ITAM, MAESTRÍA EN CIENCIA DE DATOS

MÉTODOS NUMÉRICOS Y OPTIMIZACIÓN

Implementación de la matriz de Hashimoto

Ollin Demian Langle Chimal, Ana Isabel Millán Careaga, Raúl Zagal Rojo

30 de mayo de 2017

1. INTRODUCCIÓN

1.1. COMPLEJIDAD

Las ciencias de la complejidad tiene como principal objetivo el estudio de los sistemas. No es en sí una sola teoría, sino una colección de ellas, que se basa en herramientas multidisciplinarias. Tiene como objetivo el estudio de la dinámica colectiva, son un área de investigación interdisciplinaria que busca mejorar nuestro entendimiento de los sistemas complejos, es decir, sistemas compuestos por un gran número de partes interactuantes que además tienen una actividad agregada que es no lineal, es decir, no es una combinación lineal de la dinámica de sus partes individuales.[5]
Las ciencias de la complejidad están en un rápido crecimiento a nivel mundial gracias al reto de entender problemas en economía, medicina, epidemiología, transporte, neurociencia, genómica, reconocimiento de patrones, detección de comunidades, crimen, ecología, cambio climático, redes sociales, publicidad y demás temas de interés. A pesar de las diferencias en la naturaleza de los casos anteriormente descritos, se han econtrado muchas cuestiones en común. Por ejemplo, podemos encontrar que sistemas complejos dinámicos que van de ecosistemas a mercados finacieros y el clima, suelen tener cambios repentinos de comportamiento en los cuales puede ocurrir un giro inmediato a un régimen dinámico contrastante. Aunque predecir dichos puntos críticos, es decir, donde ocurren dichos cambios drásticos, antes de que sean alcanzados es extremandamente difícil. Trabajos en diferentes campos científicos sugieren la existencia de señales tempranas genéricas que pueden indicar si se está acercando un umbral crítico para una amplia variedad de sistemas.[3].

Consideramos la mecánica estadística como una metodología matemática para entender las propiedades de los sistemas en cuestión así como sus interacciones. Esto a menudo involucra el uso de probabilidad y diversas ramas de las matemáticas, que al ser universalmente utilizadas, generan ideas para la aplicación en otras áreas.
Los experimentos confinados a laboratorios y simples muestras estadísticas, son muchas veces demasiado simplificados e idealistas como para poder estudiar varias de las preguntas más interesantes en el área de la complejidad, por lo que se debe buscar obtener datos reales de los sistemas, así, la investigación teórica de las ciencias de la complejidad deben tener una base de datos obtenida de sistemas reales para poder hacer posible la identificación de cuestiones comunes entre ellos.

1.2. REDES COMPLEJAS

Las redes complejas son conjuntos de elementos conectados llamados nodos, además de sus interacciones que son representadas por aristas. A los nodos de una red también se les llama *vértices* y los representaremos por los símbolos $v_1, v_2, ..., v_N$, donde N es el número total de nodos en la red. Si un nodo v_i está conectado con otro nodo v_j, esta conexión se representa por una pareja ordenada (v_i, v_j), o simplemente $v_i v_j$ y se le llama arista como se mencionó anteriormente. De esta manera definimos:

Definición 1. *Una red es el par ordenado $G = (V, E)$ donde V es el conjunto de nodos y E es el de aristas existentes en ella.*

Existen dos tipos de redes, las dirigidas y las no dirigidas. Las últimas son las que tienen conexiones simétricas entre nodos, es decir, dada una red G, si $v_i v_j \in G$, entonces, $v_j v_i \in G$. Mientras que en las primeras no todas las conexiones son simétricas.

En una red, los nodos además de estar conectados también interactúan y estas interacciones dan lugar a fenómenos dinámicos interesantes. Por esto es que además de estudiar las propiedades estructurales de una red también es importante hacerlo a sus propiedades dinámicas una vez que sabemos de qué manera interactúan los nodos. Por ejemplo, las enfermedades en una sociedad no son estáticas, sino que se propagan por toda la población dando lugar a epidemias.[1]
Algunos ejemplos de propiedades dinámicas son los cambios en las distribuciones de grado en el tiempo, evolución, adaptación, decaimiento y en particular, dentro de estas propiedades dinámicas se encuentra el tema central de estudio de este trabajo que es la percolación como transición de fase y auxiliar para encontrar un proceso difusivo óptimo.

Para hacer análisis de redes se cuenta con varios tipos de representaciones algebráicas que nos dan la posibilidad de estudiar los tipos de propiedades mencionados anteriormente. Por ejemplo, se puede representar a la red por medio de una matriz que contenga la información de cuántos nodos componen a la red en cuestión y cuáles de ellos están conectados.

Definición 2. ***Matriz de Adyacencia.*** *Cualquier red puede ser representada por una matriz de adyacencia que está definida como:*

$$A_{ij} = \begin{cases} 1 \textit{ si hay un enlace del nodo j al nodo i} \\ 0 \textit{ si no lo hay} \end{cases}$$

Esta matriz no es la única que nos genera una representación matemática de una red, hay muchas otras que contienen la información necesaria de acuerdo al problema que se quiera resolver. Para los fines de este trabajo es conveniente definir la siguiente:

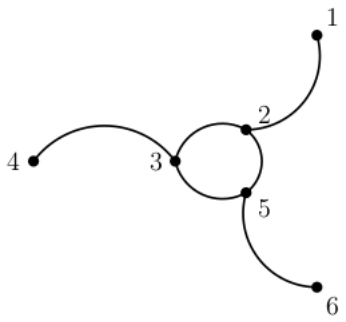
Definición 3. ***Matriz Non-Backtracking (NB).*** *También llamada Matriz de Hashimoto, es una representación de la estructura de las aristas de una red. Es usada para identificar caminos sin regresos, es decir, contiene la información de la secuencia de aristas a seguir en una caminata sobre la red tal que si se camina $k \to \ell$, la matriz da la información de qué aristas $\ell \to i$ están permitidas como siguiente movimiento excluyendo explícitamente la posibilidad $\ell \to k$.*

$$\hat{B}_{k\to\ell, i\to j} = \begin{cases} 1 \textit{ si } \ell = i \textit{ y } j \neq k \\ 0 \textit{ en cualquier otro caso.} \end{cases} \tag{1.1}$$

Esta matriz está definida para redes no dirigidas, dándole posteriormente a cada una de las aristas ambas direcciones, por lo que tenemos dos aristas dirigidas por cada no dirigida. Posteriormente se ordenan dichas aristas y estas representarán las dimensiones de nuestra matriz por lo que está definida en $2M \times 2M$.
Todas estas matrices cumplen la propiedad de ser cuadradas, por lo que se puede esperar que sus eigenvalores contengan alguna propiedad de interés. Lo que en efecto es cierto y ha dado lugar a un gran campo de investigación que son las propiedades espectrales asociadas a las redes teniendo particular interés los 2 eigenvalores de norma mayor que son la principal motivación de la realización de éste trabajo.

1.3. MATRIZ DE HASHIMOTO

Para entender mejor a la matriz de Hashimoto utilizamos el siguiente grafo como ejemplo, con $V = 6$ y $E = 6$:



La matriz NBM correspondiente es de 12×12 de la forma:

	$1\to2$	$2\to1$	$2\to3$	$2\to5$	$3\to2$	$3\to4$	$3\to5$	$4\to3$	$5\to2$	$5\to3$	$5\to6$	$6\to5$
$1\to2$	0	0	1	1	0	0	0	0	0	0	0	0
$2\to1$	0	0	0	0	0	0	0	0	0	0	0	0
$2\to3$	0	0	0	0	0	1	1	0	0	0	0	0
$2\to5$	0	0	0	0	0	0	0	0	0	1	1	0
$3\to2$	0	1	0	1	0	0	0	0	0	0	0	0
$3\to4$	0	0	0	0	0	0	0	0	0	0	0	0
$3\to5$	0	0	0	0	0	0	0	0	1	0	1	0
$4\to3$	0	0	0	0	1	0	1	0	0	0	0	0
$5\to2$	0	1	1	0	0	0	0	0	0	0	0	0
$5\to3$	0	0	0	0	1	1	0	0	0	0	0	0
$5\to6$	0	0	0	0	0	0	0	0	0	0	0	0
$6\to5$	0	0	0	0	0	0	0	0	1	1	0	0

Cuadro 1.1: Matriz Non-Backtracking para red de ejemplo.

También pensamos que es conveniente definir y ejemplificar lo que significa una caminata sin retornos.

Definición 4. *Caminatas sin retornos*

a) Los nodos iniciales y finales de la caminata no tienen que ser necesariamente diferentes.

b) Los ciclos están permitidos en la caminata NB.

c) Hay que recordar que la condición para una caminata NB es que no puede regresar por la misma arista por la que acaba de pasar, sin embargo, puede visitar los mismos nodos varias veces.

d) Las caminatas NB pueden pasar por la misma arista en repetidas ocasiones.

e) El número de nodos de la caminata NB suele llamarse el orden de la interacción.

Un claro ejemplo de este tipo de redes es el juego de *Piedra-Papel-Tijeras-Lagarto-Spock* en la que podemos encontrar caminos entre cualquiera dos nodos y todos ellos serán sin retornos.

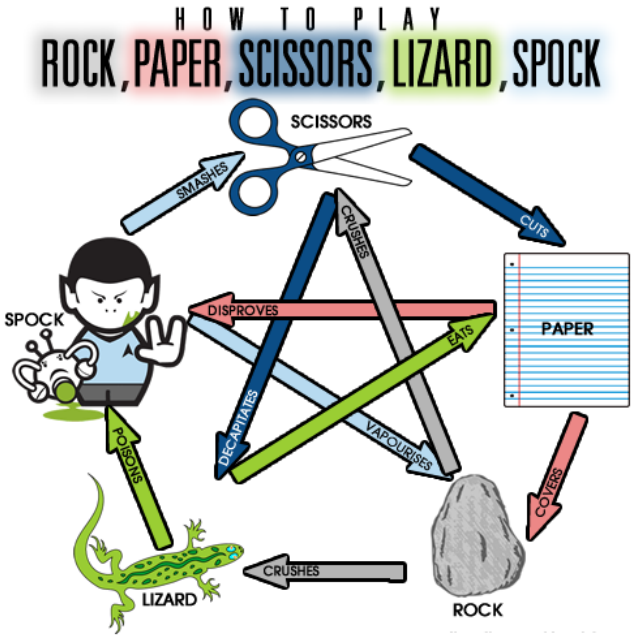


Figura 1.1: Juego de *Pidra-Papel-Tijeras-Lagarto-Spock*

En el caso de la figura 1.1, tenemos interacciones dirigidas entre cualesquiera dos nodos que son las estrategias del juego. Por la forma en la que definimos todas estas interacciones nos es posible encontrar caminos sin retornos de distintas longitudes entre 2 estrategias.

1.4. Archivos de red

Entre los diversos programas que existen para el análisis de redes ha sido complicado generar un tipo de archivo estándar para el almacenamiento e intercambio de redes. Si bien existen muchas propuestas, una de las más populares es el tipo de archivo GML (Graph Modelling Language). Este tipo de archivo permite agregar información arbitraria a los elementos de los grafos, de esta manera el autor de la red puede incluir caracterísitcas de las relaciones y de los nodos de la red. Un ejemplo de la propuesta de este tipo de archivos se muestra a continuación.

```
graph
[
  node
  [
   id A
   label "Node A"
  ]
  node
  [
   id B
   label "Node B"
  ]
  node
  [
   id C
   label "Node C"
  ]
   edge
  [
   source B
   target A
   label "Edge B to A"
  ]
  edge
  [
   source C
   target A
   label "Edge C to A"
  ]
]
```

Figura 1.2: Forma de archivo GML

Como podemos observar, el grafo está formado por nodos que se representan usando la palabra reservada node seguido de las propiedades del nodo. Mientras que las aristas se denotan con la palabra reservada edge seguido de los nodos origen y objetivo que conectan junto con sus respectivas propiedades. Como se puede observar este archivo sigue un formato de texto plano que es fácil de leer por un usuario y fácil de interpretar por una máquina.

2. MOTIVACIÓN

La implementacion de la matriz Non-Backtracking en el lenguaje C, surge del interés de por los algoritmos de detección de comunidades de redes complejas basadas en métodos espectrales [7][6][2]. Cuando se realizó ese ejercicio para redes de orden y tamaño grandes se observó que el tiempo y la demanda computacional eran muy elevados y que el cuello de botella del mismo estaba al generar de la matriz Hashimoto. De esta manera se pensó que el realizar dicha tarea *from scratch* nos permitiría tener mayor control del uso de los recursos, hacerlo más eficiente y de esta manera abrir paso a las investigaciones que son la motivación del interés por implementar.

3. IMPLEMENTACIÓN

En el proyecto buscamos transformar una red guardada en un archivo GML en su representación matricial de Hashimoto, para lo cual se realizaron los siguientes pasos:

- Lectura del archivo GML
- Construcción de la estructura de la red
- Obtención de la matriz de Hashimoto
- Paralelización del algoritmo

3.1. LECTURA DEL ARCHIVO GML Y CONSTRUCCIÓN DE LA RED

La lectura de los archivos GML está comprendida en 3 partes, 2 librerías y un script de lectura. La primera librería es donde se define la estructura de la red, el script es donde se encuentran las funciones de las que depende la lectura y la tercera librería es la que se importa a las funciones de creación de matrices y su tarea es llevar a cabo las funciones del script.

```
typedef struct {
  int entrada;        // Será un arreglo de nodos que son vecinos
                                                  //de un nodo.
  double peso;       // Peso de la arista, 1 si es no pesada.
} ARISTA;

typedef struct {
  int id;            // Identificador en el archivo gml
  int grado;         // Grado del nodo, grado de entrada
                                       //para dirigidas
  char *etiqueta;    // Etiqueta del nodo en el archivo gml.
                     //NULL si no especifica
  ARISTA *arista;    // Arreglo de structs ARISTA, una para cada vecino
} NODO;

typedef struct {
  int nnodos;        // Número de nodos en la red
  int dirigida;      // 1 = si es dirigida, 0 = si es no dirigida
  NODO *nodo;        // Arreglo de structs NODO, uno para cada arista
} RED;
```

Figura 3.1: Structs para generar la red

En la figura 3.1 podemos observar que tenemos una estructura principal llamada RED, la cual está compuesta por el número de nodos, un booleano que indica si la red es dirigida y un arreglo de structs NODO.
El struct NODO se compone de un identificador para cada uno de ellos, muchas veces en los archivos GML esto empieza en 1 entonces para facilidad de escritura en C si esto sucede lo bajamos a cero. Cada nodo también tiene la propiedad grado que es un entero que nos dice el número de vecinos, etiqueta porque hay archivos como el de Les Miserables que incluye los nombres de los personajes y también nos interesa tener esta información y un arreglo de structs ARISTA que es una lista de nodos que están conectados con el nodo en cuestión.
El struct ARISTA guarda la información de los nodos conectados al nodo en el que *estemos parados*, en el caso de ser una red dirigida son los nodos de entrada cuya arista tiene nodo de salida el antes mencionado. Y también guarda la información de un peso si nos encontramos en el caso de estar leyendo una red pesada, en caso contrario el valor es 1 para todas las aristas.
Con el script de lectura lo que se hace es leer línea por línea, identificando cada tipo de información que hay en ella y agregando las propiedades que se encuentren a cada una de nuestras estructuras definidas.

Obtención de la matriz de Hashimoto Como se mencionó anteriormente, la matriz tiene dimensiones de $2M \times 2M$ y debido a que la mayor parte de sus entradas son cero utilizamos la función *calloc* que nos permite inicializar arreglos con el valor cero en todas las entradas, esto nos otorga un gran incremento en el performance del algoritmo porque entonces sólo pasamos por los valores que potencialmente pueden ser uno. La forma en la que podemos actualizar los valores de la matriz es[7]:

$$B_{j\to i,\ell\to k} = \delta_{i\ell}\left(1-\delta_{jk}\right) \tag{3.1}$$

La implementación en C de dicha matriz tiene dos pasos.

```
int kronecker(int x, int y){
    if(x==y) return 1;
    else return 0;
}
```

Figura 3.2: Función delta de Kronecker en C

```
int generar(int i, int j, int k, int l, int ** matriz){
  for(i=0;i<red.nnodos;i++){
    for(j=0;j<red.nodo[i].grado;j++){
      arista_2=0;
      for(k=0;k<red.nnodos;k++){
        for(l=0;l<red.nodo[k].grado;l++){
          matriz[arista_1][arista_2] = \
           kronecker(red.nodo[k].id,red.nodo[i].arista[j].entrada)* \
           (1-kronecker(red.nodo[i].id,red.nodo[k].arista[l].entrada));
        arista_2++;
        }
      }
      arista_1++;
    }
  }
}
```

Figura 3.3: Función de creación de la matriz de Hashimoto

Como podemos observar en la figura 3.3 fue una implementación directa de la ecuación 3.1 y por la forma en que se recorren las estructuras lo que estamos haciendo tiene una complejidad algorítmica de $\mathcal{O}(e^2)$ con e el número de aristas. Esto es debido a que por cada arista

se recorren todas las aristas, de esta manera entramos a los espacios de la matriz de manera ordenada y obteniendo resultados deterministas gracias al etiquetado de los nodos en la estructura de la red.

Se compararon diferentes formas de implementar el algoritmo utilizado para la generación de la matriz Non-Backtracking (NBM) para observar el desempeño de diferentes técnicas y tecnologías. Es por eso que para éste análisis decidimos utilizar:

- Implementación secuencial en C
- Implementación secuencial en Julia
- Paralelización usando OpenMP

El algoritmo para generar la matriz de Hashimoto se basa en hacer varios barridos sobre la matriz de adyacencias, por lo que se presentan varios ciclos for anidados dentro de la implementación. Observando esta característica consideramos que la implementación de algoritmos y técnicas de paralelización nos ayudarán a aumentar la eficiencia en el cálculo de esta matriz.

4. RESULTADOS

A continuación se muestran los resultados para redes de prueba que se pueden obtener del repositorio de redes de la Universidad de California Irvine [8]. Se utilizó una computadora tipo laptop con procesador Intel i7-6700HQ a 2.6 GHz y 32GB de DRAM a 2100MHz junto con el compilador GNU Compiler Collection (GCC) versión 6.3.1-2.

Cuadro 4.1: Resultados red de Zachary's Karate Club. (34 nodos - 78 aristas)

Implementación	Tiempo User (s)	Tiempo System (s)	Tiempo Total	% CPU
C - Secuencial	0.00	0.00	0.007 s	93
Julia - Secuencial	5.72	0.06	5.785 s	99
OpenMP - 2 threads	0.01	0.00	0.011 s	108
OpenMP - 4 threads	0.01	0.00	0.007 s	208
OpenMP - 6 threads	0.05	0.00	0.017 s	304
OpenMP - 8 threads	0.08	0.01	0.018 s	475

Cuadro 4.2: Resultados red de Delfines. (62 nodos - 159 aristas)

Implementación	Tiempo User (s)	Tiempo System (s)	Tiempo Total	% CPU
C - Secuencial	0.01	0.00	0.015 s	96
Julia - Secuencial	5.81	0.04	5.856 s	99
OpenMP - 2 threads	0.06	0.01	0.041 s	170
OpenMP - 4 threads	0.06	0.00	0.022 s	300
OpenMP - 6 threads	0.08	0.02	0.023 s	422
OpenMP - 8 threads	0.31	0.01	0.061 s	521

Cuadro 4.3: Resultados red de Los Miserables. (77 nodos - 254 aristas)

Implementación	Tiempo User (s)	Tiempo System (s)	Tiempo Total	% CPU
C - Secuencial	0.02	0.01	0.032 s	98
Julia - Secuencial	6.52	0.06	6.587 s	99
OpenMP - 2 threads	0.06	0.01	0.043 s	149
OpenMP - 4 threads	0.11	0.01	0.047 s	245
OpenMP - 6 threads	0.15	0.01	0.043 s	378
OpenMP - 8 threads	0.31	0.01	0.068 s	480

Cuadro 4.4: Resultados red de NCAA. (115 nodos - 613 aristas)

Implementación	Tiempo User (s)	Tiempo System (s)	Tiempo Total	% CPU
C - Secuencial	0.08	0.02	0.108 s	99
Julia - Secuencial	8.13	0.02	8.153 s	99
OpenMP - 2 threads	0.19	0.04	0.192 s	120
OpenMP - 4 threads	0.25	0.04	0.178 s	165
OpenMP - 6 threads	0.33	0.04	0.179 s	202
OpenMP - 8 threads	0.43	0.05	0.180 s	263

Como prueba final para los algoritmos ocupamos una red de 667 nodos y 18084 aristas que consta de usuarios de Twitter junto con sus conexiones.

Cuadro 4.5: Resultados red de Twitter. (667 nodos - 18084 aristas)

Implementación	Tiempo User (s)	Tiempo System (s)	Tiempo Total	% CPU
C - Secuencial	73.77	5.28	1:19.59 min	99
Julia - Secuencial	2314.65	2.38	38:38.68 min	99
OpenMP - 2 threads	132.50	10.12	2:03.65 min	115
OpenMP - 4 threads	132.17	8.14	1:57.50 min	146
OpenMP - 6 threads	189.93	8.72	1:55.74 min	171
OpenMP - 8 threads	216.58	8.34	1:59.19 min	188

5. Discusión

Observando los resultados mostrados en la tabla 4.5 encontramos algunos resultados no tan esperados. Primeramente observamos que dentro de las implementaciones secuenciales, el desempeño de Julia es muy inferior al desempeño encontrado en la versión secuencial de C. La métrica del porcentaje de CPU utilizado nos asegura que se hizo la paralelización del código, aunque observamos que no se ocupa la capacidad total de nuestro procesador.
Vemos que los tiempos para los diferentes casos de N threads no presentan una diferencia significativa entre ellos. Además, de que a mayor número de threads no se asegura un mejor desempeño, ya que observamos una fluctuación entre los mismos pero en general se observa una curva en la que el performance del algoritmo en paralelo tiene un pico óptimo al usar entre 4 y 6 threads, aunque no es constante y seguramente depende de la topología de la red además de factores en los que no tenemos control. Se esperaba un mejor desempeño para el esquema paralelizado usando OpenMP, sin embargo, como se puede observar la versión secuencial en C tuvo un mejor rendimiento que cualquiera de las otras.
OpenMp permite al usuario paralelizar sin la necesidad de definir a detalle el método de paralelización ya que se le delega al compilador y al runtime system la forma en que cada thread será ejecutado. Consideramos que la razón del mal desempeño de OpenMP radica en la actualización de dos variables internas en los ciclos siendo independientes una de la otra.

Además de éste trabajo ya existen precedentes de que la paralelización de algoritmos de redes complejas aumenta el overhead y hace que el performance de los mismos disminuya[4].

6. Conclusión

Tener control de la implementación desde un inicio nos permite usar el tipo de estructura más orientada al problema que queremos atacar, en nuestro caso logramos vencer la marca de referencia en Julia secuencial.
Se observó que no hubo mucha diferencia en el tiempo total y tiempo system entre todas las pruebas con OpenMP sin importar el número de threads que se les asignó. Igualmente tampoco se puede asegurar que a mayor número de threads el tiempo total sea menor.
Como podemos observar, no en todos los problemas las técnicas de paralelización tienen un mejor desempeño respecto a su implementación secuencial.
Consideramos que sería conveniente revisar otra herramienta de paralelización que permita realizar afinaciones sobre el acceso a memoria y coordinación, con lo que pensamos que se podrían obtener mejores resultados.
El hecho de no paralelizar sobre arreglos alojados en memoria con tamaños definidos sino en distintos arreglos de diversos tamaños debido a que el número de vecinos de cada nodo es en general diferente, añade otra razón para la dificultad de paralelizar eficientemente.

7. CÓDIGO

Figura 7.1: El código está disponible en este repositorio.

REFERENCIAS

[1] Maximino Aldana. “Redes Complejas”. Notas. 2006.

[2] Florent Krzakala y col. “Spectral redemption in clustering sparse networks”. En: *Proceedings of the National Academy of Sciences of the United States of America* 110.52 (2013), págs. 20935-20940. ISSN: 0027-8424, 1091-6490. DOI: 10.1073/pnas.1312486110. arXiv: 1306.5550. URL: http://www.pnas.org/content/110/52/20935%7B%5C%%7D5Cnhttp://www.ncbi.nlm.nih.gov/pubmed/24277835%7B%5C%%7D5Cnhttp://www.pnas.org/content/110/52/20935.full.pdf.

[3] et. al. Martin Scheffer. “Early-warning signals for critical transitions”. En: *Nature* 461 (2009), págs. 53-59.

[4] Frank McSherry. *Scalability! But at what Cost?* 2015. URL: http://www.frankmcsherry.org/graph/scalability/cost/2015/01/15/COST.html (visitado 29-05-2017).

[5] Anna Best Michael MacDonald Brett Jackson. *Complexity Science in Brief*. 2012. URL: http://www.uvic.ca/research/groups/cphfri/assets/docs/Complexity_Science_in_Brief.pdf (visitado 03-10-2012).

[6] M. E. J. Newman. “Spectral community detection in sparse networks”. En: (2013). arXiv: 1308.6494. URL: http://arxiv.org/abs/1308.6494.

[7] Abhinav Singh y Mark D. Humphries. “Finding communities in sparse networks”. En: *Scientific Reports* 5 (2015), pág. 8828. ISSN: 2045-2322. DOI: 10.1038/srep08828. arXiv: 0512038 [q-bio]. URL: http://www.nature.com/doifinder/10.1038/srep08828%7B%5C%%7D5Cnhttp://arxiv.org/abs/q-bio/0512038.

[8] UCIrvine. *Network Repository*. URL: https://networkdata.ics.uci.edu/index.php.