidad.

Francisco Herrera	2	3	4	5	6
Ratio 0,2	31	61	142	319	495
Ratio 1,0	19	35	56	183	285
Ratio 2,0	16	26	30	146	246

Tabla 6: Fast Unfolding - Francisco Herrera

Marco Dorigo	2	3	4	5	6
Ratio 0,2	31	79	171	227	579
Ratio 1,0	16	47	67	78	323
Ratio 2,0	11	39	40	46	285

Tabla 7: Fast unfolding - Marco Dorigo

Nick Jennings	2	3	4	5	6
Ratio 0,2	32	73	138	218	300
Ratio 1,0	17	33	42	62	78
Ratio 2,0	13	19	26	31	47

Tabla 8: Fast unfolding - Nick Jennings

Frans Coenen	2	3	4	5	6
Ratio 0,2	13	74	146	230	386
Ratio 1,0	11	37	46	62	150
Ratio 2,0	10	24	25	32	119

Tabla 9: Fast unfolding - Frans Coenen

Lo primero que se ve es que a menor ratio, mayor número de comunidades. Pero no se puede establecer una relación lineal entre los ratios. El número crece progresivamente, hasta llegar a la profundidad 6, que se dispara, menos para Nick Jennings. Esto se puede deber a que al haber un número mayor de nodos, los valores de modularidad no son suficientemente elevados para crear comunidades muy grandes. Menos en el caso de Nick Jennings, que las comunidades estarán más marcadas.

Las pruebas de Chinese Whispers se han realizado solo sobre los grafos con profundidad 2, 3 y 4. Puesto que con grafos más grandes, el algoritmo no funciona bien al detectar nodos inconexos (en estos grafos no hay) y lanza una excepción que impide acabar su ejecución. Como el algoritmo tarda poco (5 segundos como máximo), se ha ejecutado con 15 iteraciones, a partir de ese valor el número de comunidades encontradas converge en todos los algoritmos encontrados.

En las tablas 10, 11, 12 y 13 se muestran los resultados.

Francisco Herrera	2	3	4
# de Comunidades	26	83	338

Tabla 10: Chinese Whispers - Francisco Herrera

Marco Dorigo	2	3	4
# de Comunidades	30	131	436

Tabla 11: Chinese Whispers - Marco Dorigo

Nick Jennings	2	3	4
# de Comunidades	38	128	455

Tabla 12: Chinese Whispers - Nick Jennings

Frans Coenen	2	3	4
# de Comunidades	13	120	419

Tabla 13: Chinese Whispers - Frans Coenen

Este algoritmo encuentra un número de comunidades muy elevado. Para profundidad 2, los valores son idénticos a los de Fast Unfolding con ratio 0,2. Esto es porque a esta profundidad hay un número muy pequeño de nodos y las comunidades están muy marcadas (en este grafo solo están los coautores del autor, con lo que, básicamente, cada comunidad corresponde al grupo de cada coautor).

Para las siguientes profundidades los valores se disparan, por lo que este algoritmo es menos estricto a la hora de formar comunidades, y detecta muchas.

En la sección 7.4 de los anexos viene representada evolución de todos los grafos, ordenados por nivel de profundidad desde 2 hasta 6 (en la sección 7.3 se explica como se obtiene la representación de dichos grafos). Los colores de los nodos corresponden a las comunidades encontradas mediante Fast Unfolding con ratio 1.0.

En todos los grafos, en la parte exterior se encuentran una serie de comunidades muy marcadas. Estas corresponden con los coautores explorados en el último nivel. Mientras que en la parte interior de los grafos, aparecen un gran número de comunidades revueltas, y para profundidad 5 y 6, apenas se puede interpretar nada por el gran número de nodos pertenecientes a distintas comunidades existentes.

Existe una comunidad predominante en el centro de cada grafo que tiene entre un 12% y un 21 % de los nodos del grafo. Esa comunidad siempre incluye al autor base. En casos como el de Frans Coenen y Nick Jennings (secciones 7.4.3 y 7.4.4), existen dos comunidades dominantes entrelazadas.

5 Conclusiones y trabajo futuro

Durante la realización de este TFG se ha desarrollado una herramienta capaz de analizar y extraer automáticamente información bibliográfica desde Google Scholar. Esta herramienta será puesta a disposición de cualquier investigador que la necesite en el siguiente enlace, además tendrá total libertad para modificarla y mejorarla:

https://mega.co.nz/#!jwJzhapD!q_8lHiUcY5TzoWs7JWc4Bdzx1SejwGWUR4QWrMNW-zQ

El diseño de la misma se ha realizado de manera modular y extensible pensando en futuras mejoras y ampliaciones. Puesto que extrae y almacena toda la información posible, aunque solo se usen algunos campos en este proyecto, para así poder tener todos los datos posibles para futuros nuevos criterios de crear grafos sin tener que modificar esa parte del programa (salvo que Google realice un cambio sustancial en el HTML y provoque que el programa quede obsoleto).

Por otra parte, el programa se ha diseñado de manera modular para que sea muy sencillo realizar pequeñas modificaciones en el mismo, sobre todo a la hora de cambiar los criterios para generar nuevos grafos, como por ejemplo en función de los años de publicación de los artículos, o filtrando los autores en base a campos de investigación (palabras clave), criterios que pueden proporcionar nuevos grafos con información interesante.

Como contribuciones concretas del trabajo desarrollado se mencionará:

- Se ha diseñado una API que permite extraer información automatizada desde GoogleScholar.

- Utilización de técnicas de similitud de cadenas (para eliminación de ambigüedades)

- El fichero obtenido mediante la API se puede introducir en cualquier herramienta de visualización de grafos (como Gephi o Cytoscape)

- Estudio experimental, para varios casos concretos, del funcionamiento del sistema.

En un futuro se prevee continuar estudiando la detección de comunidades científicas probando nuevos algoritmos, o incluso desarrollando algunos de creación propia. También se pretende seguir investigando en el campo de la similitud de strings para intentar encontrar algún algoritmo (o combinación de algoritmos) que resuelva el problema con una precisión aún mayor.

6 Referencias

[1] – Google Scholar – Sitio oficial – Disponible en: http://scholar.google.es/ [Consultado: 03/03/2014]

[2] – Google Scholar - Disponible en: http://www.georgefox.edu/offices/murdock/Tutorials/introGS.html [Consultado: 03/04/2014]

[3] – Ejemplo de perfil de usuario de Google Scholar - Disponible en: http://scholar.google.es/citations?hl=es&user=fpf6EDAAAAAJ [Consultado: 04/04/2014]

[4] – Parser Google Scholar en Python (por Christian Keibrich) - Disponible en: http://www.icir.org/christian/scholar.html [Consultado: 10/03/2014]

[5]– Parser Google Scholar en JAVA - Disponible en: http://nwb.cns.iu.edu/svn/nwb/trunk/sci2/plugins/normalplugins/edu.iu.sci2.reader.googlescholar/src/edu/iu/sci2/reader/googlescholar/search/GoogleScholarReaderHelper.java [Consultado: 17/03/2014]

[6] – Visualización de grafos - Cytoscape - Disponible en: http://www.cytoscape.org/ [Consultado: 20/04/2014]

[7] – Visualización de grafos - Gephi - Disponible en: https://gephi.org/ [Consultado: 25/04/2014]

[8] – Graphml - Disponible en: http://graphml.graphdrawing.org/ [Consultado: 22/04/2014]

[9] – Libreria JAVA Simmetrics. - Disponible en: http://sourceforge.net/projects/simmetrics/ [Consultado: 03/04/2014]

[10] – Jaro, M. A. (1989). "Advances in record linkage methodology as applied to the 1985 census of Tampa Florida". Journal of the American Statistical Association

[11] – Levenshtein, Vladimir I. (February 1966). "Binary codes capable of correcting deletions, insertions, and reversals". Soviet Physics Doklady

[12] - A. Monge and C. Elkan. The field-matching problem: Algorithm and applications. AAAI Press, 1996. Pages 1-30

[13] – Librería JAVA Jsoup – Disponible en: jsoup.org [Consultado: 05/04/2014]

[14] – DBLP - Disponible en: http://dblp.uni-trier.de/db/ [Consultado: 04/03/2014]

[15] – Cite SeerX - Disponible en: http://citeseerx.ist.psu.edu/index [Consultado: 07/03/2014]

[16] - A. J. Lait and B. Randell. An assessment of name matching algorithms. Technical report, Department of Computer Science, University of Newcastle upon Tyne, UK, 1993.

[17] – S. Needleman and C.Wunsch. A general method applicable to search for similarities in the amino acid sequence of two proteins. Molecular Biology, 48:443-458, 1970

[18] – G. I. Ivchenko and S. A. Honov. On the Jaccard Similarity Test. Journal of Mathematical Sciences. 1998

[19] – Christian Kreibich - Disponible en: http://www.icir.org/christian/ [Consultado: 17/03/2014]

[20] – Francisco Herrera - Disponible en: http://decsai.ugr.es/~herrera/Curriculum.html [Consultado: 10/06/2014]

[21] – Marco Dorigo - Disponible en: http://www.scholarpedia.org/article/User:Marco_Dorigo [Consultado: 10/06/2014]

[22] – Nick Jennings - Disponible en: http://users.ecs.soton.ac.uk/nrj/ [Consultado: 10/06/2014]

[23] – Mendeley data base - Disponible en: http://www.mendeley.com/ [Consultado: 10/06/2014]

[24] -Jacomy and Venturini. ForceAtlas2, A Graph Layout Algorithm for Handy Network Visualization. Pages 2-8. 2011

[25] -Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, Etienne Lefebvre, Fast unfolding of communities in large networks, in Journal of Statistical Mechanics: Theory and Experiment 2008 (10), P1000

[26] – Chris Biemann. Chinese Whispers - an Efficient Graph Clustering Algorithm and its Application to Natural Language Processing Problems. Pages 1-7.

[27] - J. Newman. "Detecting community structure in networks". Eur. Phys. 2004 pages 1- 9

[28] - Stijn van Dongen, Graph Clustering by Flow Simulation. PhD thesis, University of Utrecht, May 2000

[29] – Plugin Gephi – Chinese Whispers - Disponible en: https://marketplace.gephi.org/plugin/chinese-whispers-clustering/ [Consultado: 14/05/2014]

[30] – Frans Coenen - Disponible en: http://cgi.csc.liv.ac.uk/~frans/ [Consultado: 03/06/2014]

7 Anexos

7.1 Similitud de cadenas

Record linkage

Se tiene el problema de como saber cuando dos cadenas de caracteres corresponden con el mismo autor, puesto que a veces firman con el nombre completo, solo las iniciales, un apellido, etc

Evidentemente, este problema no tiene una solución exacta, lo que se va a intentar es detectar el máximo número de cadenas de caracteres que pertenecen al mismo autor, sin que se tomen dos autores diferentes como iguales.

La manera óptima de saber cual de los algoritmos de comparación de cadenas explicados anteriormente es el que mejor resultado da para el problema planteado es realizando pruebas experimentales.

Para ello se han tomado unos perfiles totalmente aleatorios de autores de distintos países, y se han cruzado todas las cadenas de caracteres (tal y como trabaja con ellas el programa, es decir, después de ser tratadas su capitazilación y eliminación de caracteres erróneos) con nombres de autores encontradas entre sí para sacar su valor de similitud (exceptuando los que hasta JAVA, mediante el método *equals(),* considera iguales).

Los valores de similitud de cadenas devueltos por los algoritmos oscilan entre 0 y 1, siendo 1 que las cadenas son exactamente idénticas, y 0 que no tienen ninguna semejanza en común.

Esta prueba se ha realizado con cada algoritmo y se han volcado los resultados a un fichero de texto con formato CSV.

T BERGMAN	B GRANT	0.3333333

Tabla 14: Ejemplo valor similitud con Levenhstein

Una vez realizadas todas las comparaciones, se tiene un fichero por cada algoritmo con más de 100.000 comparaciones, y casi todas diferentes. Una cantidad de información que debería ser suficiente para obtener unas conclusiones fiables.

Los ficheros generados se pueden abrir con cualquier hoja de cálculo. Una vez abiertos, se ordenan los datos por valor de similitud de forma descendente, y se busca el valor más alto entre cadenas de caracteres que no pertenecen al mismo autor.

Soundex es el primero en descartarse, da *falsos positivos* (considera como iguales, valor 1, cadenas que no lo son), como por ejemplo los mostrados en la siguiente tabla. Un resultado lógico, puesto que los criterios fonéticos no son la mejor forma de determinar autores iguales con tal cantidad de apellidos similares.

JD GODWIN	JD COWDEN	1.0
KG BERMAN	KK BROWN	1.0
M CINQUE	M JONES	1.0
JC HOGG	J USUKA	1.0
LG QUE	L ZHUO	1.0
AK ZAAS	A KOCH	1.0
T GANOUS	TE KING	1.0

Tabla 15: Ejemplo de valores erróneos para Soundex

Nada más abrir el fichero, uno se da cuenta de que **Monge Elkan** no es capaz de resolver este problema. Casi un 25% de las muestras son *falsos positivo.* El orden de comparación de las dos cadenas es muy importante, pero no es determinante como para dar valores aceptables.

T BERGMAN	B BRIVATI	1.0
KR LUTHER	R DUNPHY	1.0
R DUNPHY	N SITTER	1.0

Tabla 16: Ejemplo de valores erróneos para Monge Elkan

Jaccard, además de dar valores de similitud bajísimos en la mayoría de comparaciones, muestra los primeros valores erróneos muy pronto, con solo 30 comparaciones por encima, por tanto este algoritmo se desecha. Este resultado era previsible, este algoritmo en realidad es para detectar similitud de conjuntos, y el orden de los caracteres no es importante.

T BALE	T BERGMAN	0.3333333
N SITTER	N HUNTINGTON	0.3333333
A SZCZERBIAK	A KROUWEL	0.3333333
C DANN	C GREENâ€PEDERSEN	0.3333333

Tabla 17: Ejemplo de valores erróneos para Jaccard

NeedlemanWunsch da sus primeros valores erróneos también muy temprano, después de apenas 25 comparaciones mejores.

G MARTNEZ	R MARTNEZ	0.9444444
WJ MARTIN	PJ MARTIN	0.9444444
TW BAILEY	NW BAILEY	0.9444444

Tabla 18: Ejemplo de valores erróneos para NeedlemanWunsch

Para **SmithWaterman** los primeros valores corresponden a cadenas de caracteres con muchos caracteres coincidentes consecutivos. Las primeras comparaciones erróneas salen a partir de la fila 130, cuando empiezan a aparecer cadenas de caracteres muy cortas.

BL LAWSON	R LAWS	0.93333334
S MIR	S MARUOKA	0.92
S MIR	S MARKLUND	0.92
G LIU	G LIAO	0.92

Tabla 19: Ejemplo de valores erróneos para Smith Waterman

Para Levenhstein los primeras comparaciones nos triviales erróneas salen a partir de la fila 250. Son cadenas de caracteres que difieren en unicamente un valor (la inicial de un nombre), pero no se pueden dar por válidas.

R MARTNEZ	G MARTNEZ	0.8888889
MS WILSON	JS WILSON	0.8888889
WJ MARTIN	PJ MARTIN	0.8888889
TW BAILEY	NW BAILEY	0.8888889

Tabla 20: Ejemplo de valores erróneos para Levenhstein

Las primeras comparaciones erróneas para **Jaro Winkler** salen curiosamente a partir del mismo valor (0,926), con más de 400 valores por encima (sin contar comparaciones triviales), y también difieren unicamente en un valor.

JS WILSON	SH WILSON	0.9259259
WJ MARTIN	PJ MARTIN	0.9259259
TW BAILEY	NW BAILEY	0.9259259

Tabla 21: Ejemplo de valores erróneos para Jaro Winkler

Por tanto JaroWinkler exigiendo un valor de similitud de 0.926, es el algoritmo que mejor resuelve el problema planteado, y por tanto el algoritmo usado en el programa anteriormente explicado.

La solución dada es la mejor obtenida, pero no resueve el problema totalmente, quedan muchos emparejamientos con valores inferiores que deberían haber sido considerados como idénticos (tabla9). Es un campo en el que aún queda mucho por explorar.

CE HENNESSY	C HENNESSY	0.90303034
HD MENENDEZ	H MENNDEZ	0.90235686
CL WOHLFORD-LENANE	C WOHLFORD-LENANE	0.8834422
DA SCHWARTZ	D SCHWARTZ	0.86969703
E MCELVANIA	E MCELVANIA-TEKIPPE	0.8596491

Tabla 22: Ejemplo de falsos negativos

Detección de strings que no pertenecen a un conjunto

Por otra parte, se ha intentado detectar los autores erróneos de la lista de coautores (autores que no han colaborado en ninguna investigación con el investigador principal). La idea era detectar cuales de ellos tenían poca similitud con cualquier cadenas de caracteres de los autores que firman los artículos.

Para resolver el problema correctamente, un solo algoritmo se antoja insuficiente. No se puede establecer un valor de corte para resolver el problema correctamente, cada cadena es diferente, por su longitud o su parecido con otras smiliares, por lo que el problema se ha resuelto aplicando dos algoritmos.

Primero de cada autor, se busca la cadena de entre las firmas de artículos que tiene un mayor de

similitud según el algoritmo de Levenhstein. Así se tiene la cadena más parecida con la original.

Se ha escogido Levenhstein porque devuelve cadenas *intuitivamente* más parecidas que JaroWinkler, el algoritmo que mejor resuelve el problema anterior.

Ahora hay que ver si estas dos cadenas son suficientemente similares como para pertenecer al mismo elemento. El criterio más lógico para determinar esto es que deben tener un buen número de caracteres coincidentes seguidos (el nombre o el apellido), por tanto se aplica el algoritmo de Smith Waterman exigiendo un valor de similitud de 0,5 sobre estas dos cadenas.

Si tiene un valor menor, se descarta como coautor.

A continuación se muestran algunos ejemplos de coautores detectados como erróneos, junto con la cadena de caracteres detectada como más parecida según Levenhstein.

RECHAZADO RODERICH GRO M DORIGO 0.45
RECHAZADO PIER LUCA LANZI PVC CAIRONI 0.3272727
RECHAZADO DARIO IZZO R RIOLO 0.42857143
RECHAZADO MARCO DEL REY MAM DE OCA 0.32
RECHAZADO ANDREAGIOVANNI REINA A DECUGNIERE 0.3
RECHAZADO FRANS COENEN A LOPEZ 0.2
RECHAZADO SINISA TODOROVIC S GARCIA-MORATILLA 0.1875

No se aplica solo SmithWaterman porque puede encontrar muchos caracteres coincidentes seguidos en cadenas que no tienen que ver, por lo que devuelve más *falsos positivos* que aplicando estos dos algoritmos combinados.

Con este criterio no se asegura eliminar todos los falsos coautores (de hecho eso es imposible, pues en algún caso seguro que hay un coautor verdadero con una cadena de caracteres muy similar , por ejemplo mismo apellido, que un coautor falso), pero sí que es seguro que todos los autores detectados como erróneos lo son.

Nota: Al no conocer todos los autores del mundo, es imposible estar completamente seguro de que dos cadenas de caracteres corresponden al mismo autor, por tanto la valoración de las comparaciones se ha hecho de forma *intuitiva*.

7.2 Manual de ejecución de ParserScholar

A continuación se explica como ejecutar correctamente la API.

Se ha implementado una interfaz gráfica para facilitar su ejecución.

La pantalla principal es la siguiente:

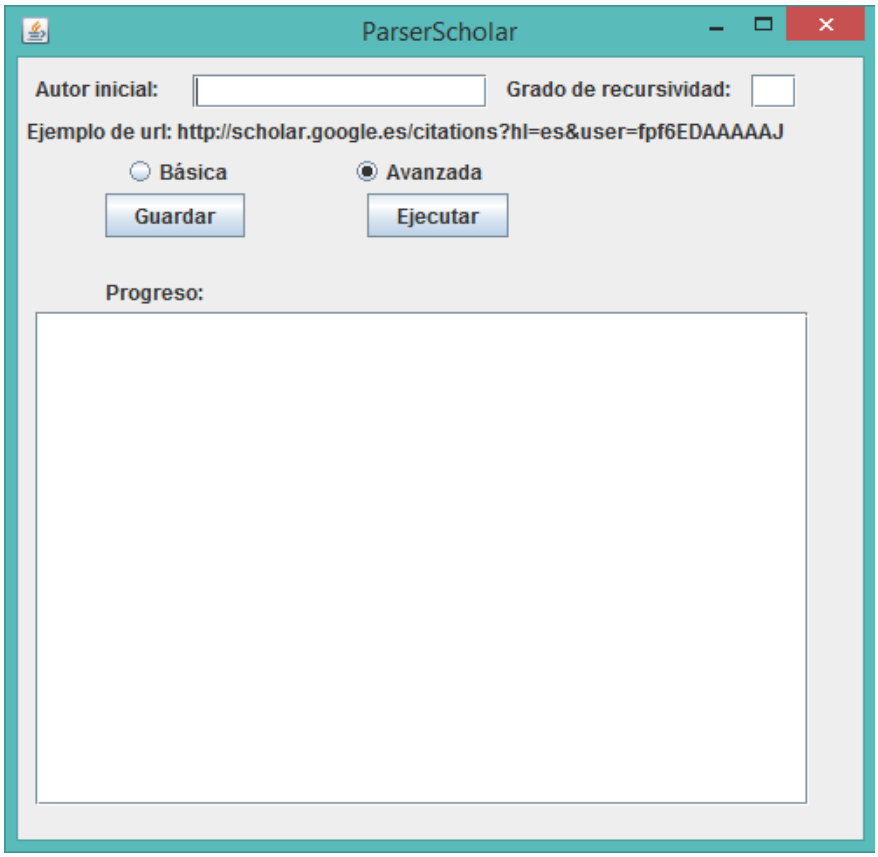

Ilustración 13: Interfaz aplicación

El ***Autor inicial*** es el id del autor sobre el que se va a iniciar la exploración de la base de datos. Se puede introducir tanto la url completa de su páigna de perfil, como simplemente su id.

El ***grado de recursividad*** indica el valor numérico de la profundidad de exploración. Con 1 o menos solo se explora el autor inicial.

Lo siguiente es elegir la versión a ejecutar, ambas están explicadas anteriormente.

El botón ***guardar***, elige el destino de guardado del fichero .graphml resultante. Como se supone que se dispone de permisos de lectura y escritura en la ruta elegida, se toma ese directorio como lugar donde almacenar los ficheros temporales necesarios.

Una vez que se ha rellenado todo lo anterior, el botón ***ejecutar*** empieza a recopilar datos y a crear el fichero del grafo con los datos indicados. Cuando esté listo. Se indica el FIN, y se muestra el número de nodos y de ramas del grafo final.

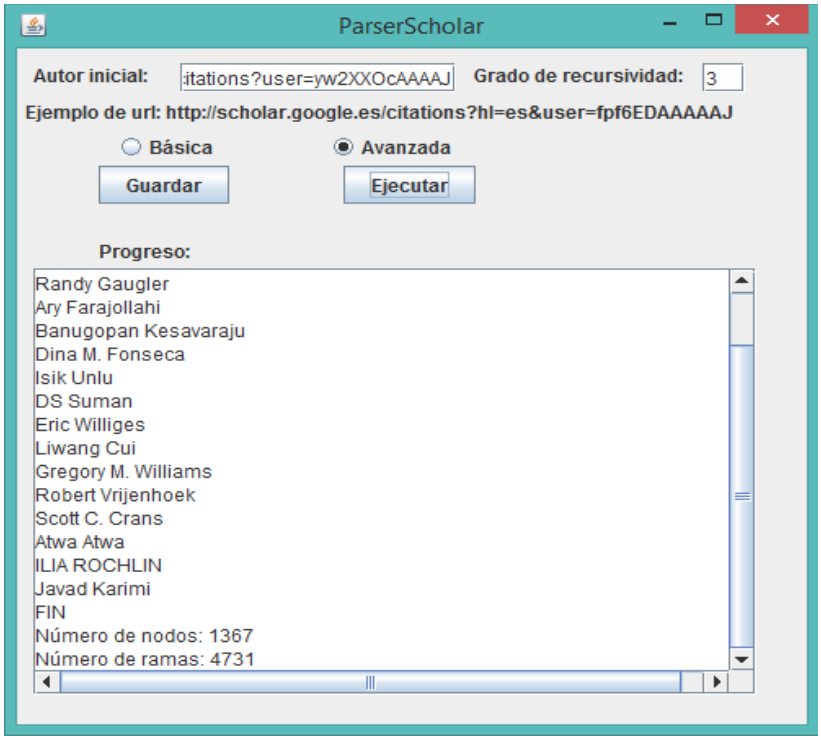

Ilustración 14: Ejemplo ejecución ParserScholar.Java

7.3 Manual de visualización

Gephi es capaz de abrir ficheros graphml.

Al abrirlo suele salir un mensaje como este (ilustración 15)en el que elige el tipo de grafo (en este caso son grafos indirectos, y muestra el número de nodos y de aristas. Si el grafo es muy grande, puede ser necesario asignar más memoria RAM al programa.

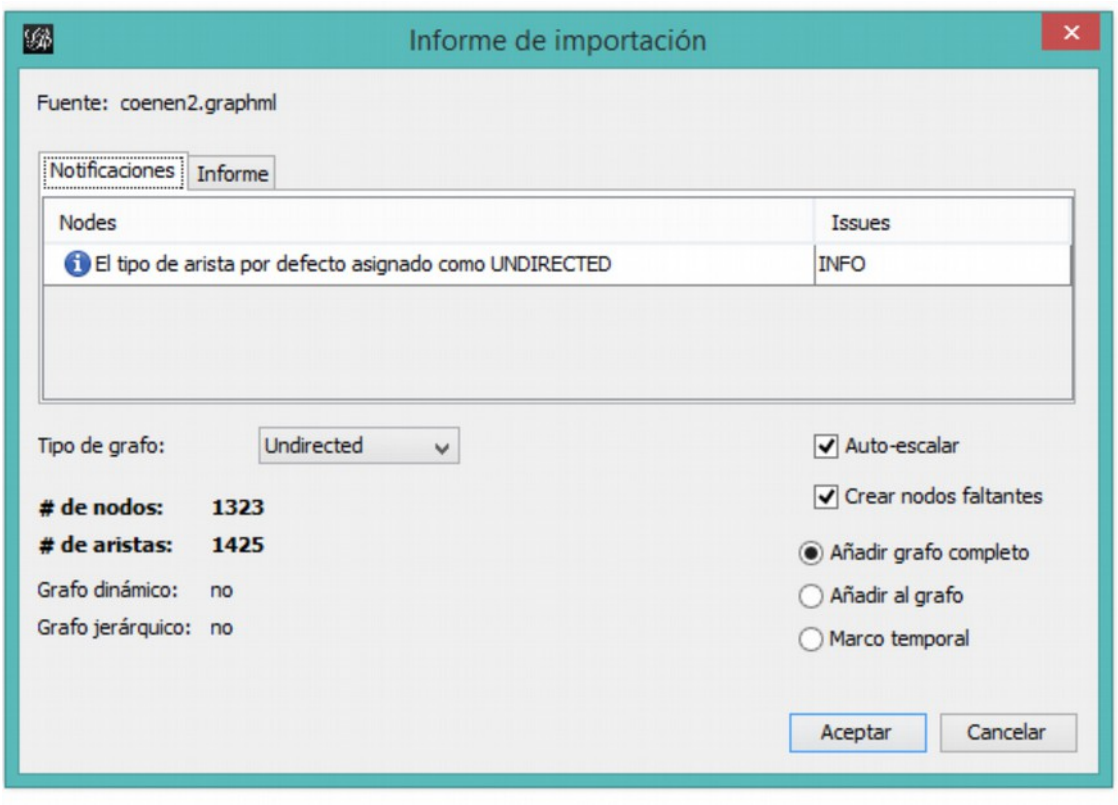

Ilustración 15: Gephi - Informe importación

Una vez que se ha aceptado, aparece el grafo en el centro de la imagen, con el layout por defecto (ilustración 16). Si hay un número elevado de nodos, con ese layout no se verá casi nada.

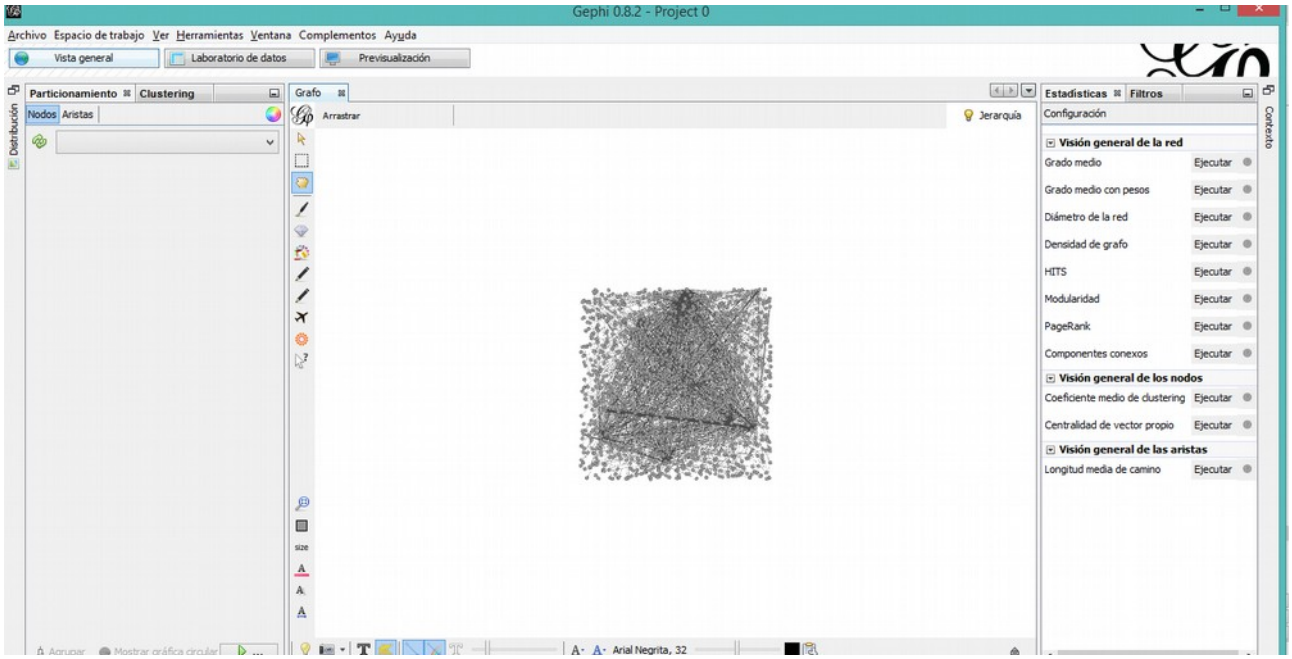

Ilustración 16: Gephi - Pantalla principal

En este caso se quieren detectar comunidades, por lo que el layout con el que mejor se visualizan es Force Atlas 2. Se cambia en distribución (a la izquierda). La ejecución ha de pararse manualmente cuando se considere oportuno.

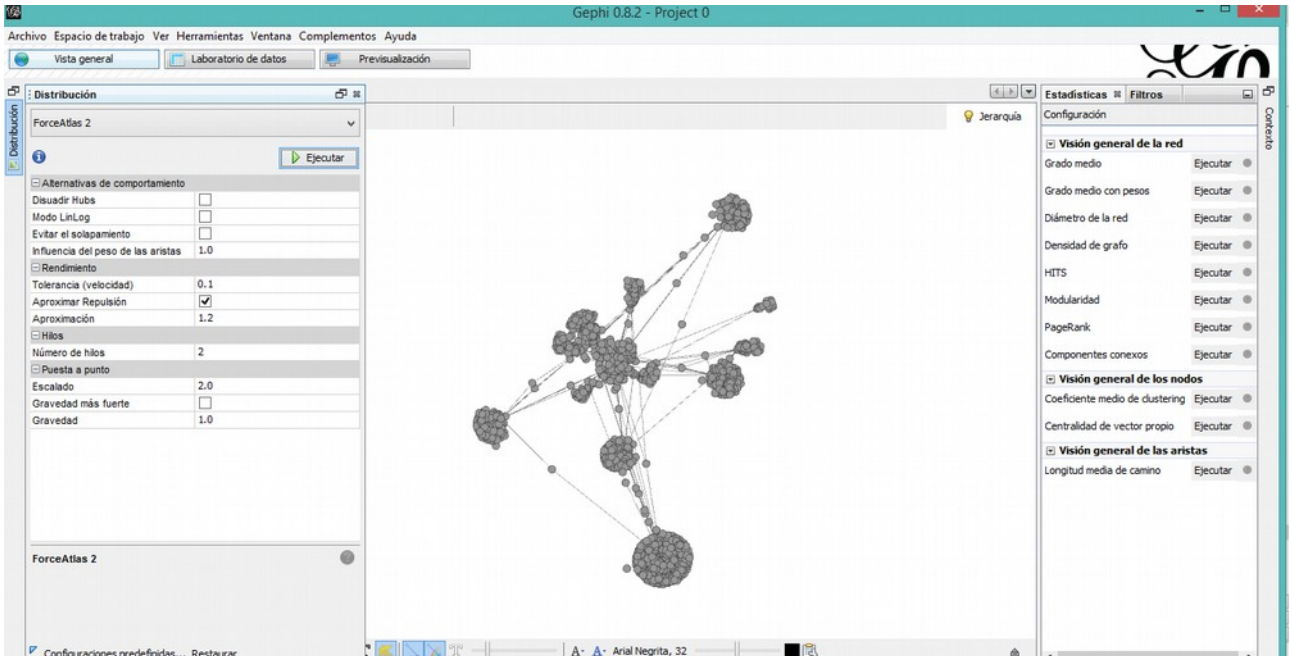

Ilustración 17: Gephi - Force Atlas 2

Ahora se va a proceder a ejecutar los algoritmos de detección de comunidades. El primero de ellos,

Fast Unfolding, viene integrado en el botón modularidad de las estadísticas (derecha). Al ejecutarlo pide como parámetro el ratio (en este caso se usará 1.0). Una vez ejecutado muestra un reporte con el número de comunidades encontradas.

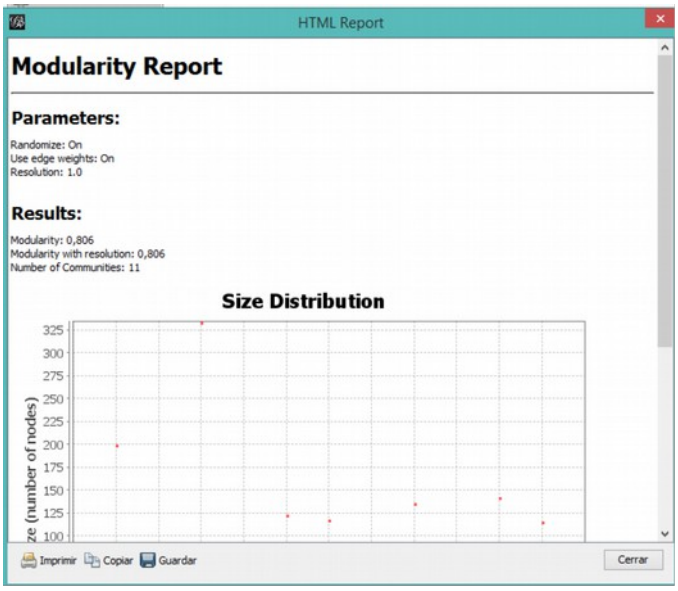

Ilustración 18: Gephi - Reporte de modularidad

Una vez descargado e instalado el plugin de Chinese Whispers, este se ejecuta desde Clustering, seleccionándolo, e introduciendo el número de iteraciones. El resultado es el número de comunidades encontradas. Además, los nodos del grafo se pintan con el color de su comunidad correspondiente.

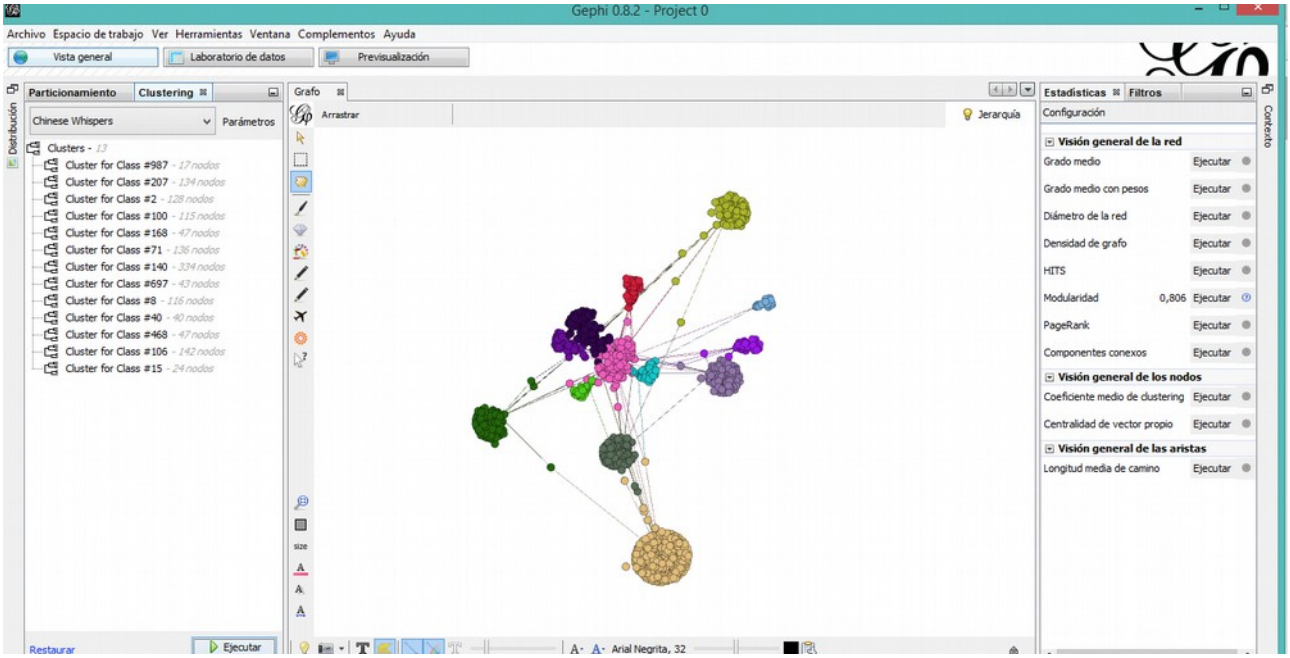

Ilustración 19: Gephi - Chinese Whispers

Se puede cambiar el color de cada comunidad, o establecer el criterio de clustering del layout en Distribución.

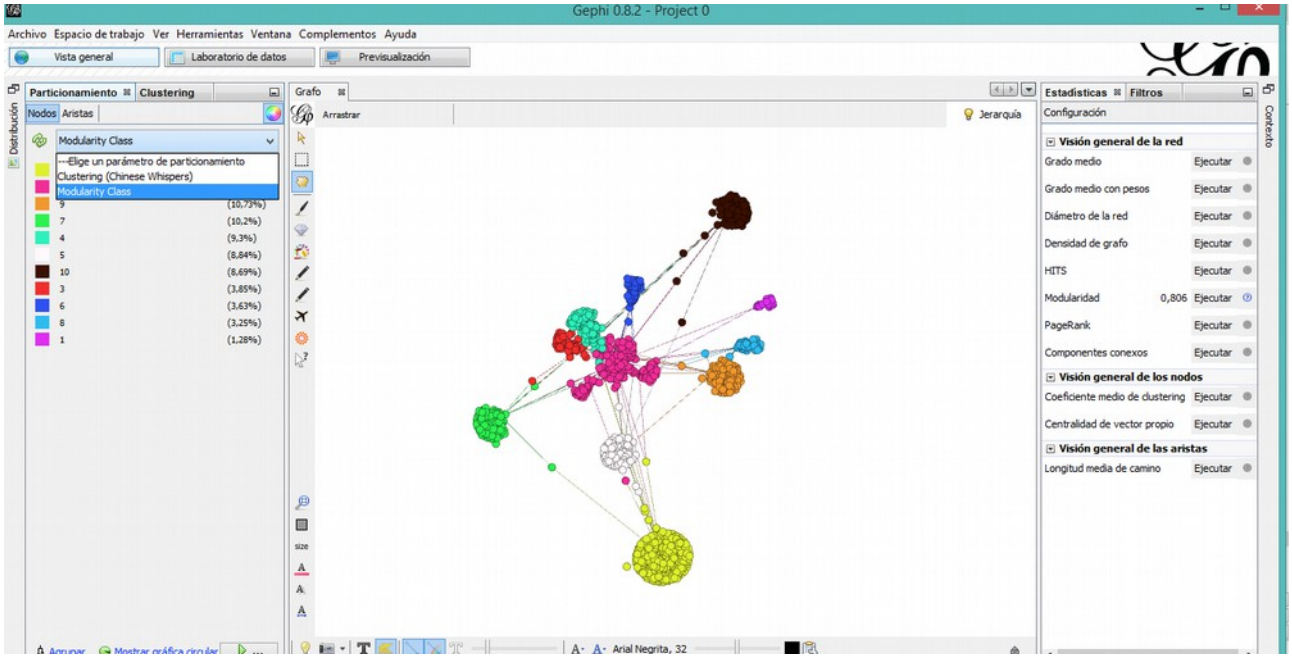

Ilustración 20: Gephi - Color de nodos

7.4 Grafos

7.4.1 Francisco Herrera

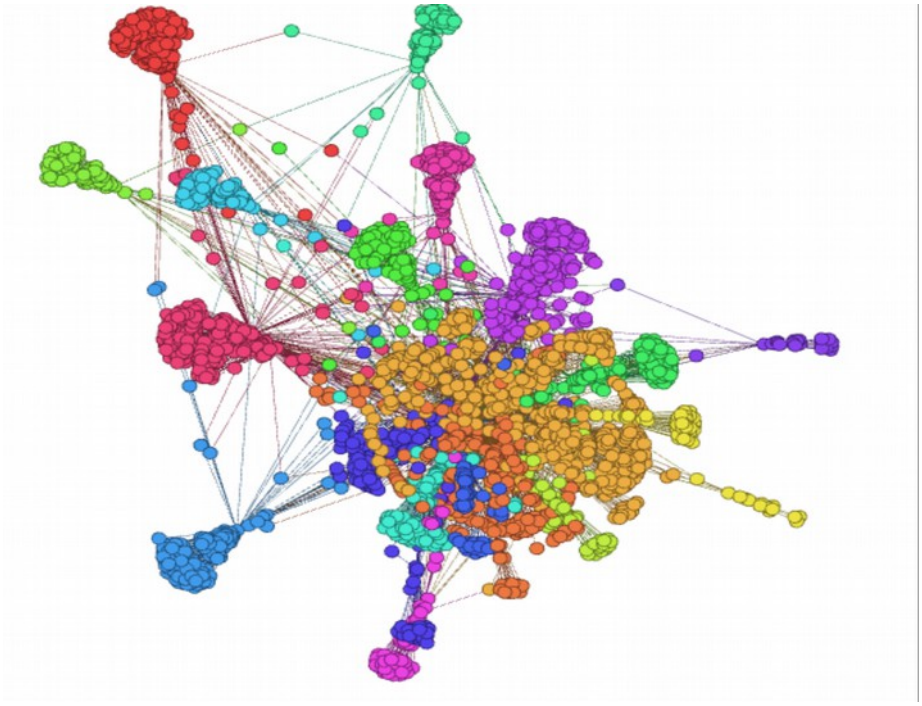

Ilustración 21: Francisco Herrera prof 2

Ilustración 22: Francisco Herrera prof 3

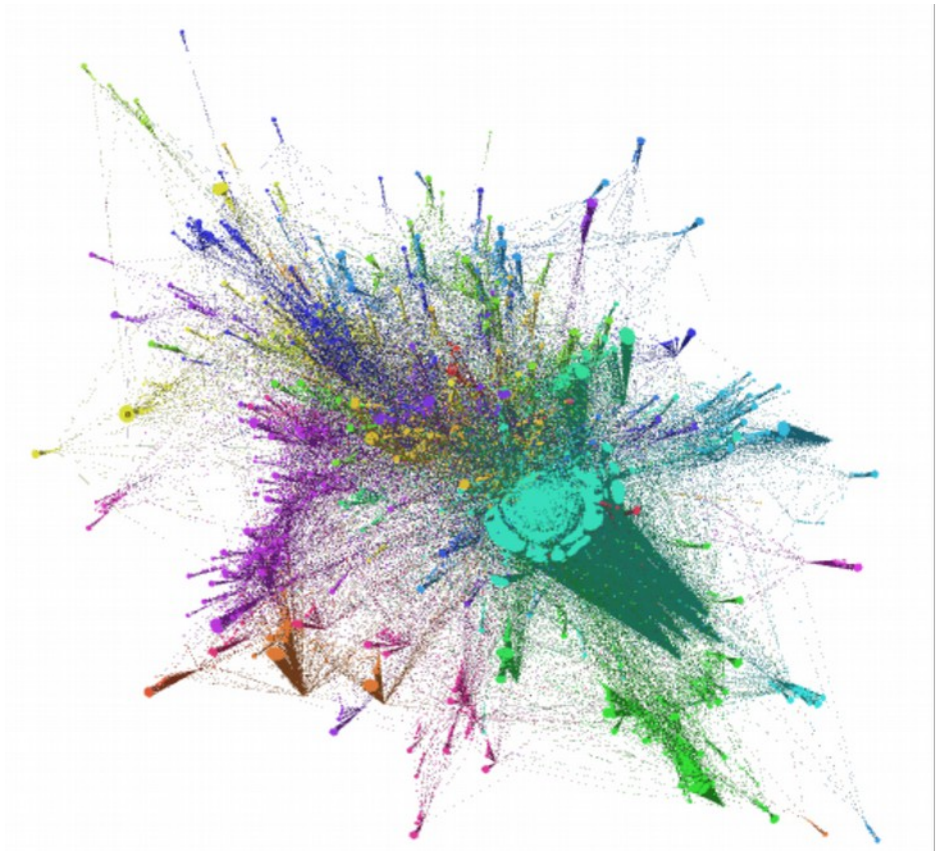

Ilustración 23: Francisco Herrera prof 4

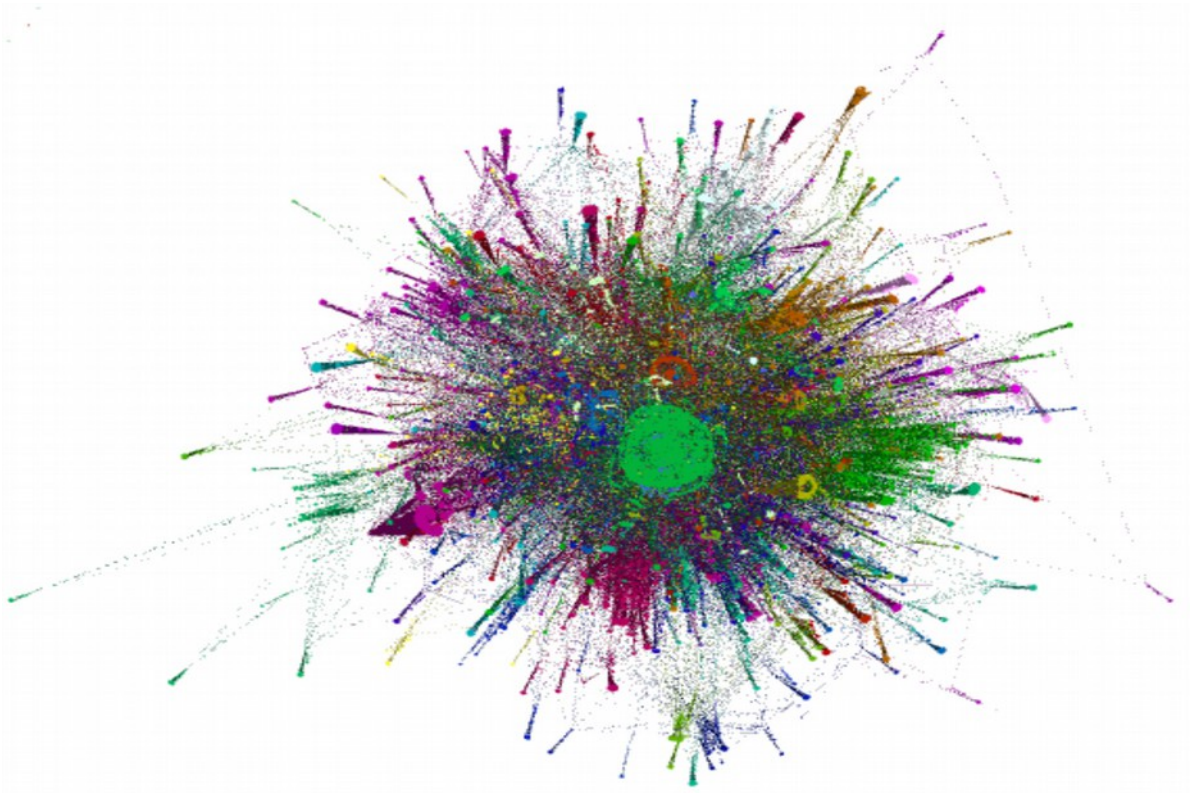

Ilustración 24: Francisco Herrera prof 5

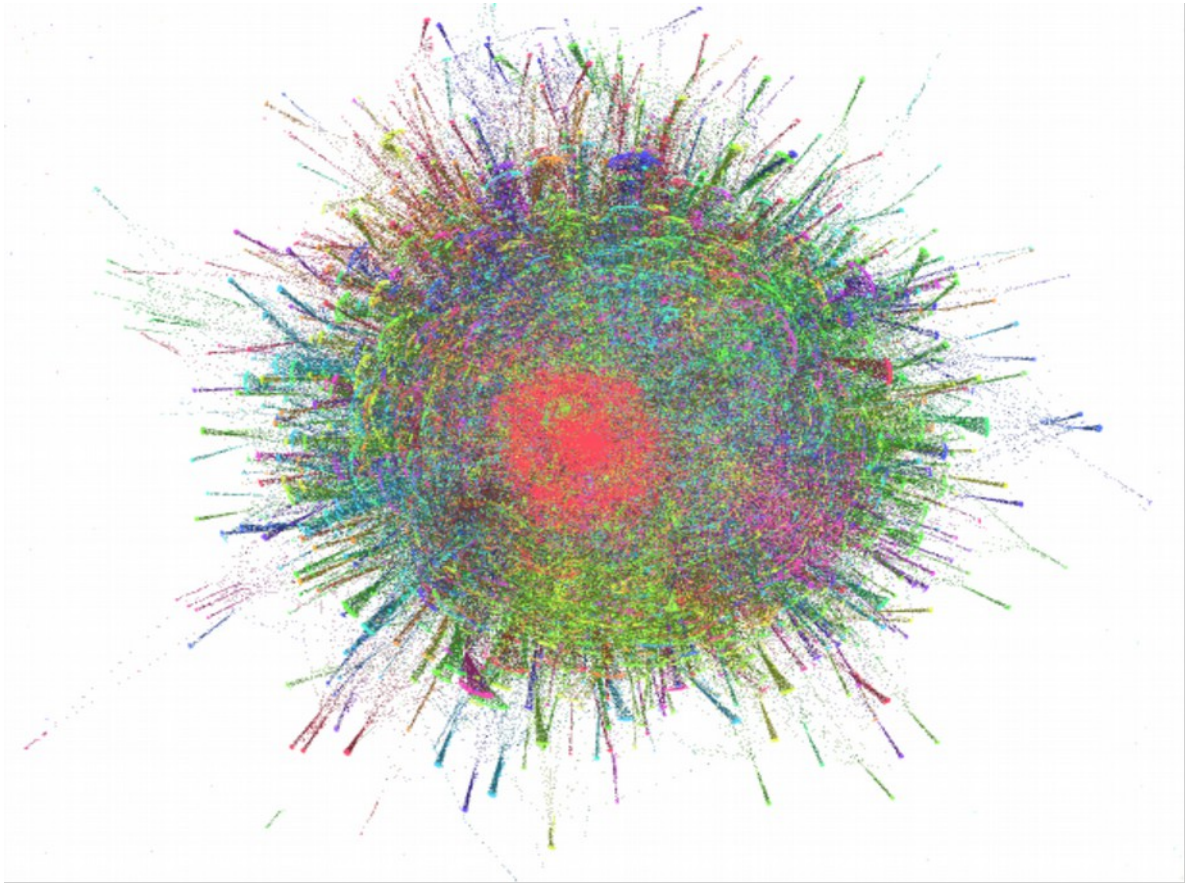

Ilustración 25: Francisco Herrera prof 6

7.4.2 Marco Dorigo

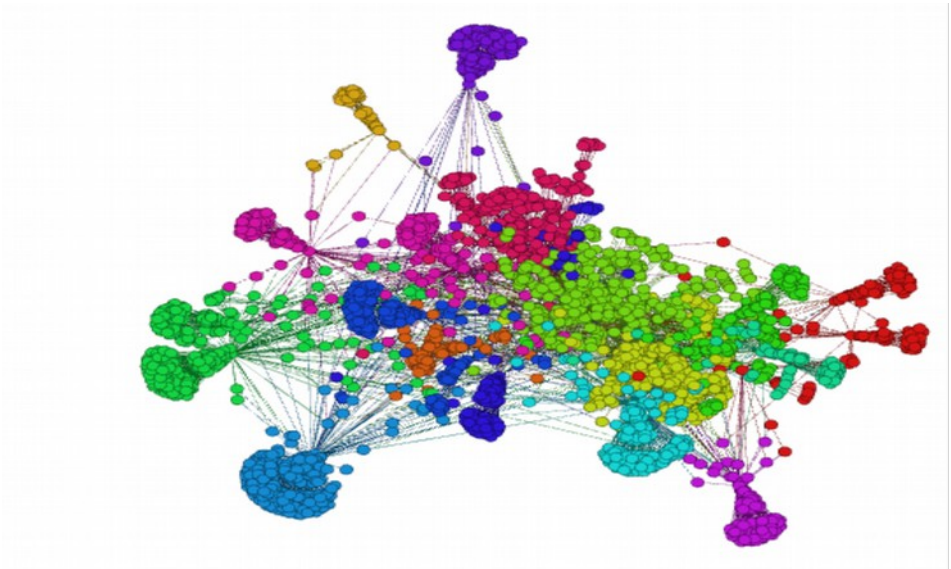

Ilustración 26: Marco Dorigo prof 2

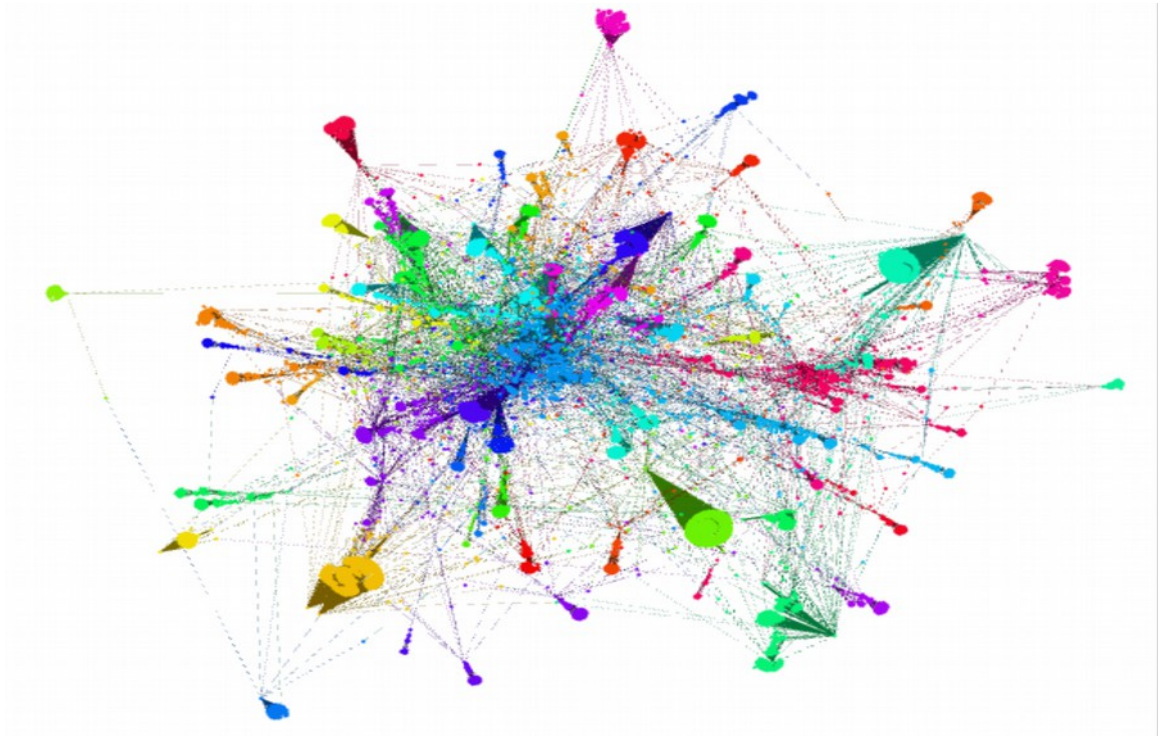

Ilustración 27: Marco Dorigo prof 3

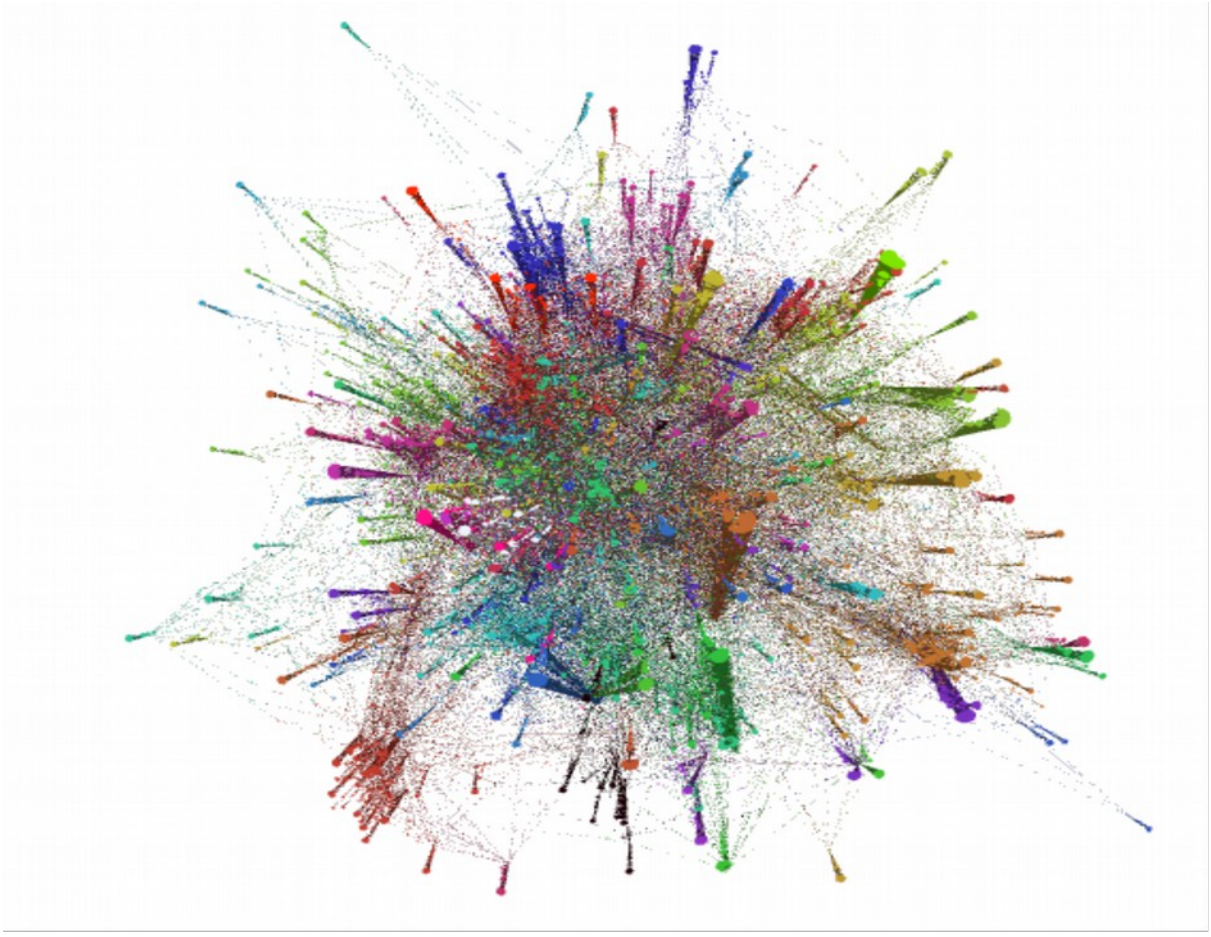

Ilustración 28: Marco Dorigo prof 4

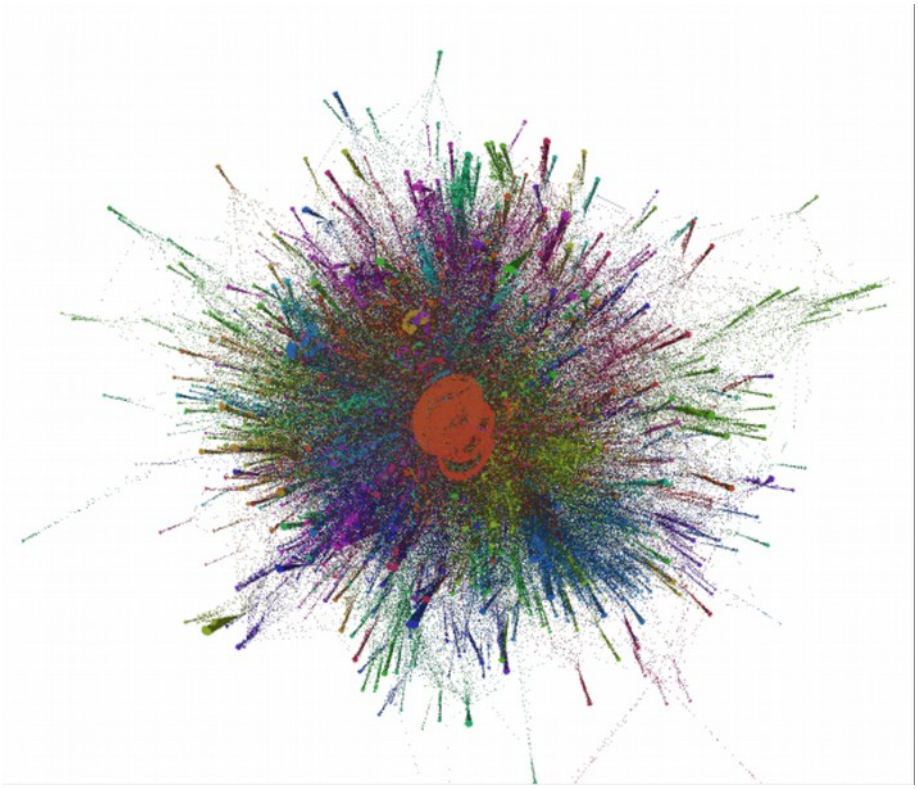

Ilustración 29: Marco Dorigo prof 5

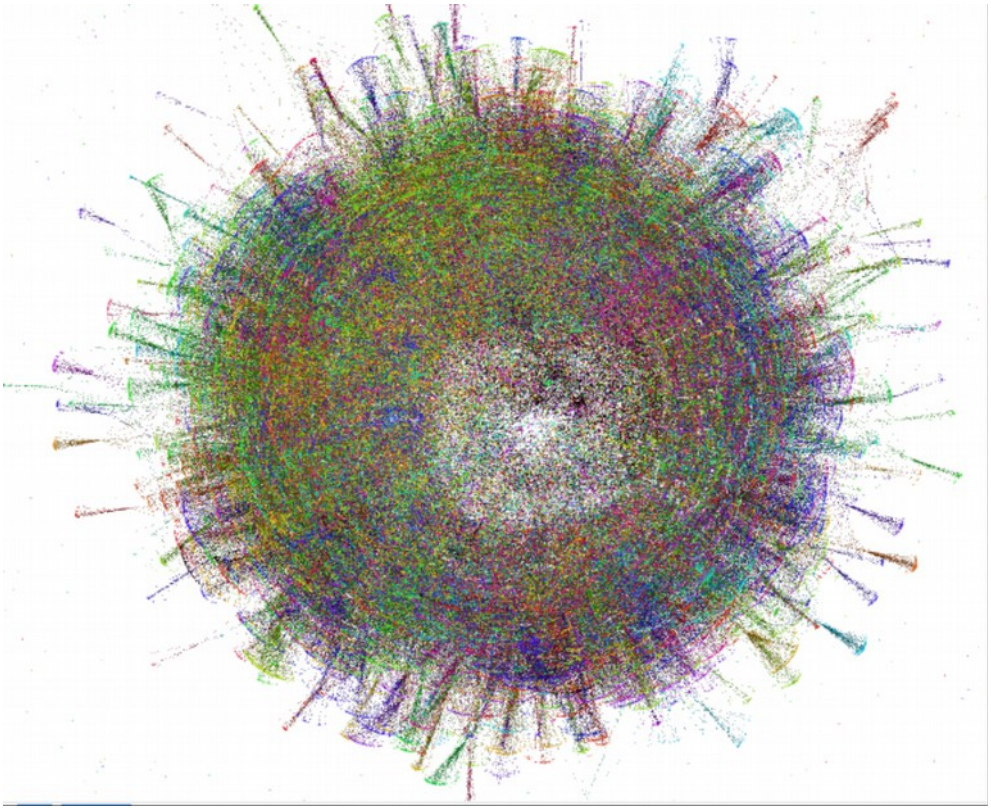

Ilustración 30: Marco Dorigo prof 6

7.4.3 Nick Jennings

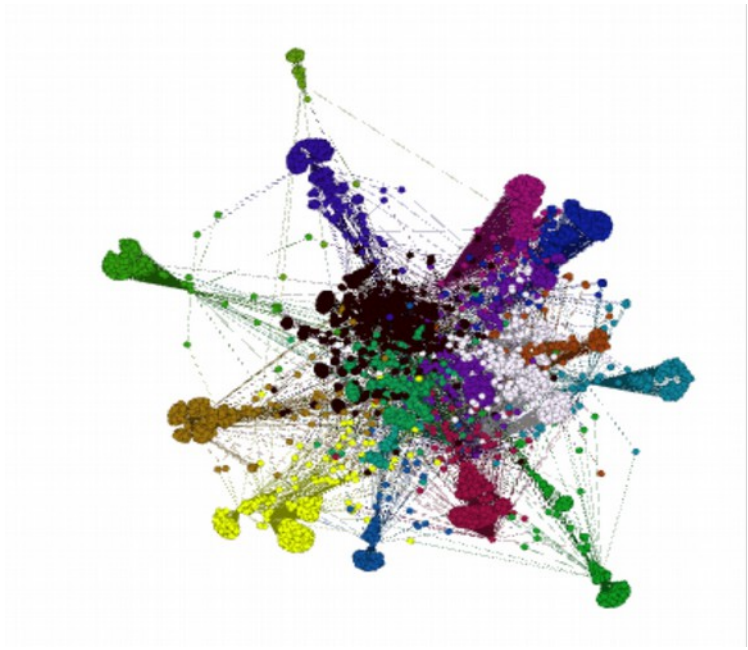

Ilustración 31: Nick Jennings prof 2

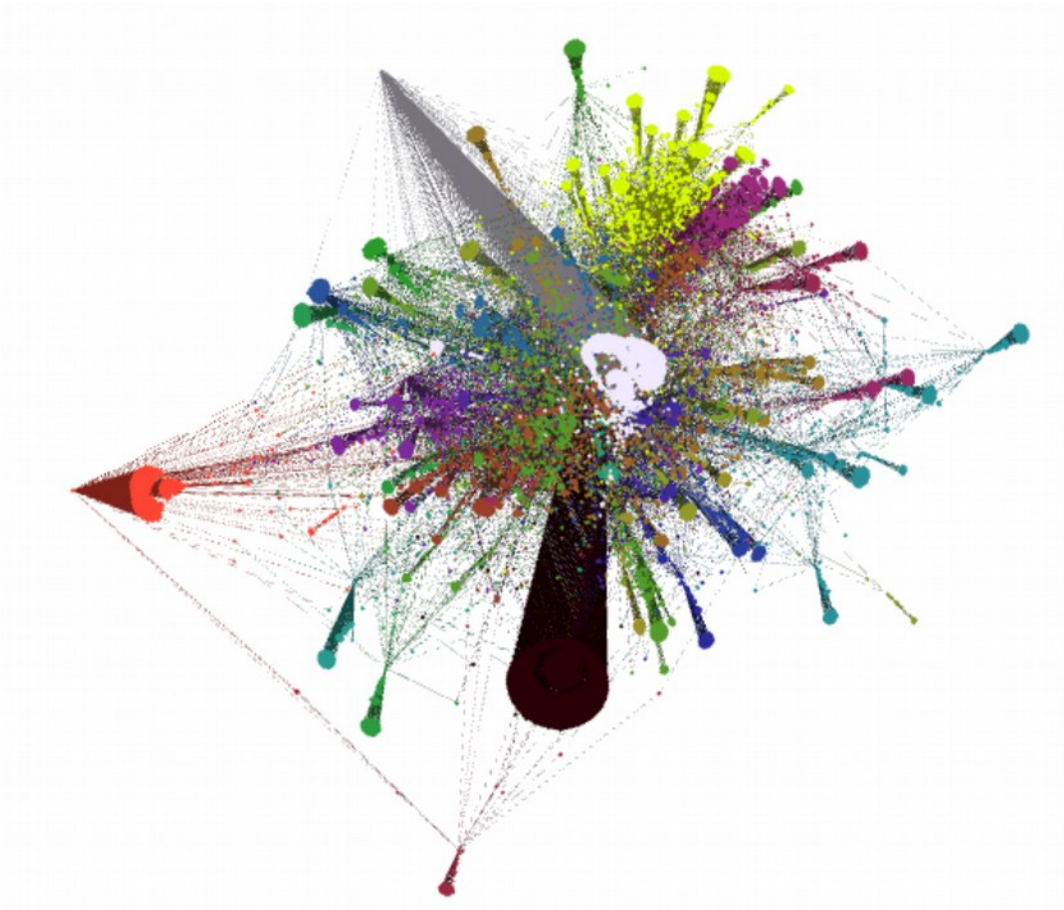

Ilustración 32: Nick Jennings prof 3

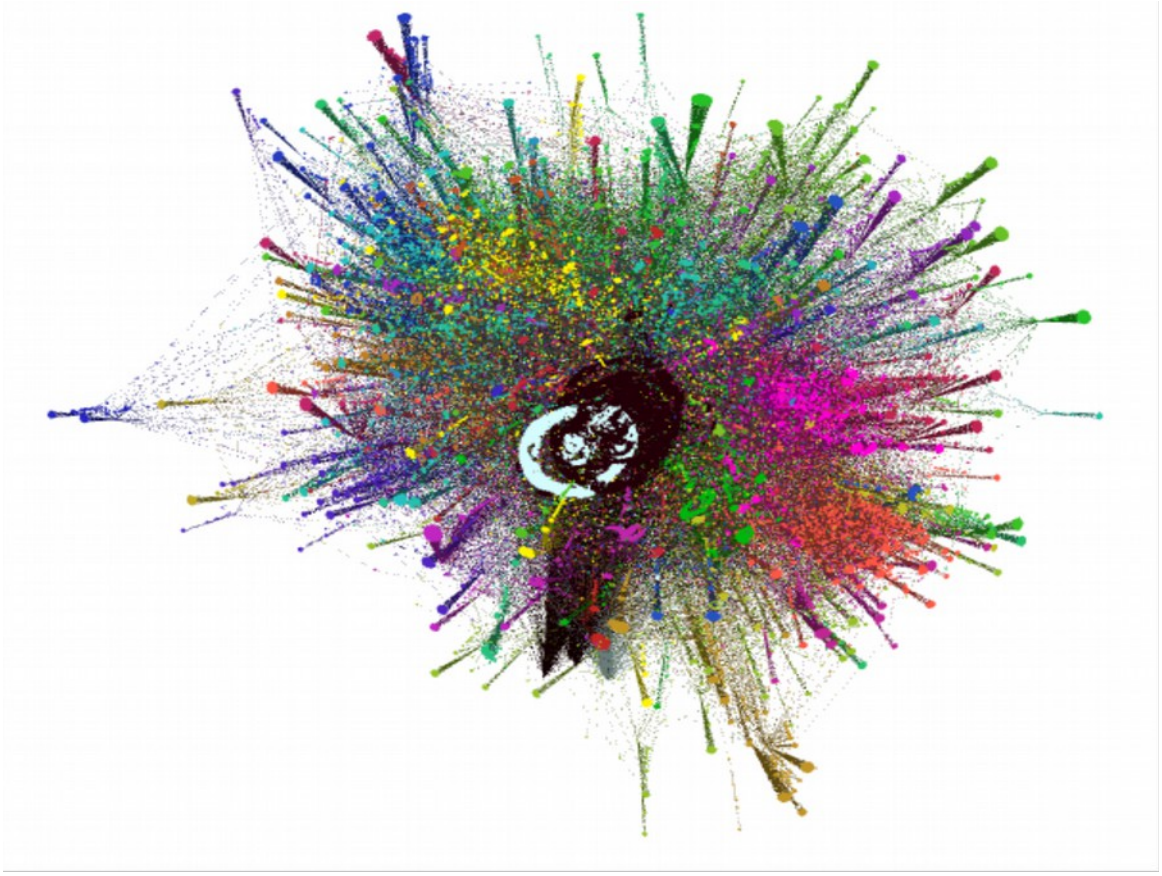

Ilustración 33: Nick Jennings prof 4

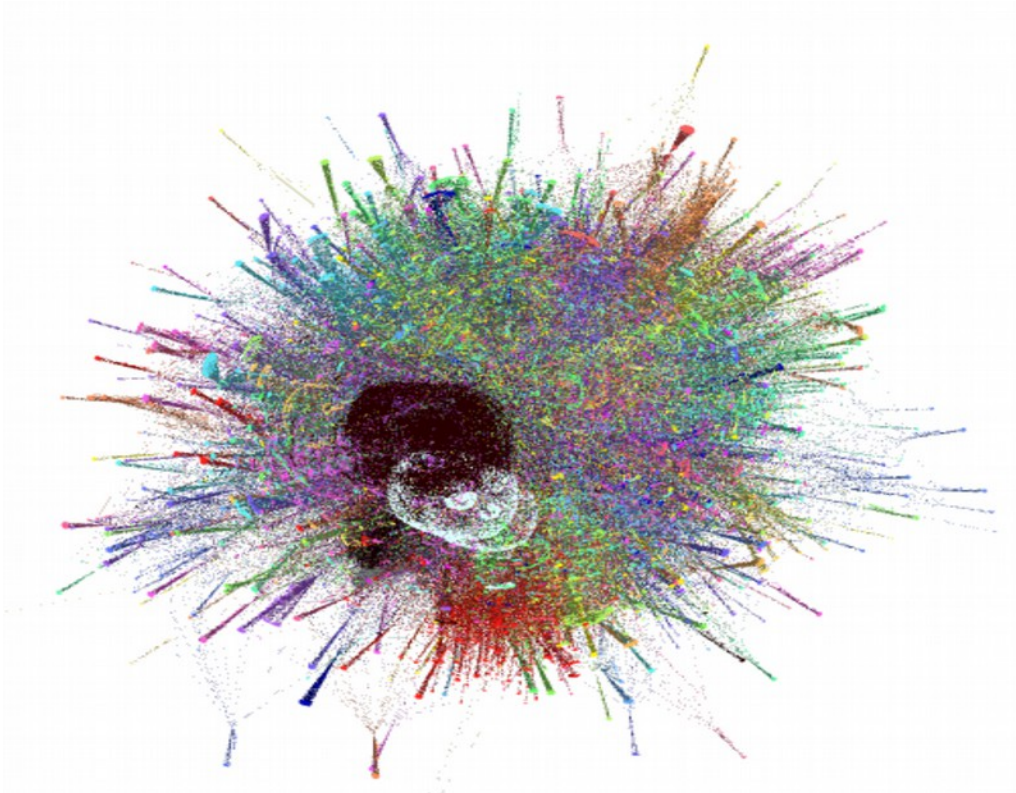

Ilustración 34: Nick Jennings prof 5

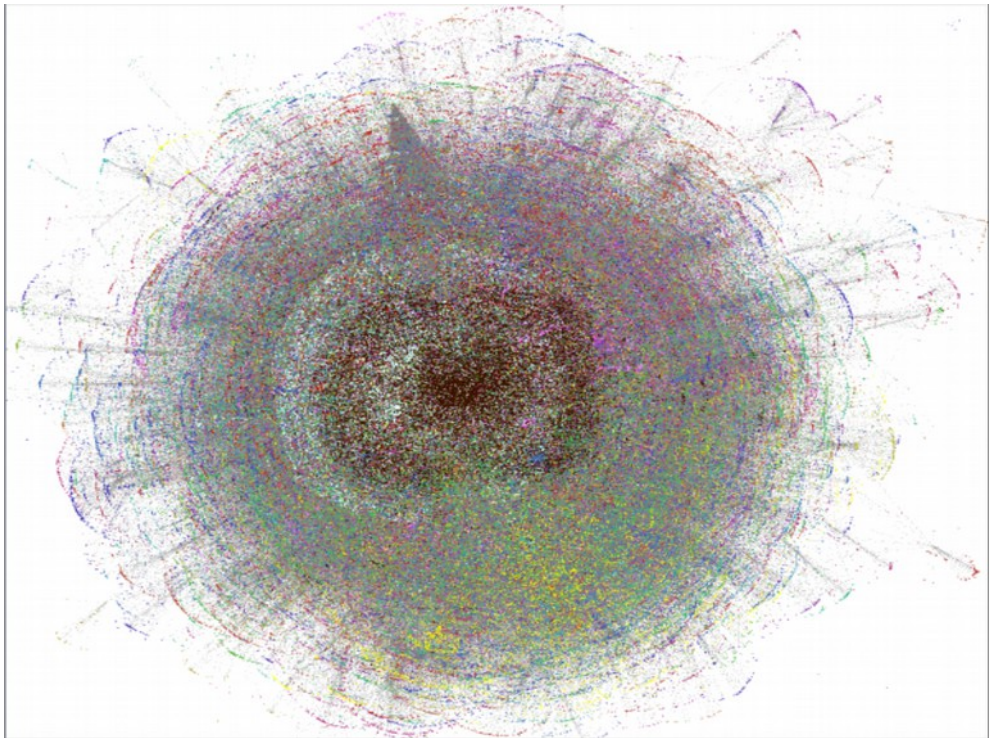

Ilustración 35: Nick Jennings prof 6

7.4.4 Frans Coenen

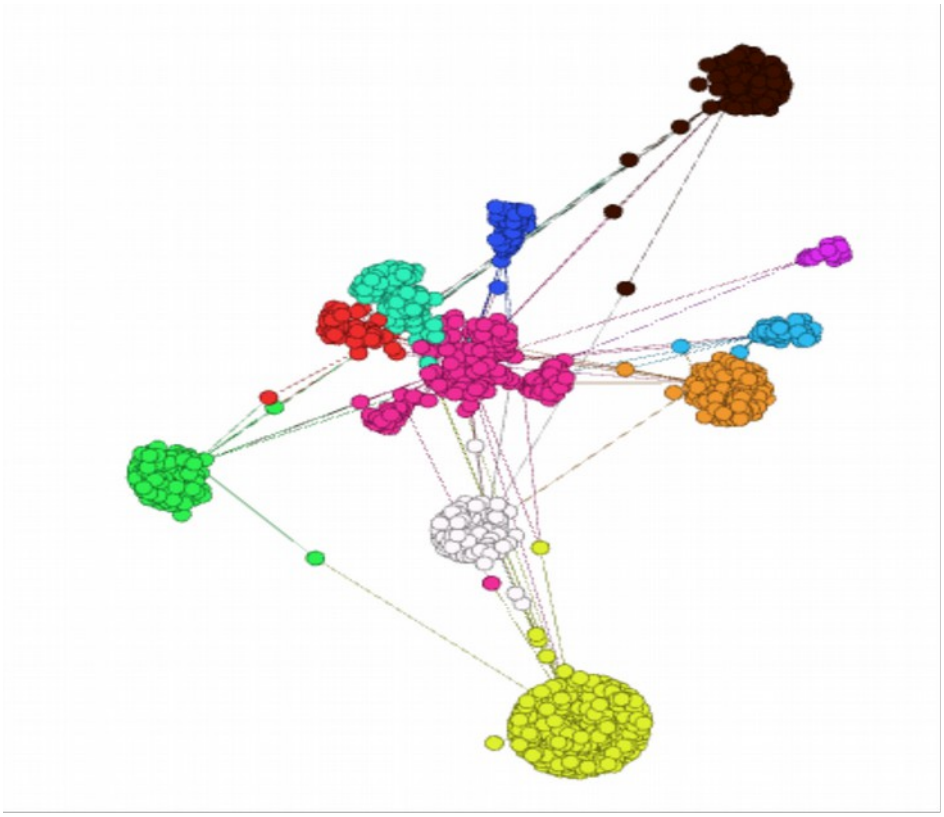

Ilustración 36: Frans Coenen prof 2

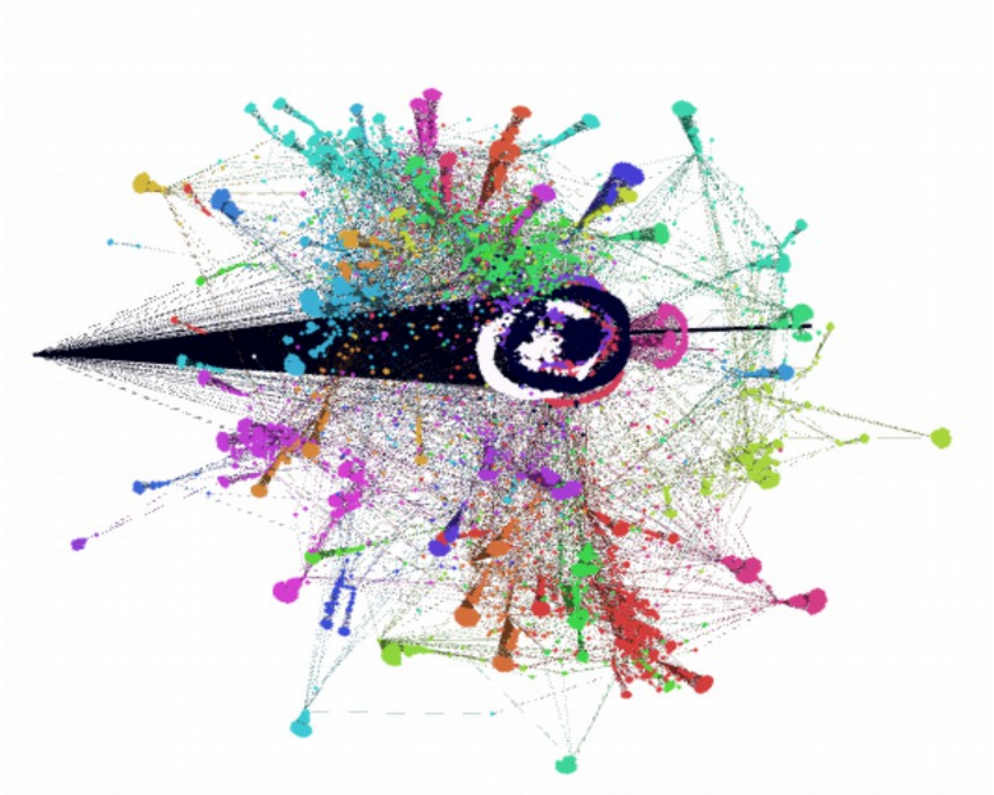

Ilustración 37: Frans Coenen prof 3

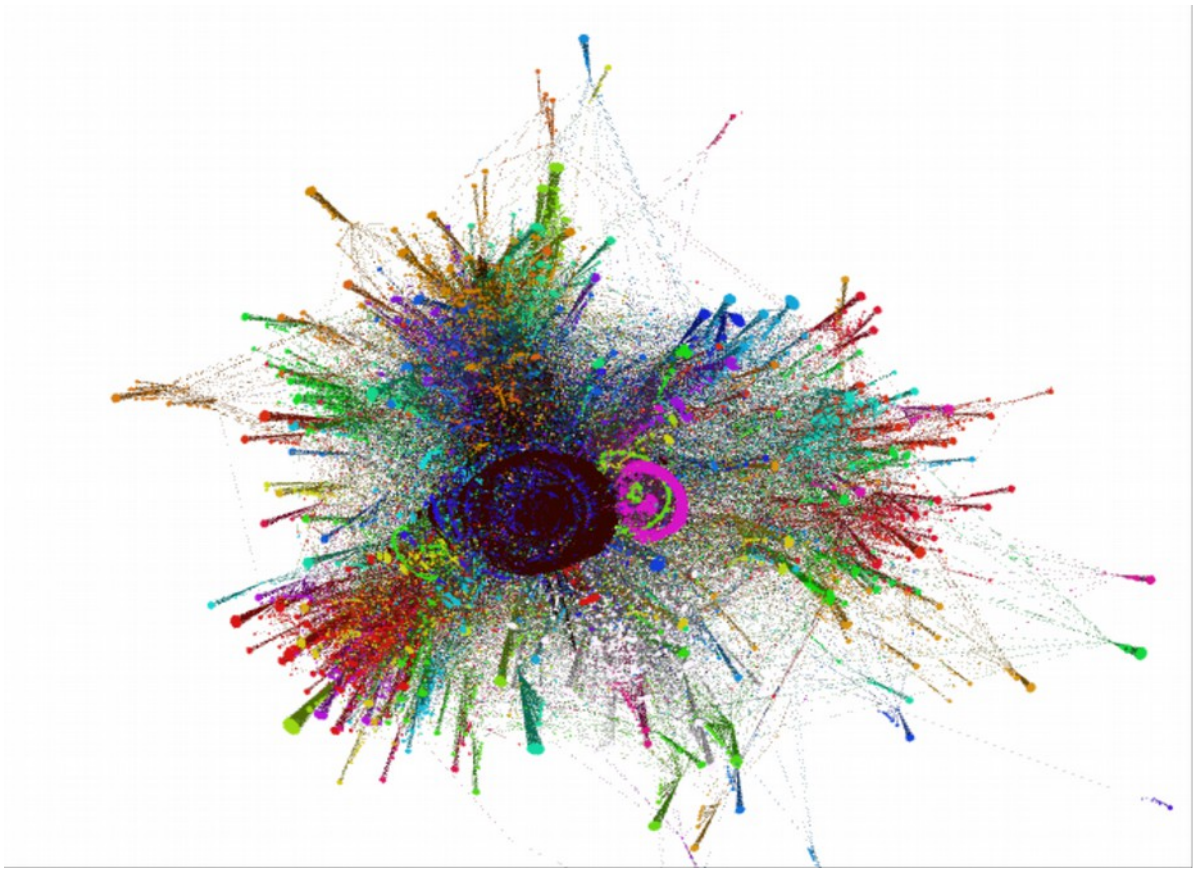

Ilustración 38: Frans Coenen prof 4

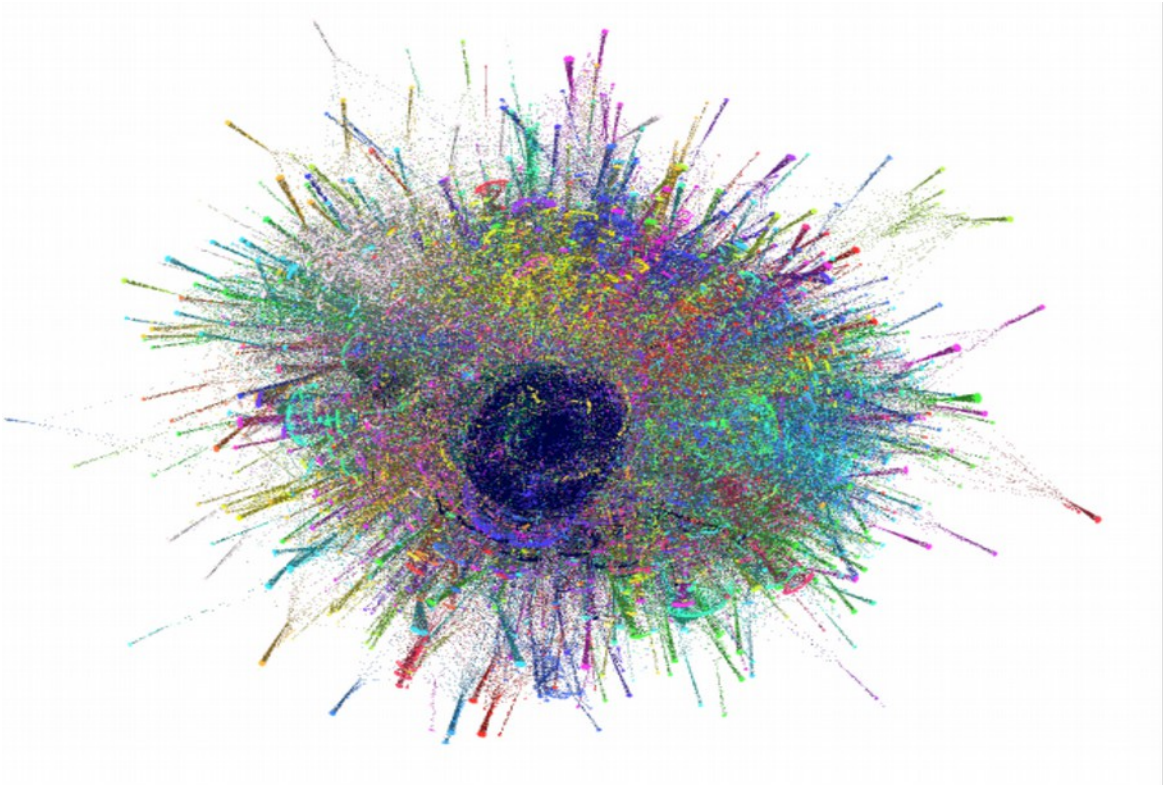

Ilustración 39: Frans Coenen prof 5

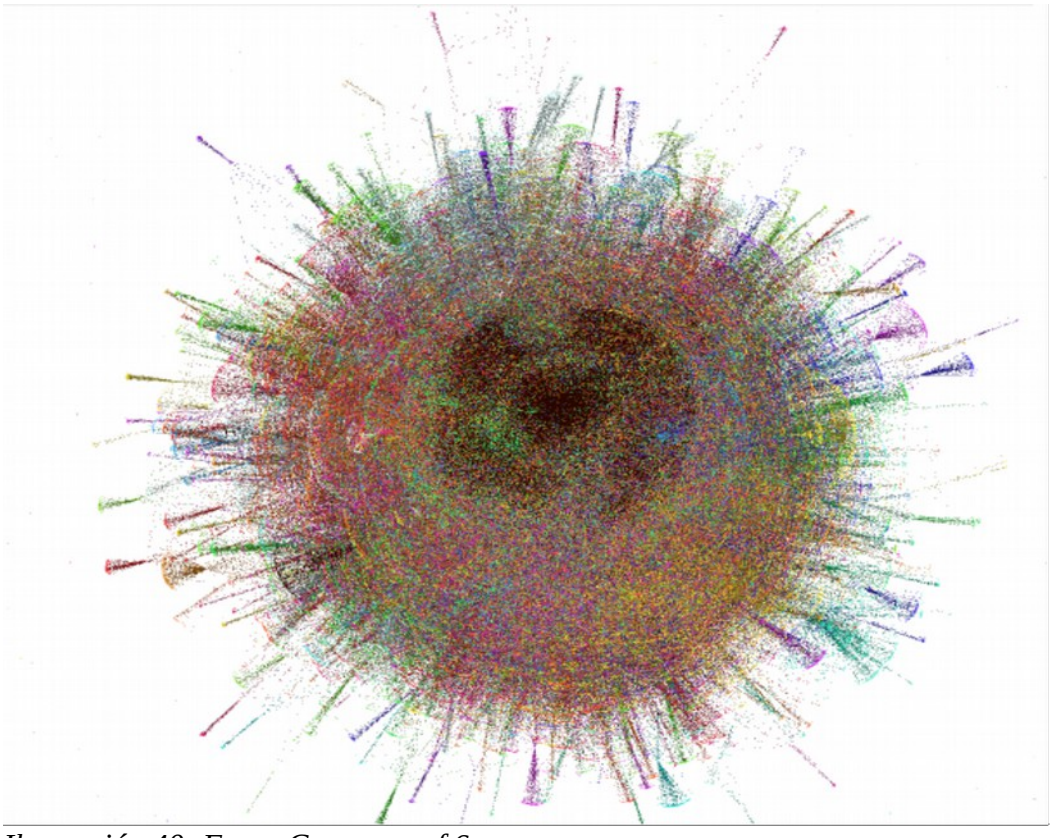

Ilustración 40: Frans Coenen prof 6