UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

ALGORITMO DE COMUNIDADE PARA INFORME DE SOSTENIBILIDAD

Fernando Modamio Miguel
Tutor: David Domínguez

JULIO 2015

RESUMEN

En los últimos años ha tomado gran importancia dentro del ámbito de la empresa el respeto al medio ambiente. Éstas han ido esforzándose cada vez más en adaptarse a los criterios de sostenibilidad recomendados por organizaciones externas.

En 1995 nació GRI (Global Reporting Initiative) para la elaboración de estos exámenes de sostenibilidad a aquellas compañías que quisieran dar a conocer su desempeño económico, ambiental y social.

Se ha requerido la evaluación de los datos publicados por esta organización para cerca de 6000 empresas durante 15 años mediante diversos algoritmos de análisis y reconocimiento de patrones para la creación de un informe de sostenibilidad ambiental.

Esta memoria recoge como se ha desarrollado una aplicación para solicitado anteriormente. La aplicación creará un grafo de empresas y conexiones entre ellas para después analizarlo y estudiar la capacidad de predecir que decisiones tomaron las empresas teniendo en cuenta las políticas realizadas en años anteriores.

Una vez desarrollada se procederá a interpretar los datos obtenidos y a compararlos dependiendo de los diferentes parámetros que se hayan tenido en cuenta a la hora de formar el grafo. También se proporcionará una representación gráfica del grafo obtenido y de las comunidades detectadas dentro de él mediante la herramienta de visualización de grafos Gephi.

Palabras Clave: Sostenibilidad, grafos, nodos, aristas, lista de adyacencia, vecinos, globalización-localizacion, sectorización, reconocimiento de patrones, clusterización.

ABSTRACT

In the last years, respecting the environment has gained a great importance inside company scope. Companies have increased their efforts in order to be complaint with the sustainability criteria recommended by external organizations.

In 1995 the GRI (Global Reporting Initiative) was founded. Its mission consisted in elaborating these sustainability exams for those companies that wanted to make public their economic, environmental and social performance.

It has been required to evaluate the data published by this organization concerning near six thousand companies and during a period of fifteen years. After processing this data with different analysis and pattern identification algorithms, an environmental sustainability report has been created.

This thesis contains the development of an application in charge of processing all the data. The application will create a graph containing the companies and the relationships among them. Afterwards, this graph will be analyzed with the aim of forecasting future decisions made by companies taking into account the policies they followed the previous years.

Once the application has been developed, the resultant information will be studied and the data will be compared depending on the different parameters included in the graph.

Furthermore, a graphical representation of the graph will be displayed. It will contain the neighborhoods detected inside it by the graph visualization tool Gephi.

Index Words: sustainability, graphs, nodes, edges , adjacency list, neighbors, globalization-localization, sectoring, pattern recognition, clustering

Contenido

Ilustraciones

Tablas

GLOSARIO

Sostenibilidad: que es compatible con los recursos que dispone una sociedad. Características del desarrollo que aseguran las necesidades del presente sin comprometer las necesidades del futuro.

GRI: "Global Reporting Initiative" es una organización independiente cuyo fin es el impulso de la elaboración de memorias de sostenibilidad en todo tipo de organizaciones. Fue fundado en 1997 en la ciudad de Boston y cuenta con la participación activa de representantes de organizaciones de derechos humanos, laborales, investigación, medioambientales, corporaciones y otras organizaciones contables.

Distancia de Hamming: método para el cálculo de la distancia entre dos palabras. Es el número de bits en que difieren dos palabras.

Grafo: conjunto de objetos, llamados nodos, unidos por aristas que permiten establecer relaciones entre ellos.

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

Arista: relación entre los nodos de un grafo.

Grafo no dirigido: grafo en el que las aristas son dobles, es decir que van de un nodo A a un nodo B y del mismo nodo B al nodo A.

Matriz de adyacencia: matriz cuadrada cuyas filas y columnas representan los nodos del grafo y el contenido las relaciones entre ellos.

Lista de adyacencia: forma de representar las relaciones entre los nodos de un grafo mediante una lista. Cada nodo dispone de una lista con los demás nodos a los que puede acceder mediante una arista.

Cutoff: distancia de corte para empresas vecinas. Si la distancia entre dos empresas es menor que el cutoff estas serán vecinas, de lo contrario no lo serán.

Patrón: modelo, prototipo o forma regular que sigue un orden. Un patrón se puede entender como un vector de características del tipo x=(x1,x2,...,xn).

Reconocimiento de patrones: área del aprendizaje automático que permite a partir de la extracción de información establecer propiedades entre conjuntos. Para esto, debe adquirir los datos, procesarlos, identificar las características y clasificar.

Red neuronal: sistema de aprendizaje automático que imita las neuronas de los seres vivos de forma artificial para el reconocimiento y clasificación de patrones.

Hub: centro de conexión en un grafo por el que pasan un máximo de nodos. En este proyecto se asocia a las empresas con mayor conexión con otras del mismo sector o región o con empresa fuera de los suyos propios.

Clusterización: parámetro de agrupamiento del grafo. El coeficiente de clusterización cuantifica qué tanto está interconectado con sus vecinos. No busca medir la completitud de éste sino más bien el alcance. Es decir, nodos que no son vecinos

Algoritmo de camino mínimo (Dijkstra): algoritmo para la detección del camino más corto entre dos nodos del grafo, en grafos con aristas de peso no negativo.

BINS: contenedores que agrupan empresas con características similares. Dentro de cada bin se encontraran una serie de empresas que tienen un número de conexiones parecido. El tamaño del bin hará de elemento de corte para la clasificación en torno a este número de conexiones.

1 Introducción

Esta memoria responde a la necesidad de documentar el Trabajo de Fin de Grado titulado "Algoritmo de comunidades para informe de sostenibilidad".

Se quiere comenzar esta introducción dando unas pequeñas nociones de la organización que produjo los datos con los que se ha trabajo.
GRI (Global Reporting Initiative) comenzó su andadura a finales de los años 90 de la mano de la fundación bostoniana CERES1 y de su consultor de cabecera, Allan White, director ejecutivo de Tellus. En aquella época existían varios informes ambientales y se dio la necesidad de estandarizarlos con el propósito de crear un consenso global sobre cómo deberían realizarse estos informes.
Según el sitio web de GRI, en el momento de la elaboración de su primer informe aproximadamente el 24% de las principales compañías ya elaboraban informes medioambientales. El aumento era cada vez mayor ya que en 1996 tan solo lo realizaban el 17% de las grandes compañías y el 13% tres años antes.

Desde la creación de la organización se han ido creando diferentes estándares para la medición de los criterios requeridos. GRI se inició con una primera versión llamada G1, en 2002 sacó a la luz la segunda versión G2, en 2006 fue G3 y actualmente se utiliza la versión G4.
Las posibles calificaciones dentro de estos informes son C, B, A con opción a + si ha utilizado verificación externa.

Una vez explicado brevemente la obtención de los datos se procede a describir el proyecto:

Se ha desarrollado un programa que a partir de un fichero con datos sobre sostenibilidad de 5897 empresas crea un grafo en el que cada nodo será una de estas empresas y cada conexión responderá a la definición de "vecino", empresas cuyas políticas de sostenibilidad han sido similares desde el año 1999 hasta el 2013.

Una vez creado este grafo, analiza sus características e intenta recuperar el patrón para un año de los 15 de los que se tienen datos después de que a éste se le haya introducido datos corruptos.

La documentación aquí presentada intenta explicar cómo funciona la aplicación a la vez que aclara como se ha diseñado y muestra los resultados obtenidos.

1.1 Motivación

La motivación de este proyecto es el análisis de una serie de valores de sostenibilidad para cerca de 6000 empresas en base a diversos algoritmos para la creación de un informe de sostenibilidad.

¿Cómo se están comportando las empresas después de la creación de métricas para la medida de la sostenibilidad e impacto en el medio ambiente?

¿Están tomando medidas unilaterales en torno a estos criterios o por el contrario se están imitando? ¿Ocurren más estás agrupaciones de políticas en algunos continentes que en otros? ¿Y respecto a los sectores de producción?

¿La crisis global hizo que cambiaran las directrices tomadas en años anteriores o no afectó a la certificación GRI de las empresas?

¿Cuáles son las empresas líderes en torno a estos criterios? ¿Se dedican a imitar a las demás o llevan la iniciativa?

¿Se puede recuperar lo que hicieron las empresas para un año determinado en función de sus empresas vecinas? ¿Es más difícil en los años en la que la crisis global azotó la producción? ¿Con la recuperación volvieron a ser predecibles o se distanciaron en la homogeneización de las políticas de medio ambiente?

1.2 Objetivos

Los objetivos fundamentales de este proyecto son responder a la mayor cantidad posible de las preguntas formuladas en el anterior apartado.

Para ello se analizarán los datos ejecutando diversos algoritmos con el fin de obtener un informe de datos que responda a las preguntas antes mencionadas.

Este informe deberá contener las respuestas a las siguientes cuestiones:

Qué empresas son las líderes en políticas de sostenibilidad dentro de cada continente y sector. No se especifica si están copiando entre ellas o si hay una líder que es imitada por el resto.

Recuperación de patrón, ¿hasta cuanto ruido se puede introducir para recuperar buena parte de los datos originales? ¿Qué media de empresas vecinas es necesaria para la correcta recuperación de un patrón? ¿Cuál es el límite de recuperación? ¿Qué años son más fáciles de recuperar y cuales más difíciles? (¿la crisis afectó a la sostenibilidad?)

Por último el programa debe calcular una serie de datos referentes al grafo formado por las empresas y las conexiones entre estas y obtener una serie de archivos de salida con los que poder trabajar en la herramienta “Gephi” para pintar el grafo resultante.

1.3 Estructura de la memoria

En la sección 2 de la memoria se describen las herramientas externas que han sido fundamentales para la realización del proyecto, Gephi y los scripts utilizados para transformar el fichero proporcionado por la universidad para poder trabajar con los datos que almacena.

En la sección 3 se describe tanto el diseño que se ha hecho para la resolución del problema como el tipo de desarrollo que se ha llevado a cabo.

En la sección 4 se detallan los resultados obtenidos de la ejecución del programa y las pruebas que se realizaron para comprobar el correcto funcionamiento del mismo.

En la sección 5 se explican las conclusiones obtenidas de la realización del proyecto además de las posibles mejoras que no se han implementado.

Para finalizar, se detallan las referencias con las que se trabajó para la ejecución del programa y de esta memoria.

2 Estudio del estado del arte o tecnologías a utilizar

2.1 Estudio del arte

Para la realización de este proyecto se ha tenido como mayor referencia el artículo "*Predicting sustainability report scoring sequences using an attractor network"* de Mario Gonzalez, María del Mar Alonso-Almeida, Cassio Avila y David Dominguez (tutor del proyecto).

Este artículo está pendiente de publicación en la revista "Neurocomputing". Gracias a él se han elegido las funcionalidades que finalmente ha tenido la aplicación.
En el trabajo referente al artículo se ha modelado una red de patrones de sostenibilidad de empresas de todo el mundo. Está red, que tiene como característica ser small-world, se compara con una configuración obtenida a partir de Información Mutua entre empresas.
En este trabajo se encontró la idea del estudio de la globalización y sectorización, aunque los resultados aquí obtenidos fueron menos concluyentes.

2.2 Técnicas y herramientas

En esta parte de la memoria se introduce de forma general las técnicas y herramientas usadas para la resolución del problema.
Además del lenguaje de programación C, se han utilizado scripts bash para modificar ligeramente el archivo de entrada que se le pasará al programa, que se entregó en formato .xls y fue convertido en .csv para poder trabajar con él con funciones propias de C.

Una vez obtenidos los resultados, se ha trabajado con la herramienta de visualización de grafos "Gephi" para lograr presentar una representación gráfica de la solución.

2.2.1 CSV y Scripts

El fichero con los datos que se facilitó era un archivo Excel con 5897 filas y 52 columnas. Al necesitar de funciones especiales para trabajar en C con archivos en formato .xls se procedió a su conversión en .csv

CSV es el acrónimo de "Comma Separated Value". En este tipo de documento de texto los datos aparecen separados por comas. Es decir, cada columna del Excel se presenta tras la conversión separada por el carácter ",".

El contratiempo que surgió a partir de esta conversión es que en el fichero original aparecían gran cantidad de comas dentro de una misma columna.

Al leer los datos del fichero se estaban obteniendo datos incorrectos: cada vez que aparecía una coma no deseada, CSV simulaba una nueva columna lo que originaba que las funciones tratamiento de cadenas en C, después de leer del fichero corrupto, que separaban cada línea leída en función del token "," almacenaran información donde no correspondía.
Para solucionar esto se procedió a analizar el fichero en busca de las cadenas donde se repetían los caracteres ",".
La mayoría de las apariciones se daban dentro de las columnas referentes a los años para separar las diferentes calificaciones GRI obtenidas en un mismo año.
En ocasiones este problema también ocurría en los nombres de algunas empresas.
Una vez detectadas las cadenas donde se repetían las comas, se programó un script bash que con ayuda del comando sed sustituía las comas por espacios.

Se incluye una captura de las primeras líneas del script:

```
#!/bin/bash

file1="gri.xls"

sed -i 's/G1,/G1 /g' $file1
sed -i 's/G2,/G2 /g' $file1
sed -i 's/G3,/G3 /g' $file1
sed -i 's/G3.1,/G3.1 /g' $file1
sed -i 's/G4,/G4 /g' $file1

sed -i 's/GRI,/GRI /g' $file1
sed -i 's/GRI-Ref,/GRI-Ref /g' $file1
```

Ilustración 1

Para las empresas que tenían comas en su nombre no se encontró otra solución que cambiarlas a mano, buscando desde Excel las comas que quedaban después de pasar el script.

También hubo otro problema con el archivo de datos. Para algunas empresas el fichero no definía el índice de región. Era un dato fácil de calcular ya que se aporta la región específica a la que pertenece cada empresa, pero se volvían a dar fallos de lectura al moverse las columnas una vez transformado el archivo en CSV.

Para corregirlo se volvió a echar mano del comando sed sustituyendo todas las cadenas "„" por ",*,". De esta forma no hubo más problemas de lectura, aunque hubo que cambiar la forma de la que se leía el dato de índice de región. Ahora en vez de leer dicha columna y guardar el índice, se leía la cadena de texto donde especificaba la región y se guardaba un índice u otro dependiendo de esta.

Una vez hecho este trabajo el fichero ya estaba listo para ser leído.

2.2.2 Gephi

Gephi es una herramienta de código abierto enfocada a análisis y visualización de grafos.
Una vez creado el grafo por la aplicación, se ha procedido a imprimirlo en un formato en que el Gephi pudiera leerlo y representarlo. Una vez cargado en Gephi, todos los nodos aparecen unidos en un mismo punto.

Se deben aplicar un par de funciones para que sea capaz de separar los nodos según proceda. Después de calcular estos parámetros, se ejecuta un algoritmo de separación llamado Force Atlas 2.
Este algoritmo separa los nodos con una función parecida a la gravedad de los planetas.
Dependiendo de los grupos de uniones entre los nodos, estos se van atrayendo y repeliendo hasta que el grafo es visible.
Force Atlas 2 no termina hasta que se le indica, asi que separará y unirá hasta que el usuario decida que es suficiente para una visualización correcta.
Una vez formado el grafo a gusto del usuario, se procede a la detección de comunidades. Para eso se ha utilizado un algoritmo llamado Chinese Whispers (en español esto es el "teléfono escacharrado").

Este algoritmo detecta grupos distinguibles de nodos unidos y pinta cada comunidad de un color diferente. En el apartado de pruebas se mostrarán resultados obtenidos con esta aplicación. El primer anexo es un manual de uso de Gephi para la detección de comunidades en redes complejas.

3 DISEÑO Y DESARROLLO

3.1 DISEÑO

Se ha diseñado la aplicación como un módulo principal que recibe el fichero de entrada, crea los archivos de salida y va llamando a funciones de otros que también escribirán en salida.

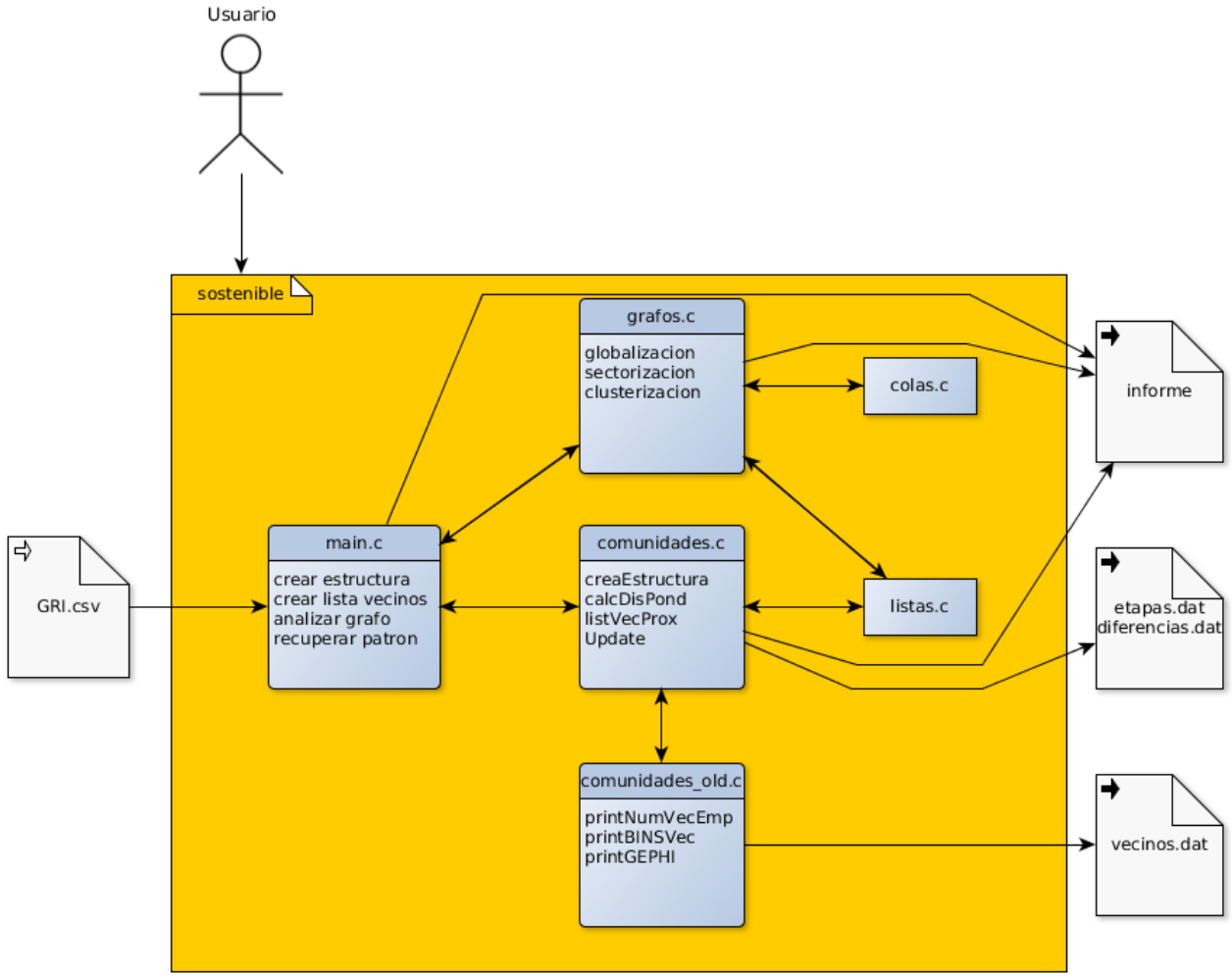


Ilustración 2

3.1.1 Módulos

El proyecto está dividido en diversos módulos escritos en lenguaje C que contendrán las funciones necesarias para la obtención de resultados.

El módulo **main.c** contiene el programa principal que irá llamando a las funciones recogidas en los demás módulos.

El módulo **comunidades.c** contiene las funciones referentes a la creación de la estructura, del grafo y las relaciones de éste. También contiene las funciones para la recuperación de patrones mediante una red neuronal.

El módulo **grafos.c** engloba las funciones referidas al análisis del grafo, como cálculo del grado máximo, sectorización y globalización, clusterización y camino medio del grafo.

En **listas.c** están las funciones que implementan la estructura de lista enlazada. Estas funciones permiten crear listas, introducir nodos en ellas, borrar nodos de ellas, buscar dentro de estas y eliminarlas.

En el módulo **colas.c** se han implementado colas para la ejecución del algoritmo de Dijkstra para el cálculo del camino mínimo entre nodos.

Por último, en **comunidades_old.c** tendremos funciones que acabaron por ser reemplazadas por otras mejores. Se han guardado para tener la contingencia de volver a un punto anterior de desarrollo y retomar otras posibilidades desde aquí.

Además, cada módulo lleva asociado su correspondiente .h con los prototipos de las funciones que utiliza. También se añade un módulo llamado **constantes.h** que contiene las definiciones de constantes y factores usados en diversas partes del programa. Estas son número de patrones (15), ruido (15), factor de ajuste en el cutoff (cuto=-0.6), factor de disparo de la red neuronal (cc=0) e infinito (99999).

3.1.2 Traza de la aplicación

En esta sección se explicará el orden en el que el programa irá calculando los diferentes resultados.

3.1.2.1 Parámetros de entrada

Los parámetros que pide el programa por línea de comandos son:

1- nombre del fichero de entrada: fichero en formato csv que contiene los datos del GRI para 5897 empresas suministrado por la universidad.

2- nombre del fichero de salida: fichero en el que se recogerán los principales resultados de la ejecución del programa tales como distancia media, desviación típica, cutoff, media de vecinos, resultados de la globalización y sectorización, recuperación del patrón y datos sobre el análisis del grafo.

3- año a recuperar: año dentro del patrón que se desea recuperar mediante la red neuronal.

3.1.2.2 Traza

El programa principal primero comprueba la rectitud de los parámetros de entrada y abre el fichero con los datos de las empresas, los lee y crea la estructura con la que se trabajará.

Esta estructura es un doble puntero al tipo de dato empresa. Cada fila del array tendrá una empresa asociada que contendrá su identificador, su índice de región y de sector, una lista enlazada de vecinos y un entero que guarda el tamaño de esta.
Una vez creada esta estructura se procede al cálculo de la tasa de 1s por año (más adelante se explicará para que sirve este vector), el cutoff de distancia y la lista de vecinos de cada empresa.

Ahora se procede al análisis de algunas características del grafo que se ha creado en la estructura.
En primer lugar se calcula el grado máximo, es decir, la empresa con mayor número de conexiones. Como van a haber varias empresas con el mismo número de conexiones que coinciden con el mayor grado (se suelen formar grandes comunidades de empresas) únicamente se muestra el valor de éste. Luego se calcula la clusterización o coeficiente de agrupamiento y después de esto se calcula el camino medio entre las empresas.
Con esto se finaliza el apartado de análisis del grafo.

A continuación se calculan los valores de globalización y sectorización. Una vez terminados, se procederá a introducir ruido en el patrón del año seleccionado como parámetro de entrada. Este ruido es una constante (igualada a 20%) y si se desea modificar se debe acceder al módulo **constantes.h.**

Después de modificar el patrón se introduce este resultado en la función **Update** que intenta recuperar el patrón original mediante la función de red neuronal. Esta operación puede hacer hasta 10000 iteraciones y es una de las más complejas en tiempo de la aplicación.

Por último se libera la memoria y se cierra el fichero de salida.

3.1.3 Ficheros de entrada

Para la ejecución de la aplicación es necesario el paso a ésta de un fichero de entrada que contenga los datos de sostenibilidad de las empresas con los que se van a trabajar.

Se ha proporcionado por la universidad un fichero Excel con datos de 5897 empresas de todo el mundo en torno a las políticas de sostenibilidad que han llevado a cabo en los últimos años.

Este fichero ha sido modificado y transformado en el formato CSV para poder trabajar con él mediante funciones C.

3.1.4 Ficheros de salida

El programa pide por línea de comandos el nombre del archivo de salida donde se visualizarán los resultados.
En este fichero se mostrarán datos relativos a la distancia de corte cutoff, la medía de vecinos (grado medio del grafo), las estadísticas de globalización y sectorización, la recuperación del patrón original y parámetros del grafo analizado.

Además de este fichero de información general se generan dos ficheros de salida que contienen información sobre cómo se ha recuperado el patrón mediante la red neuronal. Estos son:

diferencias.dat: en este fichero se muestra la diferencia entre el patrón recuperado en cada iteración y el vector original. Al pintarlo en una gráfica se dará una idea del porcentaje de recuperación que se consigue a lo largo de la ejecución del bucle.

etapas.dat: en este fichero se muestra la diferencia entre las dos últimas iteraciones del bucle. Dará una aproximación de cómo está funcionando la red neuronal. Acabará en 0 si consigue terminar la recuperación (dos etapas consecutivas en las que no hay cambios) o en la línea 10000 después de ese número de iteraciones sin llegar al resultado requerido.

Aparte de estos ficheros que siempre se generan, podremos obtener los siguientes ficheros de salida si invocamos a las respectivas funciones:

vecinos.dat: este fichero representa un histograma para el número de empresas que tienen X vecinos. Cada línea representa un número de vecinos y tendrá asociado el número de empresas que tienen tantos como indica la línea.

bins.dat: este fichero es similar al anterior, pero muestra el histograma con las empresas combinadas en contenedores (bins). Se agruparán tantas empresas como indique el tamaño del bin, que podrá ser modificado en el código.

Por ejemplo, con un tamaño de bin de 60, el fichero mostrará el número de empresas que tienen de 1 a 60 vecinas, a continuación se mostrará el número de empresas que tienen de 61 a 120 vecinas...

numvecinos.dat: este fichero representa el número de vecinos que tiene cada empresa. Cada línea representa a una empresa (la línea 1 tiene a la empresa con identificador 0 asociada) y tendrá adjunto el número de vecinos de esa empresa.

gephi.csv: este fichero imprime el grafo en el formato requerido para ser leído por la aplicación gephi que es el siguiente: una línea por conexión, siendo la primera columna el identificador del nodo de partida de la arista y la segunda el identificador del nodo de llegada. Al haber una gran cantidad de conexiones en un grafo con 5897 nodos el archivo resultante ocupará cerca de 7MB en disco.

gephinames.csv: es similar al anterior, solo que añade el nombre de los nodos en vez del identificador de cada empresa. Ocupa mucho más y no muestra información de interés en comparación con el otro (en el grafo aparecen los nombres de las empresas pero no influye en el resultado). Se ha utilizado **gephi.csv** para generar los grafos en la aplicación de visualización de grafos "gephi".

Además se podrá obtener por pantalla o fichero la lista de vecinos de cada empresa, siendo esto no recomendable ya que aporta gran cantidad de información desdeñable. Se usó en pruebas para ver la rectitud de las conexiones.

3.2 DESARROLLO

Para la realización del proyecto se ha utilizado un modelo evolutivo. Al comenzar el desarrollo se tenía una idea de las funcionalidades que iba a aportar el proyecto, pero no se tenía claro que métodos iban a utilizar éstas para alcanzar los resultados esperados. A medida que avanzaba el proyecto, nuevas funcionalidades eran desarrolladas según las recomendaciones del tutor.

De ese modo, se comenzó a trabajar desarrollando funciones que más adelante serían rediseñadas de tal forma que mejorasen las fases anteriores propias.

Cada vez que se desechaba una función por haber sido repuesta por otra se movía al módulo **comunidades_old** para tener la opción de recuperarla y mejorarla nuevamente desde un punto anterior. También se incluyeron funciones que fueron desarrolladas para hacer comprobaciones (imprimir la estructura, nombre de empresas, comprobar vecinos...) y otras que aunque se llaman en la ejecución final, no tenían cabida en los módulos **comunidades.c** y **grafos.c** ya que eran funciones de apoyo que calculaban pequeños parámetros o imprimían histogramas.

3.2.1 Lectura del fichero de datos y creación de la estructura

En un principio se contó con un archivo Excel con el contenido los estándares de sostenibilidad GRI de 5897 empresas de todo el mundo, una gran cantidad de información que debía ser tratada para poder trabajar con ella en los diferentes algoritmos que se iban a utilizar para la interpretación de los futuros resultados.

Los campos de este fichero eran: nombre de la organización, país, región, índice de región, sector, macro sector, índice de sector y los datos del GRI en tres formatos diferentes.
Primeramente se daban los datos desglosados por año, desde 1999 hasta 2013, es decir, una columna por cada año con todos los diferentes criterios y niveles que esa empresa había cumplido en ese año. A continuación se tenía la misma información pero binarizada, es decir, si para un año

se había cumplido algún criterio GRI, en esa columna se daba un 1 y por el contrario, si no se había llegado a ningún estándar de sostenibilidad se daba un valor 0. Por último, se encontraba esta información binarizada pero dando peso al número de estándares diferentes cumplidos en un vector de tamaño 20.

El primer problema que se tuvo fue trabajar con este fichero. Para poder leer los datos de éste, se tradujo a .csv. Este formato transforma un Excel a texto separando las columnas en "," o en otro carácter seleccionado (";" o "/").
Esta solución dio una corrupción de los datos al aparecer en el fichero tanto comas como los otros caracteres mencionados. Al leer de él, si por ejemplo el nombre de una organización contenía una coma, se desplazaban los datos reales una columna. De este modo, para dicha empresa, los datos del GRI del año 1999 aparecían en el año 2000, y para el año 1999 teníamos valores que correspondían al índice de sector de la empresa.
La complejidad de este problema es que los datos estaban corruptos pero parecían legítimos. Se avanzó en el desarrollo teniendo unos datos falsos, aunque cuando se empezaron a obtener los primeros resultados, se revisó la lectura de los datos y se corrigió el problema.
La forma de corregir esto fue un script que fuera cambiando las comas del fichero por espacios en blanco.
También se dio el problema de que para muchas empresas el índice de región no aparecía en el fichero. El script corrigió esto buscando dos comas seguidas ",," y sustituyendo esta cadena por ",*,".
Al tener esta corrupción en los datos referidos al índice de región, el programa desarrollado lee la región a la que pertenece la empresa y asigna un entero del 1 al 7 según el valor leído:
1 para África.
2 para Asia.
3 para Europa.
4 para Norte América.
5 para Latino América y Caribe.
6 para Oceanía.
7 para empresas en las que no aparece la región a la que pertenecen.

Sin tener en cuenta el fallo de las comas, en un primer momento se optó por almacenar toda la información contenida en el fichero. Para ahorrar en la reserva de memoria se eligió por no guardar los nombres de las empresas ni los países de cada una de estas ya que no se estaban utilizando en ninguna parte del programa.
Se estima que se ahorró 250KB al no guardar estos datos (media de 50 caracteres por 5897 empresas).
Así que se optó por almacenar únicamente un identificador entero para cada empresa, su índice de región y de sector y su vector de sostenibilidad.
Más adelante, con el avance del programa, se dio la necesidad de conocer los nombres de las empresas, pero para no cambiar la forma de la que se leían los datos se desarrolló una función que leyera de fichero los campos de nombre y país de algunas empresas relevantes. Conocer el nombre de una determinada empresa era importante, pero no hacía falta almacenar en memoria todos los nombres teniendo a disposición el fichero que los contenía.

Para el vector de sostenibilidad se optó por guardar la primera configuración de sostenibilidad que aparece en el fichero, dejando para un futuro la opción de trabajar con los estándares desglosados, es decir, como aparece en el fichero por última vez, como un vector de tamaño 20 para cada año. Realmente esta última configuración es la que aporta mayor información, pero se comenzó por algo más simple dejando la posibilidad de si fuera necesario ser adaptado para trabajar con esos datos.
Para cada empresa se reservó un vector de enteros de tamaño 15 para guardar la información de sostenibilidad de cada año binarizada.

Si en el año 2006, la empresa “Hirsch Servo” había cumplido los estándares del GRI G2 y CI en la posición 7 de su vector de sostenibilidad tendría un 1. Como antes de este año no había cumplido ningún criterio de sostenibilidad GRI, las posiciones de 0 a 6 de su vector tendrían valor -1.

Después de todo este trabajo ya se tenían unos datos con los que empezar a trabajar. La estructura para una empresa quedó de esta forma:

```
typedef struct Empresa{

int region_id;
int sector_id;
int sust[15];              //patron sostenibilidad
int id;
pNodo listaVecinos;
int num_vecinos;

}empresa;
```

region_id responde al índice de región
sector_id almacena el índice de sector
sust es el vector de sostenibilidad
id el identificador de la empresa.
Añadiéndose más adelante los dos últimos campos, que serán explicados más adelante, en el momento en que proceda.

Las empresas se irán guardando en un array para el posterior acceso a los datos. Una vez leído todo el fichero, dado que para cada línea de este se tiene una empresa se obtendrá un array de tipo “empresa” con 5897 entradas. El identificador que se le asigna a cada empresa dependerá del orden en el que sea leída del fichero. Conviene aclarar que en el fichero proporcionado las empresas aparecen ordenadas en primer lugar por sector y alfabéticamente dentro de este. En primer lugar aparecen las empresas del sector energético siendo “A2A spa” la primera empresa del fichero.
Además, se guardará en una variable el número de empresas leídas. Este valor será fundamental ya que será pasado además del array de empresas a todas las funciones del programa para poder recorrer éste y operar con los datos.

3.2.2 Calculo de distancias y lista de vecinos

Una vez obtenidos los datos se procede a crear un grafo de sostenibilidad en el que los nodos serán las empresas y las aristas serán las conexiones entre éstas.

Para que haya una conexión entre dos empresas, el vector de sostenibilidad de ambas debe ser semejante hasta un determinado límite. El primer problema es determinar este límite de corte. Cuando haya una conexión entre dos empresas diremos que estas son “vecinas”.

Como se ha referido anteriormente, el proyecto llevó un desarrollo evolutivo. Para el cálculo de la distancia de corte entre vecinas primero se desarrollaron unos métodos que más adelante fueron perfeccionados para obtener soluciones más precisas que dieran lugar a resultados más satisfactorios.

Primera aproximación

Para empezar, se comenzó midiendo la similitud entre dos empresas en torno a la sostenibilidad usando **distancia de Hamming.** Este algoritmo resta el valor de las mismas posiciones de dos vectores de dos empresas obteniendo así las diferencias entre estas. Como tenemos vectores binarios la máxima diferencia para un año es 1 en el caso de que una empresa no haya conseguido obtener ningún mínimo de GRI y otra haya obtenido un criterio como mínimo.
Así la mayor diferencia entre dos empresas es de valor 15. Si dos empresas han hecho políticas iguales (similares realmente, ya que no importa si una empresa obtuvo calificación A+ y otra C-) de sostenibilidad en los mismos años, la diferencia entre ellas será 0.

A continuación se calcula la distancia de Hamming para cada par de empresas y se calcula la distancia media y la desviación típica.

La distancia de corte se ha denominado cutoff y responde a la siguiente formula:

*cutoff=distancia media – (desviación típica * factor de ajuste)*

El factor de ajuste responde a la necesidad de la creación de grafos con más o menos conexiones. Modificando este parámetro obtendremos diferentes resultados en los siguientes apartados de análisis.

Segunda aproximación y aceptación de métrica

Después de la obtención de los resultados del anterior punto, se comprobó que éstos no daban toda la información que se podía desear.
En los primeros años la mayoría de las empresas no obtuvieron el mínimo valor necesario para la obtención de un certificado GRI. Con esta métrica, en las primeras medidas se tienen muchos valores de distancia iguales.
Es mucho más importante que dos empresas tuvieran las mismas políticas de sostenibilidad una vez que el criterio GRI estuviera más aceptado en conjunto que en los primeros años.
De este modo, únicamente la distancia de Hamming no impone la suficiente medida para determinar si dos empresas son "vecinas sostenibles".

Para solucionar esto se implementó una "distancia de Hamming ponderada". En primer lugar, se calcula la "tasa de 1s" para cada año del patrón de sostenibilidad.
Esta "tasa de 1s" responde a la suma total del número de unos para cada año dividido entre el total de empresas.
Una vez calculado este vector (cada posición es la tasa de 1s de cada año, por lo tanto tendrá tamaño 15) se procede al cálculo de las distancias con esta métrica.

Para calcular la distancia ponderada de dos empresas se mira cada año del patrón de sostenibilidad de ambas.
Si para un año las dos empresas hicieron políticas diferentes, esto es, si una obtuvo un GRI y la otra no, se suma a la distancia 2 dividido entre la raíz cuadrada de la tasa de 1 de ese año.
Si las dos empresas no obtuvieron GRI, se añade a la distancia 1 dividido entre la raíz cuadrada de la tasa de 1 de ese año.
Si las dos empresas obtuvieron GRI no se añade nada a la distancia.

El cutoff (la distancia de corte para saber si dos empresas son vecinas) se calcula de la misma manera, la media menos la desviación típica multiplicada por el factor de ajuste, sólo que ahora tendremos unas distancias ponderadas al número GRIs obtenidos para cada año. La distancia

mínima sigue siendo 0, pero ahora tendremos un número decimal de mayor precisión como medida de la distancia entre empresas.

Los resultados obtenidos con esta métrica resultaron satisfactorios para las siguientes funcionalidades del programa, así que, de este modo, se aceptó esta distancia ponderada como forma para medir distancias entre empresas. El siguiente paso era crear la lista de vecinos, la lista de adyacencia del grafo.

Una vez calculado el cutoff, se recorre el array de empresas mirando cada par de estas, empresa x y empresa y, calculando de nuevo la distancia entre ellas. Si la distancia es menor que el cutoff, se añade la empresa y como vecina de la empresa x. Cuando se calcule la distancia de la empresa y a la empresa x, como será igual que la anterior, se añadirá la empresa x como vecina de la empresa y. Para esta operación se han usado listas enlazadas. Cada empresa tiene asociada una lista enlazada con el primer nodo al identificador de la primera empresa vecina. A su vez, este nodo apunta al identificador de la siguiente empresa vecina sucesivamente hasta la última que apunta a NULL.

La opción de desarrollo más fácil era una matriz de adyacencia, de dimensión NxN (N=5897) con 1 en la posición i,j para empresa i y empresa j vecinas y 0 en los casos en los que no lo sean. El problema de esta solución es que desperdicia una gran cantidad de memoria, así que se optó por el desarrollo mediante listas enlazadas.

Dado la gran cantidad de empresas con las que se trabajaba y un cutoff relativamente bajo, aunque ajustable, se daba la situación en la que gran cantidad de empresas no tenían ningún vecino. Esta situación no era deseable para el algoritmo de recuperación de patrones, así que para estos casos se precedió a incluir el vecino más próximo de estas empresas. Una vez creada la lista de adyacencia de una empresa, se busca el vecino cuya distancia sea mayor que el cutoff pero menor que 1 mas. Si no se encuentra ninguno, se procede a la misma búsqueda aumentando en 1 el corte.

A destacar en este algoritmo que nunca añade los mismos vecinos más próximos. Por ejemplo si la empresa con identificador 512 va a añadir el vecino más próximo no va a comenzar comprobando si la empresa con identificador 513 es vecina y si no lo es su distancia con la anterior. El algoritmo comienza buscando en una posición aleatoria del array. De este modo, cada vez que el programa se ejecute se obtendrán resultados diferentes.

3.2.3 Recuperación de patrones

La siguiente funcionalidad que tiene el programa es la de recuperar un año concreto del patrón. Esto se hace gracias a una función de red neuronal.

Para empezar se selecciona un año a recuperar. Como se ha mencionado antes, los primeros años contienen menos información, ya que la mayoría de las empresas no consiguió el mínimo para optar a un GRI, asi que para obtener resultados concluyentes se recomienda intentar recuperar años con mayor tasa de 1s.

A este año se le añade un porcentaje de ruido, es decir, se contaminan los datos para posteriormente intentar recuperar los originales.

Por ejemplo, al año 8 (2007) se le añade un 20% de ruido, o lo que es lo mismo, se cambian el 20% de sus valores; para 5897 empresas, habrá 177 cambios. Estos valores que se cambian son aleatorios y tendrán más influencia dependiendo de si caen en empresas con muchas conexiones.

Debido a que la función de la red neuronal recupera a partir de los valores de las empresas vecinas será más fácil recuperar los datos para empresas con muchas empresas vecinas. Pero si la mayoría del ruido ha entrado en las mismas vecinas el valor para ese año será irrecuperable.

Hay que tener en cuenta el porcentaje de ruido que se introduce en el patrón. Para ruidos bajos (1-5%) es más fácil recuperar los valores originales. Para ruidos superiores a 50% tendremos un patrón demasiado contaminado donde habrá más datos incorrectos que correctos.
Una vez calculado el vector con ruido, se intenta recuperar el vector original. Para ello se utiliza la función **Update** que implementa la red neuronal.

En esta función se va actualizando el vector con ruido intentando quitar los valores erróneos. Para ello se ejecuta la función de la red neuronal un máximo de 10000 veces o hasta que la diferencia entre las dos últimas etapas sea 0, es decir, que haya llegado al máximo errores subsanados posible y no haya forma de mejorar.

La función de la red neuronal es la siguiente:

Para todas las empresas
Para todas las empresas j vecinas de la empresa i
Para todos los años del patrón
Para el año y del patrón.

*w = w + año y del patrón la empresa i * año y de la empresa j vecina de i*

Una vez calculado w, se multiplica por el valor en el vector (1 o -1) y se pasa a una función de corte.
Si el valor supera el corte, el valor se actualiza a 1, si no se actualiza a -1.

El valor de corte viene definido por la siguiente formula:

k= Numero de patrones * número de vecinos medios del grafo * tasa1 del año con ruido al cuadrado * factor de corrección.

El factor de corrección se estableció a 0 a base de realizar gran cantidad de pruebas buscando cual mejoraba

Primera aproximación

El primer desarrollo de la función de la red neuronal, esta actualiza los valores de manera secuencial. Es decir, se comienza por la empresa con identificador 0, se actualiza el valor en el vector con ruido respecto a sus vecinas y se continúa con la empresa con identificador 1.
Cuando se termina con todas las empresas se continúa con la siguiente iteración.

El fallo de esta aproximación es que el resultado depende mucho de la posición que ocupa dentro de la estructura. Se da mucho más peso a las primeras apariciones siendo esto incorrecto, todas las empresas deberían tener el mismo valor independiente de su posición en la estructura, que se debe exclusivamente a su sector y al orden alfabético.

Se procede a una segunda aproximación en la que se reduzca este factor.

Segunda aproximación

En la segunda versión de la función, se intenta eliminar la dependencia de la posición en el array utilizando un vector auxiliar. En la primera pasada se actualiza este vector auxiliar. Una segunda pasada sobre este vector determina el resultado de cada iteración.

Los resultados usando este método son similares a la primera aproximación, mejorándolos escasamente. Se procede a una tercera aproximación que perfeccione estos resultados.

Tercera aproximación

En la tercera versión se procede a actualizar en orden aleatorio. Se obtiene un número aleatorio entre 0 y el número de empresas y se actualiza esa posición. Si la posición ya ha sido actualizada se calcula otro número aleatorio hasta que se obtiene uno que no haya sido actualizado. Cuando se han actualizado todas las posiciones del patrón se procede con la siguiente iteración.

El problema con esta versión es el tiempo de procesamiento, muy superior a los anteriores. Incluso podría darse el caso de no terminar nunca al extraer un número aleatorio ya actualizado una y otra vez.
En la última versión se intenta buscar un equilibrio entre tiempo de procesamiento y secuencialidad.

Versión final

La última versión de la función elige aleatoriamente el primer elemento que se actualiza y continúa secuencialmente a partir de ese en cada iteración.
Esta versión consigue los mejores resultados por tiempo de procesamiento.

3.2.4 Análisis del grafo

3.2.4.1 Globalización/Sectorización

Esta funcionalidad aporta información sobre con quien están conectadas las empresas.

En la parte de globalización se da una aproximación de cuales continentes tienen mayor asociatividad entre ellos. Es decir, si las empresas de un determinado continente están tomando políticas similares entre ellas.

Los datos del GRI recogen 6 regiones principales: África, Asia, Europa, Norte América, Latino América y Caribe y Oceanía.

Se recorre la lista de vecinos de cada empresa y se mira cuantas de ellas son de su misma región. Cuando se ha hecho esta operación para todas las empresas, se calculan las estadísticas de globalización y las empresas con más conexiones con empresas de su misma región para cada continente. La empresa con más vecinos del mismo continente se denomina **Hub** del continente. La empresa con más vecinos fuera de su contiene se denomina **Hub** del continente fuera de él.
Lógicamente los Hubs serán empresas con muchas conexiones. Estas se consideran las empresas que han llevado la iniciativa dentro de la región ya que tener más vecinas significa tener políticas comunes con mayor número de empresas.

La parte de sectorización es similar a la anterior sólo que ahora el estudio se centra en sectores en vez de en regiones.

En esta ocasión también tenemos 7 macro sectores distintos: Energía, Finanzas, Industria, Sector Primario, Servicios, Tecnológicas y "Otros". Estos últimos responden a empresas que no pueden ser catalogadas en los otros 6 sectores.

Del mismo modo se recorren las empresas buscando vecinas del mismo sector y se calculan las estadísticas. También se encontrarán los Hubs de cada sector de la economía.

Cabe destacar cómo se accede a la información de los nombres de las empresas dado que ésta no se guarda en memoria durante la ejecución.
Debido a que los nombres de las empresas sólo eran deseables en este apartado del programa, en vez de reservar memoria para todos ellos se optó por consultar en el fichero cada vez que se hallaba un Hub, ya fuera de globalización o de sectorización.
En total se calculan 2 Hubs por cada continente (6 continentes) (con él mismo y fuera de él) y 2 por cada sector (7 sectores) (ídem), en total 26 accesos a fichero.

3.2.4.2 Coeficiente de agrupamiento

El coeficiente de agrupamiento o clusterización determina como de interconectado está el grafo. Un grafo completo es aquel en el que cada par de nodos están unidos por una arista, por tanto, un grafo completo tiene n(n-1)/2 aristas.
Pero este parámetro no mide tanto completitud si no alcance de cada vértice, es decir, la proporción entre los enlaces conectados con sus vecinos entre el número que habría si fuera completo.
Según el modelo Watts-Strogatz, el coeficiente de agrupamiento de un vértice es

número de triángulos/número de vecinos(número de vecinos-1)

El algoritmo se basa en la búsqueda de triángulos, o ciclos de tres nodos, es decir de conexiones que van desde la empresa A a la empresa B, desde B hasta una tercera empresa C y vuelven de nuevo a A.

3.2.4.3 Camino mínimo/camino medio

En este apartado se ha implementado el algoritmo de Dijkstra para hallar el camino mínimo entre cada par de nodos.
El algoritmo necesita una serie de estructuras para la realización del cálculo. Se tendrá un vector de distancias que se irá llenando conteniendo las distancias de cada nodo al nodo fuente (el inicial desde donde se calculan los caminos). Además existirá otro vector de apoyo con los nodos desde los que se ha accedido a cada nodo desde el inicial, (los nodos padres). De este modo podremos definir un camino saltando desde cada nodo hasta su nodo padre hasta llegar al nodo inicial. También se han utilizado colas de prioridad para establecer la precedencia en la visita de nodos.
El algoritmo, básicamente, va recorriendo el grafo en altura desde el nodo fuente, añadiendo en cada visita a un vecino a este a la cola, calculando la distancia añadiendo 1 en cada altura hasta llegar a todos los nodos que tenga accesibles.
Al final, suma todas las distancias, desechando las que sean infinito al no haber podido acceder a esos nodos y divide entre el número de conectados para dar una estimación media de distancias.

Para hallar el camino medio entre los nodos del grafo, se ha calculado el camino mínimo para cada nodo al resto sin tener en cuenta los nodos inaccesibles. Al existir nodos a los que no podemos llegar desde otros, el camino medio sería infinito. Que surjan estos nodos inaccesibles depende del número de vecinos medio que es establecido por el cutoff.

4 Pruebas y resultados

4.1 Pruebas

En esta sección se comentan los errores corregidos mediante pruebas.

4.1.1 Formato archivo entradas

Ya se ha comentado con anterioridad los problemas encontrados con el formato.
Para probar que los datos que se estaban leyendo se diseñaron una serie de funciones:

imprimirPatronEmpresa
Está función imprime por pantalla el vector de sostenibilidad de una empresa, a continuación se compara la salida con el vector escrito en el fichero csv.
Evidentemente no se pudo comprobar para las 5897 empresas, pero si que fue útil para detectar los errores antes mencionado.

imprimirPatronAnho
Imprime el patrón de un año de todas las empresas. Igualmente se compara la salida con el archivo csv. Se probó para las columnas conflictivas, 1999 y 2013 (la primera y la última) en archivos con subconjuntos de empresas escogidas al azar del fichero.

imprimirRegionIndexEmpresa
Imprime la el índice de región de una empresa. Se probó con varios identificadores de empresas al azar y se comprobó que los datos eran correctos en el csv.

4.1.2 Nombre y país de la empresa

En un principio se guardaba el nombre y el país de la empresa en la estructura empresa. Además de ocupar memoria que se desperdiciaba (unos 250-300KB), la función de lectura daba errores de memoria al no reservar +1 carácter para el /0 en el malloc.
A pesar de ser un fallo de fácil detección y corrección, se pasó por alto y se decidió no almacenar en memoria estos campos.
Más tarde, cuando se dio la necesidad de conocer estos campos se desarrolló una función que leía los datos de una empresa del fichero de entrada cuando fuera necesario, en total 26 veces durante ejecución para conocer los Hubs de globalización/sectorización.

4.1.3 Lista adyacencia

Se pensaron varias formas de representar el grafo una vez creados los vecinos. La primera prueba se hizo con una matriz de adyacencia. Esta matriz cuadrada tenía 5897 filas y 5897 columnas. Si en la casilla i,j hay un 1 significa que hay una unión entre los nodos i y j. Si hay un 0 significa que no estos no son vecinos.
El problema de este método es que desperdicia mucha memoria. Las pruebas con los algoritmos resultaban muy lentas así que se decidió implementar listas enlazadas para representar listas de adyacencia.
Cada nodo tiene adjunta una lista de adyacencia que apunta al primer vecino y este al siguiente hasta el final.
Sobretodo facilitó la implementación, ya que era simple recorrer los vecinos de un nodo.

4.1.4 Umbral

El umbral de corte para la recuperación de patrones fue difícil de estimar.
Para recuperar un dato del patrón de un año en una posición del vector, que corresponde a una empresa, se hacen una serie de operaciones con los vecinos de la empresa a la que corresponde la posición. Si el valor resultante es mayor que el corte, se recuperará un 1, si es menor, un -1.
Este umbral k está definido como el número total de patrones por la media de vecinos por la tasa de 1 de ese año al cuadrado por un factor de ajuste.
Se procedió a estimar este factor a base de distintas pruebas sobre distintos años.
Después de muchas pruebas se decidió dejar k =0 igualando el factor de ajuste a 0.
Lo cierto es que para algunos años se estaban consiguiendo mejores resultados que para otros. Esto hizo difícil la elección del valor de k.

4.1.5 Clusterización

La función de clusterización estaba mal diseñada ya que al probar los valores de retorno a menudo eran mayores que 1, que correspondería a un grafo completo. Esto se dio por que se intentaba calcular un promedio que no era exacto.
La fórmula dada por Watts-Strogatz es:

> *para todos los nodos*
> *triángulos desde nodo i / # vecinos de i * (numero vecinos i -1)*

y se estaba usando una aproximación para el total de esta formatos

> triángulos totales del grafo/ #vecinos medio * (#vecinos medio -1)

Se pensaba que no habría mucha diferencia con la aproximación pero los resultados daban valores muy por encima de 1.
Se cambió el diseño para hacerlo acorde a la fórmula general y se corrigió el problema.

4.1.6 Globalización

En la globalización/sectorización se dio un error similar. Por sentido común parecía que para una región la suma vecinos de la misma región con los vecinos de fuera entre el total empresas de ésta debía ser 1.

Los resultados no indicaban eso. La fórmula para calcularlos estaba siendo una aproximación:

vi=vecinos interiores/media vecinos del grafo
ve=vecinos exteriores/media vecinos del grafo

vi+ve nunca era cercano a 1. Ni siquiera sumando todas las regiones, por tanto la fórmula pensada era incorrecta.

Se sustituyó por:

vi=vecinos interiores/total de la región
ve=vecinos exteriores/total de la región

Ahora la suma de estos era 1, 100% en porcentaje.

4.2 Resultados

4.2.1 Globalización/Sectorización

Los ratios están calculados de esta forma:

Para todas las empresas
Vecinos de su misma región / empresas región
Vecinos de fuera de su región / empresas región

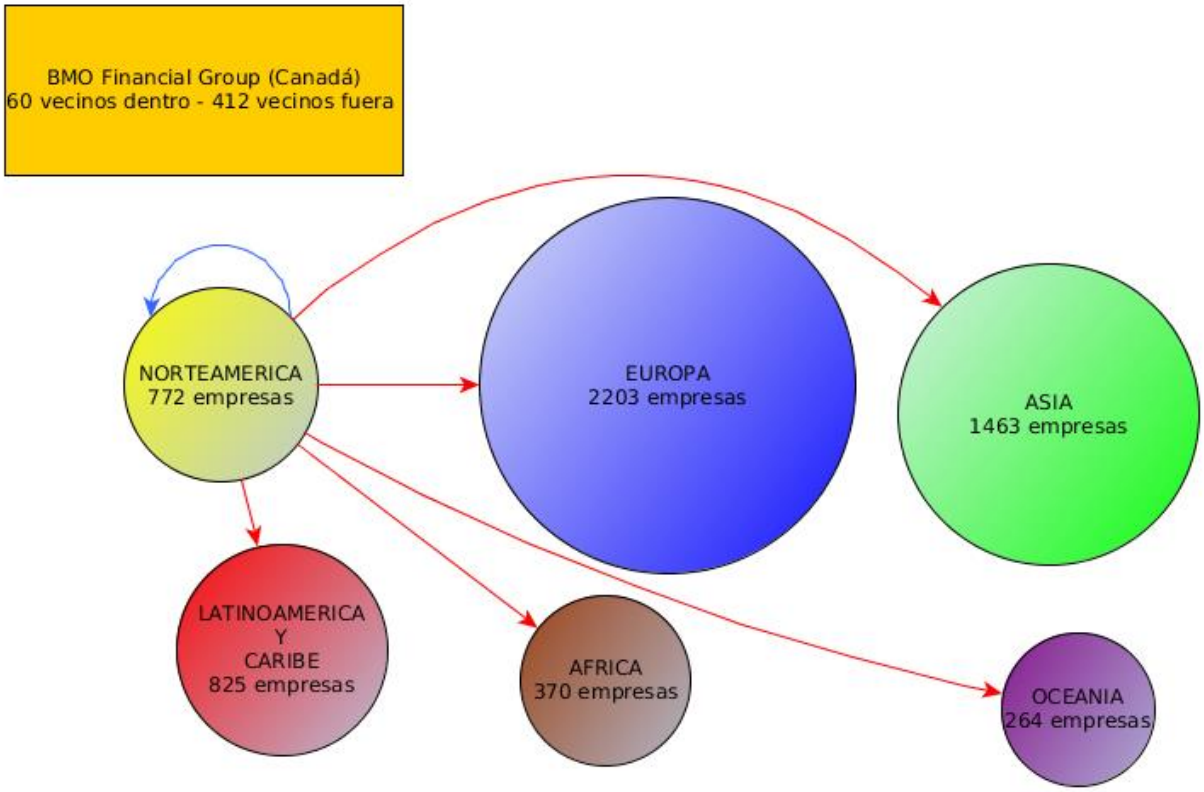


Ilustración 3

Región	Número empresas	Localización	Hub interior	Globalización	Hub exterior
África	370	10.414%	Firestone Energy Ltd (South Africa)	89.586%	DRDGold Limited (South Africa)
Asia	1463	28.206%	Development Bank (China)	71.794%	BPCL (India)
Europa	2203	37.784%	Enagas S.A. (Spain)	62.216%	BG Group (United Kingdom)
Norteamérica	772	13.452%	BMO Financial Group (Canada)	86.548%	BMO Financial Group (Canada)
Latinoamérica	825	16.213%	Banco Galicia (Argentina)	83.787%	Tractebel Energia (Brazil)
Oceanía	264	5.390%	AGL (Australia)	94.610%	CitiPower and Powercor (Australia)

Tabla1

Las conclusiones que podemos sacar de estos datos es que a mayor número de empresas para una región está va a estar más localizada.
Es meramente un tema probabilístico, si hay más empresas europeas, la probabilidad de que cualquier empresa sea vecina de una europea es más alta que de que sea vecina de una empresa oceánica.
Hubiera sido interesante probar para unos conjuntos de regiones del mismo tamaño.

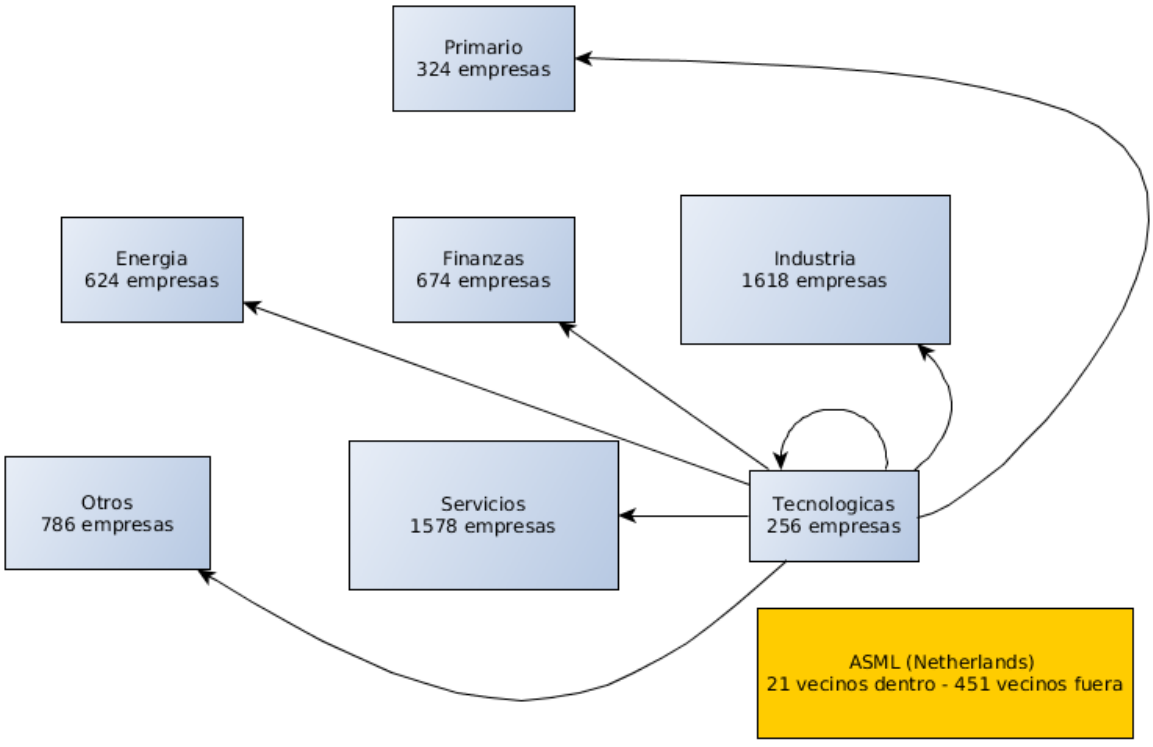


Ilustración 4

Sector	Numero empresas	Sector dentro	Hub dentro	Sector fuera	Hub fuera
Energía	624	14.373%	MOL Group (Hungary)	85.627%	BG Group (United Kingdom)
Finanzas	674	12.394%	IBERCAJA BANCO S.A.U (Spain)	87.606%	UCA Funds Management (Australia
Industria	1618	24.111%	ArcelorMittal (United Kingdom)	75.889%	Carris (Portugal)
Primario	324	4.971%	Rio Tinto (United Kingdom)	95.029%	DRDGold Limited (South Africa)
Servicios	1578	23.543%	Ahold (Netherlands)	76.457%	Accenture Spain (Spain)
Tecnología	256	2.188%	ASML (Netherlands)	97.812%	ASML (Netherlands)
Otros	786	9.992%	Protos (Belgium)	90.008%	Adecco (Switzerland)

Tabla 2

Para la sectorización ocurre lo mismo, los sectores con más empresas tienden a tener más vecinos en común.
Probablemente mejorando la precisión a la hora de capturar los patrones del fichero estos datos hubieran mejorado.
Según se capturan ahora no se tiene en cuenta el número de criterios que cumplieron cada año, ni la calificación que obtuvieron. Se pensó dejar para más adelante otra versión que estableciera importancia en el número y calificación de los criterios, pero no hubo tiempo para implementarla.
De este modo los vecinos que se creasen serían más precisos.

4.2.2 Recuperación de patrones

En esta sección se presentarán dos tipos de gráfica:

1- Diferencia con el patrón original: mide el número de valores diferentes en cada etapa entre el vector que se está recuperando y el vector original.

2- Diferencia entre las dos últimas etapas: mide el número de valores diferentes entre las últimas dos etapas de la recuperación. Hasta que no llega a 0 no se termina la recuperación.

Se probarán recuperaciones en dos años distintos (2008 y 2012), para mismo ruido (15%) y con dos valores diferentes de número de vecinos medios (50 y 200).

AÑO 2008
50 vecinos

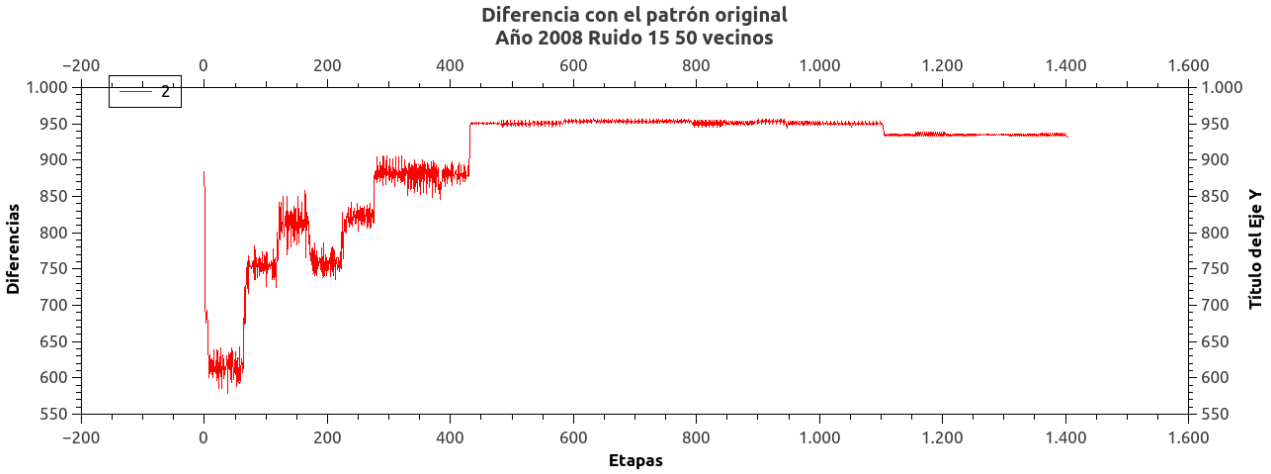


Ilustración 5

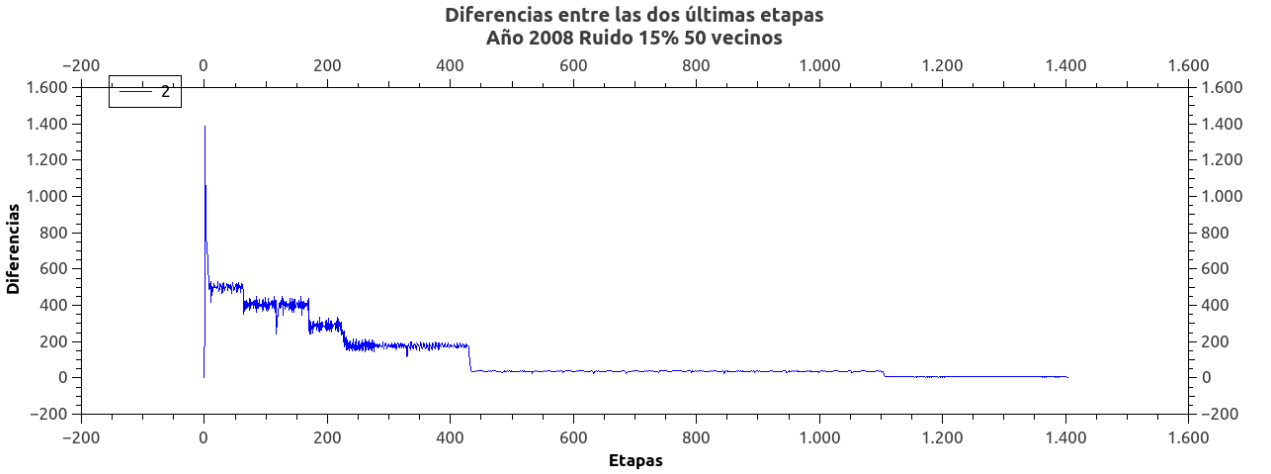


Ilustración 6

El error inicial de 884 valores diferentes en las primeras 50 etapas baja, pero después se incrementa e incluso empeora el porcentaje original. La diferencia entre etapas está funcionando, pero la red no está interpretando bien los atributos de la red, esto se debe a que existen pocos vecinos por empresa.

200 vecinos

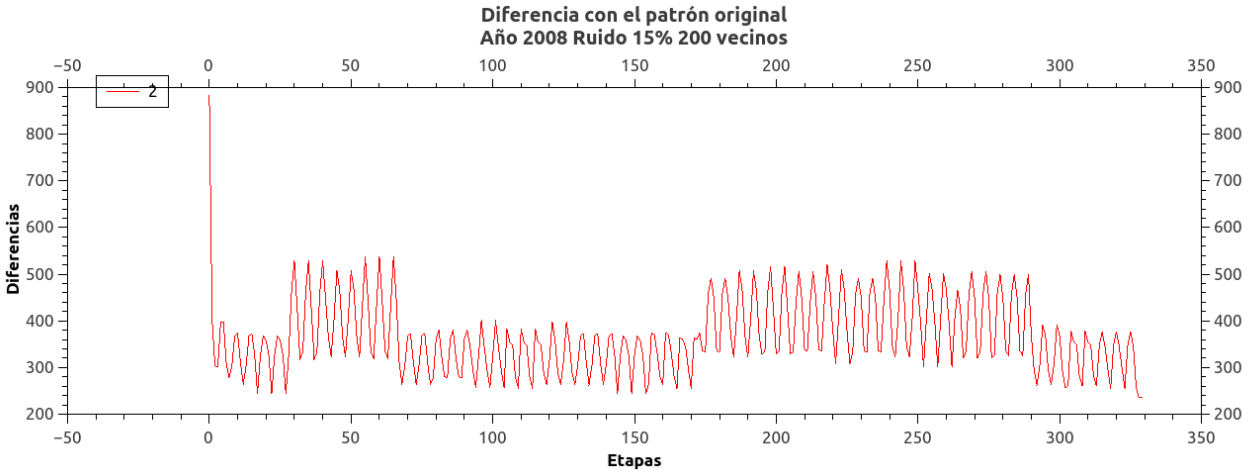


Ilustración 7

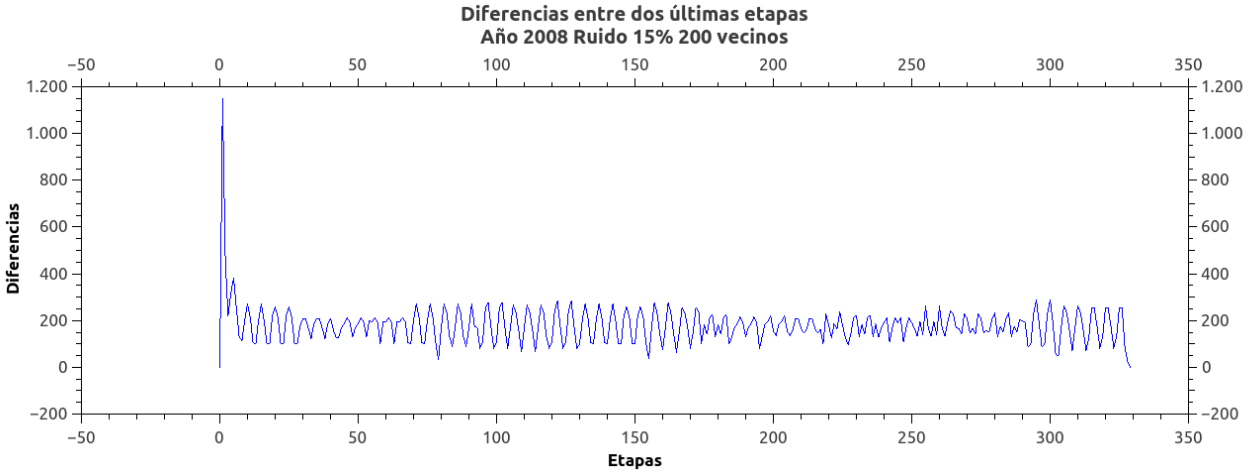


Ilustración 8

Con más vecinos el porcentaje de recuperación es mucho mejor. Se llega a recuperar desde un 15% hasta un 3% de errores.

Año 2012
50 vecinos

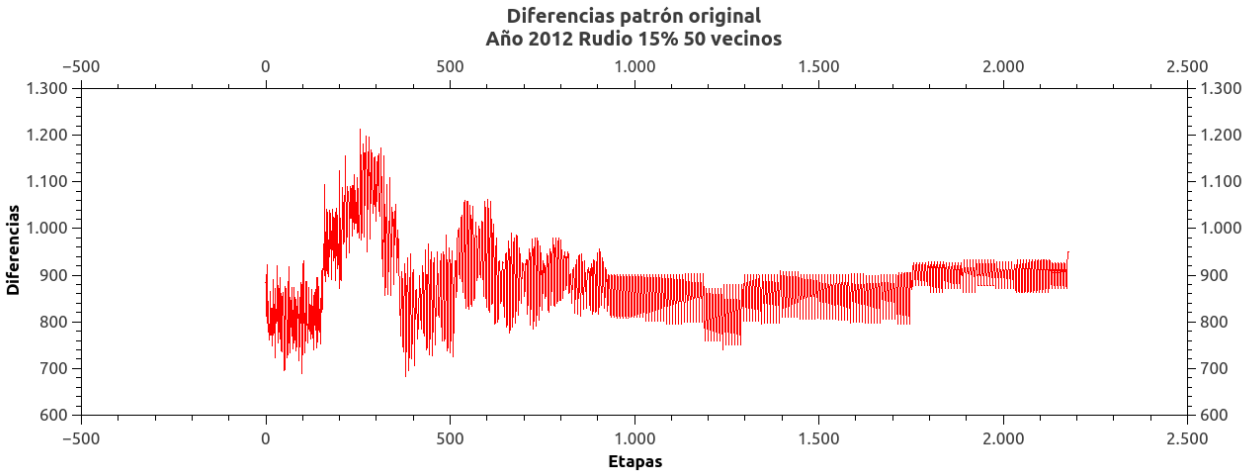


Ilustración 9

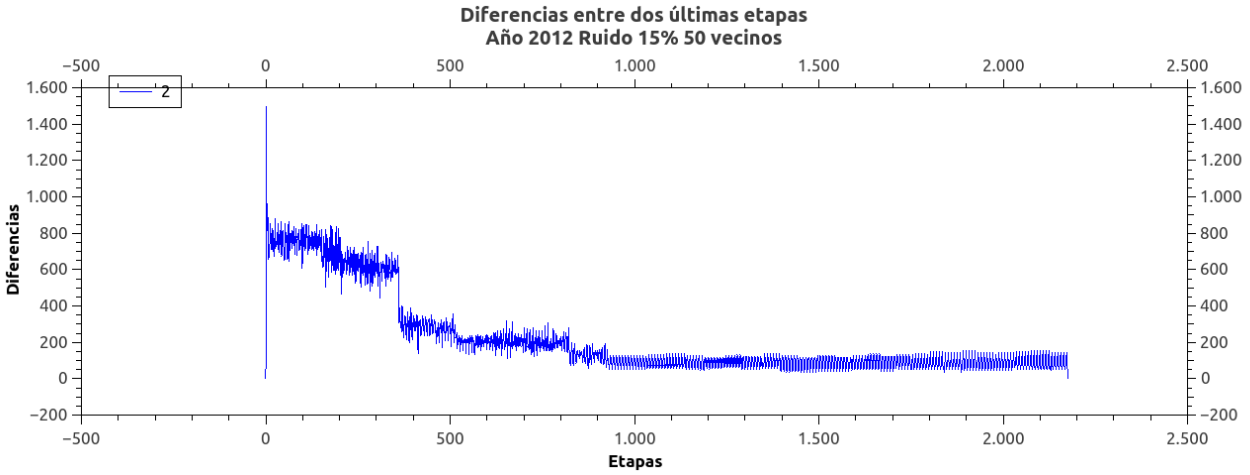


Ilustración 10

Para este año, con 50 vecinos, el resultado es aún peor que en el 2008. Además de que no se consigue recuperar, en ningún momento llega a estar cerca de hacerlo.

200 vecinos

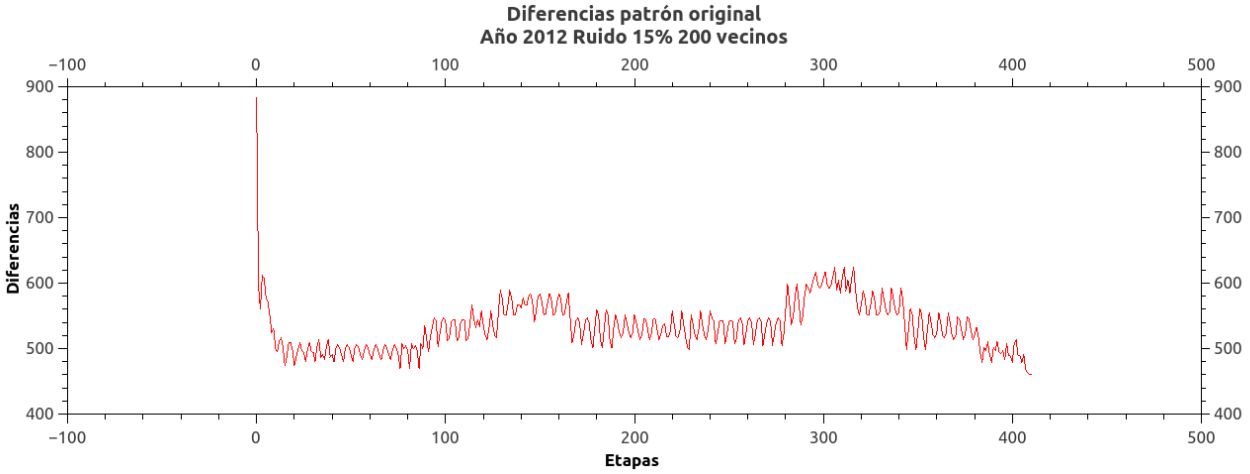


Ilustración 11

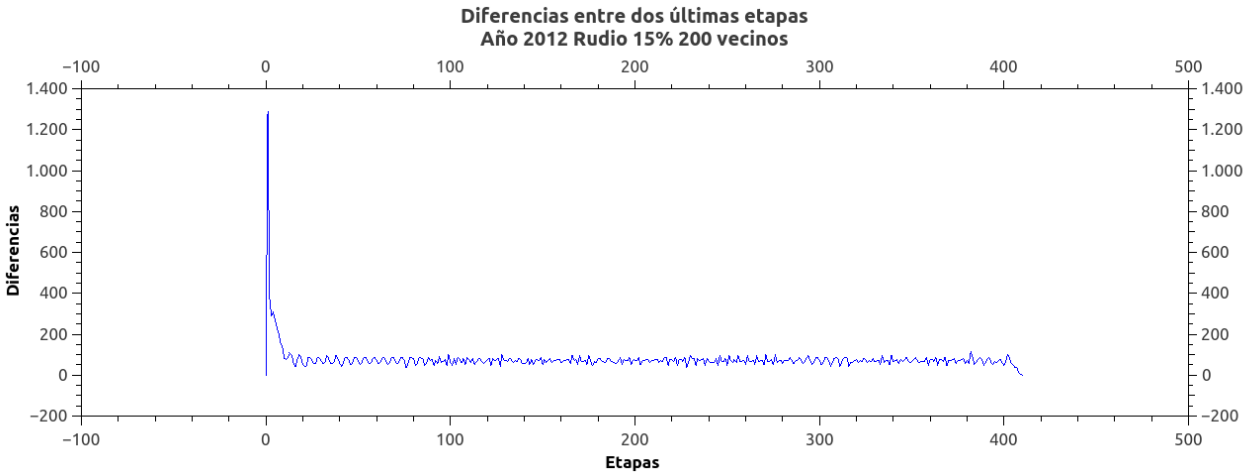


Ilustración 12

Con más vecinos la recuperación se hace efectiva. Baja de un 15% a un 7% errores. Además los resultados parciales dan menos picos.

Porcentaje de error tras recuperación con ruido 15%

Vecinos medios	2004	2008	2012
50 vecinos	16% (empeora)	14%	10%
90 vecinos	15%	8%	10%
115 vecinos	6%	7%	12%
200 vecinos	6%	2%	9%

Tabla 3

En los años centrales se obtienen mejores porcentajes de recuperación.
Con menos de 115 vecinos de media la mayoría de veces no consigue recuperar y en algunos casos incluso la recuperación empeora el porcentaje de error del vector con ruido.

4.2.3 Clusterización y Caminos medios

Para ilustrar lo que es el coeficiente de agrupamiento o clusterización se han diseñado los siguientes modelos explicatorios:

Coeficiente agrupamiento=1

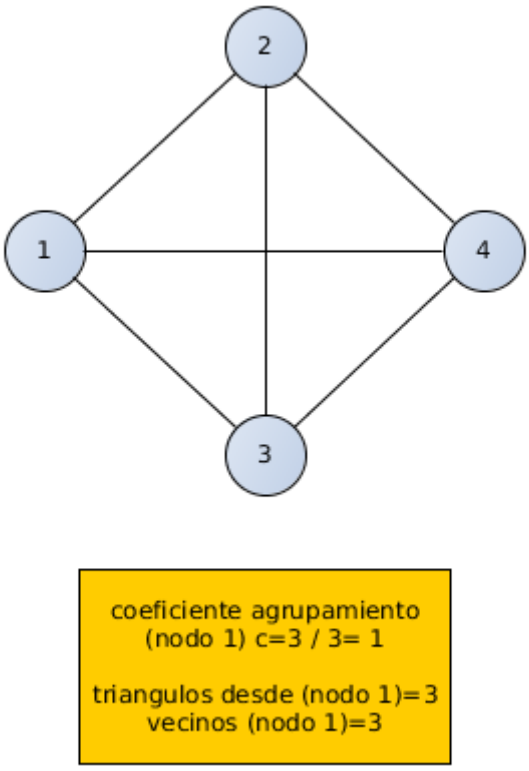


Ilustración 13

Como puede verse en la imagen, tenemos un grafo de 4 nodos con 6 aristas. El grafo es completo ya que cada nodo tiene una arista al resto de los nodos que lo componen.
En la clusterización, buscaremos el número de triángulos de cada nodo, es decir, todos los caminos que saliendo del nodo 1, pasan por un segundo y un tercer nodo volviendo al nodo 1 y los dividiremos entre el número de vecinos del nodo.

Podemos observar que los triángulos del nodo completo saliendo desde el nodo 1 son los formados por:
1-2-3, 1-2-4, 1-3-4 (y los repetidos 1-3-2, 1-4-2 y 1-4-3).
Haciendo la misma operación para todos los vértices tendremos un coeficiente de agrupamiento de 1.

Coeficiente de agrupamiento=1/4

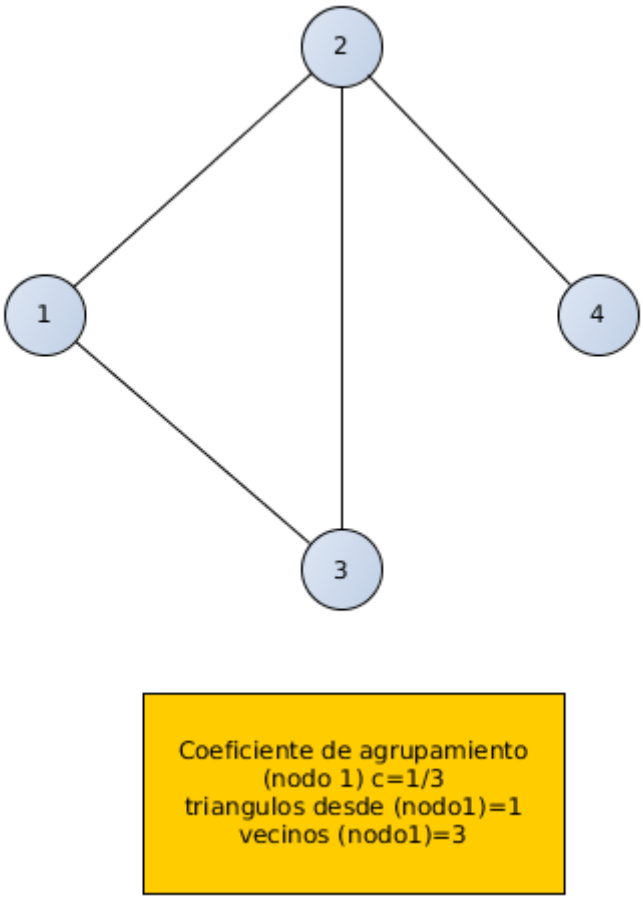


Ilustración 14

En este caso, puede verse que el grafo no es completo. Solamente se encuentra el triángulo formado por 1-2-3 (o 1-3-2).
Haciendo el cálculo desde todos los nodos: tendremos:
1- 1triangulo (mas el repetido)
2- 1 triangulo (ídem)
3- 1 triangulo (ídem)
4- 0 triángulos

Según la fórmula de Watts-Strogatz se tendría un coeficiente de agrupamiento de:

$$1/n \, \Sigma \, Ci$$
$$Ci=2 \cdot \#\blacktriangle / k(k-1)$$

Donde Ci es el coeficiente de agrupamiento en el nodo i, #▲ es el número de triángulos desde el nodo i y k el número de vecinos de i. Está multiplicado por 2 porque se cuentan los triángulos repetidos si el grafo es no dirigido.

El grafo anterior tiene ¼ de coeficiente de clusterización.

Coeficiente de agrupamiento = 0

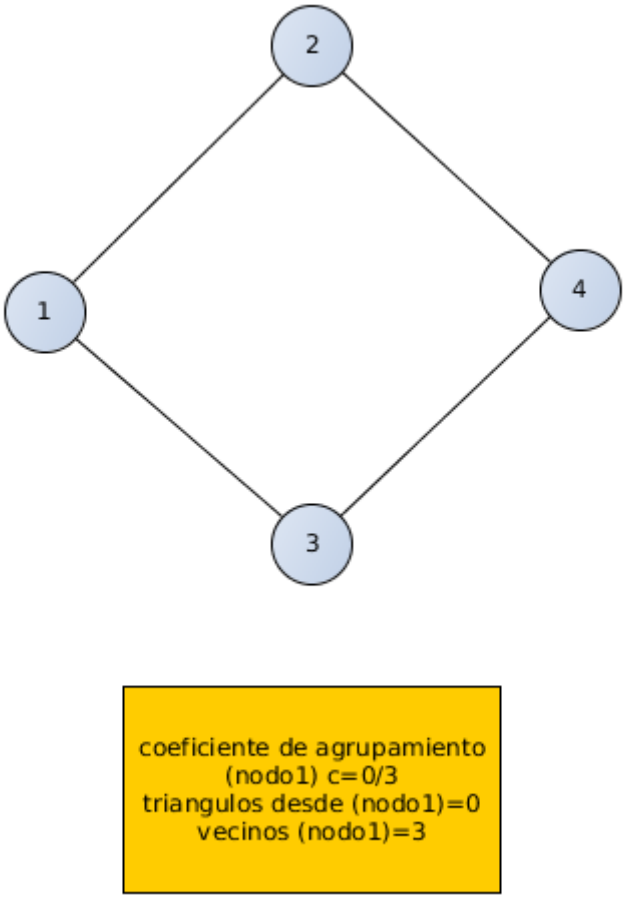


Ilustración 15

En este caso no se encuentra ningún triangulo, por tanto el grafo tiene clusterización 0.

Se muestran una serie de datos tomados de diferentes ejecuciones del programa para distintos valores de k (Vecinos medios).

Vecinos medios	1	2	3	Media
50 vecinos	0,223	0,223	0,223	0.223
90vecinos	0,340	0,339	0,339	0.339
115 vecinos	0,342	0,343	0,343	0,342
200 vecinos	0,520	0.520	0.520	0.520

Tabla 4

Según aumenta el número de vecinos, el grafo tiende a estar más conectado. Con 200 vecinos se obtiene un coeficiente de agrupamiento de 0,52. El grafo no puede llegar a considerarse small-world al no ser este valor lo suficientemente grande.

Se muestran una serie de datos tomamos para la ejecución del programa para distintos k (Vecinos medios).

Vecinos medios	1	2	3	4	5	6	7	8	9	Media
50 vecinos	3,669	5,815	3,763	6,292	4,768	4,308	4,450	4,868	4,588	4,724
90 vecinos	4,079	4,069	4,385	5,236	4,572	4,266	4,300	3,700	4,061	4,296
115 vecinos	4,017	4,502	4,129	4,196	3,754	3,890	4,322	3,621	4,412	4,093
200 vecinos	3,268	3,477	3,999	4,535	3,448	3.706	3.649	3.621	3,621	3,702

Tabla 5

Cuantos más vecinos medios tiene la red, más bajo es su camino medio. Al obviar el número de nodos desconectados se está perdiendo información en la medida, pero al aumentar el número de conexiones bajará el promedio de nodos desconectados.

4.2.4 Distribución de vecinos

Media vecinos=50

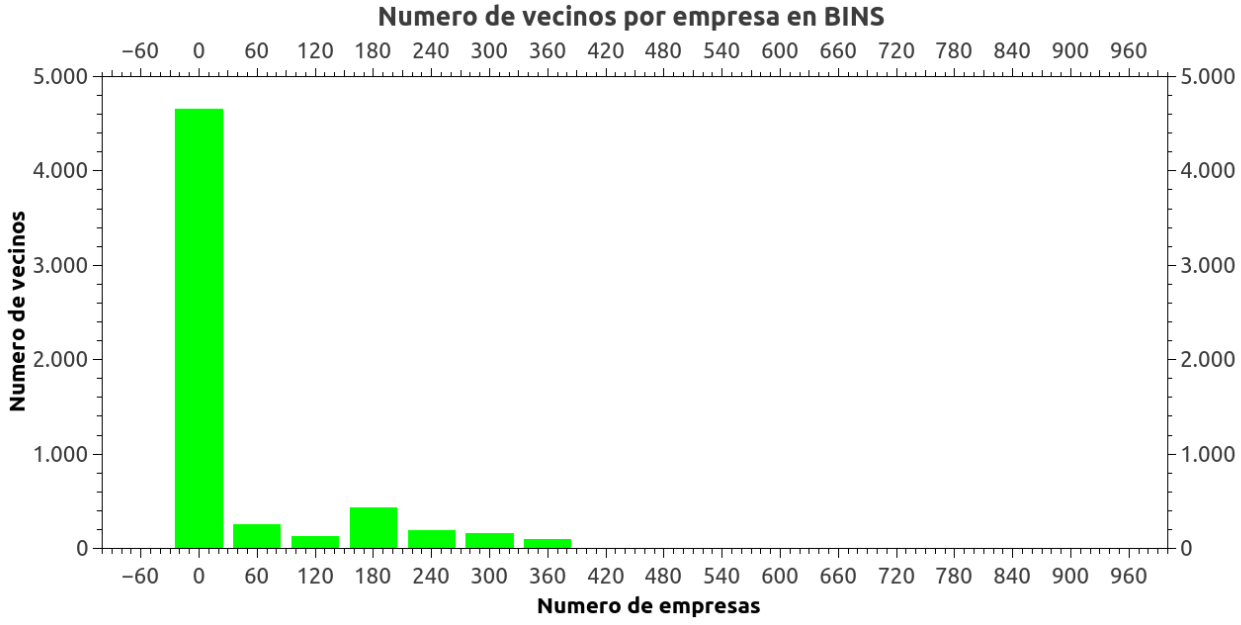


Ilustración 16

Este histograma muestra el número de vecinos por empresa. Se tienen más o menos 4800 empresas que tienen entre 1 y 60 vecinos. Los valores van disminuyendo hasta llegar a 180-240 vecinos, que vuelve a aumentar. A partir de ahí se asemeja a una distribución exponencial.

Media vecinos=115

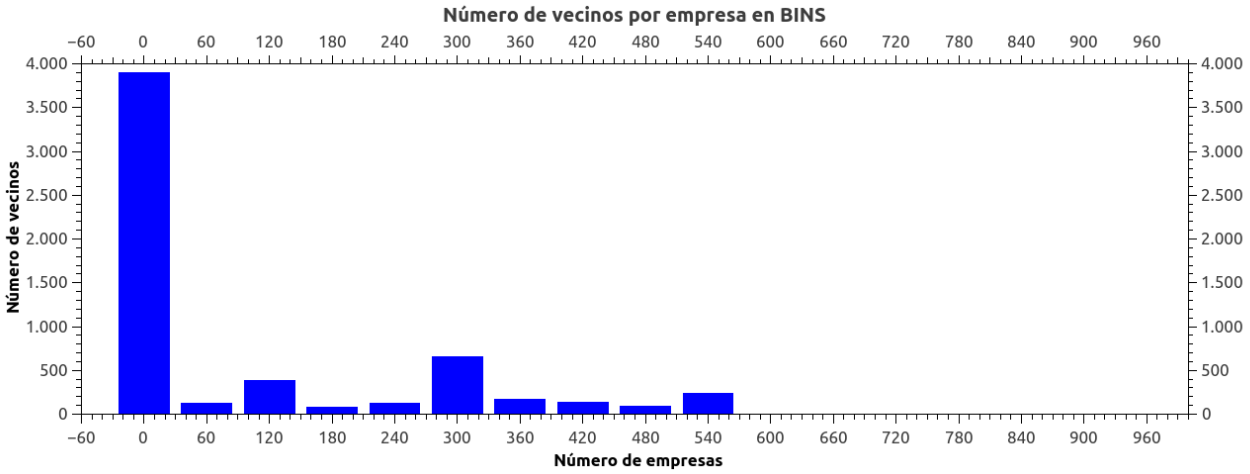


Ilustración 17

Subiendo el valor de cutoff, aumentará el número de vecinos medios. Habrá menos empresas con 1-60 vecinos, pero la distribución es similar, aumentando el número de empresas que tienen muchas conexiones.

Media vecinos=200

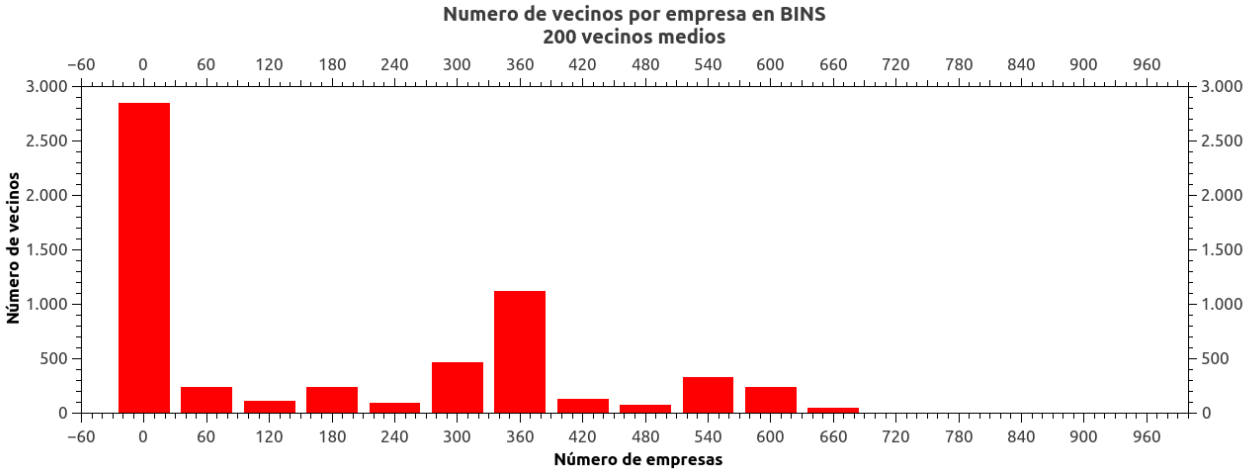


Ilustración 18

Por último, aumentando el cutoff hasta tener una media de vecinos de 200, aumentan el número de empresas con muchos vecinos. Los grandes grupos de empresas con número de vecinos de 300-360 y 360-420 responden a comunidades de empresas vecinas muy interconectadas entre ellas. Algunas de estas tendrán conexiones con otros grupos, disminuyendo el número de nodos desconectados y aumentando el valor de clusterización.

4.2.5 Visualización del grafo y detección de comunidades

Se ha utilizado la herramienta Gephi para representar los grafos visualmente.
Aplicando un algoritmo de dispersión y otro de detección de comunidades se han obtenido los siguientes resultados.

50 vecinos

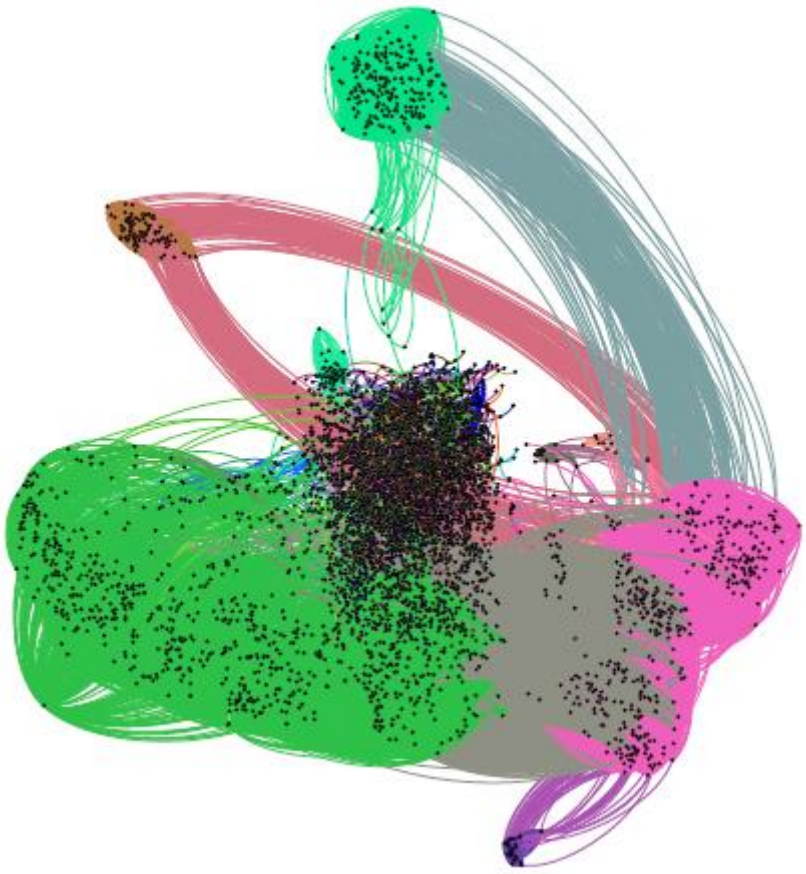


Ilustración 19

Con una media de 50 vecinos se observa la creación de 6 comunidades y una gran cantidad de nodos que quedan en medio de estas.
La comunidad verde parece la más numerosa. Aparece una pequeña comunidad violeta de empresas muy conectadas con la comunidad rosa.

115 vecinos

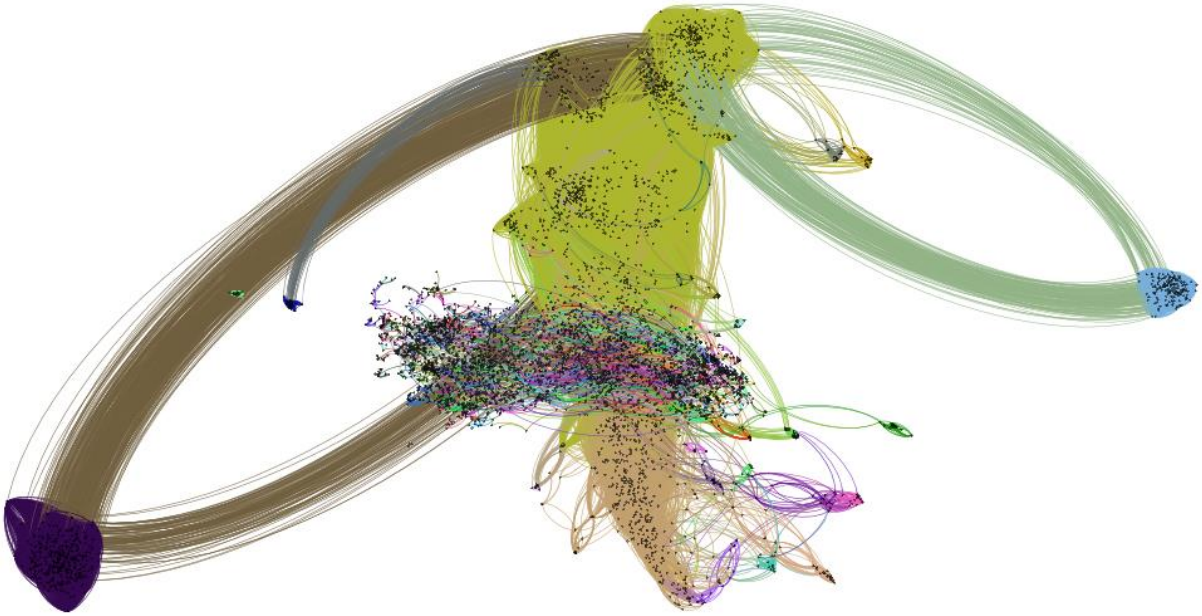


Ilustración 20

Con 115 vecinos media se crea una gran comunidad amarilla y dos más pequeñas muy conectadas con esta. En la parte inferior se observan pequeñas comunidades conectadas entre sí. Igual que en el caso anterior, en el centro aparecen muchos nodos no agrupados.

200 vecinos

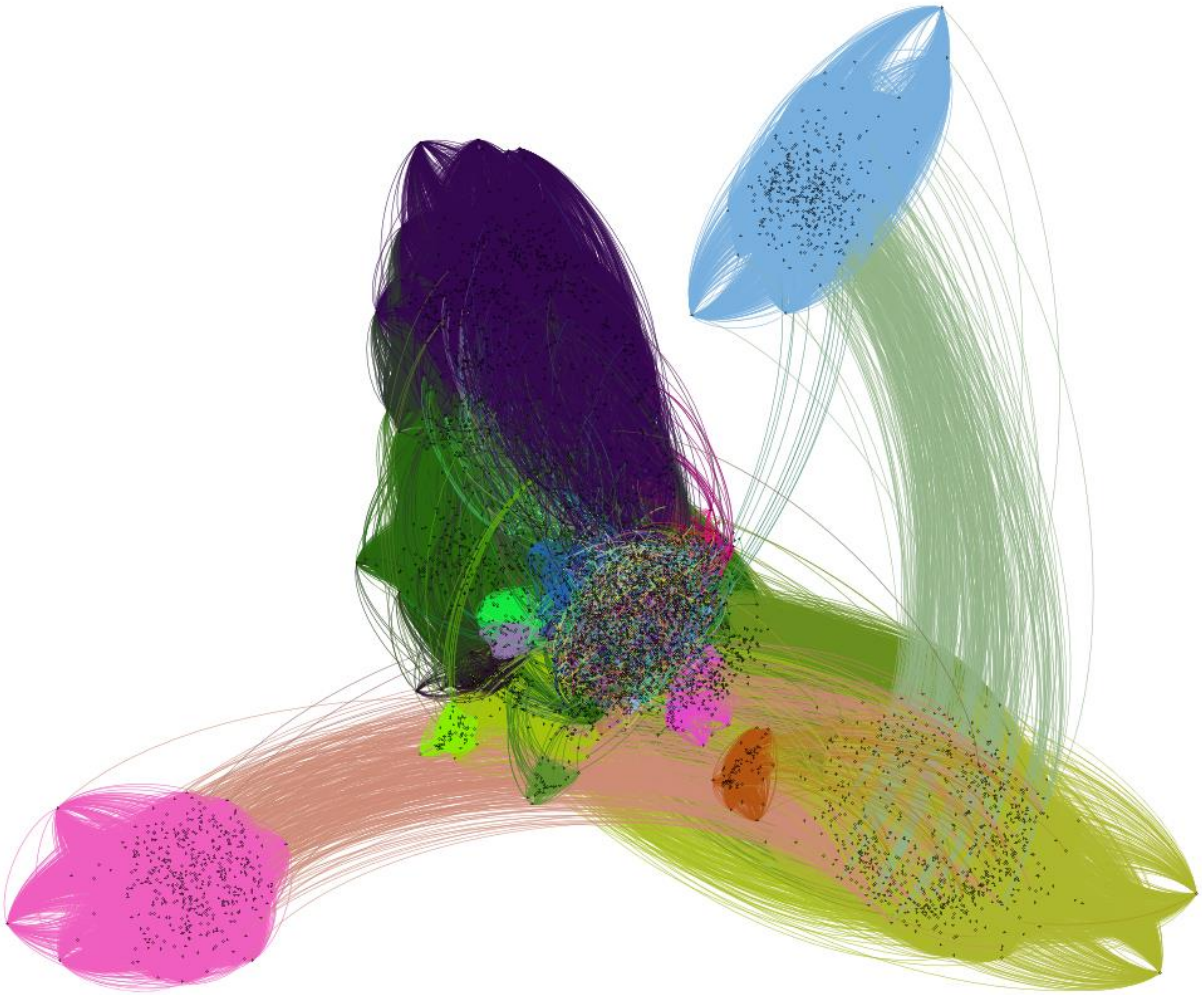


Ilustración 21

Por último, con 200 vecinos media se observan comunidades más grandes. Parece que la comunidad amarilla está más conectada con las comunidades rosa y turquesa y la azul con la verde y con las comunidades más pequeñas.

4.2.6 Tiempo de ejecución.

Vecinos medios	Tiempo mínimo	Tiempo máximo
50 vecinos	5min 18s	7min 49s
90 vecinos	7min 6s	8min 23s
115 vecinos	11min 38s	13min 7s
200 vecinos	23min 7s	25min 26s

Tabla 6

Las partes del programa que más complejidad en tiempo tienen son la creación de listas de adyacencia con un cutoff bajo, clusterización y la recuperación de patrones que en algunos casos hace pocas iteraciones y en otros llega a 10000 dependiendo de en qué partes del patrón ha entrado el ruido (aleatorio).

4.2.7 Tasas de 1

Año	%1s en el patrón
1999	0.002
2000	0.007
2001	0.021
2002	0.025
2003	0.028
2004	0.049
2005	0.066
2006	0.091
2007	0.125
2008	0.199
2009	0.264
2010	0.351
2011	0.524
2012	0.615
2013	0.380

Tabla 7

Esta tabla representa el porcentaje de empresas que aprobaron al menos un criterio GRI en un año. Se puede observar que en los primeros años muy pocas empresas de las que tenemos datos lograron cumplir algún examen de sostenibilidad. A medida que van pasando los años este valor va creciendo hasta el año 2013 que vuelve a bajar.

5 CONCLUSIONES

Se han llegado a una serie de conclusiones que intentarán responder a las preguntas lanzadas en el apartado de motivaciones.

Sobre cómo se están comportando las empresas con el avance de los criterios de sostenibilidad, 2012 es el patrón con más 1s y al año siguiente hay una caída hasta niveles de 2010. Parece que desde 1999 hay una tendencia al alza en intentan cumplir los estándares de sostenibilidad e incluso obtener mejores calificaciones que el anterior.
Esto responde en parte a una concienciación de las empresas con el medio ambiente pero sobre todo a que cumplir estos criterios mejora la confianza de la sociedad.

Además parece que todas las empresas van de la mano
En torno a los parámetros de globalización y sectorización no se pudieron obtener resultados concluyentes al tener subconjuntos de diferentes tamaños. Para los subconjuntos de mayor tamaño se daban ratios interiores más altos debido a que la probabilidad de ser vecino con una empresa también depende del número de empresas que haya en cada subconjunto.

Los años más difíciles de recuperar son los últimos. En los primeros la mayoría son 0 y quizá habría que mejorar la función de corte para obtener mejores resultados en estos años.

El intervalo de años con mejores resultados de recuperación es de 2007-2011. La distribución de 1s en estos años es más uniforme. El año en que comenzó la crisis, 2008, obtiene buenos índices de recuperación. El año 2013 es más impredecible.

Consideraciones a mejorar:

- Determinar si el grafo es small-world: para esto hubiera sido necesario estudiar más a fondo los datos de clusterización y camino medio.

- Creación de algoritmo para detectar comunidades

- Estudiar la recuperación de patrones pasando incrementos del número de patrones a la red neuronal.

6 REFERENCIAS

PREDICTING SUSTAINABILITY REPORT SCORING SEQUENCES USING AN ATTRACTOR NETWORK - MARIO GONZÁLEZ, MARÍA DEL MAR ALONSO-ALMEIDA, CASSIO AVILA, DAVID DOMINGUEZ

COLLECTIVE DYNAMICS OF 'SMALL-WORLD' NETWORKS -DUNCAN J. WATTS, STEVEN H. STROGATZ.

FUNDAMENTALS OF COMPUTATIONAL NEUROSCIENCE. - THOMAS TRAPPENBERG.

ALGORITHMS FOR GRAPH PARTITIONING ON THE PLANTED PARTITION MODEL. - CONDON, A. Y KARP, R.

GRAPH THEORY APPLICATIONS – L.R. FOULS

GRAPH THEORY – W.T. TUTTE

A NOTE ON TWO PROBLEMS IN CONNEXION WITH GRAPH – E.W. DIJKSTRA

MODERN APPLIED STATISTICS WITH S - W.N. VENABLES Y B.D. RIPLEY. SPRINGER.

AN INTRODUCTION TO NEURAL NETWORKS – PROF. LESLIE SMITH HTTP://WWW.CS.STIR.AC.UK/~LSS/NNINTRO/INVSLIDES.HTML

AUDITORÍA DE INFORMES DE RESPONSABILIDAD SOCIAL EMPRESARIA – GABRIELA C. CARRIZO

GUÍA PARA LA ELABORACIÓN DE MEMORIAS DE SOSTENIBILIDAD G4 – GRI - HTTPS://WWW.GLOBALREPORTING.ORG/RESOURCELIBRARY/SPANISH-G4-PART-ONE.PDF

GUÍA PARA LA ELABORACIÓN DE MEMORIAS DE SOSTENIBILIDAD G3 – GRI - HTTPS://WWW.GLOBALREPORTING.ORG/RESOURCELIBRARY/SPANISH-G3.1-COMPLETE.PDF

REVISTA DE RESPONSABILIDAD SOCIAL DE LA EMPRESA – FUNDACIÓN LUIS VIVES HTTP://LUISVIVESCES.ORG/UPLOAD/32/25/RSE3_COMPLETA_VF.PD

GEPHI TUTORIAL VISUALIZATION – GEPHI- HTTPS://GEPHI.GITHUB.IO/USERS/TUTORIAL-VISUALIZATION/

7 ANEXOS

7.1 Manual Gephi

Para pasarle a Gephi el grafo tendremos que tener un archivo que describa las uniones de la siguiente forma:

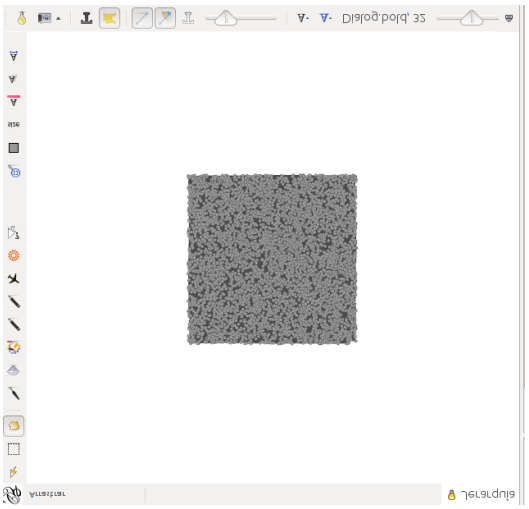
$$\begin{array}{cc} Nodo\ inicial_i & Nodo\ final_j \\ Nodo\ inicial_i & Nodo\ final_k \end{array}$$

Es decir, por cada arista tendremos una línea del fichero que describa el nodo del que sale al nodo que llega.
Una vez que tenemos el grafo en este formato procedemos a abrir Gephi y cargarlo.
Es importante que la extensión del fichero sea .csv.

Abrimos Gephi y pinchamos en Archivo/Abrir. Se abrirá una ventana donde seleccionaremos el archivo con el grafo.

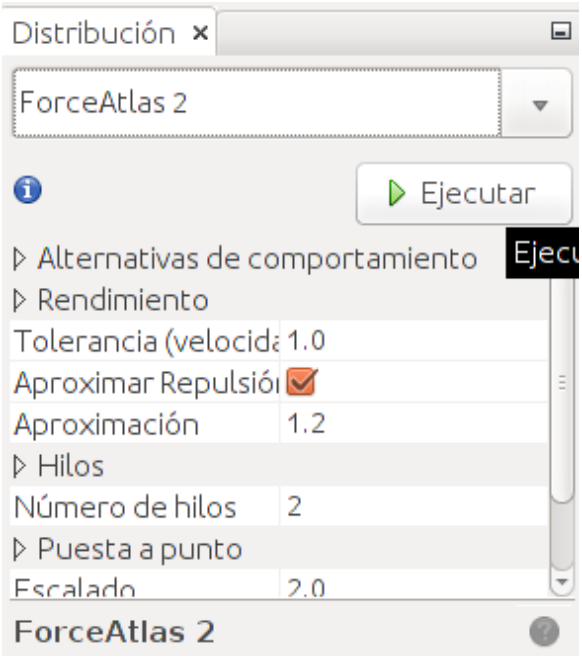
Una ventana emergente nos indicará si ha habido algún error al cargar el grafo. Si no ha habido problemas elegimos "Undirected",ya que en nuestro caso estamos pintando un grafo no dirigido.

Aceptamos y se mostrará en pantalla una nube de nodos apilados en un bloque.



Se deberán calcular un par de parámetros antes de dispersar los nodos.
A la derecha de la pantalla se mostrará la ventana Estadísticas. Se hará doble click para calcular la Modularidad y opcionalmente el Grado Medio.

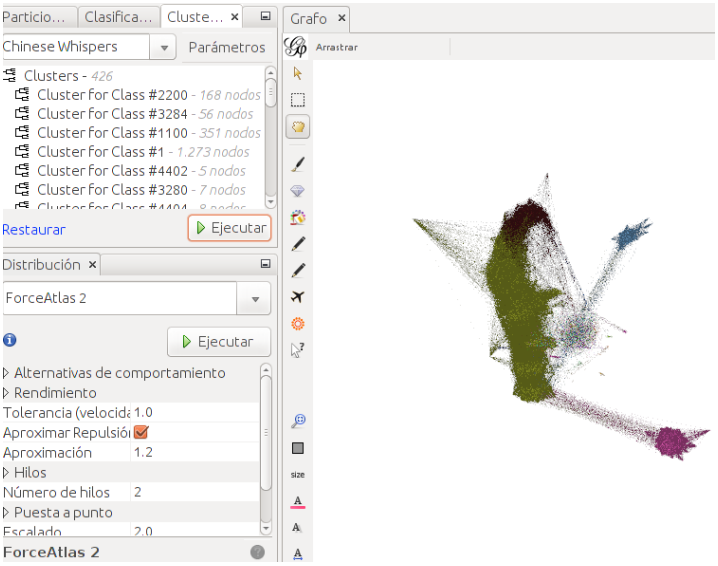
Una vez calculados procedemos a aplicar el algoritmo de dispersión. En la pestaña de Distribución, situada en la esquina izquierda elegiremos “Force Atlas 2” y pincharemos en Ejecutar.



Detendremos la ejecución cuando veamos que los distintos nodos se han separado lo suficiente, formando pequeños grupos.

Ahora, iremos a la pestaña en la esquina superior izquierda llamada "Clustering" y elegiremos "Chinese Whispers" (puede que haya que descargar este plugin por separado).

Pasados unos segundos el grafo aparecerá con las diferentes comunidades pintadas de colores.



Podremos exportar la imagen como foto o PDF en Archivo/Exportar/"Archivo SVG/PDF/PNG..."

7.2 Código C de la aplicación

En un principio se pensó adjuntar todo el código de la aplicación, pero como ocupaba 30 páginas en un tamaño de letra pequeño se optó por añadir únicamente el módulo principal y comunidades.c

main.c

main.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "comunidades.h"
#include "comunidades_old.h"
#include "grafos.h"
#include "listas.h"
#include "colas.h"
#include "constantes.h"


int main(int argc, char** argv){


		FILE* entrada=NULL;
		FILE* salida=NULL;
		FILE* vecinosporempresa=NULL;
		FILE* numvecinosporempresa=NULL;
		FILE* bricks=NULL;
		FILE* nombres=NULL;
		empresa** mEmpresas=NULL;
		int i,j,num_empresas,anho,dif;
		int* patronRudio;
		float cutoff,mediavecinos;
	float* tasas;
		char* nm;


		if(argc!=4){
			printf("Error en el numero de argumentos: './sostenible <fichero_entrada> <fichero_salida> <anho a recuperar>'\n");
			return -1;
		}
		entrada=fopen(argv[1],"r");
		if(entrada==NULL){
			printf("Error al cargar el fichero de entrada\n");
			return -1;
		}
		salida=fopen(argv[2],"w");
		if(salida==NULL){
			printf("Error al cargar el fichero de salida\n");
			return -1;
		}

		anho=atoi(argv[3]);
		if((anho<0)||(anho>14)){
			printf("Error al introducir el anho a recuperar: 'valor entre 0 y 14'\n");
			return -1;
		}

		mEmpresas=crearEstructura(entrada,&num_empresas);
		if(entrada!=NULL){
			fclose(entrada);
		}
	tasas=tasasde1s(mEmpresas, num_empresas);
		fprintf(salida,"ANALISIS DEL GRAFO\n\n");
		cutoff=calcDistPond(mEmpresas,num_empresas,cuto,tasas,salida);
		listVecPond(mEmpresas,num_empresas, cutoff,&mediavecinos,tasas);
		//fprintf(salida,"Media de vecinos:%.3f\n",mediavecinos);
		fprintf(salida,"Numero de nodos: %d\n",num_empresas);
		fprintf(salida,"Numero de conexiones: %d\n",numconexiones(mEmpresas,num_empresas));
		fflush(salida);
		fprintf(salida,"Grado Maximo: %d\n",degreeMax(mEmpresas,num_empresas));
		fprintf(salida,"Grado medio: %.3f\n",mediavecinos);
		fflush(salida);
		fprintf(salida,"Coeficiente de agrupamiento: %.3f\n",clusterizacion(mEmpresas,num_empresas,mediavecinos));
		fflush(salida);
		fprintf(salida,"Camino medio entre nodos conectados: %.3f\n\n",caminoMedio(mEmpresas,num_empresas));
		fflush(salida);


		printGEPHI(mEmpresas,num_empresas);


		globalizacion(mEmpresas,num_empresas,mediavecinos,argv[1],salida);
		sectorizacion(mEmpresas,num_empresas,mediavecinos,argv[1],salida);


		printVecEmp(mEmpresas,num_empresas,vecinosporempresa);
		/*printNumVecEmp(mEmpresas,num_empresas,numvecinosporempresa);
		printBINSVec(mEmpresas,num_empresas, 60,bricks);*/


		patronRudio=ruidoEnVector(mEmpresas,num_empresas,anho,RUIDO,salida);
		dif=Update(mEmpresas,num_empresas,anho,patronRudio,mediavecinos,salida);

		/***** LIBERAR MEMORIA *****/
		if(salida!=NULL){
			fclose(salida);
		}
		for(i=0;i<num_empresas;i++){
			LiberarLista(mEmpresas[i]->listaVecinos);
			free(mEmpresas[i]);
```

```
	}
	free(mEmpresas);
	free(patronRudio);


	return 0;
}
```

comunidades.c

```
/*****************************************************************************************************
comunidades.c


Funciones para la creacion de comunidades


Autor: Fernando Modamio
*****************************************************************************************************/
#include "comunidades.h"


/*****************************************************************************************************
CrearEmpresa


Crea una empresa leida de fichero


parametros de entrada:


empresa em, la empresa que se va a guardar
char nm, el nombre de la empresa
char msec, el macrosector de la empresa
int ri, region index
int si, sector index
int isus[], vector de sostenibilidad
int id, id de la empresa (entero unico que describe la empresa en la estructura)


salida:
empresa em, empresa creada
*****************************************************************************************************/
empresa* crearEmpresa(empresa* em,int ri, int si,int isus[],int id){
	int i;


	em=malloc(sizeof(empresa));
	em->region_id=ri;
	em->sector_id=si;
	for(i=0;i<NUM_PATRONES;i++){
		em->sust[i]=isus[i];
	}
	em->id=id;
	em->listaVecinos=NULL;
	em->num_vecinos=0;


	return em;
}


/*****************************************************************************************************
crearEstructura


Lee de fichero las empresas y crea la estructura con ellas


parametros de entrada:
FILE* file, el archivo donde se encuentran las empresas
int* num, entero donde guardará el numero total de empresas leidas y almacenadas en la estructura


salida:
empresa** mEmpresas, matriz de empresas
*****************************************************************************************************/
empresa** crearEstructura(FILE* file, int* num){
	empresa* em=NULL;
	empresa** mEmpresas=NULL;
	int num_empresas=0;
	char buffer[1000];
	char* ptr=NULL;
	char coma[2]=",";
	int aux_sus[15];
	int i,ri,si;


	mEmpresas=(empresa **)malloc(5900*sizeof(empresa*));


	fgets(buffer,1000,file);
	while(fgets(buffer,1000,file)){
		ptr=strtok(buffer,coma);
		//strcpy(nm,ptr);					//name


		ptr=strtok(NULL,coma);
		//strcpy(cn,ptr);					//country
```

```
                        ptr=strtok(NULL,coma);                  //region

                        if(strcmp(ptr,"Africa")==0)
                                ri=1;
                        else if(strcmp(ptr,"Asia")==0)
                                ri=2;
                        else if(strcmp(ptr,"Europe")==0)
                                ri=3;
                        else if(strcmp(ptr,"Northern America")==0)
                                ri=4;
                        else if(strcmp(ptr,"Latin America & the Caribbean")==0)
                                ri=5;
                        else if(strcmp(ptr,"Oceania")==0)
                                ri=6;
                        else
                                ri=7;


                        ptr=strtok(NULL,coma);
                        //ri=atoi(ptr);                         //region index


                        ptr=strtok(NULL,coma);                  //sector


                        ptr=strtok(NULL,coma);
                        //strcpy(msec,ptr);                     //macrosector


                        ptr=strtok(NULL,coma);
                        si=atoi(ptr);                           //sector index


                        /***** PATRON *****/
                        for(i=0;i<NUM_PATRONES;i++){
                                ptr=strtok(NULL,coma);
                                if(strcmp(ptr,"0")==0)
                                        aux_sus[i]=-1;
                                else
                                        aux_sus[i]=1;
                        }

                        em=crearEmpresa(em,ri,si,aux_sus,num_empresas);
                        mEmpresas[num_empresas]=em;
                        num_empresas++;
                }
                *num=num_empresas;
                return mEmpresas;
}

/*****************************************************************************************************
getNombreEmp()



parametros de entrada:

*****************************************************************************************************/
char* getNombreEmp(char* nombrefichero, int id){


                int i;
                char buffer[1000];
                char* nombre;
                char* pais;
                char* nom_pais;
                char* ptr;
                FILE* entrada;


                entrada=fopen(nombrefichero,"r");


                for(i=0;i<id+2;i++){
                        fgets(buffer,1000,entrada);
                }
                ptr=strtok(buffer,",");
                nombre=malloc(sizeof(char)*(strlen(ptr)+1));
                strcpy(nombre,ptr);
                ptr=strtok(NULL,",");
                pais=malloc(sizeof(char)*(strlen(ptr)+1));
                strcpy(pais,ptr);
                nom_pais=malloc(sizeof(char)*(strlen(nombre)+strlen(pais)+4));          //4=" ()\0"
                strcpy(nom_pais,nombre);
                strcat(nom_pais," (");
                strcat(nom_pais,pais);
                strcat(nom_pais,")");


                if(entrada!=NULL){
                        fclose(entrada);
                }


                free(nombre);
                free(pais);
                return nom_pais;
}

/*****************************************************************************************************
```

Tasasde1s

Calcula las tasas de 1 del patron dentro de un año, es decir, el porcentaje de apariciones de 1s dentro
en un año.

parametros de entrada:

salida:
float* tasas, vector con las tasas de 1 para cada año
**********/
```
float* tasasde1s(empresa** mEmpresas,int num_empresas){

	int i,j;
	float* tasas=NULL;

	tasas=malloc(sizeof(float)*NUM_PATRONES);

	for(i=0;i<NUM_PATRONES;i++){
		tasas[i]=0;
	}

	for(i=0;i<num_empresas;i++){
		for(j=0;j<NUM_PATRONES;j++){
			if(mEmpresas[i]->sust[j]==1)
				tasas[j]++;
		}
	}

	for(i=0;i<NUM_PATRONES;i++){
		tasas[i]=(float)tasas[i]/num_empresas;
	}

	return tasas;
}
```

/**********
distPond

calcula la distancia ponderada de una empresa X a una empresa Y

parametros de entrada:
empresa* x, empresa X
empresa* y, empresa Y
float* tasas, vector con las tasas de 1 de cada año

salida:
int distancia, distancia de empresa X a empresa Y
**********/
```
float distPond(empresa* x, empresa* y, float* tasas){

	int i,j;
	float* distancia=NULL;
	float distPond=0.0;

	//distancia=malloc(sizeof(float)*NUM_PATRONES);

	for(i=0;i<NUM_PATRONES;i++){
		if(x->sust[i]!=y->sust[i])
		  distPond=distPond+=2.0/pow(tasas[i],0.5);
	else if(x->sust[i]==-1&&y->sust[i]==-1)
	   distPond=distPond+=1.0/pow(tasas[i],0.5);
		}

	//free(distancia);
	return distPond;
}
```

/**********
calcDistPond

calcula el parametro M que discrimina la distancia minima para que dos empresas sean vecinas

parametros de entrada:
empresa** mEmpresas, estructura de empresas
int num_empresas, numero total de empresas en la estructura
float factor,

salida:
float M, distancia minima para que dos empresas sean vecinas
**********/
```
float calcDistPond(empresa** mEmpresas, int num_empresas,float factor, float* tasas,FILE* salida){

	int i,j;
```

```
	float acumulado=0,media,tipica,M;
	float** distancias;


	distancias=malloc(sizeof(float*)*num_empresas);
	for(i=0;i<num_empresas;i++){
		distancias[i]=malloc(sizeof(float)*num_empresas);
	}


	/************ CALCULO DE MEDIA ************/
	for(i=0;i<num_empresas;i++){
		for(j=0;j<num_empresas;j++){
			if(mEmpresas[i]->id!=mEmpresas[j]->id){
				distancias[i][j]=distPond(mEmpresas[i],mEmpresas[j],tasas);
				acumulado=acumulado+distancias[i][j];
			}
			else
				distancias[i][j]=0;
		}
	}


	media=acumulado/(num_empresas*num_empresas);
	fprintf(salida,"Distancia media:%.3f\n",media);
	acumulado=0;
	/************ CALCULO DE DESVIACION TIPICA ************/
	for(i=0;i<num_empresas;i++){
		for(j=0;j<num_empresas;j++){
			acumulado=acumulado+pow((distancias[i][j]-media),2);
		}
	}
	tipica=sqrt(acumulado/(num_empresas*num_empresas));
	fprintf(salida,"Desviacion Tipica:%.3f\n",tipica);
	M=media-(factor*tipica);
	fprintf(salida,"Cutoff:%.3f\n",M);
	fflush(salida);
	/**** LIBERAR MEMORIA ****/
	for(i=0;i<num_empresas;i++){
		free(distancias[i]);
	}
	free(distancias);


	return M;
}


/*****************************************************************************************************
listaVecinosPonderados


crea la lista de vecinos de cada empresa en la estructura y calcula la media de vecinos


parametros de entrada:
empresa** mEmpresas, estructura de empresas
int num_empresas, numero total de empresas en la estructura
float distanciaVecino, resultado de la funcion calculaDistancias
float* mediavecinos, numero de vecinos medio
*****************************************************************************************************/
void listVecPond(empresa** mEmpresas, int num_empresas, float distanciaVecino,float* mediavecinos,float* tasas){


	int i,j;
    float dis=.0;
	float num_vecinosTotal=0;

	for(i=0;i<num_empresas;i++){
		for(j=0;j<num_empresas;j++){
			if(mEmpresas[i]->id!=mEmpresas[j]->id){
				dis=distPond(mEmpresas[i],mEmpresas[j],tasas);
				if(dis<distanciaVecino){
					if(mEmpresas[i]->listaVecinos==NULL)
						mEmpresas[i]->listaVecinos=CrearLista(j);
					else
						InsertarElemento(j, mEmpresas[i]->listaVecinos);
					mEmpresas[i]->num_vecinos++;
					num_vecinosTotal++;
				}
			}
		}
		addVecProxPond(mEmpresas,num_empresas,mEmpresas[i],distanciaVecino,tasas);
	}
	*mediavecinos=(num_vecinosTotal+num_empresas)/(float)(num_empresas);
	return;
}


/*****************************************************************************************************
addVecProxPond


Añade un vecino extra a una empresa, siendo la distancia entre estos de uno mas a la distancia de corte.
Genera un valor aleatorio dentro de los limites de la matriz de empresas para empezar a buscar el vecino
desde ese punto. De ese modo no siempre añadirá el vecino mas cercano en el array


parametros de entrada:
empresa** mEmpresas, estructura de empresas
int num_empresas, numero total de empresas en la estructura
float distanciaVecino, resultado de la funcion calculaDistancias
empresa* em, la empresa a la que añadirá el vecino
float* num_vecinosTotal, numero de vecinos actual, para calcular la media de vecinos
*****************************************************************************************************/
```

```
int addVecProxPond(empresa** mEmpresas,int num_empresas,empresa* em,float distanciaVecino,float* tasas){

	int i,aleatorio,anadido=0;
	float dis,disCorte=1.0;

	srand(time(NULL));

	aleatorio=rand()%num_empresas;

	while(anadido==0){
		for(i=aleatorio;i<num_empresas;i++){
			if(em->id!=i){
				dis=distPond(em,mEmpresas[i],tasas);
				if((dis<distanciaVecino+disCorte)&&(dis>distanciaVecino)){
					if(em->listaVecinos==NULL)
						em->listaVecinos=CrearLista(i);
					else
						InsertarElemento(i, em->listaVecinos);
					em->num_vecinos++;
					anadido=1;
					break;
				}
			}
		}

		if(anadido==0){
			for(i=0;i<aleatorio;i++){
				if(em->id!=i){
					dis=distPond(em,mEmpresas[i],tasas);
					if((dis<distanciaVecino+disCorte)&&(dis>distanciaVecino)){
						if(em->listaVecinos==NULL)
							em->listaVecinos=CrearLista(i);
						else
							InsertarElemento(i, em->listaVecinos);
						em->num_vecinos++;
						anadido=1;
						break;
					}
				}

			}
		}
		disCorte++;
	}
	return anadido;
}

/*************************************************************************************************
comprNoRep

comprueba que un numero no este en un vector

parametros de entrada:
empresa** mEmpresas, estructura de empresas
int numero, numero a comprobar
int* vector, vector en el que mirar
int tamano, tamaño del vector
*************************************************************************************************/
int comprNoRep(int numero,int* vector,int tamano){

	int i,repetido=0;

	for(i=0;i<tamano;i++){
		if(vector[i]==numero){
			repetido=1;
			break;
		}
	}
	return repetido;
}

/*************************************************************************************************
ruidoEnVector

funcion que devuelve el vector de un año modificado que sirve como entrada a la funcion Update

parametros de entrada:
empresa** mEmpresas, estructura de empresas
int year, año del patron que queremos recuperar
int num_empresas, numero total de empresas en la estructura
int ruido, porcentaje de ruido que queremos introducir en el vector original para luego recuperarlo
*************************************************************************************************/
int* ruidoEnVector(empresa** mEmpresas,int num_empresas,int year,int ruido,FILE* salida){

	int i,numeroCambios,aleatorio;
	int* patronYear;
	int* cambio;

	srand(time(NULL));
	numeroCambios=num_empresas*ruido/100;
```

```
	patronYear=malloc(sizeof(int)*num_empresas);
	cambio=malloc(sizeof(int)*numeroCambios);


	for(i=0;i<num_empresas;i++){
		patronYear[i]=mEmpresas[i]->sust[year];
	}


	for(i=0;i<numeroCambios;i++){					//anadido
		cambio[i]=-1;
	}


	for(i=0;i<numeroCambios;i++){
		aleatorio=rand()%num_empresas;
		while(comprNoRep(aleatorio,cambio,numeroCambios)!=0)		//con ruido=100 no sale de aqui
			aleatorio=rand()%num_empresas;
		cambio[i]=aleatorio;
		if(patronYear[aleatorio]==1)
			patronYear[aleatorio]=-1;
		else
			patronYear[aleatorio]=1;
	}
	fprintf(salida,"RECUPERACION DE PATRON: AÑO %d\n",(year+1999));
	fprintf(salida,"Ruido:%d%%\n",ruido);
	fprintf(salida,"Cambios:%d\n\n",numeroCambios);
	fflush(salida);
	free(cambio);
	return patronYear;
}

/*****************************************************************************************************
difEnVec



parametros de entrada:
*****************************************************************************************************/
int difEnVec(int* p1,int*p2,int tamano){


	int i,dif=0;


	for(i=0;i<tamano;i++){
		if(p1[i]!=p2[i])
			dif++;
	}
	return dif;
}



/*****************************************************************************************************
actualizar4



parametros de entrada:
*****************************************************************************************************/
int actualizar4(empresa** mEmpresas,int num_empresas, int* g1, float k){


	int i,y,w,dif=0,aleat,vueltas=0;
	int h[num_empresas];
	int g2[num_empresas];
	pNodo ListaVecinos;


	srand(time(NULL));
	aleat=rand()%num_empresas;

	while(vueltas<num_empresas){
		i=aleat%num_empresas;
		g2[i]=g1[i];
		ListaVecinos = mEmpresas[i]->listaVecinos;
		h[i]=0;
		while(ListaVecinos!=NULL){
			w=0;
			for(y=0;y<NUM_PATRONES;y++){
				w=w+mEmpresas[i]->sust[y]*mEmpresas[ListaVecinos->vecino]->sust[y];
			}
			h[i]=h[i]+(w*g1[ListaVecinos->vecino]);
			ListaVecinos=ListaVecinos->pNext;
		}
		g1[i]=corte(h[i],k);
		vueltas++;
		aleat++;
	}
	dif=difEnVec(g1,g2,num_empresas);
	return dif;
}

/*****************************************************************************************************
corte
```

```
parametros de entrada:
**********************************************************************************************************/
int corte(int valor,float k){


	if(valor>k)
		return 1;
	else
		return -1;
}


/**********************************************************************************************************
Update


recupera el patron de un año dado en base a los patrones de los vecinos de cada empresa


parametros de entrada:
empresa** mEmpresas, estructura de empresas
int num_empresas, numero total de empresas en la estructura
**********************************************************************************************************/
int Update(empresa** mEmpresas, int num_empresas,int year,int* modificado,float mediavecinos,FILE* salida){


	int* g0=NULL;
	int* g1=NULL;
	int i,dif=0,vueltas=0;
	float tasa1=0,k;
	FILE* puntos1=NULL;
	FILE* puntos2=NULL;


	g0=malloc(sizeof(int)*num_empresas);
	g1=malloc(sizeof(int)*num_empresas);
	puntos1=fopen("etapas.dat","w");
	puntos2=fopen("diferencia.dat","w");


	for(i=0;i<num_empresas;i++){
		g0[i]=mEmpresas[i]->sust[year];
		g1[i]=modificado[i];
		if(g1[i]==1)
			tasa1++;
		if(g0[i]!=g1[i])
			dif++;
	}


	tasa1=tasa1/num_empresas;
	k=NUM_PATRONES*mediavecinos*tasa1*tasa1*cc;


	fprintf(salida,"Vueltas \tDif etapa anterior \tDif con original\n");
	fprintf(salida,"%d\t\t0\t\t\t%d\n",vueltas,dif);
	fprintf(puntos1,"%d\t0\n",vueltas);
	fprintf(puntos2,"%d\t%d\n",vueltas,dif);
	while((dif>0)&&(vueltas<10000)){
		dif=actualizar4(mEmpresas,num_empresas,g1,k);
		vueltas++;
		fprintf(salida,"%d\t\t%d\t\t\t%d\n",vueltas,dif,difEnVec(g0,g1,num_empresas));
		fprintf(puntos1,"%d\t%d\n",vueltas,dif);
		fprintf(puntos2,"%d\t%d\n",vueltas,difEnVec(g0,g1,num_empresas));
		fflush(salida);
	}


	/*fprintf(salida,"\nVector original+ruido:\t [ ");
	for(i=0;i<num_empresas;i++){
		fprintf(salida,"%d ",modificado[i]);
	}
	fprintf(salida,"]\n");
	fprintf(salida,"Vector recuperado:\t [ ");
	for(i=0;i<num_empresas;i++){
		fprintf(salida,"%d ",g1[i]);
	}
	fprintf(salida,"]\n");
	fprintf(salida,"Vector original:\t [ ");
	for(i=0;i<num_empresas;i++){
		fprintf(salida,"%d ",g0[i]);
	}
	fprintf(salida,"]\n");*/
	free(g0);
	free(g1);


	fclose(puntos1);
	fclose(puntos2);
	return dif;
```