MASTER EN DATA SCIENCE

Curso Académico 2018/2019

Trabajo Fin de Master

Divider Greedy algorithm for performing community detection in social networks

Autor : Michel Maes Bermejo

Tutor : Jesús Sánchez-Oro Calvo

Resumen

En este proyecto abordaremos la creación de un algoritmo capaz de detectar comunidades en redes sociales, un problema en el que los algoritmos tradicionales no ofrecen soluciones rápidas. Para ello usaremos un enfoque metaheurístico basado en la división de comunidades e intentaremos mejorar sus prestaciones a lo largo de varias iteraciones.

Índice general

Capítulo 1

Introducción y motivación

Desde su nacimiento, Internet ha logrado conectar a las personas de forma sencilla. El ejemplo más claro son las redes sociales, que han crecido de forma contundente los últimos años. La rápidez con la que se transmite la información ha provocado que muchas personas las utilicen como medio de información de referencia frente a los tradicionales. Esto ha desencadenado un creciente interés de diferentes marcas e incluso de otros medios de aprovechar este fenómeno para su beneficio, aprovechando la estructura de red (o grafo) en la que se basa.

Desde el punto de vista de un científico de datos, resulta interesante estudiar la estructura que formas estas redes, que constituyen un ejemplo perfecto de un grafo. Una de las áreas más interesantes que utiliza como base este tipo de redes es la detección de comunidades, que cuenta con multitud de aplicaciones.

Actualmente existen multitud de algoritmos que abordan el problema de la detección de comunidades. Los más tradicionales y exactos no son viables para abordar los grandes volumenes de datos que generan las redes sociales, por lo que se ha optado por soluciones heuristicas, no tan exactas, pero mucho más rápidas.

La motivación de este proyecto nace de la idea de crear una solución heurística diferente a las preexistentes (normalmente basadas en algun algun algoritmo destructivo), optando por un enfoque destructivo.

Capítulo 2

Objetivos

El objetivo principal de este proyecto será la creación de un algoritmo de detección de comunidades. Para ello utilizaremos un enfoque destructivo (comenzar con todos los nodos del grafo en la misma comunidad e ir haciendo particiones) contrario a un enfoque más utilizado, constructivo (comenzar con cada nodo en su propia comunidad e ir agrupandolos). Durante la construcción, usaremos la modularidad como métrica para evaluar nuestra solución. Realizaremos una serie de iteraciones en las cuales iremos mejorando el algoritmo con el fin de obtener la mejor versión del mismo.

Finalmente, compararemos nuestro algoritmo con otros preexistentes para valorar si es efectivo el enfoque elegido. En esta fase usaremos otras métricas sobre las particiones obtenidas como la conductancia o covertura.

Capítulo 3

Tecnologías, Herramientas y Metodologías

3.1. Tecnologías

3.1.1. Java

Figura 3.1: Logo Java

Java (Figura 3.1) es un lenguaje de programación de propósito general, concurrente y orientado a objetos. Su sintaxis deriva en gran medida de C y C++. Uno de los principales atractivos de Java es su máquina virtual (JVM) que nos permite ejecutar nuestro código Java en cualquier dispositivo, independientemente de la arquitectura.

3.2. Herramientas

3.2.1. Control de versiones: Git

Figura 3.2: Logo Git

Git[1] (Figura 3.2) es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente.

3.2.2. Entorno de desarrollo: IntelliJ

Figura 3.3: Logo IntelliJ

IntelliJ[2] (Figura 3.3) es un IDE para Java desarollado por JetBrains ideado para mejorar la productividad del programador.

3.3. Metodologías

Dado que pretendemos obtener la mejor versión de nuestro algoritmo, el desarrollo seguirá una metodología iterativa-incremental[3], dónde en cada iteración se propondrán y evaluarán diferentes mejoras sobre el algoritmo.

[1] https://git-scm.com/
[2] https://www.jetbrains.com/idea/
[3] https://proyectosagiles.org/desarrollo-iterativo-incremental/

Capítulo 4

Descripción informática

En este apartado se abordará la construcción del proyecto. Este proyecto consta de una colección de algoritmos de detección de comunidades, que parte desde un algoritmo de detección aleatoria, pasando por la evolución de un algoritmo destructivo hasta obtener la mejor versión del mismo aplicando una búsqueda local.

4.1. Requisitos

4.1.1. Requisitos funcionales

Los requisitos funcionales de la aplicación son simples dado que se trata de un algoritmo:

- Debe poder leer un fichero de entrada (correspondiente al grafo a tratar).
- Dado esa entrada y un algoritmo, debe devolver los clústers resultantes al realizar la detección de comunidades.

4.1.2. Requisitos no funcionales

Los requisitos no funcionales de la aplicación se reducen a la calidad de la solución ofrecida y al tiempo de ejecución:

- Maximizar la modularidad.

- Ofrecer el menor tiempo de cálculo posible[1].

Modularidad

Las soluciones de los algoritmos de detección de comunidades se construyen máximizando su valor de modularidad, una medida utilizada para medir la fuerza de division de un grafo en clústers. Esta métrica se define cómo la fracción de enlaces que caen dentro de los clústers dados menos el valor esperado que dicha fracción hubiese recibido si los enlaces se hubiesen distribuido al azar. De forma matemática:

$$Md(S,G) = \sum_{j=1}^{max(S)} (e_{jj} - a_j^2)$$

dónde S es la solución y G el grafo, siendo e_{jj} la fracción de enlaces con ambos vértices finales en el mismo grupo:

$$e_{jj} = \frac{|\{(v,u)\epsilon E : S_v = S_u = j\}|}{|E|}$$

y a_j la fracción de enlaces con al menos un extremo en la misma comunidad:

$$a_j = \frac{|\{(v,u)\epsilon E : S_v = j\}|}{|E|}$$

siendo en ambos caso E el conjunto de las aristas del grafo.

4.2. Diseño e implementación

A continuación, se detallará la construcción incremental del algoritmo, explicando cada versión creada:

- ConstRandom
- ConstDivider
- ConstDividerGreedy
- ConstDividerGreedyLS

[1]Existen algoritmos que ofrecen soluciones óptimas a este problema, pero no son aplicables a grafos grandes dado que requieren demasiado tiempo de computo.

4.2.1. ConstRandom: Algoritmo aleatorio

En esta primera aproximación que nos sirve como punto de partida y para posterior comparación, generamos los clústers de manera aleatoria; se crea un número aleatorio de clústers y se distribuyen los nodos de forma aleatorio en ellos.

Algorithm 1 ConsRandom algorithm

```
1: procedure CONSTRANDOM(g)                          ▷ The graph 'g'
2:     S ← EmptySolution(G)
3:     numClusters ← Random(N(g))                    ▷ N returns nº of graph nodes
4:     for i = 0 to numClusters do
5:         createEmptyCluster(S)
6:     for i = 0 to N(S) do
7:         rnd ← Random(numClusters)
8:         AssignToCluster(S, i, rnd)    ▷ Assign node i to cluster rnd in solution S
9:     return S
```

Lorem ipsym

Capítulo 5

Estudio comparativo

Capítulo 6

Conclusiones del proyecto y trabajos futuros

estigmérgicos [1].

Bibliografía

[1] J. Sánchez-Oro and A. Duarte. Iterated greedy algorithm for performing community detection in social networks. 2018.