FACULTAD DE INGENIERIA
Universidad de Buenos Aires

VISUALIZACIÓN DE REDES COMPLEJAS

TESIS DE GRADO DE
INGENIERÍA INFORMÁTICA

FACULTAD DE INGENIERÍA
UNIVERSIDAD DE BUENOS AIRES

TESISTA: MARIANO GASTÓN BEIRÓ
DIRECTOR: DR. ING. JOSÉ IGNACIO ALVAREZ-HAMELIN

AGOSTO 2008

Agradecimientos

Al terminar esta tesis después de todo un año de trabajo, no puedo dejar de agradecer a aquellas personas vinculadas a la misma:

A los miembros del jurado, por haber aceptado la lectura y evaluación de este trabajo.

A José Ignacio por enseñarme tantas cosas y por tenerme inmensa paciencia. Por incentivarme al estudio y a la investigación.

A Jorge, por sus correcciones y sugerencias.

También en lo personal quiero agradecer:

A mis padres, Mary y Marcelo, por apoyarme siempre en todo, por confiar en mí y por su enorme afecto. Y a mi hermano Sergio por soportarme. Gracias por haberme dado tanto, y por todo lo que se sacrifican día a día.

A toda mi familia, que es parte de mi, me aconseja y empuja.

A mis profesores de la carrera y del cole, de quienes he aprendido mucho y les estaré eternamente agradecido.

A mis amigos, sin quienes se haría muy difícil caminar y enfrentar los problemas y responsabilidades. Gracias por su alegría, por aguantarme y por estar siempre!

A mis compañeros de la carrera con quienes compartimos días de estudio y noches sin dormir terminando tps, y que me ayudaron para llegar hasta acá.

A mis compañeros del trabajo, por entenderme cada vez que les dije "tengo que irme a trabajar con la tesis...".

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. La complejidad de los sistemas en el mundo actual

El permamente desarrollo de la tecnología en las últimas décadas, ha permitido poner en práctica el conocimiento científico de una forma nunca antes vista. Entidades que antes sólo podían conocerse en un nivel macroscópico, pueden ahora explorarse en profundidad gracias a los avances en la informática y la electrónica.

El Proyecto Genoma Humano, por ejemplo, ha permitido conocer los 27.000 genes presentes en los seres humanos, contenidos en 2900 millones de pares de bases de ADN, y su disposición en los cromosomas del núcleo celular [33].

La Internet cuenta hoy en día con una cantidad estimada de 1000 millones de computadoras [2], y la WWW (World Wide Web) con 100 millones de sitios [1].

Entender el funcionamiento de estos sistemas, su estructura e interconexión requiere de herramientas que permitan simplificar la información a un nivel comprensible por el ser humano, discriminando lo esencial de lo que es accesorio.

1.2. Visualización de la Información

Visualizar significa formar en la mente una imagen visual de un concepto abstracto [1], es decir algo incapaz de ser visto. Esa imagen mental excede a la percepción de los sentidos, y como tal nos aproxima al conocimiento; es una

[1]Diccionario de la Real Academia Española

construcción que va más allá de la percepción sensorial. La Visualización de la Información es entonces la utilización de representaciones visuales e interactivas por computadora de datos abstractos, con el fin de aumentar su comprensión [14].

Esta área abarca el estudio de técnicas y algoritmos para plasmar en una imágen una cantidad normalmente abundante de datos, de forma que se pueda obtener rápidamente la información esencial que subyace en ellos a partir de la simple observación del elemento visual. De esta forma constituye una herramienta valiosa para los expertos de distintas áreas, que pueden a través de ella validar modelos o teorías, sintetizar la información, comprender, expresar resultados y comunicarse.

La Visualización de la Información difiere de la Visualización Científica, en tanto que esta última disciplina busca a través de la imágen obtener una representación realista de datos generalmente espaciales, concretos, como lo son el cuerpo humano o la superficie de un sólido.

1.3. Notación utilizada y definiciones previas

En este trabajo, las redes complejas se representarán a través de grafos. Un *grafo* G se notará como $G = (V, E)$, en donde (V,E) simboliza una pareja de conjuntos:

1. V es el conjunto de *vértices*

2. E es el conjunto de *aristas*, es decir conexiones entre vértices

El *cardinal* de un conjunto A se símbolizará $|A|$. Por ejemplo la cantidad de vértices de un grafo $G = (V, E)$ es $|V|$, mientras que $|E|$ es el número de aristas.

Cada *vértice* se representará como v_i, en donde $1 \leq i \leq |V|$. A lo largo del trabajo, se empleará el término *nodo* como sinónimo de *vértice*.

Las *aristas* (o *ejes*) se notarán e_{ij}, simbolizando una conexión entre el vértice v_i y el vértice v_j. Los vértices v_i y v_j se denominan *extremos* de la arista e_{ij}, y se dice que la arista *incide* en ellos. Una arista que conecta a un vértice consigo mismo se denomina *bucle*.

Un *grafo simple* es aquel que no posee aristas múltiples entre un mismo par de vértices, ni tampoco bucles.

Se dice que dos vértices v_i y v_j son *vecinos* o *adyacentes* si existe una conexión entre ellos, es decir si $e_{ij} \in E$.

El *grado* de un vértice v_i se define como la cantidad de aristas que inciden en él, y se simboliza $d(v_i)$. En el caso de grafos simples coincide numérica-

mente con la cantidad de vecinos que posee. Asímismo, el conjunto formado por los vecinos de v_i se denomina *vecindad* y se denota N_i o $N(v_i)$.

El *grado máximo* de un grafo, $d_{\text{máx}}$, es el máximo de todos los grados de sus nodos, es decir

$$d_{\text{máx}} = \text{máx}\{d(v_i), v_i \in V\} \tag{1.1}$$

Un *camino* entre dos vértices v_i, v_j es una secuencia ordenada de aristas $\{e_{i,\alpha_1}, e_{\alpha_1,\alpha_2}, ..., e_{\alpha_{n-1},\alpha_n}, e_{\alpha_n,j}\}$ tal que cada arista en la secuencia comparte un vértice extremo con la arista siguiente, la primera arista tiene a v_i como vértice, y la última posee a v_j. Un *camino mínimo* entre dos vértices es aquél que tiene la mínima cantidad de aristas. Dos caminos se dicen *arista-disjuntos* cuando no comparten aristas. Dos vértices están *conectados* cuando existe al menos un camino entre ellos.

La *distancia* entre dos vértices v_i y v_j, $r(v_i, v_j)$ es el menor número de aristas que conforman un camino entre ellos. El *diámetro* de un grafo, D, es la mayor de las distancias entre todos los pares de nodos. Formalmente:

$$D = \text{máx}\{r(v_i, v_j), v_i, v_j \in V\} \tag{1.2}$$

El *coeficiente de clustering* (cc_i) de un vértice es la cantidad de conexiones entre sus vecinos con respecto al máximo posible, es decir

$$cc_i = \frac{2 \cdot n_{link_i}}{d(v_i) \cdot (d(v_i) - 1)} \ , \tag{1.3}$$

en donde n_{link_i} es la cantidad de aristas entre vecinos de v_i:

$$n_{link_i} = |\{e_{jk} \in E / j, k \in N_i\}| \tag{1.4}$$

Representa la probabilidad de que dos vecinos de un vértice estén conectados entre sí.

Un grafo se dice *conexo* cuando todos sus vértices se encuentran conectados entre sí. En general un grafo es *k-arista-conexo* cuando existen al menos k caminos arista-disjuntos entre todo par de vértices.

Un subgrafo, conectado, maximal de G se denomina *componente conexa (connected component)*.

Dado un grafo $G = (V, E)$ y un conjunto de vértices $V_1 \subseteq V$, se llama *subgrafo de G inducido por V_1* al grafo $H = (W, F)$ tal que $W = V_1$ y F es el subconjunto de aristas en E que tienen ambos extremos en V_1.

Dado un conjunto S, un subconjunto $V \subseteq S$ es *maximal* con relación a una propiedad P si no existe ningún otro conjunto en S que contenga propiamente a V y que satisfaga la propiedad P. Análogamente, V es *minimal* con relación a P si no existe otro conjunto en S que sea contenido por V y que satisfaga P.

Una *partición* de un conjunto V es una familia de subconjuntos $\{V_1, V_2, ..., V_n\}$ tal que

1. $V_i \neq \emptyset, 1 \leq i \leq n$
2. $\bigcup_i V_i = V$
3. $V_i \cap V_j = \emptyset, 1 \leq i \leq n, i \neq j$

Un *corte* $[A, B]$ es una partición de los vértices de un grafo en dos conjuntos A y B. Una arista se dice que *cruza el corte* si tiene un extremo en A y el otro en B. El *tamaño de un corte* es la cantidad de aristas que lo cruzan, y se simboliza $|[A, B]|$.

En el caso de tratar varios grafos o subgrafos en forma simultánea, se empleará un subíndice para indicar a cuál de ellos se refiere una expresión. Por ejemplo, $d_G(v_1)$ es el grado del vértice v_1 en G, y $r_{G_1}(v, w)$ es la distancia entre v y w en el grafo G_1.

1.4. Sistemas Complejos

Un sistema complejo está formado por varios entes independientes que se interrelacionan, surgiendo de ello un comportamiento en común. Normalmente se representan mediante su red de interacción denominándose *red compleja*, y su topología reúne las siguientes características [8]:

1. Comportamiento emergente (*emerging behavior*), es decir la capacidad de lograr una complejidad o sofisticación global a través de la integración de vínculos relativamente simples. Está ligado a la capacidad de auto-organización (*self-organization*).

2. Una distribución de grados de cola larga: generalmente del tipo ley de potencias (*power law*), denominada libre de escala (*scale-free*).

Según Barabási y Albert [8], la distribución de grados de estas redes es consecuencia de dos hechos:

1. El permanente crecimiento de la red, debido a la incorporación de nuevos nodos.

2. Los nuevos nodos suelen conectarse preferencialmente a aquellos nodos ya bien conectados. Esta propiedad se conoce como *preferential attachment* (vinculación preferencial).

Sin embargo, puede surgir también debido a la optimización de varios parámetros de desempeño para la conexión de los nuevos nodos, tal como lo expresan Fabrikant *et al.* [17]. No se conoce una explicación general hasta el momento.

1.4.1. Distribución de grados *scale-free*

Las redes complejas habitualmente presentan nodos cuyos grados siguen una distribución de probabilidades de tipo ley de potencias, que constituyen un caso particular de las distribuciones de cola larga. Estas distribuciones tienen una función de densidad de probabilidad como la siguiente:

Dado un grafo $G = (V, E)$, tomando $v \in V$ y llamando d_v al grado de v,

$$p_G(d_v) = A \cdot d_v^{-B} \tag{1.5}$$

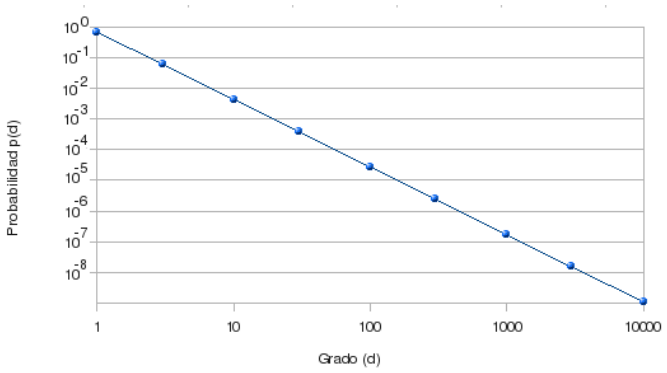


Figura 1.1: Distribución potencial de grados con $B = 2,2$.

Estudios realizados por los hermanos Faloutsos muestran que en el caso de la Internet, el coeficiente B se encuentra comprendido entre 2,1 y 2,4 [18]. Mientras que en general, las redes complejas presentan coeficientes entre $2 \leq B \leq 3$. Es importante remarcar que para estos exponentes las distribuciones no poseen momento de segundo orden, es decir que la varianza tiende a infinito cuando el tamaño de la red aumenta.

La distribución se denomina también libre de escala (*scale-free*) porque la curva de la distribución potencial es autosimilar, es decir que acercándose a una parte de la curva a nivel microscópico, se percibe la misma forma geométrica que a un menor nivel de detalle.

1.4.2. La vinculación preferencial

El modelo de vinculación preferencial fue propuesto por Price en 1976 [30], mientras estudiaba la red de citas entre artículos científicos. Tratando de explicar la razón de que esta red tuviera una distribución de grados de ley de potencias, propuso un modelo de ventaja acumulativa (*cumulative advantage*): La probabilidad con que un artículo es citado es directamente proporcional a la cantidad de citas que ya tiene. Es decir que al escribir un artículo,

su autor preferirá citar otros bien conocidos a través de sus citas, en lugar de escoger artículos poco conocidos.

Su explicación es actualmente aceptada por la comunidad científica, y se conoce con el nombre de *preferential attachment* (vinculación preferencial) acuñado por Barabási y Albert [8].

1.4.3. *Assortative mixing*

Un concepto interesante de ser evaluado en los sistemas complejos es el de la correlación de conexidad (en inglés, *assortative mixing*), que mide el nivel en que los nodos de mayor grado están conectados.

Aquellas redes en que los nodos de grado alto muestran preferencia por conectarse a otros nodos de grado alto, se denominan de tipo *assortative*. Este comportamiento es propio de las redes sociales: una persona muy popular tiene contactos que a su vez son muy populares.

En cambio, las redes como Internet y la Web tienen un comportamiento de tipo *disassortative*, en que los nodos de grado elevado están mayormente conectados a nodos de grado menor que el propio [25].

1.5. Ejemplos de Sistemas Complejos

A continuación se describirán algunos de los Sistemas Complejos estudiados en la actualidad [26].

1.5.1. Redes Sociales

Los sociólogos han sido pioneros en el estudio de redes en el mundo real, a partir de las redes sociales. Ya en 1930 Elton Mayo había estudiado en Chicago las relaciones informales entre los trabajadores en las fábricas. Una red social es una red de personas vinculadas entre ellas, ya sea por amistad, contacto o consanguinidad, por ejemplo.

En la actualidad, el crecimiento de Internet generó el surgimiento de comunidades virtuales con millones de integrantes, como MySpace y LinkedIn.

Una característica frecuente en estas redes es la propiedad de mundo pequeño (*small world*), que indica que tomando un par de personas en el mundo, se puede llegar de una a otra con una secuencia de sólo cuatro intermediarios, en donde cada intermediario es conocido por el anterior, y conoce a su vez al siguiente.

Propiedad de mundo pequeño *(Small world)*

En las redes de tipo *small world* se puede llegar de un nodo a otro a través de un camino corto, incluso cuando la cantidad de nodos es muy grande. Lo que sucede es que existen típicamente un grupo de nodos muy conectados entre sí y que actúan como *hubs*, es decir que forman parte del camino mínimo entre muchos pares de nodos. Formalmente, estas redes deben cumplir con las dos siguientes características [34]:

1. Un elevado coeficiente de clustering.

2. Una distancia promedio entre nodos pequeña.

1.5.2. Redes Biológicas

Los biólogos han aplicado herramientas del análisis de redes al estudio de las cadenas alimentarias, las redes neuronales y las redes de interacción entre proteínas, entre otras. Todas ellas han mostrado en general tener comportamiento *disassortative* [25].

1.5.3. Redes Tecnológicas

Son aquellas creadas por el hombre para la distribución de recursos. Han sido objeto de estudio en cuanto a su topología las redes eléctricas, las redes de aeropuertos y de transporte, las redes de telefonía y la Internet, entre otras.

La Internet es una red que evoluciona a una velocidad mucho mayor que cualquiera de las otras redes tecnológicas, ya que sus nodos se conectan y desconectan permanentemente, a la vez que se agregan o cancelan rutas. Esta dinámica, sumada al tamaño de la red y al hecho de que el control no esté centralizado (es decir no hay un único organismo que que mantenga toda la infraestructura) hacen que sea imposible tomar una "fotografía" de la Internet en un momento concreto. Los científicos deben reconstruir la red a partir de datos de exploraciones que sólo muestran conexiones punto a punto, y la estructura se estudia entonces con distintos niveles de detalle:

1. A nivel de IPs: Algunas exploraciones de Internet que se basan en datos obtenidos por traceroute no son capaces de reconocer la subyacencia de un mismo ruteador en nodos con IPs distinta: es decir, que varias IPs pertenezcan a un mismo ruteador físico. Entonces a cada IP se le asigna un nodo distinto, lo que implica un sesgo a la hora de interpretar la conexidad de la red.

2. A nivel de ruteadores: Los nodos de la red son computadoras de propósito especial que controlan el movimiento de datos (ruteadores).

3. A nivel de Sistemas Autónomos: Los Sistemas Autónomos o *Autonomous Systems* (ASes) son conjuntos de computadoras cuyo ruteo interno de datos se realiza a través de un protocolo local, pero el intercambio de datos entre ellos se realiza a través de la Internet pública. Normalmente un AS se corresponde con un ISP (*Internet Service Provider*: Proveedor de Servicios de Internet) o una universidad, por ejemplo. Un AS se compone de ruteadores conectados entre sí y con una administración central.

También se pueden definir otras vistas, pero ésto escapa a los objetivos de la presente tesis.

1.5.4. Redes de Información

El ejemplo más importante de estas redes es la WWW (World Wide Web). Cada nodo de la red es un sitio web, y los enlaces son direccionales y reflejan los hipervínculos entre ellos. Estudios realizados por Barabási-Albert [3] mostraron que la Web cumple con la propiedad de mundo pequeño, siendo su diámetro de alrededor de 19, y que la distribución de grados de sus nodos sigue una ley de potencias. Si el tamaño de la Web se aumentara en un 1000 %, su diámetro pasaría de 19 a tan sólo 21.

Las exploraciones científicas de la WWW se realizan siguiendo los hipervínculos, y son por lo tanto sesgadas al igual que las de Internet.

También son redes de información las redes de citaciones entre artículos mencionadas anteriormente, y que dieron origen al estudio de la vinculación preferencial, y las redes de colaboraciones científicas.

1.6. Organización

La presente tesis abarca el desarrollo de un algoritmo de visualización de Sistemas Complejos, LaNet-vi. El presente capítulo ha introducido al lector en la necesidad del estudio de los sistemas complejos, describiendo sus principales características. El próximo capítulo hace una revisión de algunos de los algoritmos y métodos de visualización de sistemas complejos más conocidos. En el capítulo 3 se desarrollan los fundamentos de LaNet-vi, sus funcionalidades y los algoritmos aplicados. El capítulo 4 muestra los resultados de aplicar LaNet-vi para una amplia variedad de redes. El capítulo 5 extrae las

conclusiones de la tesis y el trabajo futuro que puede surgir a partir del presente aporte. Finalmente, los apéndices incluyen la descripción de las clases que conforman el código fuente y el artículo surgido de este trabajo y publicado en el *NJP (New Journal of Physics)* en un especial sobre visualización en física (*Focus Issue on Visualization in Physics*).

Capítulo 2

Estado del arte

A continuación se hará un repaso por algunos de los algoritmos de visualización de redes más relevantes existentes en la actualidad. Estos algoritmos buscan reflejar en la imagen propiedades de la red, tratando a la vez de:

1. Posicionar los nodos en forma dispersa en toda la imagen, de manera que todos sean visibles.

2. Minimizar en lo posible la cantidad de cruces entre ejes, con el objetivo de mostrar la totalidad de ejes.

Para abordar su estudio, se realizará una clasificación a partir del concepto empleado por cada algoritmo. No obstante, existen algoritmos mixtos que toman elementos de más de un método a la vez. Los métodos que trataremos son los siguientes:

1. Métodos dirigidos por fuerzas.

2. Métodos basados en descomposición en clusters.

3. Métodos basados en descomposición en k-núcleos.

2.1. Dirigidos por fuerzas

Los métodos de visualización de redes que emplean posicionamiento dirigido por fuerzas aplican la metáfora de asignar a cada nodo una esfera de acero, y a cada enlace un resorte, comenzando con los nodos en algún estado inicial y dejando que el sistema evolucione conforme a las leyes físicas, hasta alcanzar un estado estable de energía mínima (equilibrio estático). El primer algoritmo de este tipo es el desarrollado por Eades [16].

Existen variantes del mismo, como el algoritmo de Kamada-Kawai [24] y el de Fruchterman-Reingold [19]. Todos ellos se aplican en forma iterativa, una cantidad predeterminada de veces, o hasta que se cumplan condiciones energéticas de terminación.

El método tiene serios inconvenientes:

1. La cantidad de iteraciones necesarias no ha sido determinada, y no está claro en qué forma depende de la red como para acotarla. Existe un criterio ad-hoc de realizar 100 iteraciones que para la mayoría de los grafos sería suficiente [19], pero nuevamente se trata de un criterio totalmente empírico.

2. La complejidad es del orden de $O(n^2)$ para una sóla iteración, lo cual impide su utilización para redes muy grandes.

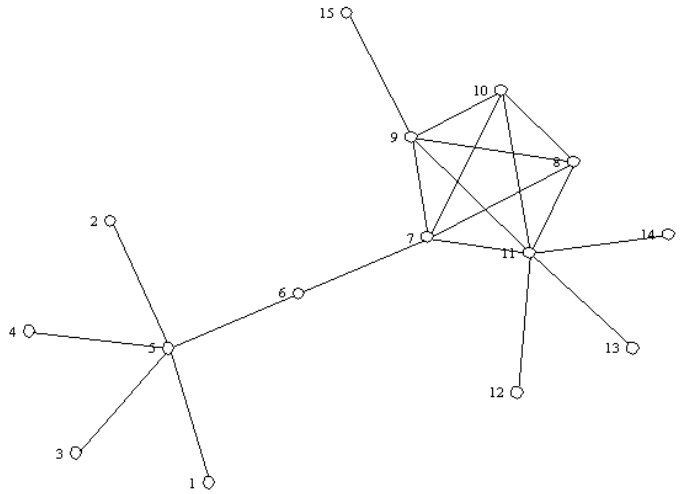


Figura 2.1: Red de 15 nodos visualizada con un algoritmo dirigido por fuerzas. (Imagen generada con Network Workbench [28]).

2.2. Descomposición en clusters

Es conveniente aclarar que en la jerga de estos métodos el concepto de *cluster* representa a un conjunto de nodos muy conectados entre sí. Sin embargo, siendo este concepto bastante amplio, el significado utilizado en el resto de la tesis (ver definición 4 en sección 3.1.2) es distinto al de esta sección.

2.2.1. Descomposición espectral

El análisis espectral de redes parte de la matriz de adyacencias del grafo de la red y realiza la descomposición en autovalores y autovectores. Se muestra que los autovectores asociados a los autovalores más grandes son efectivos para particionar la red en clusters. Se observan los pesos de cada nodo dentro de esos autovectores, y se agrupan los nodos de pesos similares. Normalmente los autovalores más pequeños no se utilizan porque tienden a resaltar características locales, por ejemplo una conexión simple entre dos nodos que pertenecen a clusters distintos. [20].

Ejemplo

Para la red de la sección anterior, la descomposición en autovalores y autovectores genera el siguiente conjunto de autovalores:

$$\Lambda = \{4{,}25; 2{,}26; 1{,}09; 0{,}57; 0{,}20; 0{,}00; -1{,}00; -1{,}31; -1{,}55; -2{,}17; -2{,}33\} \tag{2.1}$$

Para el mayor autovalor, $\lambda_1 = 4{,}25$, el autovector asociado es

$$v_1 = \begin{bmatrix} 0{,}0078 \\ 0{,}0078 \\ 0{,}0078 \\ 0{,}0078 \\ 0{,}0331 \\ 0{,}1095 \\ 0{,}4327 \\ 0{,}4119 \\ 0{,}4312 \\ 0{,}4119 \\ 0{,}4758 \\ 0{,}1119 \\ 0{,}1119 \\ 0{,}1119 \\ 0{,}1014 \end{bmatrix}, \tag{2.2}$$

La figura 2.2 muestra la matriz de adyacencias, y la 2.3 los pesos que asocia el primer autovector a cada nodo. Se observa que a partir de este autovector se podría fácilmente agrupar a los nodos de la red en 4 clusters, como se ve en la figura 2.4. Inclusive, si se continuara con el segundo autovalor, $\lambda_2 = 2{,}26$, se podría subdividir cada cluster, separando por ejemplo el nodo 6 del resto de los nodos del cluster amarillo.

Este particionamiento que en nuestro ejemplo se realiza a simple vista, puede automatizarse empleando algún algoritmo de clustering como K-Means [23], empleando el peso de cada nodo en el autovector como único atributo.

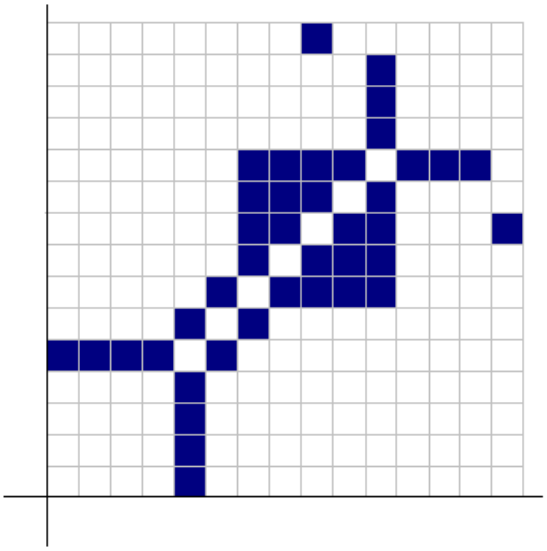


Figura 2.2: Matriz de adyacencias de la red.

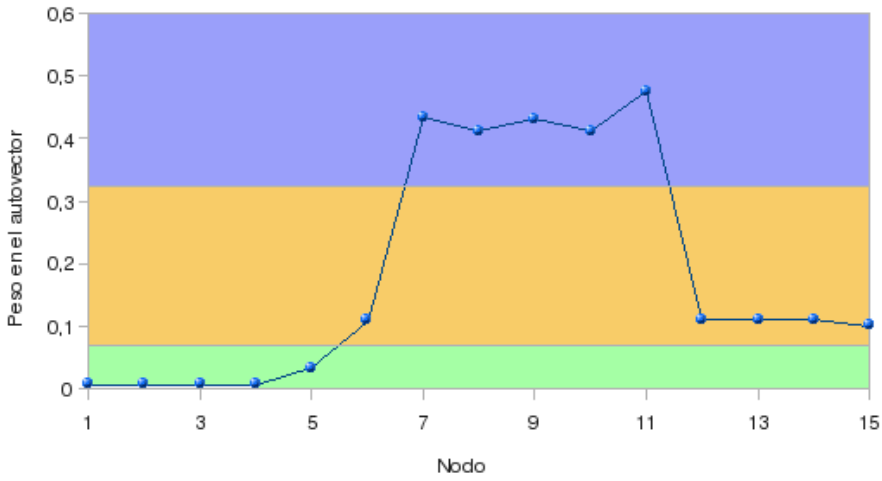


Figura 2.3: Componentes del autovector asociado al mayor autovalor.

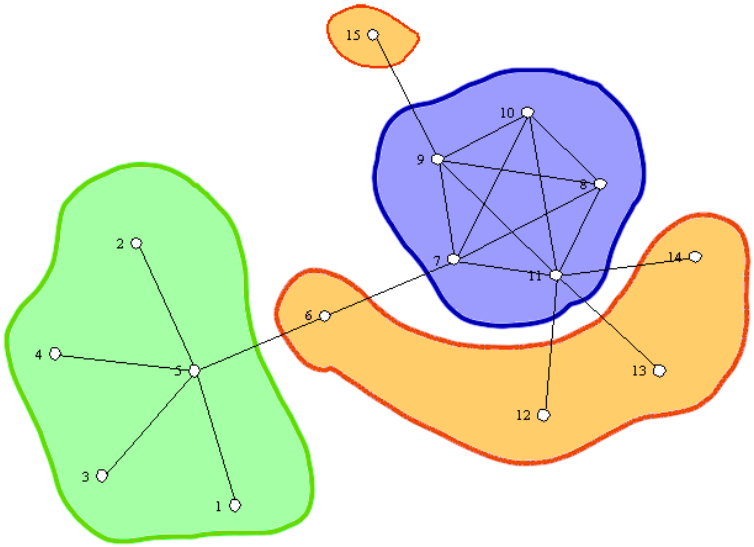


Figura 2.4: Descomposición en clusters de la red basada en el autovector asociado al mayor autovalor.

En el caso de redes con distribuciones de grados de cola larga, el método se ve deteriorado, y se suele realizar una normalización en frecuencias de la matriz de adyacencias. Esta normalización consiste en dividir cada valor de la matriz por la suma de todos los valores de su fila, con lo cual se obtiene una matriz estocástica (es decir, que todas sus filas suman 1) [20].

Por otra parte la complejidad computacional del problema para un caso general es difícil de determinar, dado que los autovalores son soluciones de ecuaciones polinómicas que no pueden ser siempre resueltas en forma exacta sino sólo a través de procedimientos iterativos. Para matrices simétricas, se pueden hallar los mayores autovalores y sus autovectores asociados en un tiempo de $O(n^3)$ [22], que igualmente es excesivo a la hora de aplicarlo en redes con miles de nodos; nuevamente, se hace necesario aplicarlo sólo a una pequeña parte de la red, como puede serlo el núcleo central.

2.2.2. Algoritmos divisivos

El trabajo de Newman [27] sobre el descubrimiento de estructuras comunitarias, logra formar clusters a través de la remoción de ejes. El punto de partida es el interesante concepto de contar cuántos caminos mínimos atraviesan cada eje de la red. Esta propuesta se basa en una medida de centralidad denominada *edge betweenness*, que definimos a continuación:

Definición 1 *Sea $G = (V, E)$ un grafo. El coeficiente de* betweenness *de un*

eje $e_{ij} \in E$ es la cantidad de caminos mínimos en G de los cuales e_{ij} es parte.

El procedimiento es el siguiente: Se analizan los caminos mínimos entre todos los pares de nodos de la red, incrementando un contador en cada eje que es parte de un camino mínimo. Luego, se elimina el eje por donde pasan el mayor número de caminos, con el fundamento de que son justamente los ejes entre clusters (intercomunitarios) los que deben aparecer en gran cantidad de caminos, mientras que los ejes internos a los clusters no deberían participar de muchos caminos.

El método logra muy buenos resultados y logra independizarse del tipo y tamaño de la red, aunque con una complejidad computacional de $O(n^3)$.

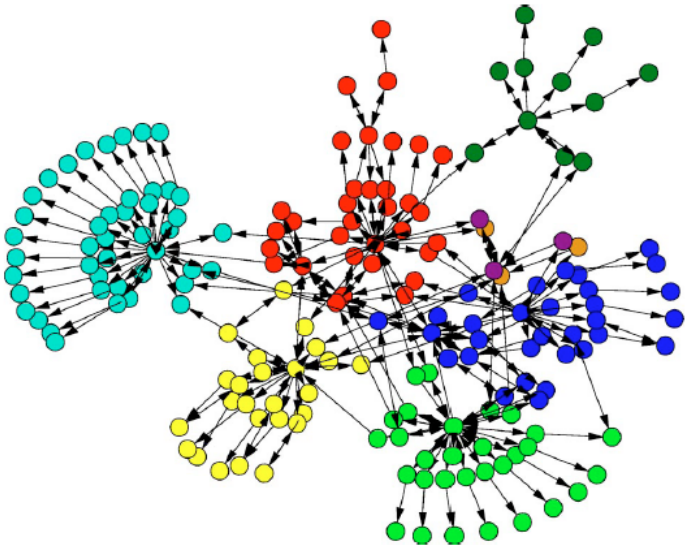


Figura 2.5: Descomposición en clusters de las páginas de un sitio Web, descubriendo estructuras comunitarias. Los distintos colores representan a los clusters. Las flechas indican el sentido de los hipervínculos (imágen extraída de [27]).

2.3. Descomposición en k-núcleos

2.3.1. Algunas definiciones

Se introducirá el concepto de k-núcleo, ideado por Seidman [31] en 1983 y luego generalizado por Batagelj y Zaversnik [10].

Definición 2 *Sea $G = (V, E), E \subseteq VxV$ un grafo no dirigido, en donde V denota el conjunto de vértices y E el conjunto de aristas.*

Un subgrafo $H = (C, E|C)$ inducido por $C \subseteq V$ es un k-núcleo (en inglés, k-core) si y sólo si:

$$\forall v \in C : d_H(v) \geq k \tag{2.3}$$

y H es el subgrafo máximo que cumple esta propiedad [10] (ver figura 2.6).

Un k-núcleo no necesariamente es un subgrafo conexo, sin embargo los núcleos de un grafo están anidados, es decir:

$$i < j \rightarrow H_j \subseteq H_i \tag{2.4}$$

Existe un algoritmo eficiente para descomponer un grafo en k-núcleos, que se basa en la siguiente propiedad:

> Dado un grafo $G = (V, E)$, si se remueven recursivamente los vértices cuyo grado es menor a k, y los ejes que inciden en ellos, entonces el grafo resultante es el k-núcleo.

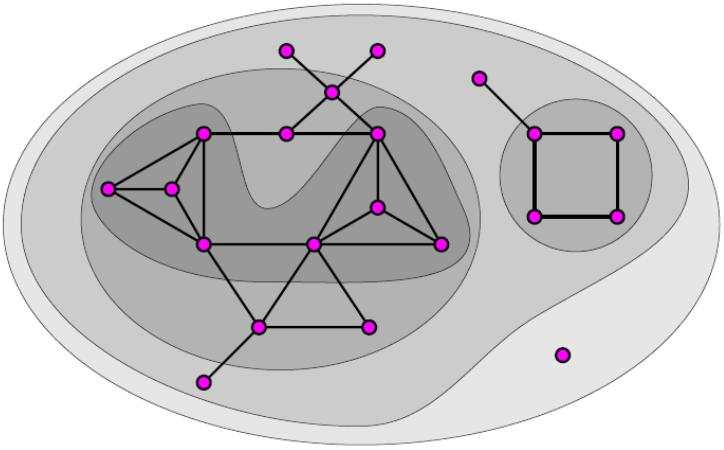


Figura 2.6: Descomposición en k-núcleos de un grafo (imagen extraída de [10]).

2.3.2. Visualización en 2.5 dimensiones, Batagelj *et al.*

Luego, Baur *et al.* [11] combinaron los k-núcleos con el enfoque dirigido por fuerzas y la descomposición espectral, para visualizar

sistemas autónomos en 2,5 dimensiones. El acomodamiento de los nodos comienza por el núcleo central, que se organiza a partir de la descomposición espectral (ver sección 2.2.1). Al emplear la descomposición espectral sólo en dicho núcleo la complejidad se reduce respecto a los métodos espectrales "puros". El resto de los núcleos se posiciona de a uno por vez, a través de fuerzas que interaccionan con los nodos de núcleos anteriores. Cuando un núcleo ya tiene una posición estable, sus nodos se dejan fijos y se pasa a graficar el núcleo siguiente. El algoritmo no permite distinguir núcleos formados por varias componentes conexas.

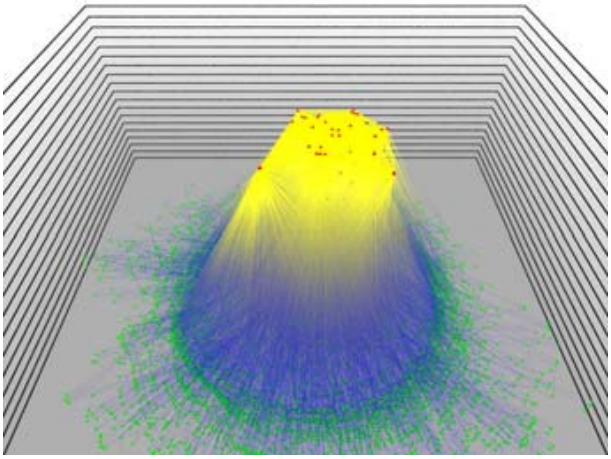


Figura 2.7: Visualización de la red de sistemas autónomos en 2,5 dimensiones. Baur *et al.* [11]

Finalmente, se asigna una coordenada z a cada k-núcleo, de manera de formar capas horizontales, generando así una imágen en el espacio. Sin embargo, si bien las imágenes en 3 dimensiones son útiles en la Visualización Científica, no son las más apropiadas para visualizar información, en donde pueden haber cientas de variables en juego. A la hora de descubrir patrones, clusters y explorar relaciones, es más sencillo para el sistema de percepción con que contamos los seres humanos analizar algunas variables por separado en una imagen bidimensional [32].

2.3.3. LunarVis

La visualización Halfmoon [7] (y su versión actual LunarVis [21]) también emplea k-núcleos, ubicando los vértices en un semicírculo organizados según el k-núcleo al que pertenecen. Luego, dentro de cada núcleo los nodos se posicionan utilizando el método dirigido por fuerzas de Fruchterman-Reingold [19]. El método utiliza transparencias para mostrar todos los ejes de la red. Al igual que el algoritmo anterior, tampoco puede reconocer la existencia de distintas componentes dentro de un núcleo.

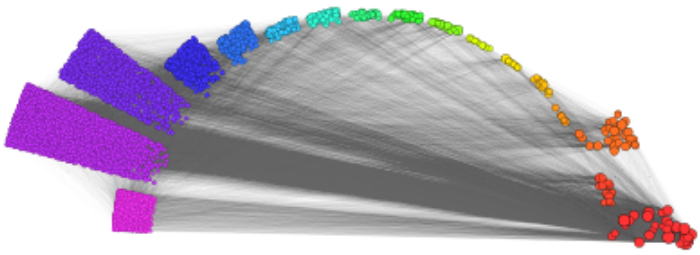


Figura 2.8: Visualización de la red de sistemas autónomos con LunarVis [21].

En esta visualización, los colores más cálidos representan nodos de grado alto, mientras que los fríos están asociados a nodos poco conectados. Observando que los núcleos centrales tienen mayor cantidad de nodos rojos, y los lejanos tienen nodos azules o violetas, se concluye que la correlación grado-capa es alta. También se descubre que hay conexiones desde todos los núcleos hacia el núcleos centrales, en donde se encuentran los nodos más conectados (comportamiento *disassortative*).

2.3.4. LaNet-vi 1.0

La primer versión de LaNet-vi [6] realiza la descomposición en k-núcleos de la red, asignando a cada núcleo un círculo o anillo. Dentro de cada círculo o anillo los nodos que pertenecen a un mismo cluster (ver definición 4 en sección 3.1.2) se agrupan en un sector de tamaño proporcional al tamaño del cluster. El radio de cada nodo representa su grado, y su posición dentro del anillo

indica cómo está compuesta su vecindad. Si sus vecinos pertenecen en su mayor parte a núcleos centrales, entonces el nodo estará más cerca del centro, y viceversa.

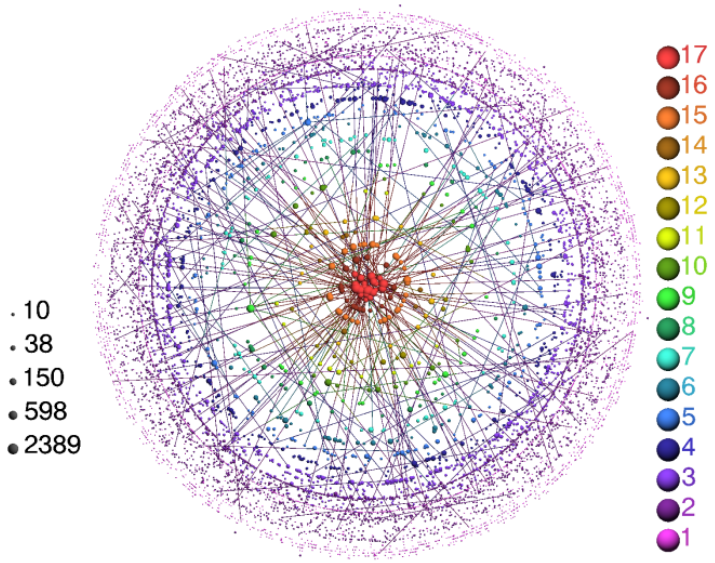


Figura 2.9: Visualización de la red de sistemas autónomos con LaNet-vi 1.0 [6].

En la figura 2.9 se puede observar cómo el número de capa y el grado de los nodos están muy correlacionados, una característica propia de los sistemas autónomos y contraria al mapa de ruteadores (ver [6]). También se verifica claramente que hay muchas conexiones entre los nodos de grado elevado (en los núcleos superiores) y los nodos poco conectados en la periferia. Es un comportamiento de tipo *disassortative*, típico de redes complejas como Internet, la Web y las redes biológicas, aunque no de las redes sociales [25].

Capítulo 3

Fundamentos de LaNet-vi

En el presente capítulo se desarrollará cada una de las funcionalidades de LaNet-vi. Algunas de ellas, como la descomposición en k-núcleos, ya estaban presentes en la primera versión, y se describen en la sección 3.1. Las secciones restantes corresponden a todas las innovaciones de LaNet-vi 2.0, entre ellas:

1. Análisis de la núcleo-conexidad
2. Descomposición en cliqués y reordenamiento de los nodos
3. Soporte para grafos pesados y multigrafos
4. Etiquetado de los nodos
5. Renderización con transparencias

Las contribuciones de la presente tesis son las que incorpora la versión 2.0 de LaNet-vi [12]. El código fuente de LaNet-vi2 puede descargarse desde SourceForge[1].

3.1. Visualización con LaNet-vi1

El concepto de k-núcleo fue introducido en la sección 2.3.1. En esta sección se explicará cómo se emplea esta descomposición para graficar la red en LaNet-vi 1.0.

Para ello es necesario introducir el concepto de *shell index*:

[1] http://sourceforge.net/projects/lanet-vi/

Definición 3 *Un vértice v_i tiene* shell index *k si pertenece al k-núcleo pero no al $(k+1)$-núcleo. Se simbolizará como s_i ó $s(v_i)$.*

Es decir que el *shell index* indica en qué capa de la red se encuentra el nodo, si imaginamos a los k-núcleos como subconjuntos cada uno del k-núcleo inferior.

La capa S_k está formada por todos los vértices de $G = (V, E)$ cuyo *shell index* es k:

$$S_k = \{v_i \in V / s(v_i) = k\} \tag{3.1}$$

Al súbindice de la capa máxima, es decir el máximo valor de k tal que S_k no sea vacía, lo denominaremos $k_{\text{máx}}$.

3.1.1. Visualización en capas

A partir de esta idea de capa, el algoritmo ubica a cada nodo en una corona cuyo radio medio depende de la capa a la que pertenece. Se generan entonces coronas concéntricas; la corona central incluye a los nodos en la capa máxima, $S_{k_{\text{máx}}}$, mientras que las coronas más lejanas contienen a los nodos de capas inferiores.

A su vez, dentro de cada capa, la posición radial de cada nodo varía en función de las conexiones que dicho nodo posee, acercándolo al centro si está conectados a nodos de capas superiores. Esta variación puede regularse a través de un parámetro ϵ. La fórmula que asigna el radio a cada nodo es [6]:

$$\rho_i = (1-\epsilon)\cdot(k_{\text{máx}} - s(i)) + \frac{\epsilon}{|V_{c_j>c_i}(i)|} \cdot \sum_{j \in V_{c_j>c_i}(i)} (k_{\text{máx}} - s(j)) \tag{3.2}$$

El conjunto $V_{c_j>c_i}(i)$ es el conjunto formado por aquellos vecinos de v_i que pertenecen a capas superiores.

A continuación se tratarán distintos aspectos de la visualización.

3.1.2. Distribución dentro de las capas

Dentro de cada capa, los nodos se distribuyen agrupados en *clusters*.

Definición 4 *Un* cluster *es cada una de las componentes conexas dentro del subgrafo inducido por los vértices de una capa. Los j clusters distintos dentro de cada capa se simbolizarán* $Q_1, Q_2, ..., Q_j$.

El subíndice del *cluster* al que pertenece el vértice v_i dentro de su capa se denotará como q_i. Su cluster será entonces Q_{q_i}.

Una vez terminada la descomposición en k-núcleos y conocida la capa a la que pertenece cada vértice, se arman los *clusters* Q_j dentro de cada capa, y se asigna a cada cluster un sector circular acorde a su tamaño. Dentro de cada cluster, sus nodos se ubican en forma aleatoria empleando una distribución normal de probabilidades con media en el centro del sector circular.

La fórmula empleada es [6]:

$$\alpha_i = 2\pi \cdot \sum_{1 \leq m < q_i} \frac{|Q_m|}{|S_{s(i)}|} + N\left(\pi \cdot \frac{|Q_{q_i}|}{|S_{s(i)}|}, \pi \cdot \frac{|Q_{q_i}|}{|S_{s(i)}|}\right) \tag{3.3}$$

El primer término de la sumatoria indica el inicio del cluster, que comienza después de todos los sectores circulares de los clusters anteriores. El segundo término le asigna su ubicación aleatoria dentro del sector. El ancho de cada sector, en radianes, es $2\pi \cdot \frac{|Q_{q_i}|}{|S_{s(i)}|}$.

En el caso del núcleo central, se emplea un círculo como caso particular de una corona. En la primera versión de LaNet-vi los nodos se ubicaban en forma aleatoria dentro de este círculo, dificultando el conocimiento del centro de la red porque los nodos quedaban normalmente superpuestos en un espacio reducido. Esto se resolvió en la presente versión, a partir de la descomposición en cliqués.

3.1.3. Escala de colores

Para aumentar la riqueza de la visualización, se asignó una escala de colores a las capas, siguiendo los colores de un arcoiris. Los nodos en el k-núcleo superior se colorean de rojo mientras que los más lejanos se pintan color violeta.

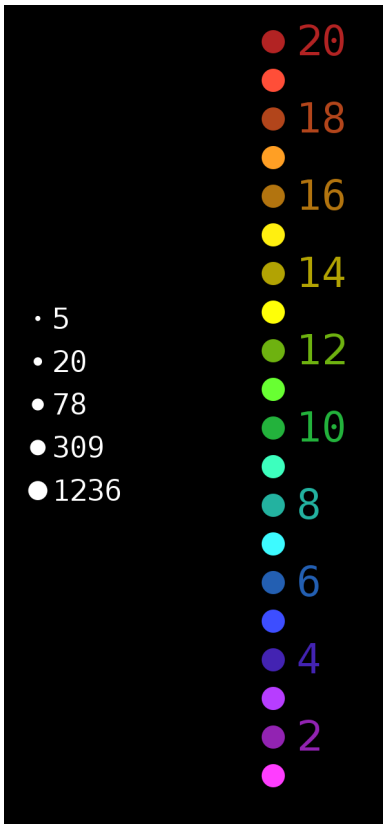


Figura 3.1: Escala de grados (izquierda) y de colores (derecha) de LaNet-vi. Ejemplo para una red con $k_{\text{máx}} = 20$ y grado máximo $d_{\text{máx}} = 1236$.

3.1.4. Visualización de los grados de los nodos

En LaNet-vi la información sobre el grado de cada nodo se encuentra implícita en su tamaño. Los nodos de mayor grado tienen asignado un círculo de mayor radio, a través de una escala logarítmica (ver figura 3.1).

De esta forma, observando gráficamente la relación entre las capas y el tamaño de los nodos, se puede determinar fácilmente si la red visualizada tiene un comportamiento de tipo *assortative* ó *disassortative* (ver sección 1.4.3).

3.2. Descomposición en cliqués

Un punto que había quedado sin resolver en LaNet-vi1 es la organización del núcleo central. En dicha versión, los nodos pertenecientes a este núcleo simplemente se posicionaban de forma aleatoria dentro de un círculo.

En esta nueva versión se adoptó la idea de descomponer al núcleo en cliqués. Un cliqué es un subgrafo completo, es decir un grafo en el cual todos sus nodos son adyacentes de a pares, o dicho de otra forma, todos son vecinos de todos.

La idea de descomponer en cliqués surgió a partir de la observación de que en la mayoría de las redes complejas analizadas, el núcleo central tiene diámetro 2, es decir que sus nodos están muy conectados entre sí, con lo cuál resultaría interesante agrupar a aquellos nodos que forman cliqués. La figura 3.2 ejemplifica la diferencia entre ambas versiones.

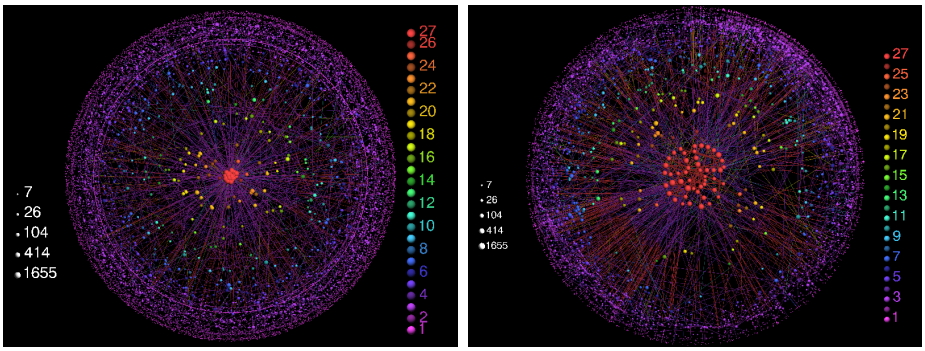


Figura 3.2: Visualización de la red de sistemas autónomos (ASes) con LaNet-vi (izquierda), y con la nueva versión: LaNet-vi2 (derecha). Obsérvese el nuevo despliegue del núcleo central.

Definición 5 *Dado un grafo* $G = (V, E)$*, un subgrafo* $H = (C, E|C)$ *inducido por el subconjunto* $C \subseteq V$ *es un* cliqué *si y sólo si:*

$$\forall v_1, v_2 \in C \rightarrow (v_1, v_2) \in E \tag{3.4}$$

Descomponer un grafo $G = (V, E)$ en cliqués se entiende como hallar una *partición* del conjunto C, es decir armar una familia de subconjuntos $C_1, C_2, ..., C_n$, tales que:

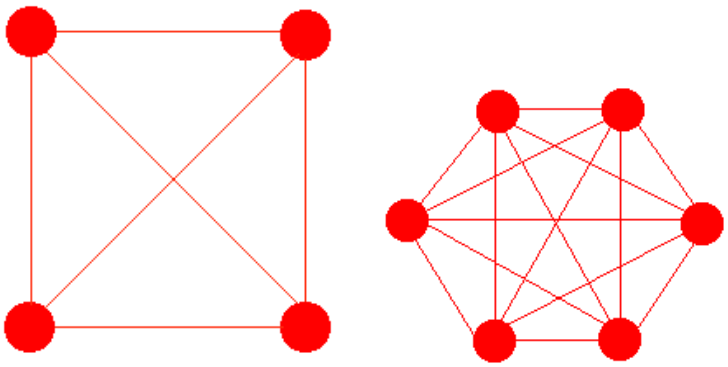


Figura 3.3: Grafos completos (cliqués) de 4 y 6 nodos respectivamente.

1. Ningún subconjunto es vacío:

$$C_i \neq \emptyset, 1 \leq i \leq n \tag{3.5}$$

2. La unión de todos los subconjuntos forma C:

$$C = \bigcup_{1 \leq i \leq n} C_i \tag{3.6}$$

3. La intersección de todos los subconjuntos es vacía:

$$C_i \cap C_j = \emptyset \tag{3.7}$$

Este problema no tiene en general una única solución. Sin embargo, no interesa cualquier descomposición del grafo en cliqués, sino que se trata obtener en la descomposición los cliqués de mayor tamaño. Hallar el mayor cliqué en un grafo es un problema NP completo, es decir que los algoritmos de resolución existentes emplean un tiempo exponencial con respecto al tamaño del grafo [15]. Presentaremos entonces algunos algoritmos existentes y luego el que se propone en esta tesis.

3.2.1. Algoritmos de búsqueda de cliqués conocidos

Enfoque por fuerza bruta

Un primer método que podría plantearse es el enfoque por fuerza-bruta: Se hace una enumeración de todos los subgrafos posibles,

y se verifica para cada uno si es un cliqué o no. La cantidad de subgrafos en un grafo es 2^n, con lo cual se requiere claramente de tiempo exponencial para analizar todas las posibilidades y así hallar el cliqué máximo. Incluso, una vez hallado el cliqué máximo se deberá reperir el procedimiento recursivamente con los nodos restantes, dado que el objetivo final es obtener una partición del grafo conformada por cliqués.

Algoritmo de Bron-Kerbosch [13]

Es un algoritmo de backtracking, de tipo goloso (*greedy*), para enumerar los cliqués máximos en un grafo. Es un refinamiento de la técnica de fuerza-bruta, y normalmente se implementa en forma recursiva. Sin embargo, al no analizar la conexidad de los nodos antes de agregarlos, realiza una gran cantidad de pruebas resultando en una alta complejidad. El algoritmo insume un tiempo lineal respecto a la cantidad de cliqués, lo cual es excesivo si se considera que la cantidad de cliqués puede crecer exponencialmente con cada nodo agregado.

3.2.2. Heurística de búsqueda de cliqués propuesta

Observando la alta complejidad de los algoritmos existentes, se optó por desarrollar una heurística que, midiendo parámetros de conexidad de los nodos, obtenga los cliqués de manera eficiente. Se busca que el algoritmo comience a construir los cliqués decidiendo qué nodos conviene incorporar con el objetivo de armar los cliqués más grandes. Sin embargo, dado que es una heurística, no se garantiza que se encontrará el mayor cliqué, ni tampoco que el resultado sea único. De todas formas descubrir los mayores cliqués resulta útil para las redes complejas ya que hay una distribución del tamaño de los cliqués en la cual muchos de ellos son chicos. Por ejemplo, encontrar un cliqué grande puede significar una colaboración entre un grupo de nodos. El algoritmo no utiliza recursión ni back-tracking, y si bien como se dijo no asegura que se obtendrá dentro de la descomposición el cliqué más grande, en la mayoría de las pruebas realizadas sí lo ha obtenido correctamente, gracias a la distribución de cliqués de diversos tamaños.

Definiciones previas

Definición 6 *La* vecindad *de un vértice $v_i \in V$ es el subgrafo inducido en G que incluye a todos los vértices conectados a v_i. Se simbolizará*

$$N_i = \{v_j \in V / e_{ij} \in E\} \quad . \tag{3.8}$$

Definición 7 *El* conjunto de conexiones entre vecinos T_i *es el conjunto de ejes cuyos vértices pertenecen a la vecindad de v_i, es decir*

$$T_i = \{e_{jk} \in E / v_j, v_k \in N_i\} \quad . \tag{3.9}$$

Definición 8 *El* coeficiente de clustering $cc(v_i)$ *de $v_i \in V$ es el cociente entre la cantidad de conexiones entre los vecinos de v_i: $w_1, w_2, ..., w_{N_i}$, y la máxima cantidad posible de conexiones entre ellos, exactamente $\frac{|N_i|\,(|N_i|-1)}{2}$. Entonces,*

$$cc(v_i) = \frac{2\,|T_i|}{|N_i|\,(|N_i|-1)} \quad . \tag{3.10}$$

Definición 9 *La* vecindad común C_{ij}*, es el subgrafo inducido de G compuesto por todos los vértices adyacentes tanto a v_i como a v_j.*

El siguiente resultado es de utilidad para el algoritmo desarrollado: la cantidad de conexiones entre vecinos de un nodo, $|T_i|$, puede calcularse a partir de los subgrafos C_{ij} para todos los v_j adyacentes a v_i, de la siguiente manera: tomando v_j, v_k adyacentes a v_i, si están mutuamente conectados entonces $v_j \in C_{ik}$ pero también $v_k \in C_{ij}$, por lo tanto:

$$|T_i| = \sum_{j/v_j \in N_i} \frac{|C_{ij}|}{2} \quad . \tag{3.11}$$

Descripción del algoritmo

Se parte del pensamiento intuitivo de que los vértices que se hallan en el cliqué más grande deberían tener vecinos con una gran cantidad de conexiones entre ellos. Como esta idea está vinculada

con el conjunto T_i de conexiones entre vecinos definido anteriormente, entonces se ordenará inicialmente el conjunto V en base al valor de $|T_i|$.

Luego se toma el primer vértice en V, es decir aquel que tiene la mayor cantidad de vecinos conectados entre sí, convirtiéndose así en el primer vértice del primer cliqué a formar, V_1. A continuación se agregan vecinos con la condición de que estén conectados a todos aquellos anteriormente conectados a V_1. Los primeros vecinos en ser agregados son aquellos que comparten más vecinos con v_i, es decir aquellos cuyo coeficiente $|C_{ij}|$ es mayor, ya que son ellos los que tienen posibilidades de ser parte de un gran cliqué. En cambio, si se eligiera como vértice a cualquiera, podría ocurrir que éste sólo estuviera conectado a v_1 e impidiera entonces alcanzar un cliqué más grande del cual v_1 era parte.

El procedimiento se repite hasta que no puedan agregarse más nodos a V_1. Entonces se termina la construcción de este primer cliqué, y se continúa con la construcción de V_2 con los nodos restantes y empleando la misma lógica.

Finalmente, se habrá obtenido una partición de V compuesta por los cliqués $V_1, V_2, ..., V_n$.

Una descripción más formal puede verse en el cuadro 3.1.

Ejemplo

Con el siguiente ejemplo (ver figura 3.4) se ilustrará la lógica del algoritmo de descomposición en cliqués. Entre parentesis se indica la línea correspondiente a la especificación del algoritmo en el cuadro 3.1.

Se observa que existe un subgrafo completo de orden 5, conformado por los nodos 5, 6, 7, 8 y 9. Este es el cliqué de tamaño máximo, y sería deseable que el algoritmo lo encuentre.

En segundo lugar, interesaría encontrar el cliqué formado por 1, 2, 3 y 4.

Cálculo inicial de los coeficientes $|C_{jk}|$ (Línea 7)

Comenzamos tomando el nodo 1. Como está conectado a los nodos 2, 3 y 4, se incrementan $|C_{2,3}|$, $|C_{2,4}|$ y $|C_{3,4}|$, porque el nodo 1 es un vecino común a cada uno de estos pares de nodos.

Algoritmo 3.1 DESCOMPOSICION EN CLIQUES

```
1  Entrada: Grafo G = (V, E)
2  Salida: Conjunto de cliqués V_1, V_2, ..., V_n que forman una partición de V
3  inicializar |C_ij| (vecindad común) en 0 para v_i, v_j ∈ V
4  inicializar |T_i| en 0 para v_i ∈ V
5  inicializar n=0
6
7  Para cada vértice v_i ∈ V hacer
8      Para cada vecino v_j ∈ N_i hacer
9          Para cada vecino v_k ∈ N_i, k > j hacer
10             Incrementar |C_jk|
11
12 Para cada vértice v_i ∈ V hacer
13     calcular conexiones entre vecinos |T_i| empleando la ecuación 3.11
14
15 Ordenar V por |T_i| decreciente, obteniendo W
16
17 Mientras W no sea vacío hacer
18     Comenzar cliqué V_n
19     v_f = {Primer vértice en W}
20     Insertar el primer vértice v_f en el cliqué V_n, y eliminarlo de W
21     Generar N con todos los vértices adyacentes a v_f que están en W (aquellos
       que aún no tienen cliqué asignado)
22     Ordenar N por |C_fj| decreciente
23     Para cada vértice v_j ∈ N (en orden) hacer
24         Si es adyacente a todos los vértices en V_n (V_n ⊂ N_j) entonces
25             Agregar v_j al cliqué V_n
26             Eliminar v_j de W
27
28     n=n+1
29
30 resultado: {V_i}
```

Cuadro 3.1: Algoritmo de descomposición en cliqués

El nodo 2 está conectado a 1, 2, 4 y 5, entonces se incrementan $|C_{1,2}|$, $|C_{1,4}|$, $|C_{1,5}|$, $|C_{2,4}|$, $|C_{2,5}|$ y $|C_{4,5}|$, es decir todas las combinaciones de a pares de los vecinos del nodo 2.

Análogamente se procede con todos los nodos del grafo.

Cálculo de conexiones entre vecinos $|T_i|$ (Línea 12)

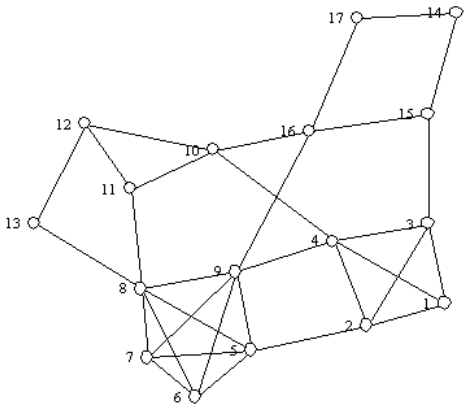


Figura 3.4: Grafo de la red a partir de la cual se obtendrán cliqués. (Imagen generada con Network Workbench [28]).

Comenzando por el nodo 1, se toman sus vecinos: 2, 3 y 4. Entre ellos existen 3 conexiones: (2,3), (2,4) y (3,4), es decir que todos están conectados con todos. Pero si no se conocieran los coeficientes $|C_{ij}|$ se debería probar para cada par de vecinos si están conectados o no, aumentando así el tiempo de la heurística. En cambio, empleando los $|C_{ij}|$ y el resultado de que cada conexión entre vecinos v_j y v_k debe estar reflejada en los conjuntos $|C_{i,j}|$ y $|C_{i,k}|$ expresado en la fórmula 3.11, se acorta el tiempo de cálculo.

$$|T_1| = \frac{|C_{1,2}| + |C_{1,3}| + |C_{1,4}|}{2} = \frac{2+2+2}{2} = 3 \ . \qquad (3.12)$$

De la misma forma se continúan tomando todos los pares de vecinos v_j, v_k de cada nodo v_i, para incrementar C_{jk}. La condición $k > j$ adicional se emplea para que el par no aparezca dos veces en el recuento, en diferente orden.

La tabla final W de coeficientes $|T_i|$ ordenada en forma decreciente (como indica el paso 15) queda:

Armado del primer cliqué

Se comienza con el nodo 5, que es el primero en la tabla 3.2. Luego se arma el conjunto N con todos los vecinos de 5, es decir

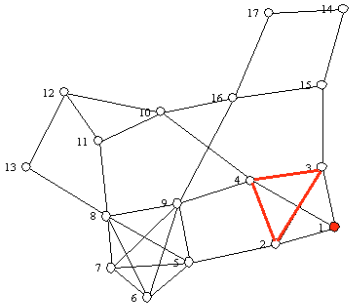


Figura 3.5: Ejes entre vecinos del nodo 1. El eje (2,3) contribuye unitariamente con $C_{1,2}$ y $C_{1,3}$. El eje (2,4) contribuye con $C_{1,2}$ y $C_{1,4}$. El eje (3,4) contribuye con $C_{1,3}$ y $C_{1,4}$.

i	Coeficiente $\|T_i\|$
5	6
6	6
7	6
8	6
9	6
1	3
2	3
3	3
4	3
10	1
12	1
11	0
13	0
14	0
15	0
16	0
17	0

Cuadro 3.2: Tabla W: Coeficientes $|T_i|$.

$N = \{2, 6, 7, 8, 9\}$, como indica el paso 21. También el conjunto N se ordena por la cantidad de vecinos que cada uno de sus nodos comparte con 5. En este caso todos sus vértices tienen $|C_i5| = 3$ salvo el 2, cuyo $|C_25|$ es 0. Los elementos v_j de N Se agregan de a uno al cliqué comenzando por el primero, verificando con cada

uno que esté conectado a todos los que ya forman parte del cliqué. Si no pueden ingresar al cliqué, entonces se descartan. Aquellos que se agregan al cliqué, se eliminan de la tabla W, de manera que sólo queden en ella aquellos que aún no tienen cliqué asignado.

La siguiente tabla ilustra la lógica del paso 24:

$C_{1_{parcial}}$	N	v_j	¿$V_1 \subset N_j$?	Nuevo $C_{1_{parcial}}$
$\{5\}$	$\{6,7,8,9,2\}$	6	Sí	$\{5,6\}$
{5,6}	$\{7,8,9,2\}$	7	Sí	$\{5,6,7\}$
{5,6,7}	$\{8,9,2\}$	8	Sí	$\{5,6,7,8\}$
{5,6,7,8}	$\{9,2\}$	9	Sí	$\{5,6,7,8,9\}$
{5,6,7,8,9}	$\{2\}$	2	No	$\{5,6,7,8,9\}$

Cuadro 3.3: Formación del primer cliqué.

Observar que el ordenamiento de N permitió que no se incorporase el nodo 2 al cliqué. Si ésto hubiese sucedido, entonces se habría armado un cliqué V_1 sólo con los nodos 2 y 5, y se habría perdido la posibilidad de encontrar al mayor cliqué. Es decir que no alcanza sólo con comenzar el cliqué con un nodo que tenga muchos vecinos conectados entre sí, sino que además es importante controlar el orden en que sus demás vecinos se agregan, para no perder las posibilidades de formar grandes cliqués.

Armado del segundo cliqué

De los nodos que quedan en W, el primero es el nodo 1, cuyos vecinos son $N = \{2,3,4\}$, todos con el mismo coeficiente $|T_i|$. Todos ellos forman un cliqué, y el algoritmo lo reconocerá, como lo muestra la siguiente tabla.

$C_{1_{parcial}}$	N	v_j	¿$V_1 \subset N_j$?	Nuevo $C_{1_{parcial}}$
$\{1\}$	$\{2,3,4\}$	2	Sí	$\{1,2\}$
{1,2}	$\{3,4\}$	3	Sí	$\{1,2,3\}$
{1,2,3}	$\{4\}$	4	Sí	$\{1,2,3,4\}$

Cuadro 3.4: Formación del segundo cliqué.

Se ha obtenido así el segundo de los cliqués en importancia.

Descomposición final en cliqués

Al finalizar el algoritmo se habrá obtenido la siguiente partición del conjunto V:

$V_1 = \{5, 6, 7, 8, 9\}$

$V_2 = \{1, 2, 3, 4\}$ $V_3 = \{10, 11, 12\}$ $V_4 = \{13\}$ $V_5 = \{14, 17\}$ $V_6 = \{15, 16\}$

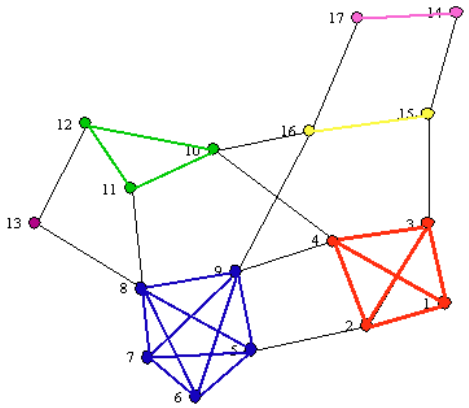


Figura 3.6: Descomposición en cliqués de la red ejemplo, empleando el algoritmo de LaNet-vi2. Observar cómo los cliqués mayores han sido detectados.

3.2.3. Estudio de la complejidad

A continuación se demostrará que la complejidad computacional de aplicar el algoritmo de descomposición en cliqués desarrollado al núcleo central de una red es del orden es de $O(n^{3/2})$ para la mayoría de las redes complejas.

El primer paso, consistente en el cálculo de los coeficientes $|C_{jk}|$ (líneas 7-10), implica considerar para cada nodo, todos sus pares de vecinos. Si un vértice v_i tiene $d(v_i)$ vecinos, entonces la cantidad de pares de vecinos es la cantidad de combinaciones de $d(v_i)$ elementos tomados de a 2, es decir

$$\binom{d(v_i)}{2} = \frac{d(v_i) \cdot (d(v_i) - 1)}{2} \tag{3.13}$$

Acotando $d(v_i) \leq d_{k_{\text{máx}}}$ y considerando que el núcleo central tiene $n_{k_{\text{máx}}}$ nodos, la cantidad de operaciones de este paso está acotada por $O(n_{k_{\text{máx}}} \cdot {d_{k_{\text{máx}}}}^2)$.

A continuación, el cálculo de los coeficientes $|T_i|$ (líneas 12-13) requiere para cada nodo v_i acumular los coeficientes $|C_{ij}|$ que comparte con sus vecinos. Esta suma incluye $d(v_i)$ elementos para cada uno, y siendo $n_{k_{\text{máx}}}$ el total de nodos, la complejidad resulta $O(n_{k_{\text{máx}}} \cdot d_{k_{\text{máx}}})$.

El tercer paso (línea 15), consistente en el ordenamiento de los nodos en el núcleo central en función del coeficiente $|T_i|$, implica una complejidad de $O(n_{k_{\text{máx}}} \log n_{k_{\text{máx}}})$ empleando un algoritmo de ordenamiento de $O(n \log n)$ como Heapsort.

Luego comienza la descomposición propiamente dicha. Comenzar el primer cliqué e insertar el primer nodo requiere de tiempo constante.

Generar el conjunto N (línea 21) implica $d(v_f)$ inserciones, que están acotadas por $O(d_{k_{\text{máx}}})$. El ordenamiento de N (línea 22) puede acotarse por $O(d_{k_{\text{máx}}} \log d_{k_{\text{máx}}})$.

Ahora, en el bucle **"para"** (líneas 23-27) se verifica para cada nodo en N si es adyacente a todos los nodos que ya están en el cliqué, y en caso que así sea, se lo añade dicho cliqué. Para el primer nodo v_1, sólo se hace verifica si está conectado a v_f (en realidad podría omitirse); debe estar conectado a v_f por el hecho de pertenecer a N_{v_f}. Para el segundo nodo v_2 se realizan 2 verificaciones: si está conectado a v_f y si está conectado a v_1: si está conectado a ambos se lo agrega al cliqué. Para el tercer nodo se harán 2 o 3 verificaciones, dependiendo del resultado anterior. Claramente el peor caso es aquél en que todos los nodos están conectados entre sí, en cuyo caso se deben hacer $1+2+....+d(v_f)$ verificaciones, resultando una complejidad de $d^2_{k_{\text{máx}}}$.

Finalmente, la eliminación de v_j del conjunto W e inserción en W requieren tiempo constante dentro del bucle **"para"**. Al terminar el bucle **"mientras"** la complejidad resultante en función de la cantidad de cliqués es $O(n_{cliques}\ d^2_{k_{\text{máx}}})$.

El cuadro 3.5 resume la complejidad de cada paso del armado de los cliqués, y el cuadro 3.6 la complejidad del algoritmo completo.

Se observa en el cuadro 3.6 que los pasos que determinan el tiempo del algoritmo son el primero y el último. Y teniendo en cuenta que

Paso	Complejidad
Crear cliqué - Insertar primer nodo	$O(c)$
Generar N	$O(d_{k_{\text{máx}}})$
Ordenar N	$O(d_{k_{\text{máx}}} \log d_{k_{\text{máx}}})$
Bucle **"para"** (líneas 23-27)	$O(d^2_{k_{\text{máx}}})$
Complejidad del bucle **"mientras"** (líneas 17-28):	$O(n_{cliques}\, d^2_{k_{\text{máx}}})$

Cuadro 3.5: Complejidad de los pasos del armado de cliqués.

Paso	Complejidad
Cálculo de los coeficientes $\lvert C_{jk}\rvert$	$O(n_{k_{\text{máx}}} \cdot {d_{k_{\text{máx}}}}^2)$
Cálculo de los coeficientes $\lvert T_i\rvert$	$O(n_{k_{\text{máx}}} \cdot d_{k_{\text{máx}}})$
Ordenamiento de nodos según $\lvert T_i\rvert$	$O(n_{k_{\text{máx}}} \log n_{k_{\text{máx}}})$
Armado de cliqués	$O(n_{cliques}\, d^2_{k_{\text{máx}}})$
Complejidad del algoritmo 3.1:	$O(n_{k_{\text{máx}}} \cdot {d_{k_{\text{máx}}}}^2)$

Cuadro 3.6: Complejidad de los pasos del algoritmo de descomposición en cliqués.

la cantidad de cliqués no puede superar a la cantidad de nodos del núcleo central, podemos concluir que la complejidad es de $O(n_{k_{\text{máx}}} \cdot {d_{k_{\text{máx}}}}^2)$.

Ahora bien, el grado máximo de los nodos en el núcleo central $d_{k_{\text{máx}}}$ puede acotarse con $d_{\text{máx}}$, es decir el grado máximo de la red completa. Por otra parte comprobamos que la cantidad de nodos en el núcleo central también puede acotarse con $d_{\text{máx}}$ para la mayoría de las redes complejas [5]. Con esta información, podemos acotar la complejidad del algoritmo para una red compleja en $O(d^3_{\text{máx}})$. Nos resta estudiar la relación entre el grado máximo $d_{\text{máx}}$ y el tamaño de la red completa, n. Para ello haremos uso de la distribución de probabilidades de ley de potencias característica de las redes complejas, presentada en la sección 1.4.1, y que puede expresarse de la siguiente forma:

$$n(d) = A \cdot d^{-B} \ , \tag{3.14}$$

en donde $n(d)$ es la cantidad de vértices de grado d. Y como se mencionó anteriormente, el coeficiente B se encuentra comprendido entre $2 \leq B \leq 3$ en las redes complejas.

A partir de la ecuación anterior, se puede llegar a la relación

$$\frac{n(1)}{n(d)} \propto d^B \ , \tag{3.15}$$

de donde surge que

$$d \propto \left(\frac{n(1)}{n(d)}\right)^{1/B} \ . \tag{3.16}$$

Entonces, aplicando la fórmula al grado máximo $d_{\text{máx}}$:

$$d_{\text{máx}} \propto \left(\frac{n(1)}{n(d_{\text{máx}})}\right)^{1/B} < \big(n(1)\big)^{1/B} < n^{1/B} \ . \tag{3.17}$$

Llegamos a la conclusión de que en las redes complejas en general, el grado máximo no supera a la raíz B de la cantidad de nodos. Como resultado, podemos acotar la complejidad del algoritmo en $O(n^{3/B})$. En el peor de los casos, en que $B = 2$, obtenemos $O(n^{3/2})$.

3.3. Posicionamiento en LaNet-vi2

A partir de los cambios en el núcleo central explicados en la sección previa, se modificaron las ecuaciones de posicionamiento de los vértices. Dado que la cantidad de clusters en cada componente es elevado en las redes complejas, en la práctica no se llegan a distinguir. Entonces se optó por utilizar las coordenadas polares de los vécinos en capas superiores para organizar la ubicación de los nodos.

3.3.1. Despliegue del núcleo central

Una vez particionado el núcleo central en cliqués, los mismos se posicionan en sectores circulares. Cada cliqué recibe un sector circular proporcional a su cantidad de nodos, con respecto al tamaño del núcleo central.

Luego los nodos se ubican sobre la periferia de este sector, equidistanciados. La figura 3.7 ilustra este despliegue en el caso en que existen dos cliqués de tamaño 6, y dos cliqués de tamaño 4.

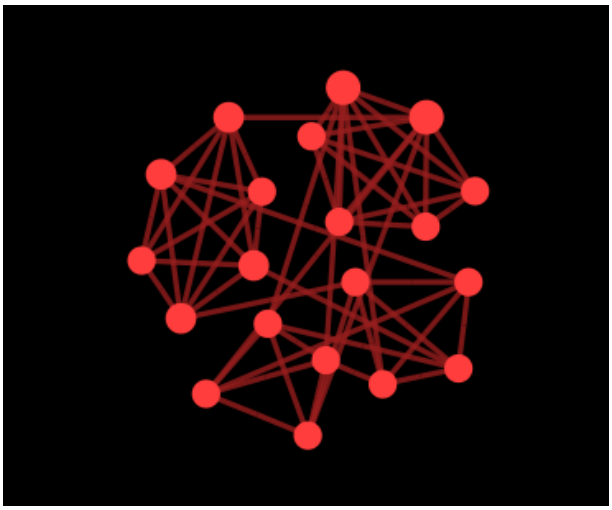


Figura 3.7: Posicionamiento de los nodos del núcleo central ($k_{\text{máx}}$).

3.3.2. Nuevo cálculo del radio

LaNet-vi2 busca expandir el núcleo central en el gráfico para mostrar su estructura. Observando la figura 3.2 ya se veía que en la nueva versión se destina más espacio al núcleo central. De hecho, ahora se calcula el espacio necesario para desplegar todos los nodos centrales, teniendo en cuenta el radio de cada uno (ligado a su grado) y la cantidad de nodos. La siguiente fórmula muestra cómo se calcula el radio del núcleo central:

$$R^2_{k_{\text{máx}}} = K \cdot \sum_{v_i \in S_{k_{\text{máx}}}} \log(1 + d(v_i))^2 \ . \tag{3.18}$$

Luego, el área sobrante de la imagen se divide equitativamente de igual forma que antes, para formar los sectores circulares de las capas. La fórmula que resume esta división es:

$$\rho_i = R_{k_{\text{máx}}} + \frac{k_{\text{máx}} - R_{k_{\text{máx}}}}{k_{\text{máx}}} f(v_i) \ , \tag{3.19}$$

en donde $f(v_i)$ es bastante similar a la expresión que se empleaba en LaNet-vi1:

$$f(v_i) = (1-\epsilon)\big(k_{\text{máx}} - s(i) + 1\big) + \frac{\epsilon}{|N_{i/\geq s(i)}|} \sum_{j \in N_{i/\geq s(i)}} \big(s(j) - s(i)\big) \tag{3.20}$$

El conjunto $N_{i/\geq s(i)}$ está formado por aquellos vecinos de v_i que pertenecen a su misma capa o bien a capas superiores.

3.3.3. Nuevo cálculo del ángulo

La estrategia empleada para asignar la posición angular de cada nodo fuera del núcleo central consiste en alinearlo con sus vecinos en capas superiores de manera que se obtengan ejes radiales, minimizando la cantidad de ejes que atraviesan la imagen. Para ello es necesario comenzar calculando la posición de los nodos de los núcleos más centrales, de manera que al llegar a los núcleos externos, sus vecinos en capas superiores ya tengan su posición calculada.

Entonces, se barren todas las capas comenzando por la capa $k_{\text{máx}-1}$ (cuya posición dependerá de los cliqués centrales), hasta terminar con la capa 1. De esta manera, la descomposición en cliqués inicial es la que organiza el despliegue de toda la red.

Las siguientes fórmulas expresan el seno y coseno del ángulo asignado a cada nodo.

$$\cos\phi_i = \frac{\sum_{j\in N_{i/\geq s(i)}\cap\phi_j^C}\big(s(j)-s(i)+1\big)\,\cos\phi_j}{\sum_{j\in N_{i/\geq s(i)}\cap\phi_j^C}\big(s(j)-s(i)+1\big)} \tag{3.21}$$

$$\sin\phi_i = \frac{\sum_{j\in N_{i/\geq s(i)}\cap\phi_j^C}\big(s(j)-s(i)+1\big)\,\sin\phi_j}{\sum_{j\in N_{i/\geq s(i)}\cap\phi_j^C}\big(s(j)-s(i)+1\big)} \tag{3.22}$$

y permiten calcular finalmente el ángulo a través de

$$\phi_i = atan2(\sin\phi_i, \cos\phi_i) \tag{3.23}$$

En la fórmula anterior, $atan2(y,x)$ es una función que calcula la arcotangente de y/x pero en un rango de $(-\pi,\pi)$, y teniendo en cuenta el cuadrante en que ellos se encuentran[2]. Concretamente:

Si $y \neq 0$:

$$atan2(y,x) = \begin{cases} \varphi\cdot sgn(y) & x>0 \\ (\pi/2)\cdot sgn(y) & x=0 \\ (\pi-\varphi)\cdot sgn(y) & x<0 \end{cases} \tag{3.24}$$

[2]Una función como ésta ya existía en el lenguaje FORTRAN, y se encuentra también en `math.h` de la biblioteca estándar de C. Además, existe en algunas planillas de cálculo populares y en el set de instrucciones de la arquitectura Intel.

Si $y = 0$:

$$atan2(y, x) = \begin{cases} 0 & x > 0 \\ 0 & x = 0 \\ \pi & x < 0 \end{cases} \tag{3.25}$$

En donde φ es el ángulo entre 0 y $\frac{\pi}{2}$ que cumple que $\tan(\varphi) = |\frac{y}{x}|$, y sgn es la función signo:

$$sgn(x) = \begin{cases} 1 & x > 0 \\ 0 & x = 0 \\ -1 & x < 0 \end{cases} \tag{3.26}$$

Promedio circular

Las fórmulas 3.21 y 3.22 expresan lo que se denomina un *promedio circular*. El promedio circular es una variación de la media convencional que se emplea para promediar ángulos, horas del día, o básicamente cualquier magnitud que pueda ciclar. Para aplicar la media circular, los ángulos deben ser convertidos en coordenadas cartesianas, es decir puntos del plano x-y, a través de la transformación

$$\phi \rightarrow (\cos\phi, \sin\phi) \tag{3.27}$$

Entonces, se calculan las medias de las coordenadas x e y por separado a través de las fórmulas

$$\cos\phi = \frac{\sum_i \cos\phi_i}{|N_i|} \tag{3.28}$$

$$\sin\phi = \frac{\sum_i \sin\phi_i}{|N_i|} \ , \tag{3.29}$$

y el resultado se convierte de coordenadas cartesianas a polares, para quedarse finalmente con el ángulo ϕ.

LaNet-vi2 realiza un promedio circular ponderado entre los ángulos de todos los vecinos de v_i en capas superiores, y de aquellos de su misma capa que ya tienen su posición calculada (simbolizados como ϕ_j^C). El peso de cada vecino en el cálculo del promedio está ponderado por la distancia en capas al nodo cuya posición se quiere calcular.

La figura 3.8 muestra un ejemplo de visualización con LaNet-vi2 en donde se puede apreciar cómo la posición de cada nodo la determinan sus vecinos en capas superiores, y se la contrasta con una imagen de la misma red obtenida con LaNet-vi1.

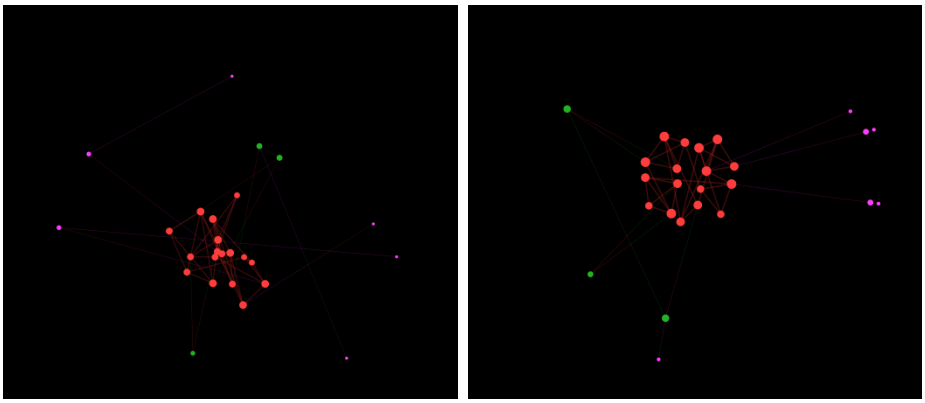


Figura 3.8: Organización en clusters que tenía LaNet-vi1 (izquierda) contrastada con la organización conforme al promedio circular en LaNet-vi2 (derecha).

3.4. Análisis de núcleo-conexidad

La conexidad de una red está íntimamente vinculada con la cantidad de caminos que permiten llegar de uno de sus nodos a otro. La existencia de múltiples variantes dentro de una red implica *robustez*, es decir la capacidad de tolerar la caída de un enlace o de un nodo. En cambio, una red en la que hay pocos caminos alternativos entre sus nodos, o en la cuál muchos caminos pasan por un mismo nodo, será mucho más vulnerable ante fallas. La multiplicidad de caminos se puede vincular también a la posibilidad de satisfacer una calidad de servicio (QoS: *Quality of Service*) requerida: teniendo más caminos aumenta la posibilidad de que alguno de ellos cumpla con la QoS necesaria.

Para estudiar la conexidad de la red y vincularla con la descomposición en k-núcleos, se desarrolló el concepto de *núcleo-conexidad* que se explicará en esta sección.

Definición 10 *Dado un grafo G y su descomposición en k-núcleos:*

1. *Dos nodos pertenecientes a un k-núcleo son* núcleo-conexos *si hay k caminos arista-disjuntos entre ellos.*
2. *El grafo es núcleo-conexo si todo par de vértices en un k-núcleo son núcleo-conexos.*

La definición anterior implica que en una red núcleo-conexa, tomando dos vértices cualesquiera la cantidad de caminos entre

ellos es de al menos el mínimo entre sus *shell indexes*. También se desprende que para que una red sea núcleo-conexa, debe primero ser conexa (es decir tener una sola componente). De lo contrario se pueden encontrar al menos dos vértices tales que no exista camino entre ellos, con lo cual no serían núcleo-conexos.

LaNet-vi2 puede determinar si un grafo es núcleo-conexo, para el caso de grafos simples y en los cuales todas sus capas tienen sólo una componente conexa.

A continuación se enunciará un teorema que establece condiciones para la k-arista-conexidad en un grafo cuyo grado mínimo es k.

Teorema 1 *Sea $G = (V, E)$ un grafo simple, y sea $\{V_1, V_2\}$ una partición del conjunto V. Considerar los subgrafos G_1 y G_2 inducidos por V_1 y V_2 respectivamente. Se define entonces la función delta:*

$$\begin{aligned} \delta(x,y) &= \text{mín}\{r_{G_1}(x,y), r_G(x,V_2) + r_G(y,V_2)\}\ ,\ x,y \in V_1 \\ \delta(x,y) &= r_G(x,V_2)\ ,\ x \in V,\ y \in V_2 \\ \delta(y,x) &= \delta(x,y) \end{aligned} \tag{3.30}$$

Para $x \in V$ and $A \subset V$, se define

$$\delta(x,A) = \min_{a \in A} \delta(x,a) \tag{3.31}$$

Y definimos la frontera de V_1, ∂V_1, y el interior de V_1, V_1^0, de la siguiente forma:

$$\partial V_1 = \{x \in V_1 : r_G(x,V_2) = 1\} \tag{3.32}$$

$$V_1^0 = V_1 \setminus \partial V_1 \tag{3.33}$$

Si se verifica que

1. $d_G(x) \geq k,\ x \in V$
2. *G_2 es núcleo-conexo*
3. $\text{máx}_{x,y \in V}\, \delta(x,y) \leq 2$
4. *Una de las siguientes*
   a) $|\partial V_1| \geq k$
   ó

b) $\sum_{x \in V_1} \text{mín}\{|[x, V_1^0]|, |[x, V_2]|\} \geq k$

Entonces G es k-arista-conexo.

Una demostración de este teorema puede encontrarse en [4]. El mismo está vinculado con un famoso teorema enunciado por Plesník (ver [29], teorema 6), que asegura que en un grafo simple de diámetro 2 la arista-conexidad es igual al grado mínimo.

- Para probar entonces la *núcleo-conexidad* de un grafo, se debe verificar que cada k-núcleo sea k-arista-conexo. Por lo tanto se debe aplicar el teorema anterior a cada uno de los núcleos. Se denotará cada k-núcleo como $H_k = (C_k, F_k)$. Como cada k-núcleo es un subgrafo de G que cumple que todos sus vértices tienen grado mayor o igual a k, la condición 1 del teorema 1 se cumple inmediatamente.
- Como segundo paso, consideramos la partición del conjunto de vértices C_k en dos conjuntos C_{k1} y C_{k2}. Como la condición 2 del teorema 1 implica encontrar un subgrafo del núcleo que sea núcleo-conexo, se considerará siempre al k-núcleo inmediatamente superior. Es decir que el análisis en LaNet-vi2 se realizará en forma recursiva, desde adentro hacia afuera, comenzando por el núcleo central. De esta forma al determinar la k-arista-conexidad de un núcleo, se dispondrá de una partición para el análisis del próximo de la siguiente forma: Se considerará cada cluster Q de la capa $k+1$ unido al k-núcleo como red G, donde la partición estará formada por el cluster $Q = G_1$ y el k-núcleo G_2.

  Luego se explicará cómo se procede cuando se encuentra un k-núcleo que no es núcleo-conexo.

  Al comenzar el análisis de un k-núcleo se parte de la hipótesis de que el núcleo inmediatamente superior es núcleo-conexo, con lo cual la segunda condición se cumple. Aquí es necesario aclarar que en el caso del núcleo central (en que no disponemos de núcleo superior alguno), no empleamos este teorema para probar la k-arista-conexidad, sino el enunciado por Plesník, considerando que en la mayoría de las redes complejas el núcleo central tiene diámetro 2.

  Utilizando la notación del teorema, al considerar el k-núcleo H_k se tiene que:

1. El grafo G es el k-núcleo: $G = H_k$
2. El conjunto V está formado por los vértices en el k-núcleo: $V = C_k$
3. El conjunto V_1 es la capa k, es decir el conjunto de vértices cuyo *shell index* es k: $V_1 = S_k$
4. El grafo G_2 es el núcleo inmediatamente superior: $G_2 = H_{k+1}$
5. El conjunto V_2 es el conjunto de vértices en el núcleo inmediatamente superior: $V_2 = C_{k+1}$

- Abordamos ahora el análisis de la condición 3, que involucra a la función delta aplicada a cada par de vértices de C_k. Se discriminarán los siguientes casos:

  1. Ambos vértices se encuentran en la capa k
  2. Ambos vértices se encuentran en el núcleo $k+1$
  3. Un vértice se encuentra en el núcleo $k+1$ y el otro en la capa k

  Ambos vértices en la capa k

  En este caso la función delta se aplica con la fórmula 3.30. Para que el valor sea menor o igual a 2, o bien ambos vértices están conectados a un vértice en la capa superior, o bien están conectados ambos a un tercer vértice en V_1.

  Como consecuencia, se deduce un método para verificar esta condición: Se divide el conjunto V_1 (capa S_k) en *clusters* (componentes conexas, ver definición 4 en sección 3.1.2), y se considera cada uno. A cada cluster se agrega un nodo ficticio, y se añade una arista hacia ese nodo desde cada nodo en la frontera ∂V_1 (notar en la fórmula 3.32 que la frontera está conformada por los vértices de V_1 que tienen al menos una arista con extremo en V_2).

  Entonces se verifica para cada cluster si su diámetro es menor o igual a 2. Pues un diámetro menor a 2 entre ellos significa que para todo par de nodos, o bien están ambos conectados a un mismo vértice en V_1 ó están cada uno conectado a algún vértice en V_2. Asímismo, un diámetro mayor a 2 implica que existen al menos un par de nodos que no cumplen con la tercera condición.

  Ambos vértices en el núcleo $k+1$

  En este caso se aplica la segunda fórmula, $\delta(v, w) = r_G(v, V_2)$ siendo $w \in V_2$, y como también $v \in V_2$ resulta $\delta = 0$.

Un vértice en el núcleo $k+1$ y el otro en la capa k
Nuevamente aplicamos la segunda fórmula, $\delta(v,w) = r_G(v,V_2)$ con $w \in V_2$. Para que se cumpla $\delta \leq 2$ o bien v está conectado a algún nodo en la frontera ∂V_1, o bien pertenece a ella. Ambas situaciones pueden resumirse en verificar que el diámetro del cluster al que pertenece v, añadiendo el nodo ficticio, no sea superior a 2.

- Resta entonces por verificar la condición 4. El caso a es fácil de comprobar: simplemente se debe observar el cardinal de la frontera y verificar que sea al menos k. Si no se cumple, se pasa al caso b. Se calcula para cada vértice $x \in V_1$ el mínimo entre dos términos:

  i. El cardinal del corte $[x, V_1^0]$, es decir la cantidad de conexiones de x con nodos internos de V_1 (que no están en la frontera).

  ii. El cardinal del corte $[x, V_2]$, que representa la cantidad de conexiones de x con nodos de V_2 (si es que está en la frontera).

  Este mínimo se acumula para todos los vértices en V_1, y si es igual o superior a k entonces la condición se cumple.

3.4.1. Implementación en LaNet-vi2

Para analizar la *núcleo-conexidad* en LaNet-vi2, se implementó un algoritmo que verifica si es diámetro de un grafo es menor o igual a 2. El algoritmo se basa en la *matriz de adyacencia* A de un grafo $G = (V, E)$, que es una matriz de tamaño $|V| \times |V|$ definida como:

$$(a_{ij}) = \begin{cases} 1 & e_{ij} \in E \\ 0 & \text{en caso contrario} \end{cases} \tag{3.34}$$

Cada fila i de la matriz de adyacencia representa al nodo v_i, y contiene una cantidad de unos igual al grado del nodo. Es decir que, si la columna j de dicha fila contiene un 1, entonces v_j está a distancia 1 de v_i. Pero además la matriz de adyacencia cuenta con una interesante propiedad: si se calcula $A + A^2$, entonces la matriz resultante tiene en la fila i tantos unos como nodos que se encuentran a una distancia menor o igual a 2 respecto a v_i.

De aquí se desprende el resultado que necesitamos para concluir si el diámetro de un grafo es o no superior a 2:

> Un grafo tiene diámetro menor o igual a 2 si y sólo si la matriz $A + A^2$ no contiene ningún elemento 0.

Es decir, si no existe ningún v_j a distancia mayor a 2 de un v_i. Aplicando este resultado evitamos calcular innecesariamente el diámetro del grafo, teniendo en cuenta que no nos interesa su valor preciso sino simplemente saber si supera a 2.

Tampoco es necesario realizar la multiplicación de A por sí misma por definición, sino que se puede simplificar la operación teniendo en cuenta que la matriz sólo contiene ceros y unos. En este tipo de matrices, se deduce que:

> La fila i de la matriz $A + A^2$ puede obtenerse como una operación OR lógica entre la fila i de A y aquellas filas j de A tales que v_j es vecino de v_i.

De este modo se determina fácilmente si el diámetro es no superior a 2, realizando en cada fila i una cantidad de operaciones OR igual a la cantidad de vecinos de v_i.

La operación se aplica a cada cluster de cada capa, determinando así qué clusters cumplen con la *núcleo-conexidad*.

El algoritmo 3.2 muestra la forma en que se determina si el diámetro del núcleo central y de cada cluster cumple con ser menor o igual a 2.

Una vez verificado el diámetro en cada cluster, se pasa a analizar la condición 4, y se determina qué clusters son núcleo-conexos y cuáles no.

El algoritmo 3.3 resume todo el procedimiento de análisis de la *núcleo-conexidad* en LaNet-vi.

El bucle **"para"** entre las líneas 10-17 calcula los cardinales de los cortes $[x, V_1^0]$ y $[x, V_2]$ analizando los vecinos de cada nodo del cluster. Luego, entre las líneas 19-22 se determina si se cumple la condición 4. En tanto, las líneas 27-28 verifican la condición 3 luego de haber agregado al cluster un nodo ficticio que se conecta con cada nodo de la frontera ∂V (líneas 24-26).

Si bien la existencia de un sólo cluster que no cumpla implica que toda la red no es núcleo-conexa, LaNet-vi2 sólo señala en la visualización a los clusters que no cumplen. Asímismo, al analizar un núcleo siempre considera que el núcleo superior es núcleo-conexo, independientemente de que existan en él clusters que no

Algoritmo 3.2 DIAMETRO 2

```
1  Entrada: Grafo G = (V, E). Los vértices se denominarán v_1, v_2, ..., v_|V|
2  Salida: Diámetro menor o igual a dos: diam = true
3  inicializar diam en false
4  inicializar Matriz de adyacencia A, vacía
5  inicializar Matriz de resultado B, vacía
6
7  Armar matriz de adyacencia A = (a_ij).
8  Para 1 ≤ i ≤ |V|
9      Fila i de B = Fila i de A
10     Para v_j ∈ N_i
11         Fila i de B = (Fila i de B) OR (Fila j de A)
12     Si Fila i de B contiene algún 0
13         return diam = false (no tiene diámetro≤ 2)
14 resultado: diam
```

Cuadro 3.7: Algoritmo de determinación de *diámetro* ≤ 2

lo sean, ya que ésto permite visualizar cuáles zonas de la red son más vulnerables y deberían ser reforzadas para mejorar la conexidad. Poder visualizar los nodos vulnerables es mucho más informativo que simplemente decir si la red es núcleo-conexa o no.

La figura 3.9 ilustra esta situación en que un cluster es núcleo-conexo a pesar de que algún cluster de un núcleo superior no lo está.

LaNet-vi2 muestra en color blanco, grisáceo, o negro (contrastando con el color de fondo respectivo) a los nodos de los clusters que no cumplen la núcleo-conexidad, diferenciándolos así del resto de la red. El análisis de núcleo-conexidad en la visualización es opcional y puede configurarse como un parámetro adicional. Si se deshabilita, entonces no se ejecuta ninguno de los algoritmos vinculados.

Ejemplo

A continuación la figura 3.11 se muestran los resultados al aplicar el análisis en una red pequeña.

Se observa que los nodos 48-49-50-51 del núcleo 3 están pintados en negro, porque constituyen un cluster que no cumple con la

Algoritmo 3.3 NUCLEO-CONEXIDAD

```
1  Entrada: Cluster H = (C, F) de un grafo G = (V, E). Sus vértices se denominarán c_1, c_2, ..., c_|C_j|.
2  Entrada: Capa k a la que pertenece el cluster.
3  Salida: núcleo-conexo
4  inicializar ∂C = ∅ (frontera de C)
5  inicializar d_i− en 0 (conexiones de cada nodo internas al cluster)
6  inicializar d_i+ en 0 (conexiones de cada nodo hacia el núcleo inmediatamente superior)
7  inicializar m en 0 (acumula el valor de la sumatoria de la condición 4 del teorema)
8  inicializar núcleo-conexo en true
9
10 Para cada c_i ∈ C hacer
11     Para cada vecino v_j ∈ N_G(c_i) hacer
12         Si v_j ∈ C
13             Incrementar d_i−
14         Si s_j > s_i
15             Si c_i ∉ ∂C
16                 ∂V = ∂C ∪ {c_i}
17                 Incrementar d_i+
18
19 Para cada c_i ∈ ∂C_j hacer
20     m = m + min(d_i−, d_i+)
21 Si m < k
22     nucleo-conexo=false
23
24 Agregar un nodo ficticio v_f al grafo H
25 Para cada nodo v_i ∈ ∂V hacer
26     Agregar arista (v_i, v_f) en F
27 Si diámetro(H) > 2
28     núcleo-conexo=false
29
30 resultado: núcleo-conexo
```

Cuadro 3.8: Algoritmo de cómputo de la *núcleo-conexidad*

núcleo-conexidad. De hecho, para ir de cualquiera de esos nodos al núcleo central (núcleo 4) sólo existen dos caminos: el que pasa por la arista (e_{10}, e_{47}) y el que atraviesa (e_4, e_{48}). La cantidad de caminos es inferior al núcleo en que se encuentran. En cambio, los nodos 51-52-53-54 también del núcleo 3 tienen 3 conexiones con el

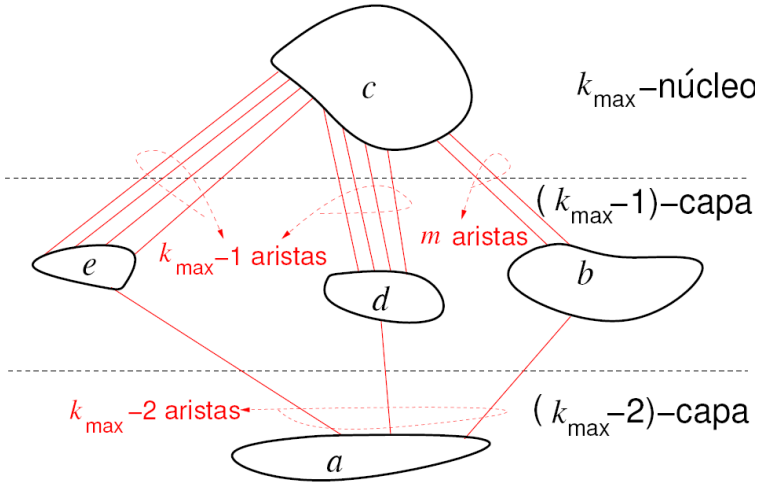


Figura 3.9: El cluster a está conectado a través de los clusters e, d y b al núcleo central. El cluster a tiene conexidad $k_{\text{máx}} - 2$, a pesar de que el cluster b tiene sólo conexidad $m < k_{\text{máx}} - 1$.

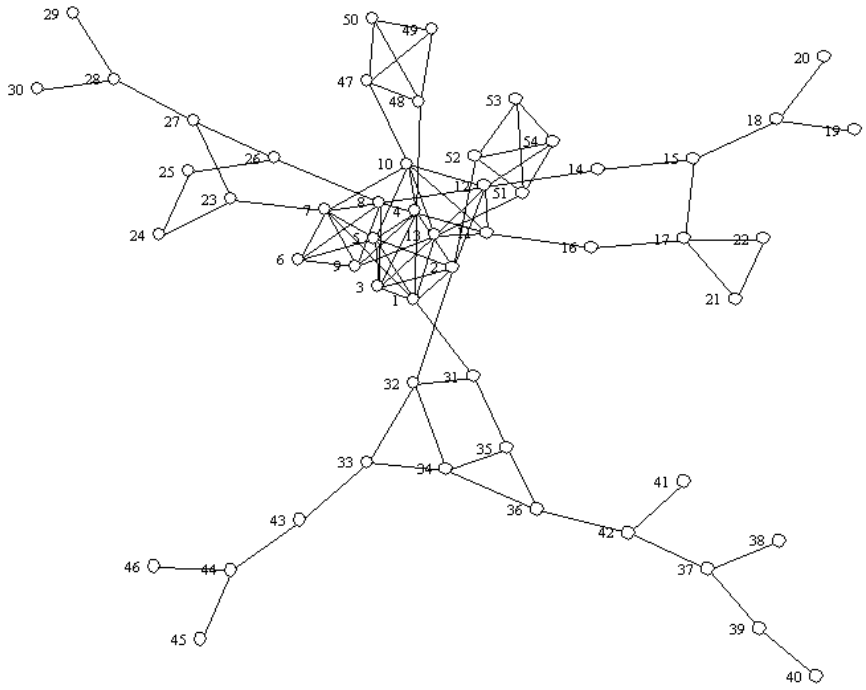


Figura 3.10: Red de ejemplo en la que se aplicó el análisis de conexidad (imagen obtenida con Network Workbench [28]).

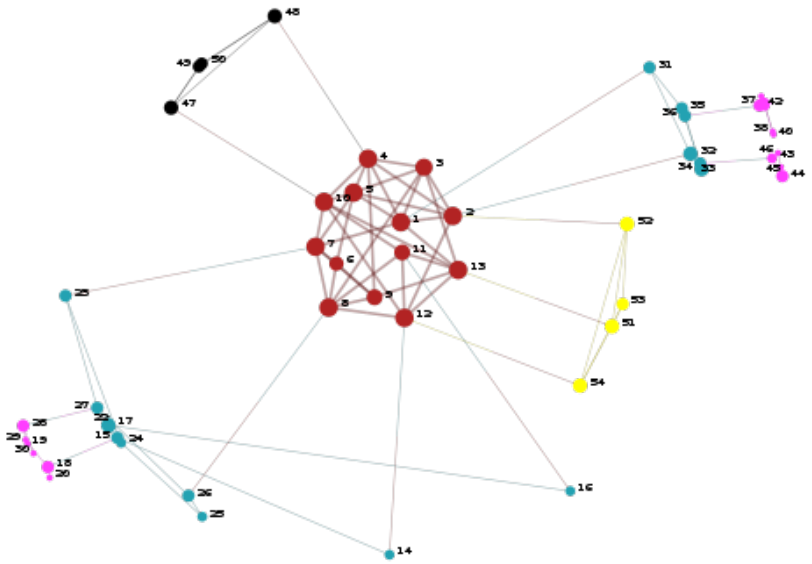


Figura 3.11: Análisis de conexidad con LaNet-vi2 para la red ejemplo de la figura 3.10.

cluster central, y por eso logran cumplir con la núcleo-conexidad.

En el núcleo 1 todos los nodos son núcleo-conexos, ya que en ellos se requiere un sólo camino hacia los demás nodos, y todos lo tienen (en efecto, el grafo es conexo).

3.4.2. Estudio de la complejidad

La complejidad computacional de la visualización no se altera con el análisis de núcleo-conexidad, que requiere un tiempo de $O(|E|)$.

Determinar si el diámetro no supera a 2, como lo requiere la condición 3, en cada cluster (C con el agregado del vértice ficticio) implica realizar una cantidad de ORs en cada fila i de su matriz de adyacencia igual a la cantidad de vecinos de v_i. Llevado a toda la matriz, ésto implica un número de operaciones OR igual a la suma de los grados de todos los nodos, lo que es de $O(|E|)$. Se considera que la operación OR entre dos filas insume tiempo constante y

que no depende del tamaño de la red, ésto se justifica a partir de la observación de que en las redes complejas que se estudiaron el tamaño de los clusters no supera los 128 nodos, y es en general inferior a 64. Hoy en día un procesador de 64 bits puede realizar entonces esta operación con una sóla instrucción de máquina y en paralelo, es decir en $O(1)$. Por lo tanto, la complejidad para la determinación del diámetro resulta en $O(|F|+|\partial C|)$, en donde $|F|$ es la cantidad de ejes del cluster y $|\partial C|$ el cardinal de su frontera. Luego, la verificación de la condición 4 requiere sólo $O(|\partial C|)$. Como conclusión, se tiene una complejidad de $O(|F|)$ en donde $|F|$ es el número de aristas del cluster.

Este proceso se repite para todos los clusters, y cada eje es visitado una cantidad constante de veces, por lo tanto la complejidad final del algoritmo 3.3 es de $O(e)$, siendo $e = |E|$, la cantidad de conexiones de la red completa.

3.5. Etiquetado de nodos

En LaNet-vi2 se ha agregado la funcionalidad de incorporar a la visualización los nombres de los nodos, o los números que tienen asignado en el grafo en caso de que los datos no incluyan nombres. Esto permite identificar por ejemplo a los nodos del núcleo central, hacer un seguimiento de la red a través del tiempo o realizar comparaciones con otras herramientas o datos conocidos.

Para controlar la superposición de nombres durante el etiquetado, se ideó una malla que divide a la imagen completa de tamaño $W(ancho) \cdot H(alto)$ en celdas de 10×12 pixeles, que es el tamaño requerido por un caracter de la fuente monoespaciada que se emplea en la imagen. Cuando se coloca una etiqueta se incrementa un contador en cada pixel ocupado por cada uno de los caracteres.

Después de posicionar y dibujar cada vértice, se analizan las celdas cercanas al mismo ubicadas:

1. A la izquierda del vértice
2. A la derecha del vértice
3. Arriba a la izquierda del vértice
4. Abajo a la izquierda del vértice
5. Arriba a la derecha del vértice

6. Abajo a la derecha del vértice

considerando una cantidad de celdas igual a la cantidad de caracteres del nombre, y se toma la posición con menor cantidad de superposiciones, las cuales se determinan a partir de los contadores de cada celda. Si dicha cantidad es menor a un cierto umbral, entonces se estampa el nombre y se incrementa en uno el contador de cada celda ocupada. De lo contrario el nombre se omite.

El método de asignar una cantidad fija de pixeles a cada letra asegura que el nombre sea legible independientemente del tamaño la red y de la resolución de la imagen renderizada. Sin embargo no se asegura que todos los nombres puedan ser estampados; lo importante es que todos los que se estampen estarán ciertamente claros. Y de todas formas, si se requiere visualizar todos los nombres siempre se puede aumentar la resolución: de esta forma se incrementa la cantidad de celdas de la malla y los nombres quedan más chicos en relación al tamaño de la imagen, permitiendo así incorporar nombres que antes no aparecían.

Por último, el usuario también puede seleccionar los nombres de qué nodos mostrar, incluyendo en el archivo de nombres sólo aquellos de dichos nodos. De esta manera puede detectar en qué capa de la descomposición en k-núcleos y en qué lugar de la imagen queda/n el/los nodo/s que le interesa/n o si cumple/n con la núcleo-conexidad, por ejemplo.

3.6. Visualización de multigrafos

LaNet-vi2 incorpora la funcionalidad de visualizar *multigrafos*. Un *multigrafo* es un grafo en el que puede existir más de una arista entre un mismo par de nodos. El abordaje de este tipo particular de grafos se realizó con las siguientes consideraciones:

1. El grado de cada vértice v_i no puede calcularse como la cantidad de vecinos, sino que debe obtenerse como la cantidad de aristas con extremo en v_i.
2. El conjunto de vecinos de un vértice v_i, N_i, no tiene vecinos repetidos. Por lo tanto cada vez que se lo recorre (por ejemplo en el algoritmo de cliqués) cada vecino sólo aparece

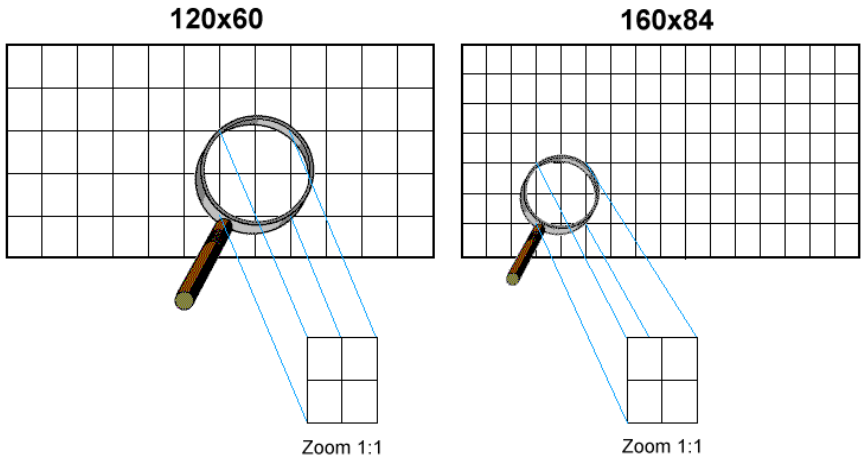


Figura 3.12: Mallas de etiquetado para imagenes con una resolución de 120x60 pixeles (izquierda) y de 160x84 pixeles (derecha). En ambas el tamaño de cada celda es el mismo (10x12 pixeles). Lo que varía es la cantidad de celdas totales en la malla, y por lo tanto el tamaño de cada caracter en relación al de la imagen, pero no su tamaño absoluto.

una vez, independientemente de la cantidad de aristas que lo conectan con v_i.

3. La multiplicidad de cada arista influye en el algoritmo de descomposición en k-núcleos, es decir que los núcleos obtenidos son diferentes respecto a los que se obtienen si se trata al grafo como simple.
4. El análisis de núcleo-conexidad no está disponible para multigrafos, ya que sólo se definieron condiciones para asegurarla en grafos simples.

La figura 3.13 ilustra la visualización de un multigrafo ejemplo.

3.7. Visualización de grafos pesados

Un *grafo pesado* es un grafo en el cual cada arista tiene asignado un *peso*. Dicho peso es un valor numérico, normalmente asociado a la capacidad, velocidad, importancia o costo de la conexión que esa arista representa. Los grafos pesados resultan muy útiles para estudiar redes de transporte como lo son las carreteras, redes de

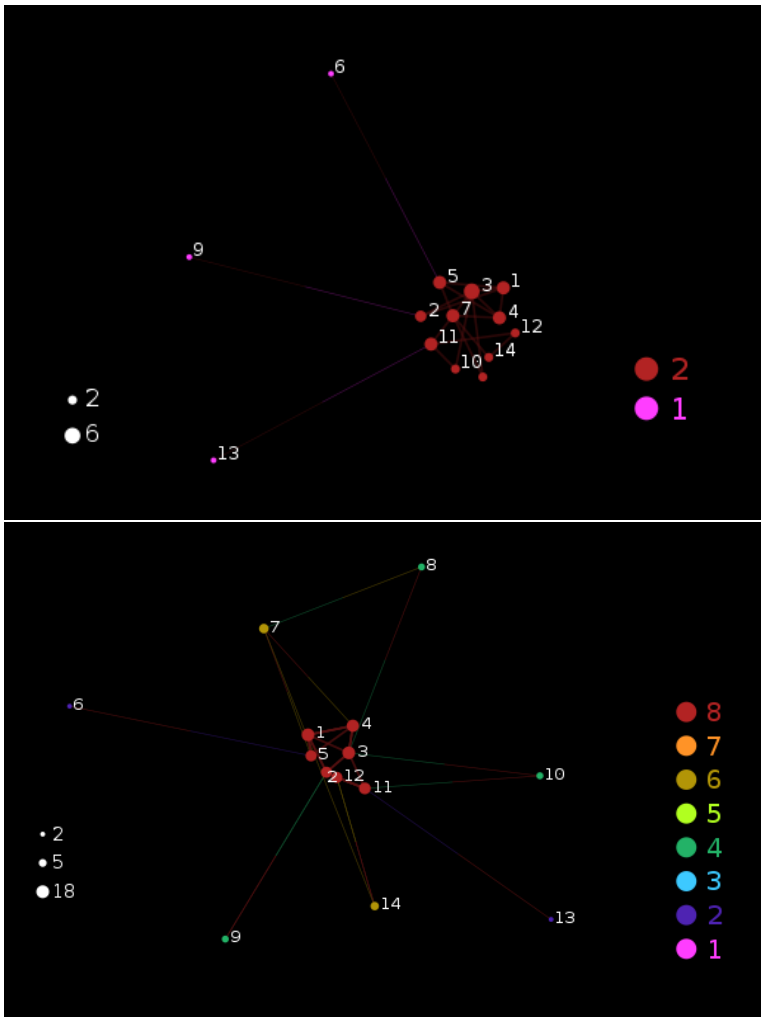


Figura 3.13: Ejemplo de grafo con múltiples aristas entre sus vértices (multigrafo). Visualización como grafo simple, sin tener en cuenta la multiplicidad de las aristas (arriba), y como multigrafo (abajo). Observar como han aumentado es este último la cantidad de capas y el grado máximo, poniendo en evidencia una jerarquía que en el primero no se percibe.

rutas aéreas y aeropuertos, las de distribución de energía, datos, etc. Al estudiar el flujo y los caminos en estas redes se hace indispensable considerar los pesos, entre otras razones porque ellos limitan el flujo entre nodos, y porque al buscar un camino suele estar presente la restricción de que sus aristas tengan un peso mínimo para así asegurar cierto flujo.

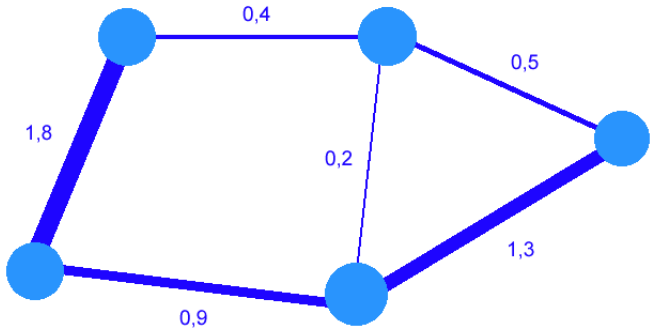


Figura 3.14: Ejemplo de grafo pesado.

Para abarcar estas redes, en LaNet-vi2 se incorporó el tratamiento de grafos pesados, indicando con un valor decimal el peso de cada arista. Los grafos pesados se representarán como $G = (V, E, w)$, en donde w es una función que asigna un peso a cada arista del grafo:

$$w : E \rightarrow \mathcal{R} \ , \tag{3.35}$$

$\mathcal{R}$ representa al conjunto de los reales.

El peso de una arista se representará como $w(e_{ij})$ o simplemente w_{ij}. Se define entonces el concepto de *strength* de un vértice [9]:

Definición 11 *Dado un grafo pesado* $G = (V, E, w)$ *el* strength *de un vértice* v_i *es la suma de los pesos de todas aquellas aristas incidentes en* v_i.

$$w(v_i) = \sum_j w(e_{ij}) \ . \tag{3.36}$$

El grado de un nodo deja ahora de tener sentido a la hora de calcular los k-núcleos, pues pretendemos que el número de capa de cada nodo, como medida de centralidad, refleje no sólo su cantidad de conexiones sino principalmente el peso de las mismas. Siguiendo la teoría de núcleos generalizada en [10], se optó por definir una *función p* o *p-function* que reemplace al grado por un valor vinculado con el *strength* a la hora de realizar la descomposición en capas.

Para que la cantidad de capas sea discreta y sólo existan k-núcleos con $k \in \mathcal{N}$, la *función p* debe asignar a cada nodo un número

natural:

$$p : V \rightarrow \mathcal{N} \ . \tag{3.37}$$

Aquí se decidió entonces armar la *función p* a partir de una subdivisión en intervalos de $\mathcal{R}$ basada en los valores de $w(v_i)$ de los vértices. La cantidad de subintervalos es un parámetro de la visualización, y lo llamamos *granularidad*. Tomando una granularidad de 5 tendríamos, por ejemplo:

$$p(v_i) = \begin{cases} 1 & w(v_i) < 18{,}1 \\ 2 & 18{,}1 \leq w(v_i) < 19{,}8 \\ 3 & 19{,}8 \leq w(v_i) < 20{,}2 \\ 4 & 20{,}2 \leq w(v_i) < 21{,}3 \\ 5 & w(v_i) > 21{,}3 \end{cases} \tag{3.38}$$

El rango de cada intervalo es determinado por LaNet-vi2 conforme al criterio de que la distribución de los pesos de las aristas en cada intervalo sea uniforme. Para ello se ordena el conjunto de pesos de las aristas, y se lo subdivide en una cantidad de intervalos igual a la *granularidad*, de modo que quede la misma cantidad de pesos w_i en cada intervalo.

Este criterio puede interpretarse de la siguiente manera: Dada la distribución de probabilidad acumulada de los pesos de las aristas, $F(w)$, se toman los valores de w que cumplen que

$$F(w) = \frac{k}{granularidad} \ , 0 \leq k \leq granularidad \tag{3.39}$$

Estos valores de w son los que determinan los límites de los intervalos para la *función p*. La figura 3.15 ilustra este procedimiento.

Por último, la descomposición en capas se realiza a partir de esta redefinición de k-núcleo para grafos pesados, basada en [10]:

Definición 12 *Un subgrafo $H = (C, E|C)$ inducido por $C \subseteq V$ es un k-núcleo si y sólo si:*

$$\forall v \in C : p_H(v) \geq k \tag{3.40}$$

y H es el subgrafo máximo que cumple esta propiedad. $p_H(v)$ es el valor de la función p *aplicada a v en el subgrafo H.*

La figura 3.16 muestra una red de colaboraciones científicas tratada como un grado pesado.

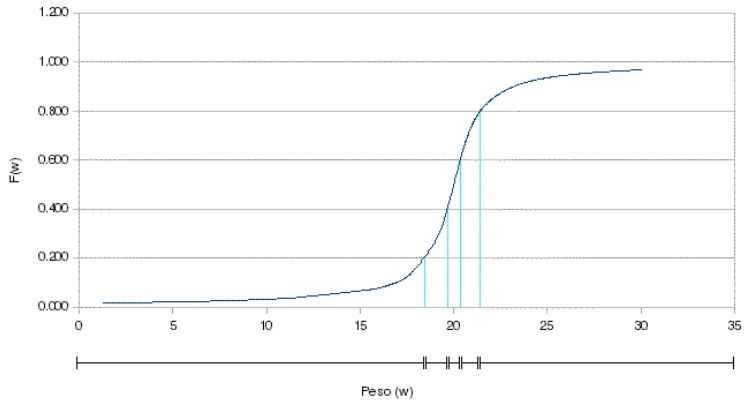


Figura 3.15: Ejemplo de subdivisión en 5 intervalos del conjunto de pesos en base a la función de distribución acumulada de probabilidad (ver ecuación 3.38).

3.8. Renderizado y transparencias

LaNet-vi emplea un *render*[3] para visualizar las imágenes, con el objetivo de poder representar a los objetos en un espacio tridimensional, asignándoles una profundidad y generar así imágenes más descriptivas que las que se obtendrían con un software de dibujo más simple.

En LaNet-vi2 las imágenes pueden obtenerse escogiendo entre dos *renders* posibles:

1. PovRay (Persistence of Vision Raytracer)[4]: Está basado en *raytracing*, una técnica que consiste en el trazado de rayos desde la cámara hacia distintos puntos de la escena, calculando las intersecciones con los objetos.
2. SVG (Scalable Vector Graphics): Es un lenguaje de descripción de gráficos vectoriales en dos dimensiones. La implementación que se utilizó es la de la librería `librsvg` para GNU/Linux[5].

[3]Un *renderizador* o *render* es un software que genera una imagen a partir de una descripción de cada uno de sus objetos en términos de su geometría, textura, iluminación, etc.

[4]http://www.povray.org/

[5]http://librsvg.sourceforge.net/

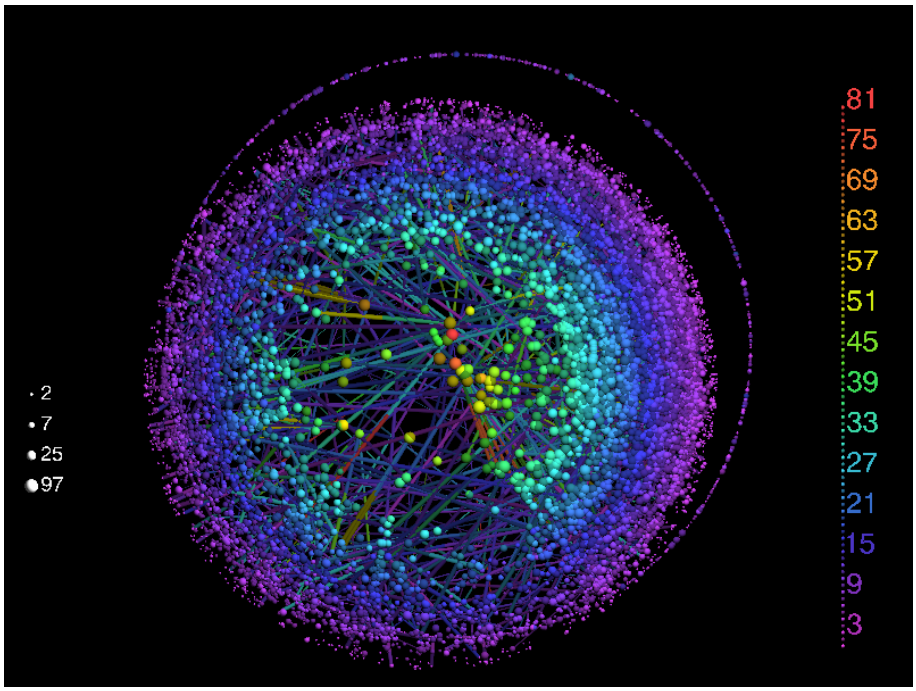


Figura 3.16: Red de colaboraciones científicas visualizada con LaNet-vi2. Los nodos representan autores, y las aristas a un trabajo conjunto. El grafo de la red es un grafo pesado, y el peso de cada arista está vinculado con la cantidad de colaboraciones entre un par de autores. La granularidad escogida es 6. Notar que el grosor de cada arista es proporcional a su peso.

PovRay ya era empleado por la primera versión, mientras que SVG fue incorporado en LaNet-vi2. En SVG se puede definir un cierto grado de transparencia en los ejes sin comprometer el desempeño, que permite llegar a visualizar el 100 % de ellos inclusive sin ocultar a los vértices y manteniendo la legibilidad de la imagen.

Capítulo 4

Visualización de redes con LaNet-vi

En este capítulo se mostrarán los resultados obtenidos del análisis de distintos tipos de redes con LaNet-vi2. Dicho análisis se enfocará en los siguientes atributos de la visualización:

Grosor de las capas: Los vértices en una misma capa se ubican en una corona, cuyo radio varía según su conexión con vecinos de capas superiores. El grosor de la corona puede entonces indicar que tan distantes en capas se encuentran sus vecinos, y puede ajustarse para la visualización a través del parámetro ϵ [6].

Correlación grado-shell index: El grado y el shell-index son medidas de centralidad, y por tanto resulta importante medir su correlación. El grado puede visualizarse a partir del tamaño de cada nodo (en escala logarítmica), en tanto los shell-indexes pueden diferenciarse por el color (desde el azul para las capas inferiores hasta el rojo para el núcleo central). Las escalas siempre aparecen referenciadas en la imagen obtenida.

Componentes disconexas: Un k-núcleo puede tener más de una componente conexa. Esto se remarca dibujando cada componente en circulos separados y "repelidos". El círculo de cada componente es proporcional a la cantidad de nodos de la misma. El parámetro γ controla el diámetro, y δ permite ajustar la distancia entre componentes [6].

Clusters: Los clusters dentro de cada capa (componentes conexas) pueden visualizarse si se desactiva la descomposición en cliqués. Los vértices que pertenecen al mismo cluster se dibujan juntos en un sector circular.

Aristas: La densidad de aristas da una idea del grado de conexión de la red, y del grado promedio de sus nodos. Una elevada cantidad de aristas no siempre implica robustez, para ésto último es más apropiado evaluar la núcleo-conexidad. El usuario puede regular la proporción de aristas a mostrar a los efectos de obtener una imagen más clara. En el caso de emplear el renderer SVG, se puede sacar provecho de la transparencia para dibujar todas las aristas sin ocultar los vértices.

Además, los colores de las aristas permiten determinar como están conectadas las capas, ya que cada extremo de la arista se pinta del mismo color que el vértice en el extremo opuesto.

Núcleo central: Se muestra como un conjunto de cliqués, cada uno ocupando un sector circular distinto y proporcional a su tamaño. Este despligue permite visualizar la proximidad entre los vértices del núcleo central. Además, como el resto de los vértices organizan su posición conforme a estos cliqués, se puede determinar si existen algunos cliqués que están más conectados a la red completa, y por lo tanto son más influyentes en la conexidad.

Núcleo-conexidad: LaNet-vi2 señala aquellos clusters cuyos vértices no verifican las condiciones para la núcleo-conexidad. Esta propiedad sólo es útil en redes con una única componente conexa. En caso de empleara en otras redes, LaNet-vi se limita sólo a analizar la mayor de las componentes. Por ejemplo, en algunos mapas de Internet, debido a sesgos en las exploraciones pueden obtenerse grafos con varias componentes, pero sólo interesa analizar la mayor de ellas; las restantes normalmente son muy pequeñas.

Etiquetado de los nodos: LaNet-vi2 puede mostrar los nombres de los nodos en la imagen, permitiendo así identificar cuáles son los nodos en el núcleo central, o cuáles no son núcleo-conexos, por ejemplo.

Grafos pesados: En el caso de redes pesadas, el peso de cada arista queda reflejado en el grosor del eje reflejado, permi-

tiendo así sacar conclusiones respecto a, por ejemplo, la correlación peso-grado de los extremos o peso-shell index de los extremos.

Por otra parte, se empleará información estadística que LaNet-vi2 genera respecto a los tamaños de las capas, cliqués, distribuciones de grados, etc, para contrastarla con las imágenes generadas.

4.1. Redes Biológicas

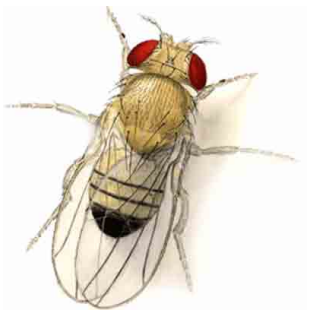


Figura 4.1: Mosca de la fruta *(Drosophila melanogaster)* (imagen extraída de http://www.life.uiuc.edu/).

Se estudiaron las propiedades de la red de interacciones entre proteínas de la mosca de la fruta *(Drosophila melanogaster)*. Se trata de 330 proteínas con 295 interacciones.

En esta red no se observa correlación grado-shell index: Existen proteínas que interactúan con muchas otras (grado alto) pero quedan en la capa 1, y también existen algunas en el núcleo central pero con pocas interacciones, seguramente porque sus interacciones son con otras proteínas del núcleo central.

En el gráfico se puede ver que la cantidad de conexiones es baja; asímismo LaNet-vi2 informa que la red no satisface las condiciones de núcleo-conexidad, y que ni siquiera el núcleo central más grande tiene diámetro 2, sino 4. Hay muchas componentes conexas, evidenciando que existen proteínas aisladas, que no interactúan con ninguna otra.

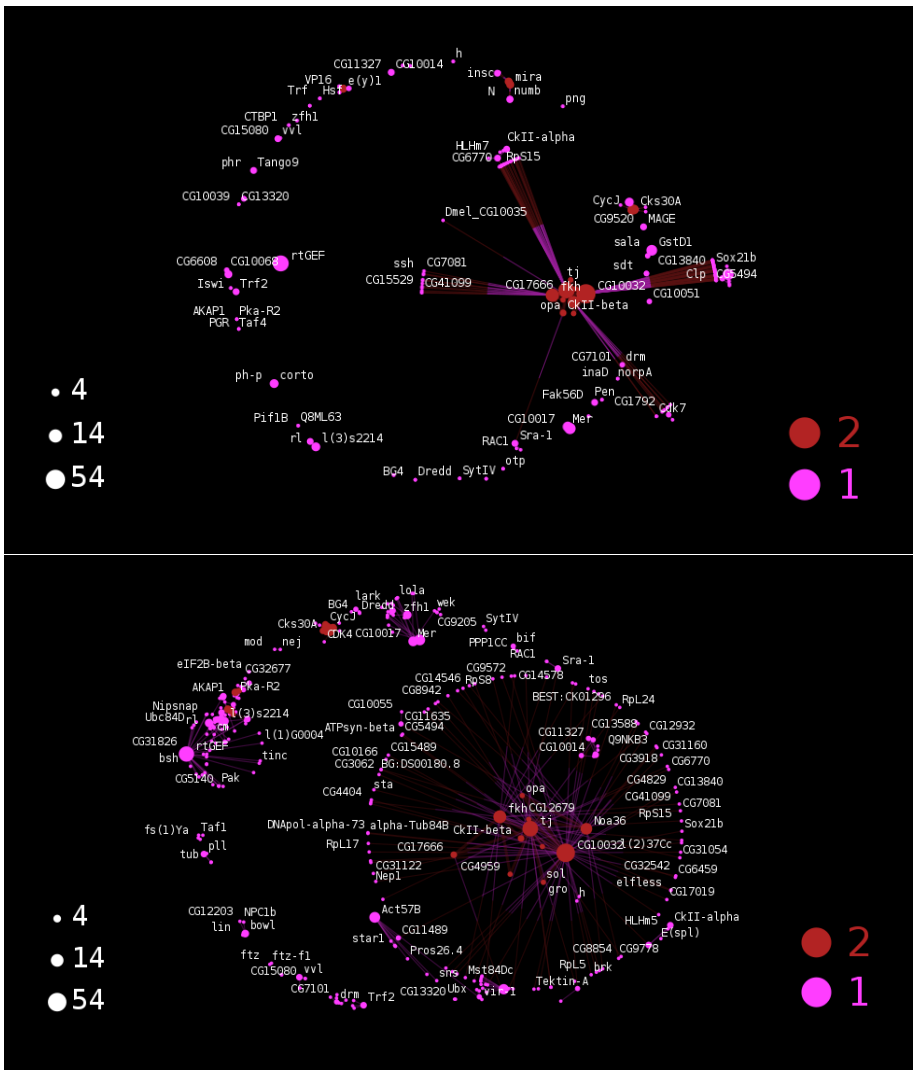


Figura 4.2: Red de interacciones entre proteinas de la mosca de la fruta. Visualización organizada por cliqués (arriba) y por clusters (abajo). Render utilizado: SVG.

También se observa el fenómeno de acoplamiento preferencial: las proteínas que tienen pocas interacciones, las tienen con proteínas de gran cantidad de interacciones. La distribución de los grados sigue una ley:

$$p(d_v) = 0{,}66 \cdot d_v^{-2{,}35} \quad . \tag{4.1}$$

La misma se obtuvo haciendo una regresión sobre la distribución de los grados, estadística que se obtuvo con un programa que

desarrollamos para tal fin.

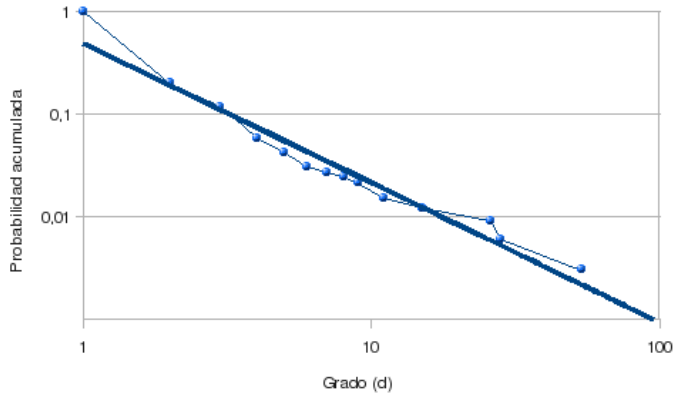


Figura 4.3: Distribución de grados de la red de interacciones entre proteínas.

El comportamiento de la red según la imagen es de tipo *disassortative*: las proteínas centrales interactúan poco entre sí, pero en gran cantidad con otras proteínas periféricas.

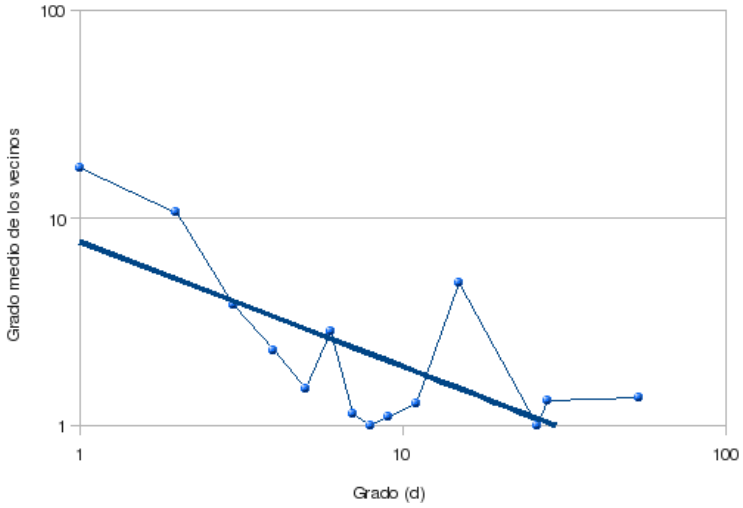


Figura 4.4: Distribución del grado medio de los vecinos para la red de interacciones entre proteínas. Si bien la escasez de nodos no permite extraer conclusiones determinantes, se nota una tendencia al comportamiento *disassortative*.

4.2. Redes de Transporte

Se analizaron las características de dos redes de transporte:

1. La red mundial de aeropuertos
2. La red de transporte de la ciudad de Berlín

Red mundial de aeropuertos

En esta red cada nodo del grafo representa un aeropuerto. Dos aeropuertos están conectados si existe un vuelo directo entre ellos. La red cuenta con 18818 conexiones entre 3880 aeropuertos, y su distribución es libre de escala, como muestra la siguiente regresión:

$$p(d_v) = 8{,}96 \cdot d_v^{-2{,}39} \tag{4.2}$$

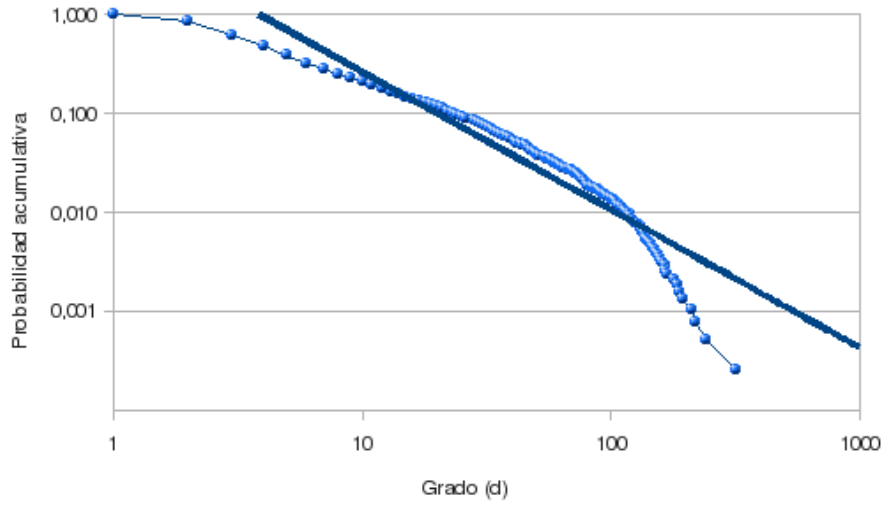


Figura 4.5: Distribución de grados de la red mundial de aeropuertos.

Se observa una importante correlación grado-núcleo: los aeropuertos con mayor cantidad de conexiones son los del núcleo central. También se cumple en gran medida con la núcleo-conexidad; sólo unos pocos nodos de capas inferiores no lo están, como se observa en la figura 4.6. Sorprende el nivel de conexidad del núcleo central: de sus 35 nodos, 27 de ellos se encuentran todos conectados

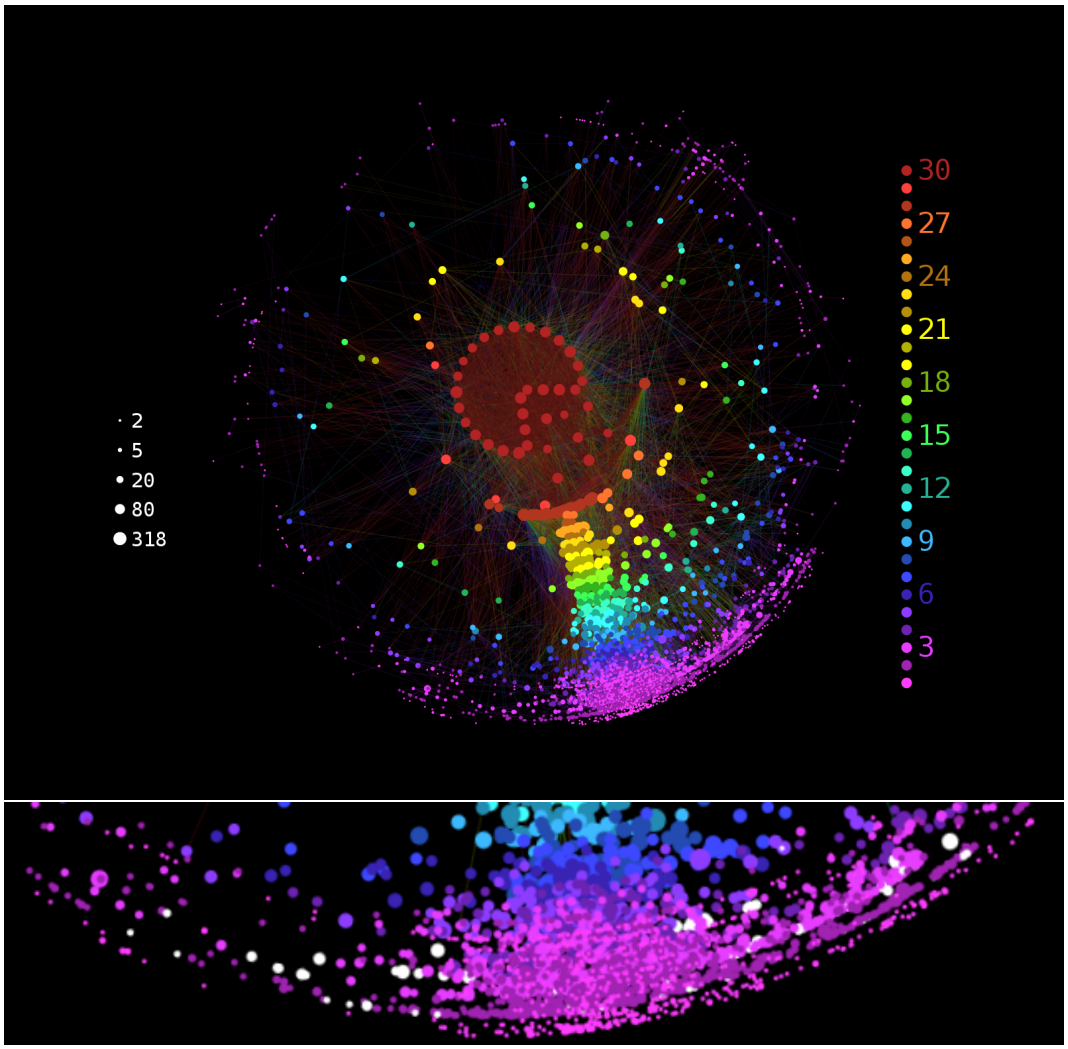


Figura 4.6: Visualización de la red mundial de aeropuertos (SVG). Mapa completo (arriba) y nodos que no cumplen con la núcleo-conexidad (abajo).

entre sí; por otra parte entre todo par de nodos centrales existen al menos 30 rutas arista-disjuntas.

Mirando los colores de los ejes se verifica una organización jerárquica y comportamiento muy levemente *assortative*. Los nodos del centro de la red tienen abundantes conexiones entre ellos mismos. Los nodos de la periferia se encuentran también muy conectados con nodos del núcleo central que son los de mayor grado, pero no conectados entre sí, lo que implica un sistema altamente jerárquico.

Finalmente, la alta conexidad del núcleo central, sumada a que prácticamente todos los nodos están conectados a dicho núcleo la convierte en una red de mundo pequeño *(small world)*: se puede llegar de una ciudad a otra con muy pocos trasbordos. Concretamente, la distancia promedio (*shortest path length)* de esta red de casi 4000 nodos es de sólo 4.35. Las ciudades del núcleo central son las que actúan como *hubs*, es decir que median en el camino mínimo entre muchos pares de aeropuertos[1].

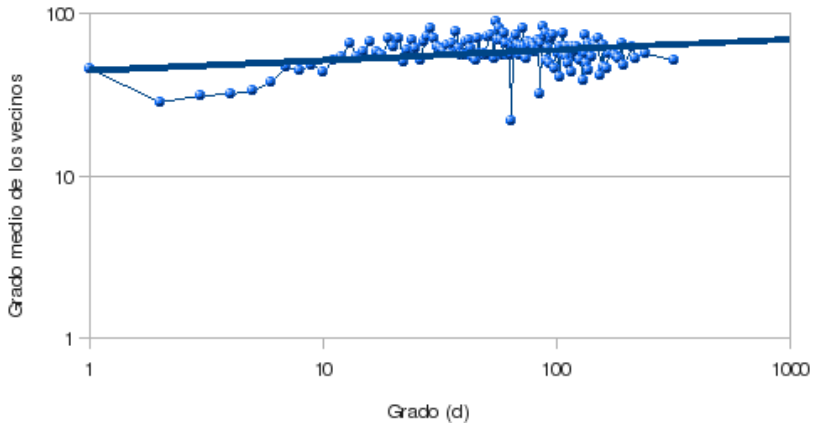


Figura 4.7: Distribución del grado medio de los vecinos para la red mundial de aeropuertos. Se observa un comportamiento muy levemente *assortative*.

Red de transporte de Berlín

La red de transporte obtenida de este centro urbano consta de 327 estaciones con 722 conexiones.

No existe una correlación significativa entre el grado y el número de capa: existen estaciones muy conectadas que no pertenecen al núcleo de la red. El núcleo central tiene diámetro 3, y no se cumplen las condiciones de núcleo-conexidad.

[1] Verificado calculando el *average shortest path length* (distancia promedio) con Network Workbench [28].

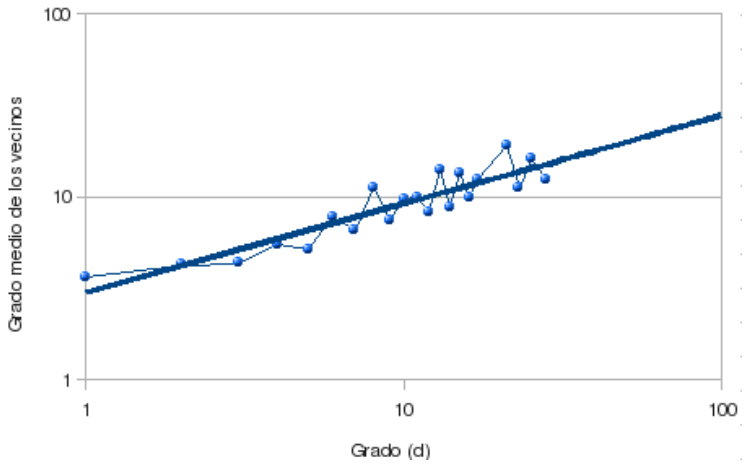


Figura 4.8: Distribución del grado medio de los vecinos en la red de transporte de Berlín. Su comportamiento es de tipo *assortative*.

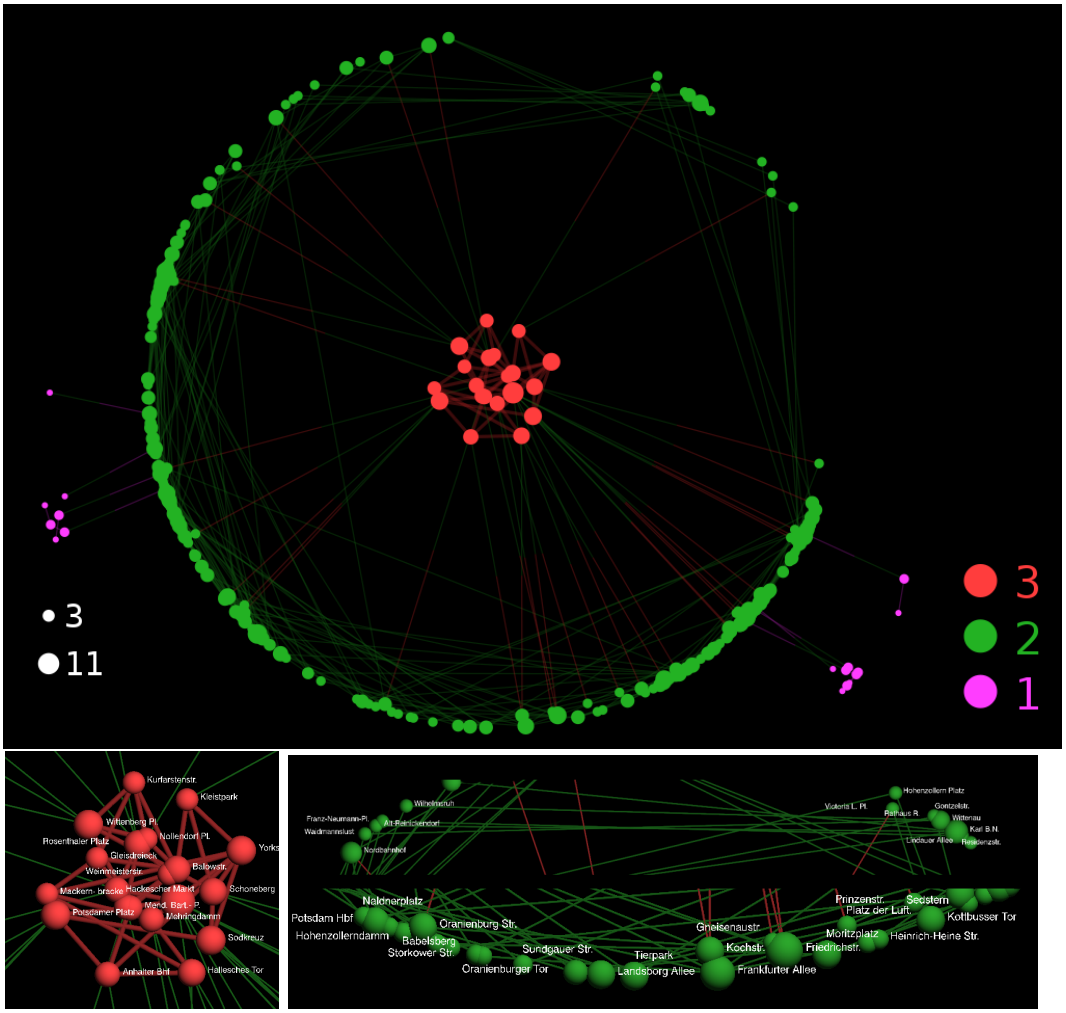


Figura 4.9: Visualización de la red de transporte de Berlín. Mapa completo (arriba, SVG), estaciones del centro de la red (abajo a la izquierda, PovRay) y de la periferia (abajo a la derecha, PovRay). Las imágenes de abajo incluyen los nombres de las estaciones.

4.3. Redes Sociales

Para observar la forma en que LaNet-vi2 refleja las características de estas redes, se empleó un conjunto de datos de intercambios

y recomendaciones de fotos en la comunidad virtual *Flickr*[2]. Este conjunto está formado por 229.709 intercambios (las aristas) entre 35.210 miembros (los nodos). La distribución de grados (es decir, de la cantidad de amigos con que cada miembro intercambia fotos) es:

$$p(d_v) = 153{,}5 \cdot d_v^{-3{,}11} \quad . \tag{4.3}$$

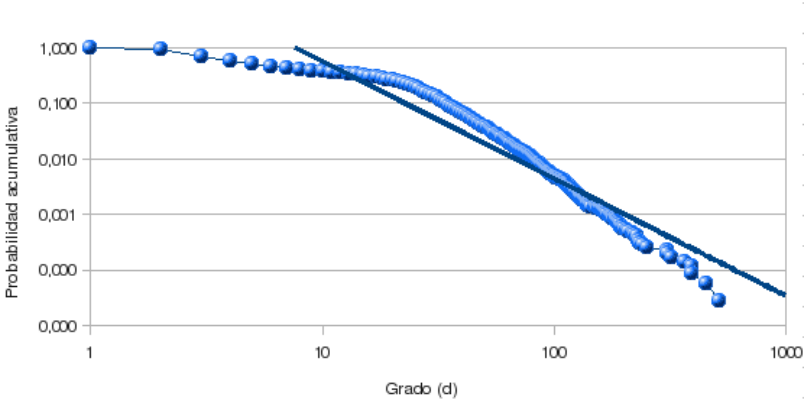


Figura 4.10: Distribución de grados de una red de intercambio de fotos en Flickr.

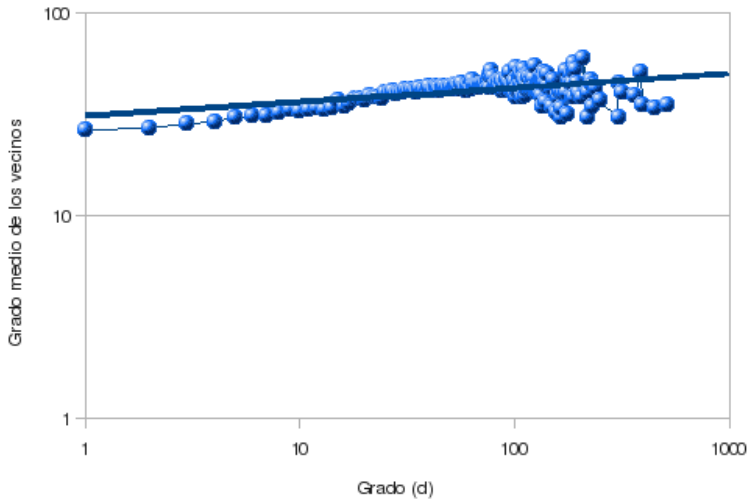


Figura 4.11: Distribución del grado medio de los vecinos para una red de intercambio de fotos en Flickr. Es muy levemente *assortative*.

[2]http://www.flickr.com/

El comportamiento es muy levemente *assortative*, como muestra la distribución del grado medio de los vecinos (ver figura 4.11). Los miembros que intercambian muchas fotos, lo hacen con otros que también son frecuentes usuarios de Flickr.

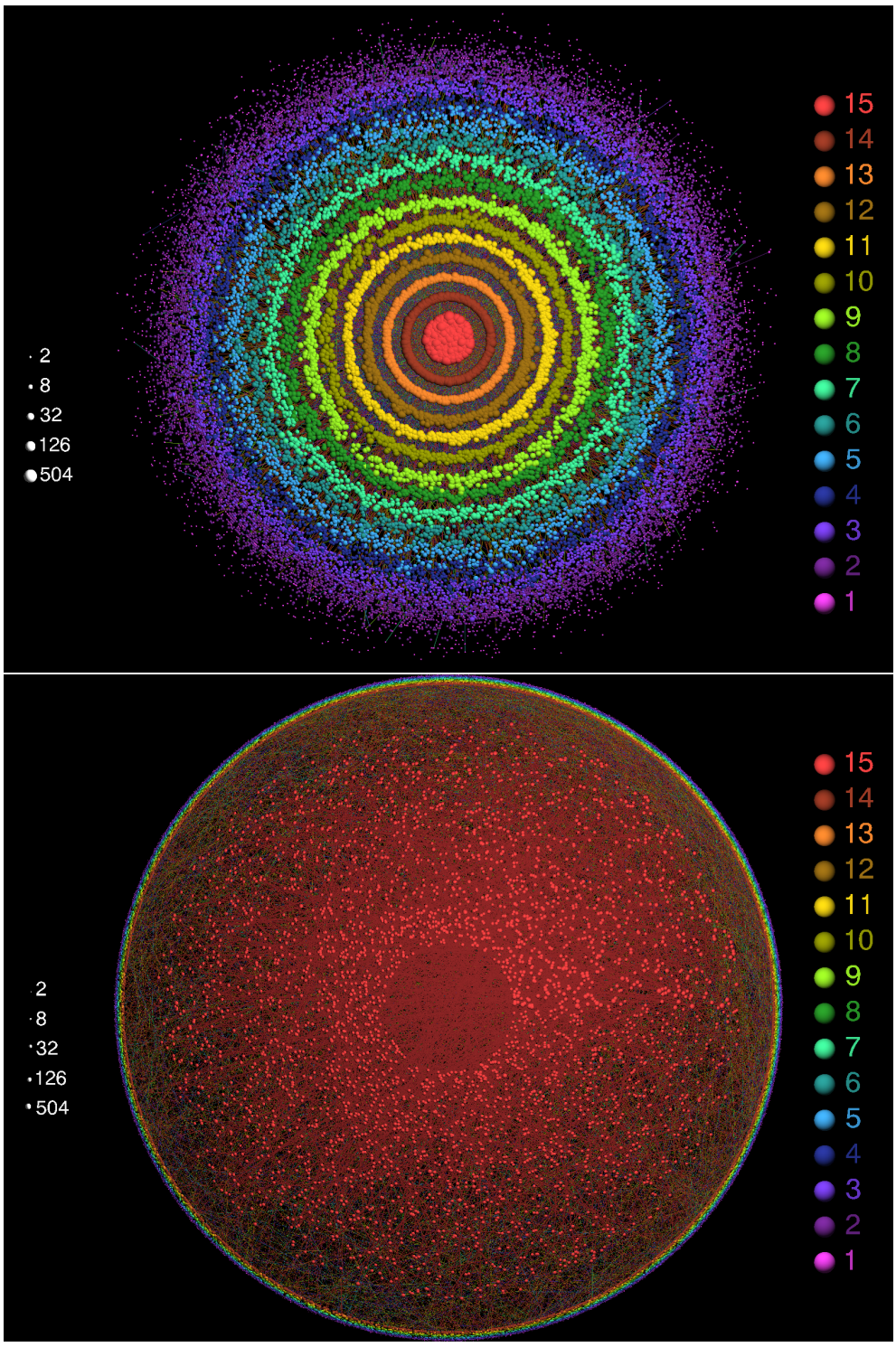


Figura 4.12: Red de intercambio de fotos en Flickr. Visualización en LaNet-vi2 con el algoritmo de LaNet-vi1 (arriba) y con descomposición en cliqués (abajo), ambas empleando el render PovRay.

Las imágenes y la información generadas por LaNet-vi2 (figura 4.12) nos permiten extraer las mismas conclusiones:

1. Los miembros de mayor grado se ubican en el centro, verificando la correlación grado-número de capa (imagen superior).

2. Hay una gran cantidad de miembros en el centro y muy conectados entre sí (imagen inferior), actuando como hubs, lo que denota el comportamiento *assortative* y el efecto de mundo pequeño característico de estas redes.

3. El análisis de núcleo-conexidad indica que el núcleo central tiene diámetro 5, por tanto no se cumplen las condiciones requeridas por el teorema 1 para asegurar la núcleo-conexidad. Ésto también es intuitivo: para que el diámetro sea 2 todos los pares de miembros de dicho núcleo deberían tener un amigo en común. Un caso de mundo pequeño tan extremo es muy poco frecuente en la realidad. Sin embargo un diámetro 5 no deja de ser sorprendente teniendo en cuenta que cerca de 4000 miembros se hallan en el núcleo central, y sólo están separados por 3 amigos intermedios.

4. No se encuentran clusters grandes: La información que loguea LaNet-vi indica que el máximo cliqué encontrado tiene tan sólo tamaño 7.

4.4. Redes de Información

Dentro de este grupo, se estudió una mapa de una porción de la World Wide Web correspondiente al dominio *.fr*, con alrededor 1 millón de páginas y 3,5 millones de enlaces. Si bien en la Web los enlaces son dirigidos, se hizo una simplificación para poder adaptarla a LaNet-vi2, tratando a las aristas como no dirigidas.

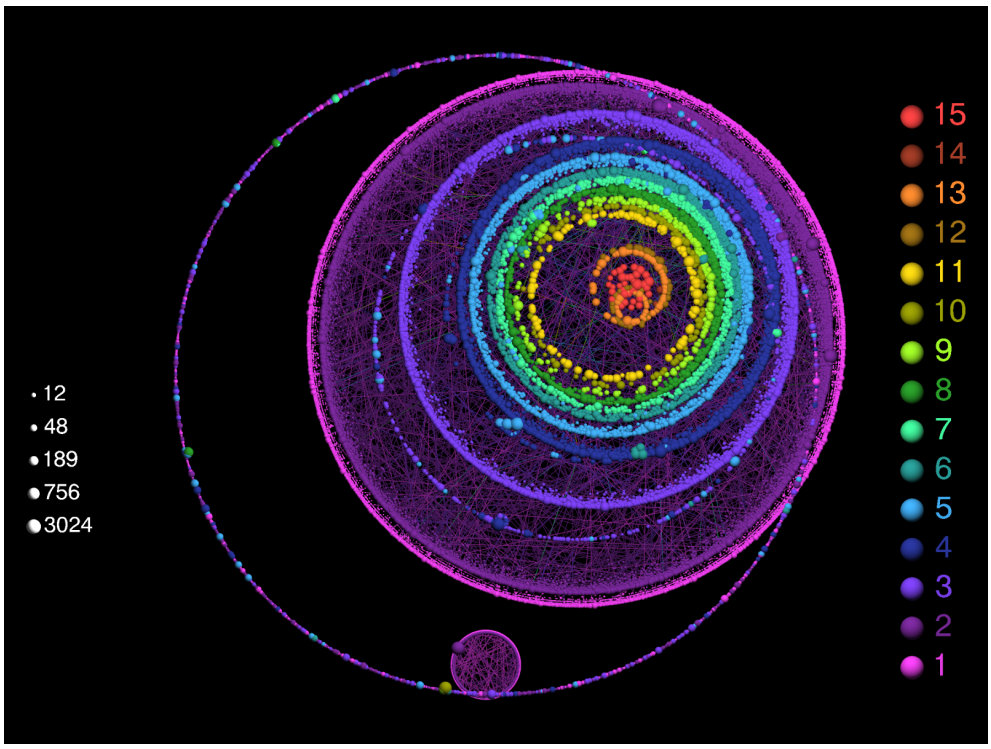


Figura 4.13: Mapa de una porción del dominio *.fr* de la Web (PovRay).

Se observa la gran cantidad de conexiones entre nodos de la primera capa, la ausencia de conexiones entre los nodos centrales y también la existencia de nodos con grado alto en la periferia. Entonces concluímos que la correlación grado-núcleo no es tan concisa como en los casos anteriores, y que el diámetro de la red seguramente es mucho mayor que en otras redes como las sociales o la de aeropuertos.

4.5. Internet

Para estudiar la topología de Internet con LaNet-vi2 se obtuvieron datos de exploraciones en dos niveles:

1. de ruteadores.
2. de Sistemas Autónomos.

Se extrajeron mapas de 3 fuentes distintas:

1. Oregon Route Views[3]: Este proyecto almacena las tablas de ruteo del algoritmo BGP en varios ruteadores BGP localizados y que tienen una gran cantidad de conexiones. De esta forma obtiene rutas públicas de Internet, aunque con la desventaja de que no todas las rutas son públicas debido a las políticas de cada sistema autónomo.
2. CAIDA[4]: Este proyecto emplea un par de decenas de sensores *skitter*, una herramienta de la familia de traceroute y ping, que permiten sondear mapas de Internet. Básicamente descubre caminos enviando paquetes a distintas direcciones de Internet, y luego a partir de esta información trata de reconstruir el mapa de la red. La ventaja respecto a la fuente anterior es que obtiene información de caminos reales, deduciendo así rutas que no están declaradas en algunas tablas públicas de BGP.
3. Proyecto DIMES[5]:Emplea un sistema distribuido compuesto por miles de sondas que obtienen también caminos reales. La diferencia con CAIDA es la cantidad de sondas, cerca de 1000 veces mayor, razón por la cual los mapas obtenidos son mucho más precisos.

4.5.1. Visualización de Sistemas Autónomos

En este nivel cada nodo o sistema autónomo *(Autonomous System, AS)* es una red cuya administración está centralizada y a cargo de una entidad, mientras que las conexiones entre ellas están dadas por políticas que suelen reflejar estrategias y acuerdos comerciales. Estas conexiones pueden ser dirigidas, pero para aplicar el algoritmo de descomposición en k-núcleos las consideramos como no-dirigidas. De hecho, al estudiar la robustez de la red y la conexidad no nos interesan las conexiones comerciales sino las posibles, y en realidad todos los enlaces de comunicación que conectan ASes son bidireccionales.

Además de analizar las características que se estudiaron en las redes anteriores, en esta sección también se verá cómo es posible

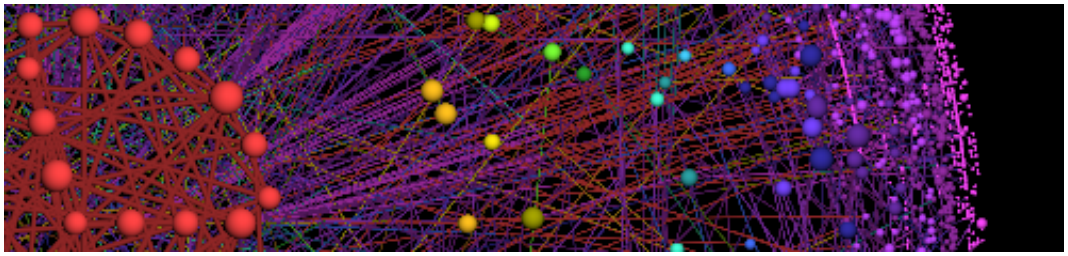
[3]University of Oregon Route Views Project. http://www.routeviews.org/.

[4]Cooperative Association for Internet Data Analysis, Router-Level Topology Measurements. http://www.caida.org/tools/measurement/skitter/.

[5]Distributed Internet MEasurements and Simulations. http://www.netdimes.org/.

a partir de la visualización detectar cuál fue la fuente de la que se obtuvieron los datos. Para que las conclusiones sean fehacientes, en cada fuente se extrajeron sets de datos de años distintos. Las imágenes fueron renderizadas con PovRay o SVG, algunas con ambos. En todas ellas se ha activado el análisis de núcleo-conexidad de LaNet-vi2.

Oregon Route Views



A través del proyecto Oregon Route Views se extrajeron exploraciones de abril de 2005 y de febrero de 2008.

De las imágenes se observa que todos los mapas tienen varios cliqués en el núcleo central, y que los ejes surgen homogéneamente en todas direcciones. Entre 2005 y 2008 el grado máximo aumentó, aunque el número de capas ha disminuído.

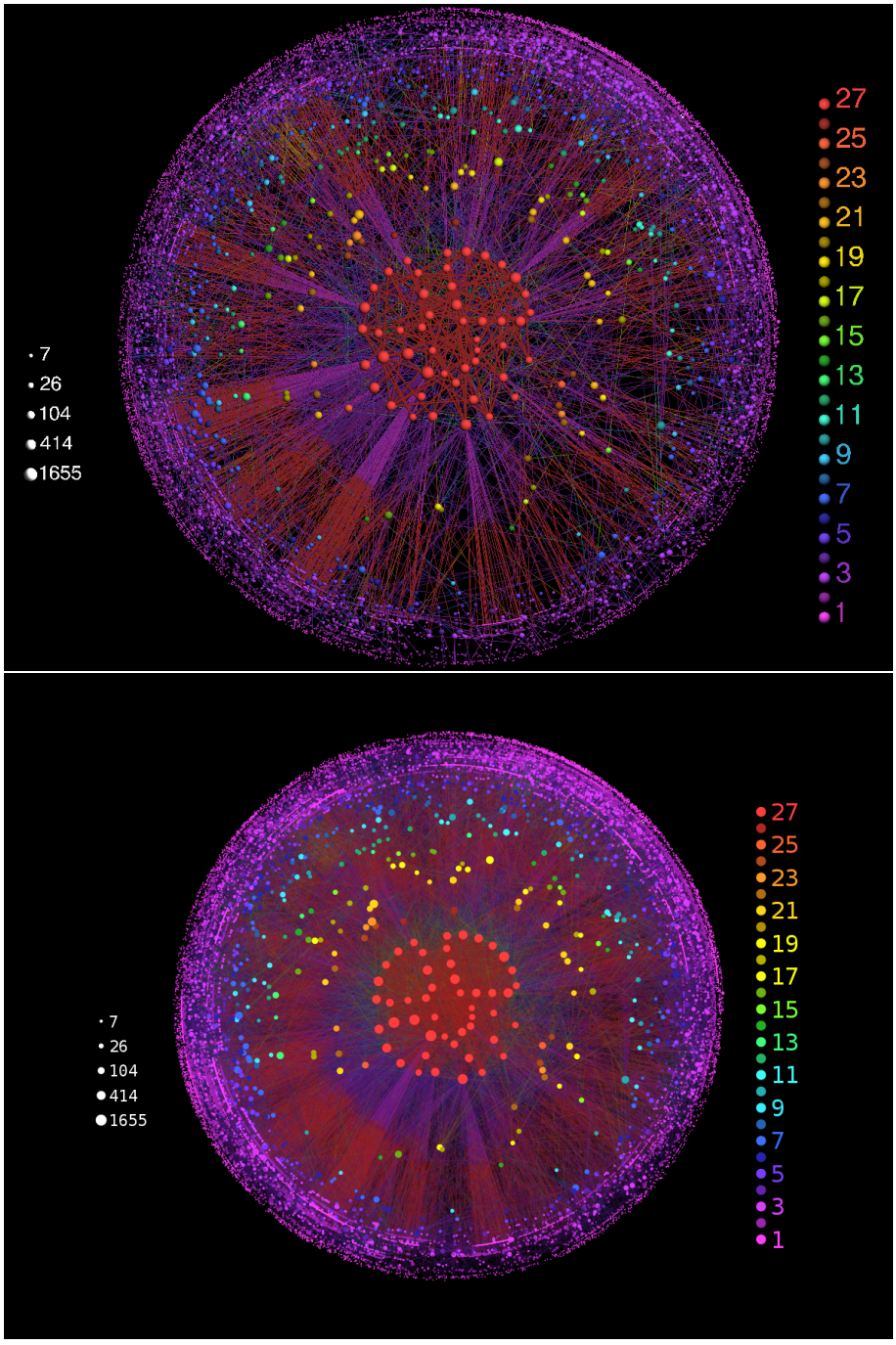


Figura 4.15: Imágenes de la red de Sistemas Autónomos en abril de 2005 obtenida del proyecto Route Views. Renderización con PovRay (arriba) y con SVG (abajo).

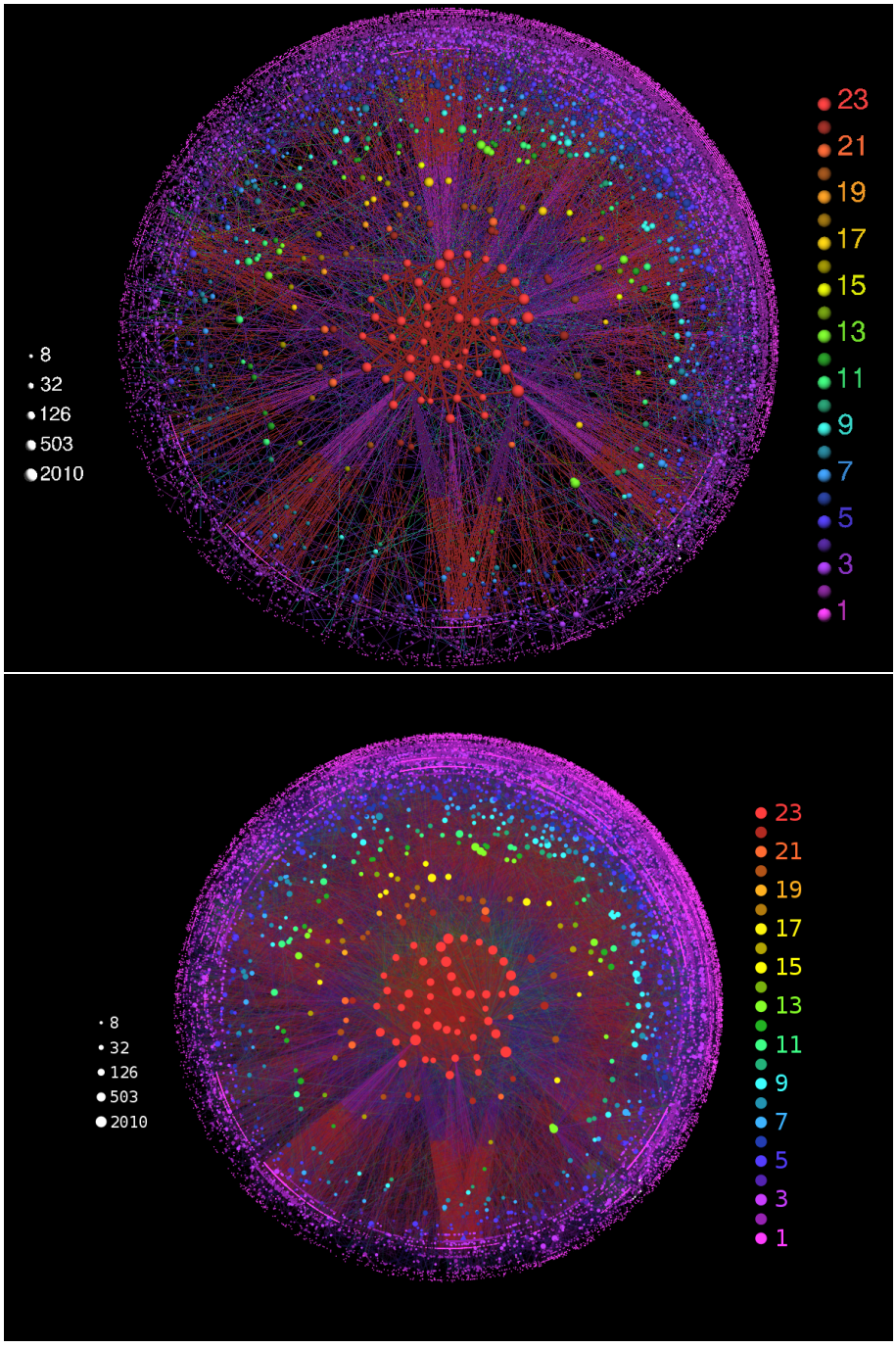


Figura 4.16: Imágenes de la red de Sistemas Autónomos en febrero de 2008 obtenida del proyecto Route Views. Renderización con PovRay (arriba) y con SVG (abajo).

En todos los gráficos las capas inferiores están muy conectadas con las superiores, por ello el predominio del color rojo en los ejes. También hay una alta correlación grado-shell index. Estas dos características implican un comportamiento *disassortative*.

Un análisis estadístico de los gráficos nos permite corroborar las propiedades deducidas a partir de las imágenes: La red en el 2008 posee 25,523 sistemas autónomos y 45,252 conexiones. La distribución de grados de los ASes es:

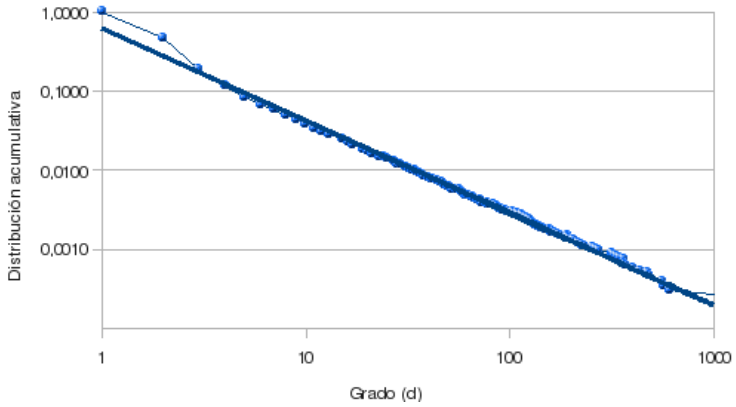


Figura 4.17: Distribución acumulativa de grados de la red de ASes obtenida con Route Views (2008).

$$p(d_v) = 0,72 \cdot d_v^{-2,17} \quad , \tag{4.4}$$

y la distribución de los grados de los vecinos en función del grado es:

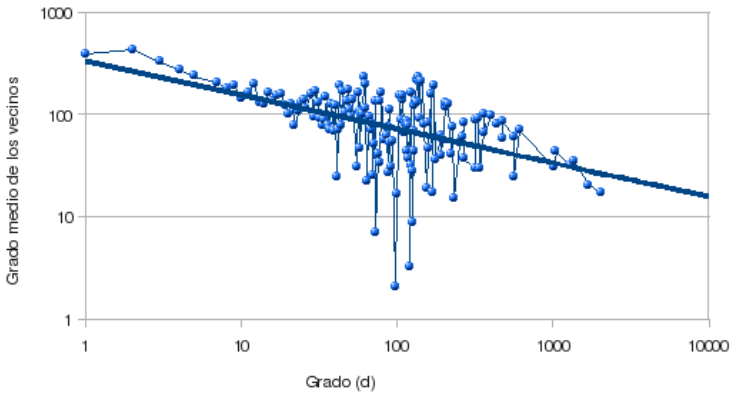
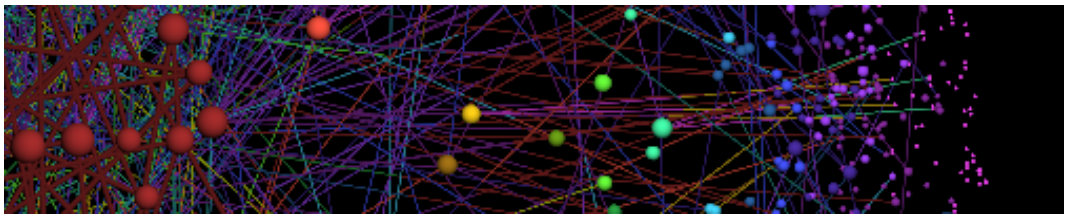


Figura 4.18: Distribución de grados de los vecinos para la red de ASes obtenida con Route Views (2008).

Esta última curva demuestra el comportamiento *disassortative*.

Por último, no se observa ningún nodo pintado de blanco, por lo tanto toda la red es núcleo-conexa.

CAIDA



Del proyecto CAIDA se tomaron datos de abril de 2005 y enero de 2008. Las imágenes muestran un único cliqué grande en el núcleo central, y que ha decrecido entre 2005 y 2008. También el grado y la cantidad de capas han decrecido, como consecuencia de la disminución en el número de sondas del proyecto: eran 25 en 2005 y sólo 10 en 2008.

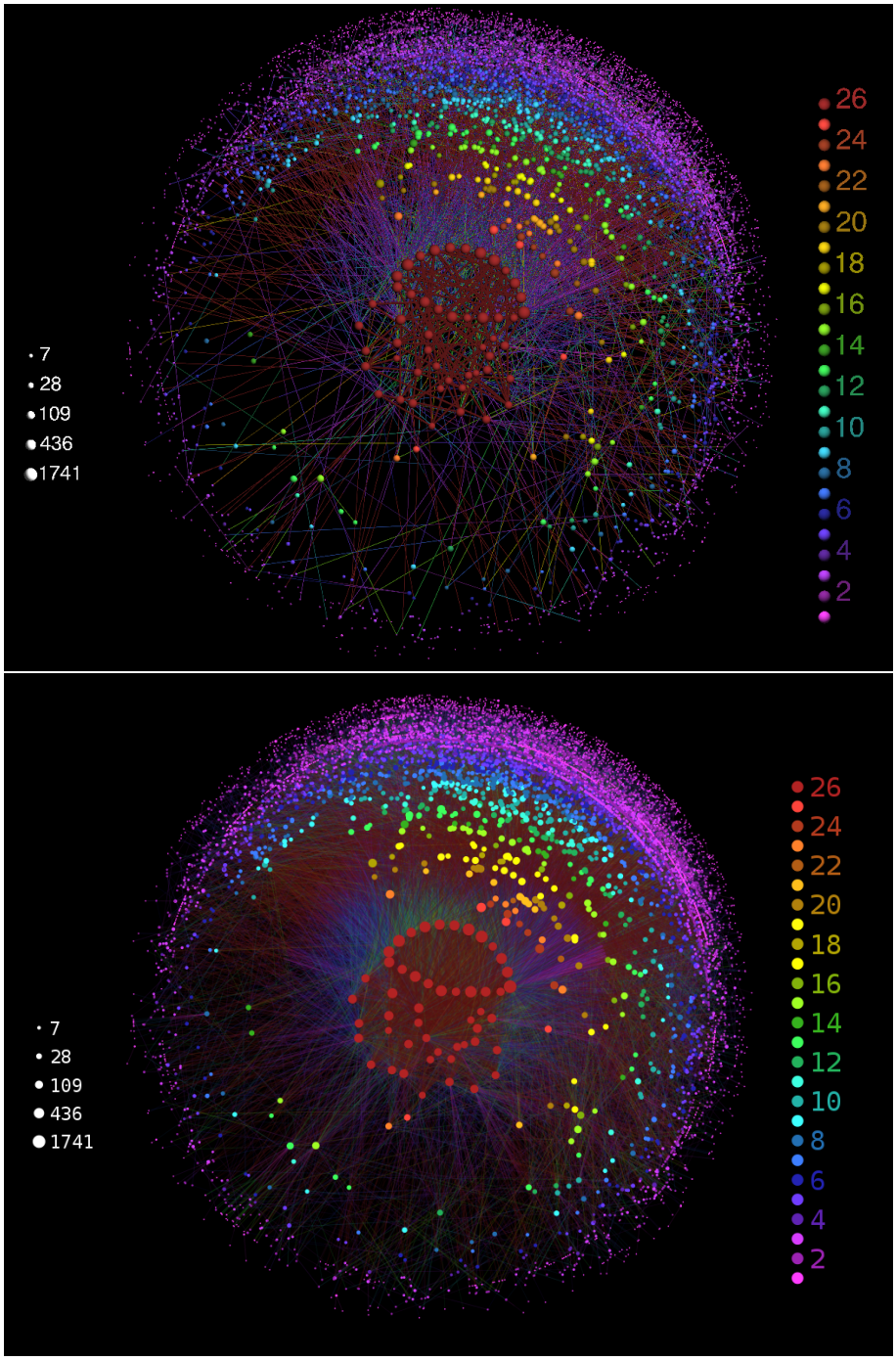


Figura 4.20: Imágenes de la red de Sistemas Autónomos en abril de 2005 obtenida del proyecto CAIDA. Renderización con PovRay (arriba) y con SVG (abajo).

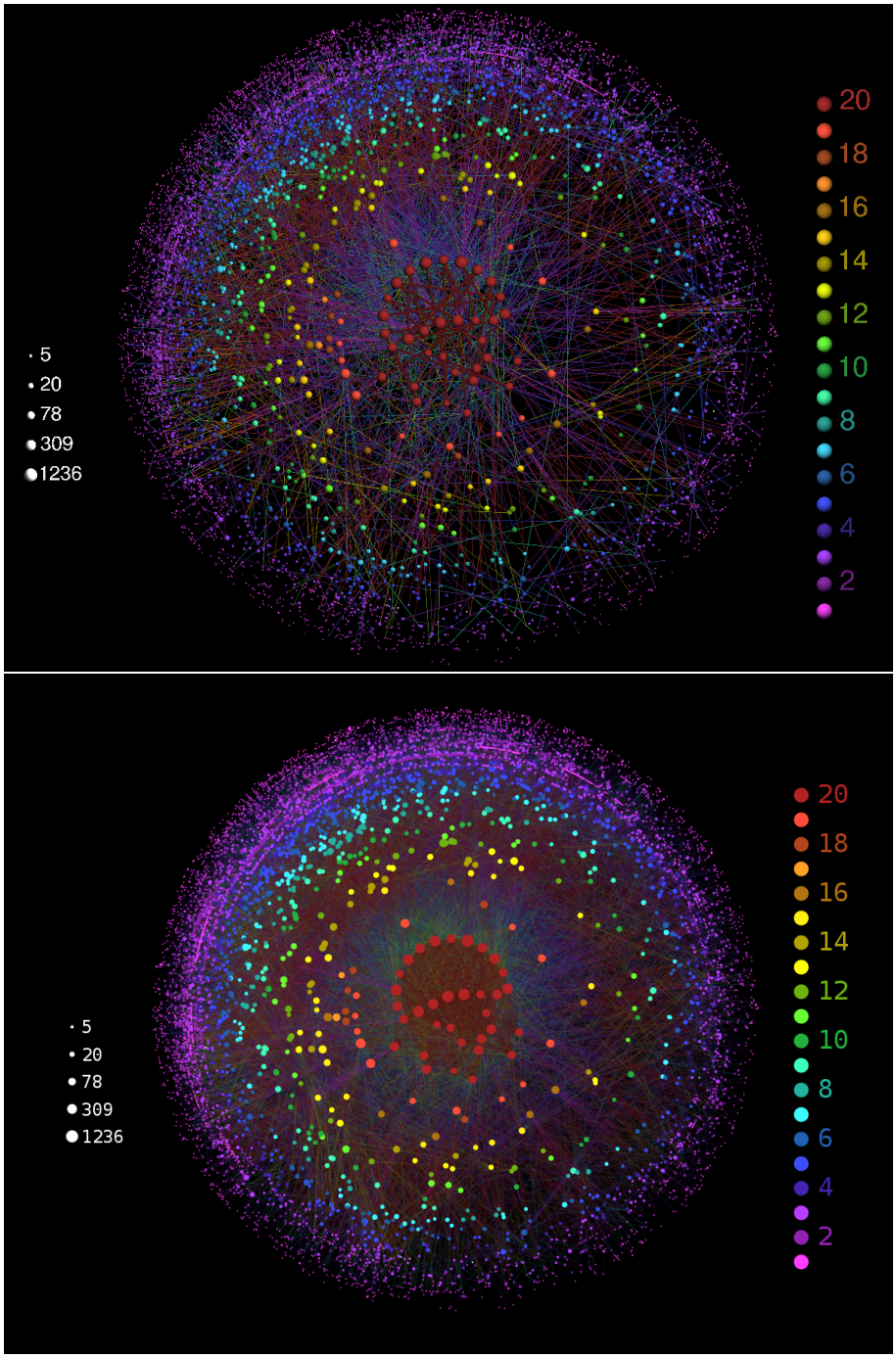


Figura 4.21: Imágenes de la red de Sistemas Autónomos en enero de 2008 obtenida del proyecto CAIDA. Renderización con PovRay (arriba) y con SVG (abajo).

Habilitando el etiquetado de los nodos y comparando los cliqués en 2005 y 2008, se descubre que el 50 % de los nodos que estaban en el cliqué anteriormente siguen estando ahora, y se han agregado otros nuevos.

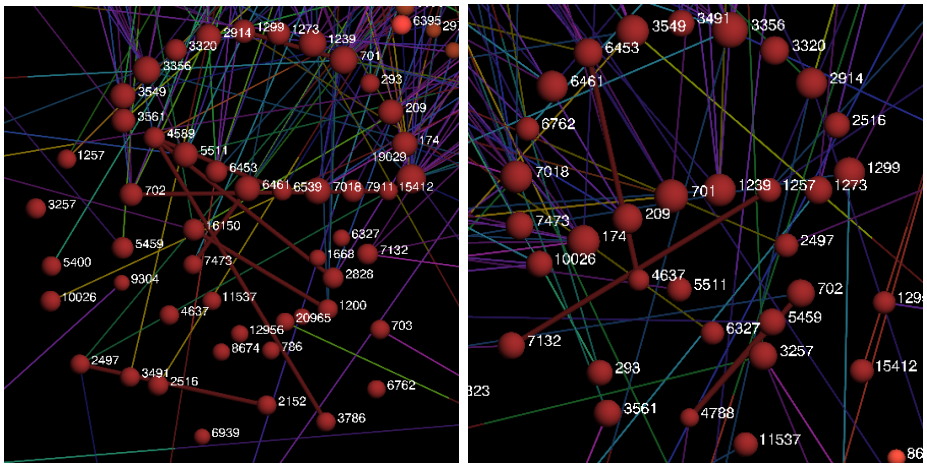


Figura 4.22: Cliqué principal del núcleo central en abril de 2005 (izquierda) y en enero de 2008 (derecha).

La red del 2008 tiene 7.897 sistemas autónomos y 20.851 conexiones, cantidad bastante inferior (menos de la mitad) que la de Route Views. Al igual que esta última, la red es núcleo-conexa. La distribución de los grados es muy similar a la obtenida en Route Views:

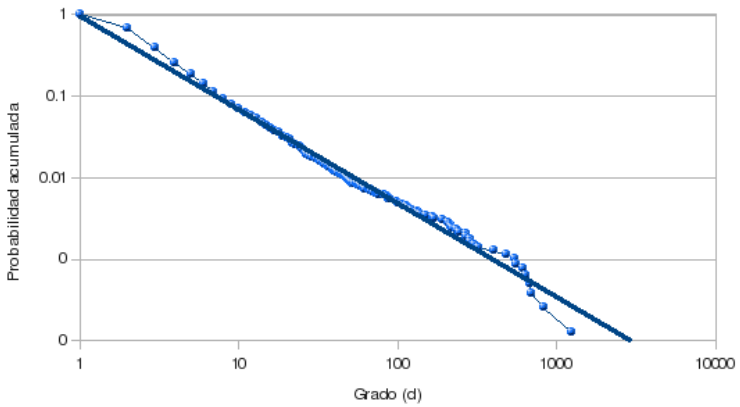


Figura 4.23: Distribución acumulativa de grados de la red de ASes obtenida con CAIDA (2008).

$$p(d_v) = 1{,}11 \cdot d_v^{-2{,}15} \quad , \tag{4.5}$$

y la distribución de los grados de los vecinos en función del grado es:

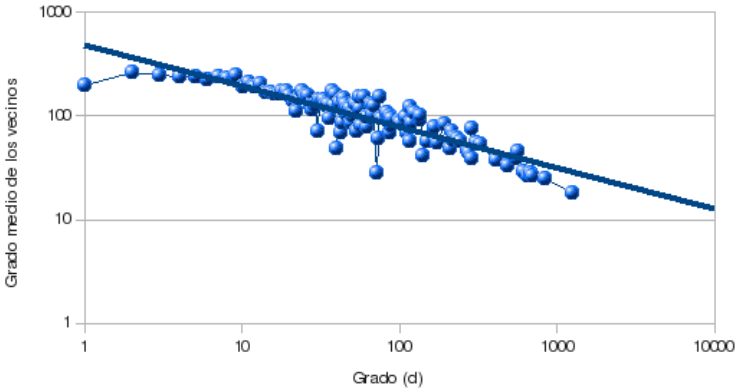
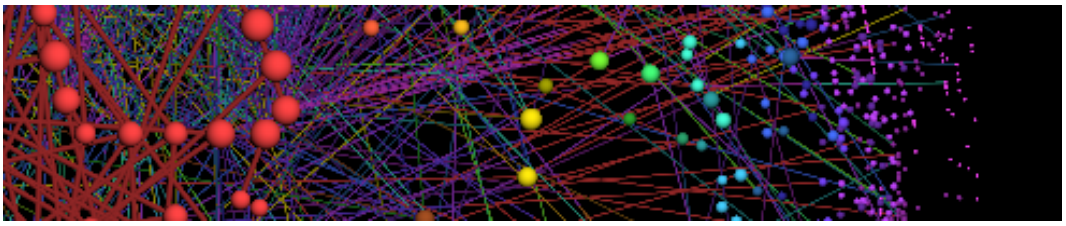


Figura 4.24: Distribución de grados de los vecinos para la red de ASes obtenida con CAIDA (2008).

DIMES



Se tomaron dos exploraciones de DIMES, de septiembre de 2007 y abril de 2005. La red del 2005 posee 20.456 nodos con 126.506 aristas, mientras que la del 2007 tiene 21.223 nodos y 68.201 aristas. A partir de la visualización (ver figuras 4.26 y 4.27) se observa que los nodos de mayor grado están en el centro y que están muy conectados con nodos de la periferia (comportamiento *disassortative*). El cliqué máximo tiene forma muy similar al obtenido con RouteViews. El análisis de núcleo-conexidad nuevamente informa que toda la red de sistemas autónomos es núcleo-conexa. La cantidad de capas también ha disminuído en DIMES entre 2005 y 2007, pasando de 39 a 27.

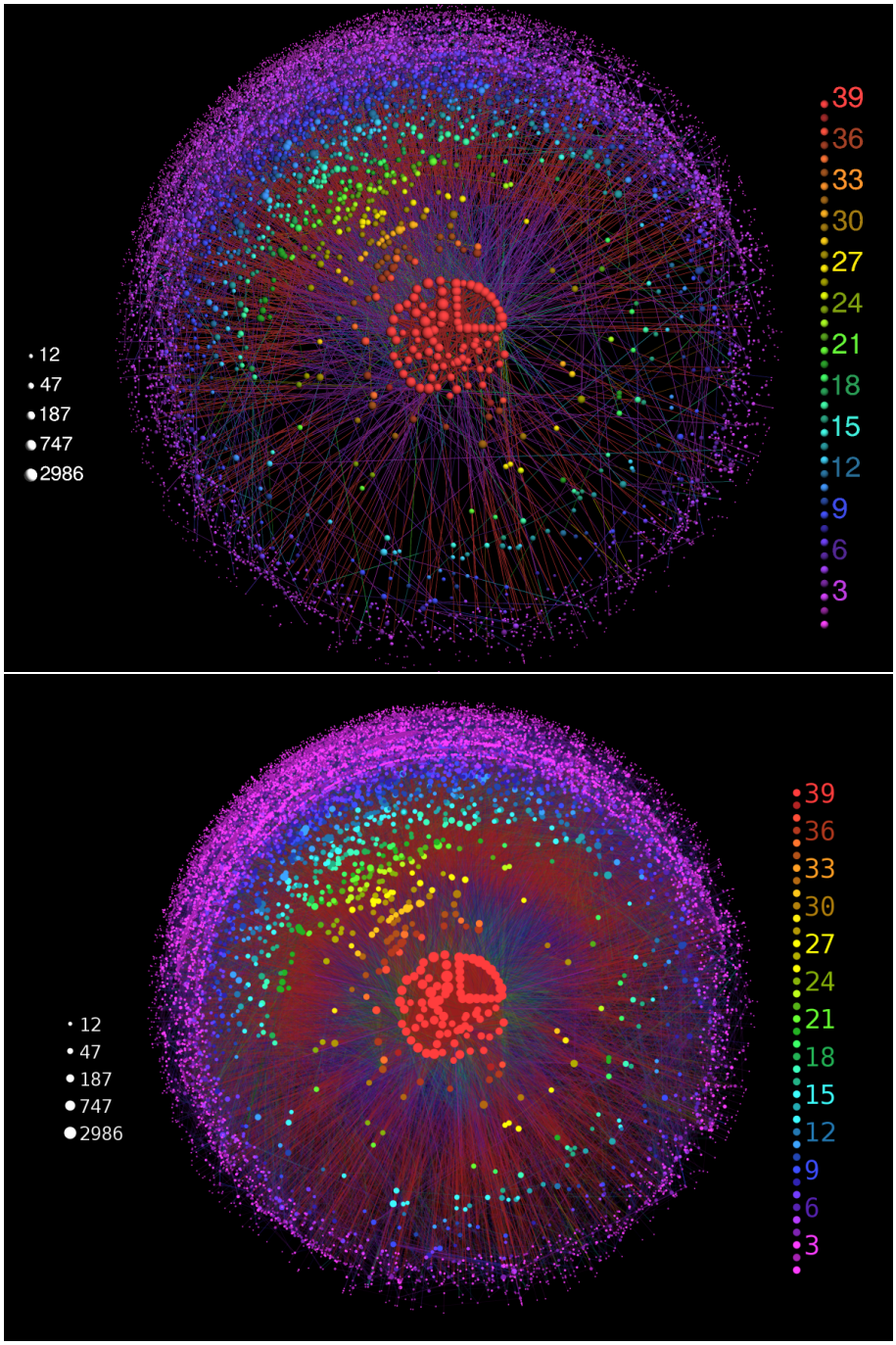


Figura 4.26: Imágen de la red de Sistemas Autónomos en abril de 2005 obtenida de DIMES, renderizada con PovRay (arriba) y con SVG (abajo).

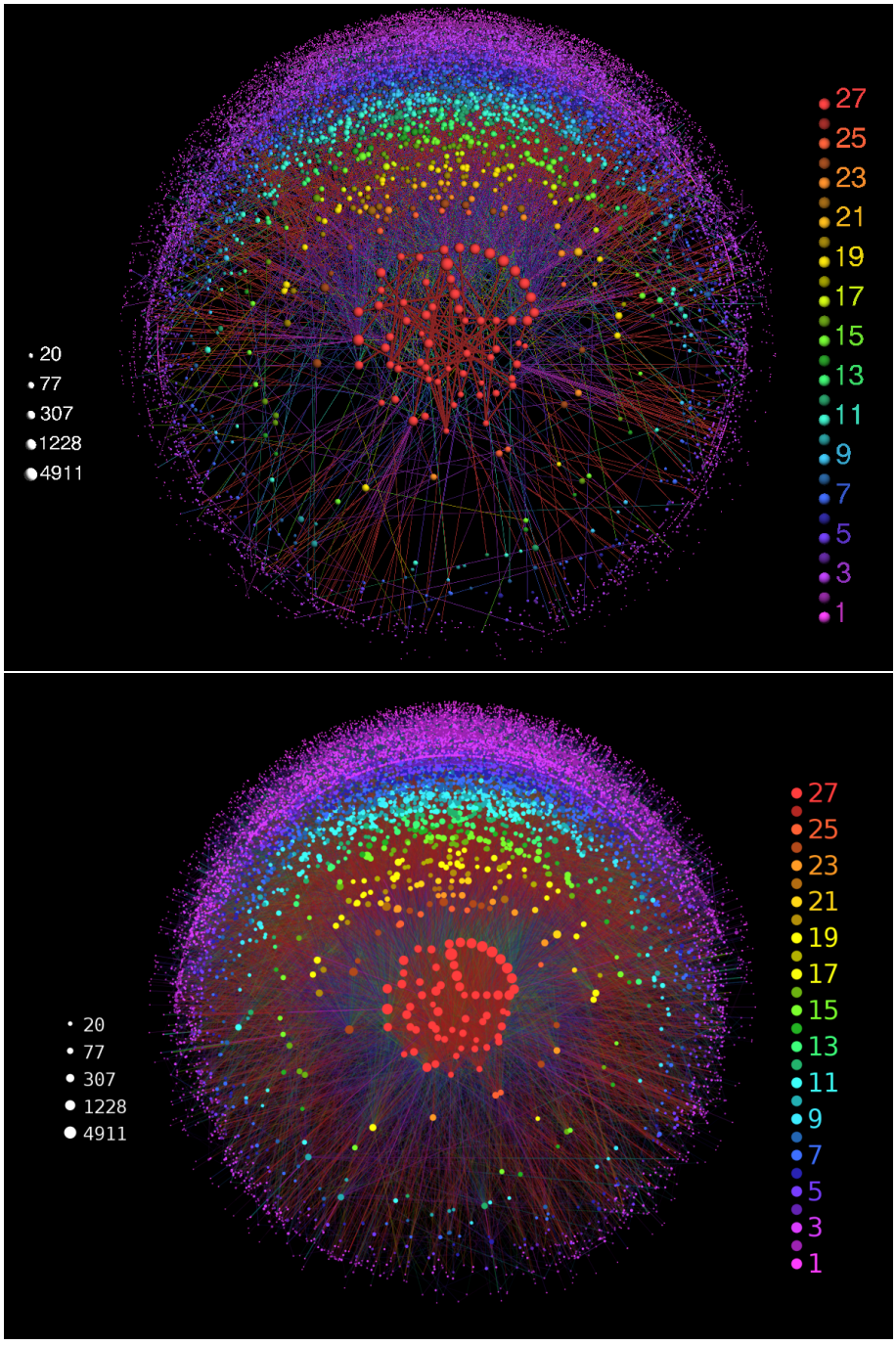


Figura 4.27: Imágen de la red de Sistemas Autónomos en septiembre de 2007 obtenida de DIMES, renderizada con PovRay (arriba) y con SVG (abajo).

También se obtuvo un mapeo entre los números de sistemas autónomos en los grafos y sus nombres, y realizando un preprocesamiento previo se generó una imagen con los nombres de los ASes que se puede observar en la figura 4.28.

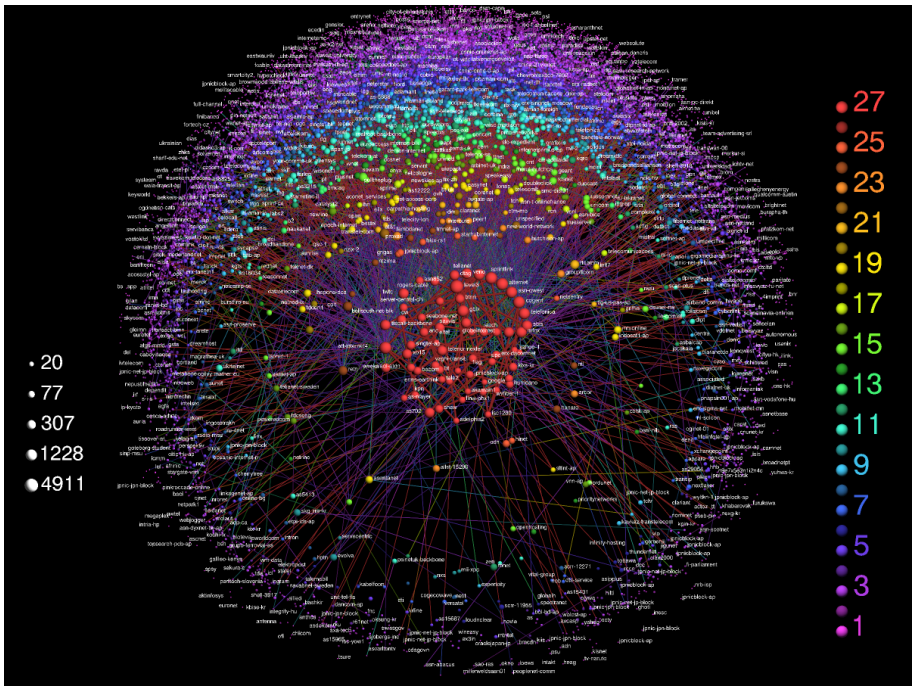


Figura 4.28: Imágen de la red de Sistemas Autónomos con sus respectivos nombres en septiembre de 2007, obtenida de DIMES.

Por último, es interesante comparar cómo están formados los núcleos de las redes obtenidas en cada una de las 3 fuentes. Identificando a los ASes por su nombre se observa (ver figura 4.29) que los nodos que conforman el núcleo central son en general los mismos, a pesar de los sesgos y particularidades de cada uno de los métodos de exploración.

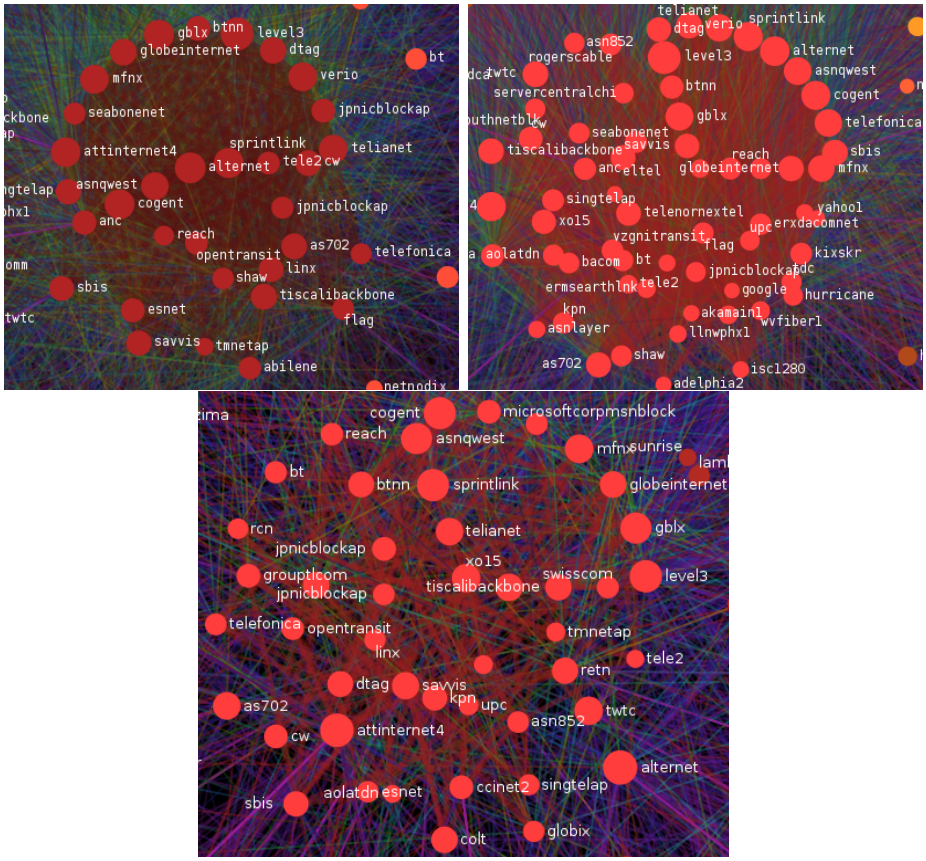


Figura 4.29: Comparación entre los núcleos de redes obtenidas con CAIDA (arriba, izquierda), DIMES (arriba, derecha) y RouteViews (abajo).

4.5.2. Visualización a nivel de Ruteadores

En este nivel, cada nodo de la red representa un ruteador, y una conexión es un enlace físico entre un par de ruteadores. Se tomó un mapa de CAIDA, con 192.244 ruteadores y 636.643 conexiones. La distribución de grados de esta red de ruteadores es:

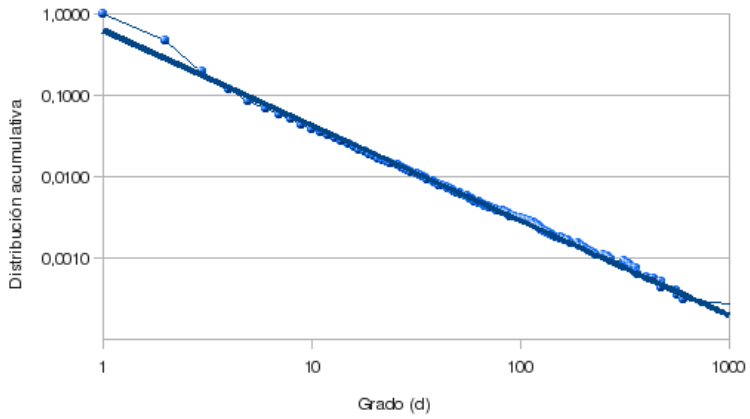


Figura 4.30: Distribución acumulativa de grados de la red de ruteadores obtenida con CAIDA.

$$p(d_v) = 33 \cdot d_v^{-2,91} \quad , \tag{4.6}$$

En tanto, la distribución de los grados de los vecinos es:

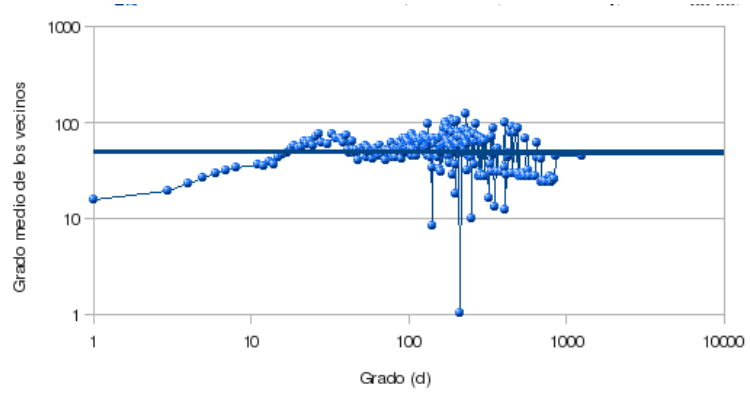


Figura 4.31: Distribución de grados de los vecinos para la red de ruteadores obtenida con CAIDA.

Se observa que los nodos de grado elevado se conectan a otros nodos de grado elevado también: ésto diferencia a los mapas a nivel de ruteadores de aquellos a nivel de Sistemas Autónomos.

La imagen obtenida (ver figura 4.32) muestra que el grado y el número de capa no están correlacionados: existen ruteadores de

grado elevado en la periferia, una característica propia de los mapas a este nivel.

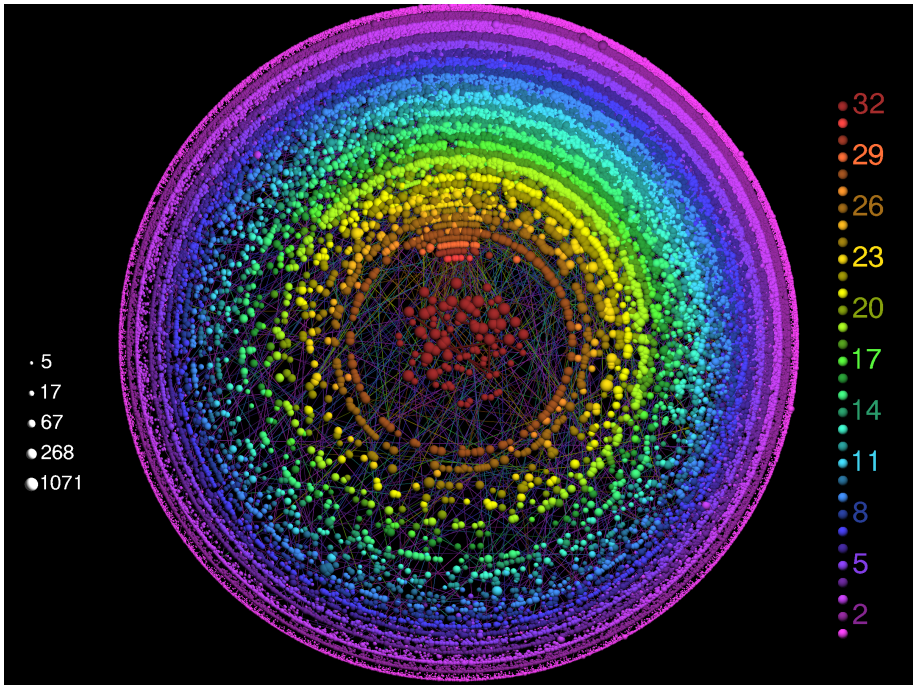


Figura 4.32: Imagen de la red a nivel de ruteadores obtenida con CAIDA.

Capítulo 5

Conclusiones

En la presente tesis se realizó un estudio de los algoritmos de visualización que existen en la actualidad, observando la debilidad de muchos de ellos para representar redes complejas debido al tiempo que demanda su ejecución y considerando el tamaño de estas redes.

Se presentó entonces el algoritmo LaNet-vi, basado en la descomposición en k-núcleos. LaNet-vi se caracteriza por su baja complejidad, lo que permite aplicarlo en la práctica en redes muy grandes.

5.1. Contribuciones

A lo largo del capítulo 3 se desarrollaron varias funcionalidades importantes, que fueron incorporadas en LaNet-vi manteniendo baja la complejidad total:

1. La descomposición en cliqués del núcleo central y organización del resto de la red en relación al mismo.
2. El análisis de la núcleo-conexidad.
3. El soporte de multigrafos y grafos pesados, abarcando así una mayor cantidad de redes.
4. El etiquetado de los nodos, que permite comparar imágenes y ubicar nodos particulares en ellas.

5. La incorporación de un nuevo *renderer*, *SVG*, que permite utilizar transparencias en las imágenes y así mostrar todos los ejes simultáneamente.

El resultado de este trabajo es la nueva versión de este software: LaNet-vi2.

En el capítulo 4 se aplicó LaNet-vi2 a la visualización de una amplia gama de redes, y se corroboró su habilidad para determinar, a partir de una imagen, varios atributos de la red como su conexidad, la correlación entre el grado y el número de capa, la mayor o menor presencia del fenómeno de mundo pequeño (*small world*), y el comportamiento de tipo *assortative* o *disassortative*.

Las contribuciones de esta tesis dieron origen a un artículo para el *New Journal on Physics*, el cual ya fue aceptado y será publicado en Diciembre de 2008, en un número especial sobre visualización [12]. Dicho artículo se encuentra en el apéndice B.

Todo el código fuente de LaNet-vi2 es abierto y ha sido publicado en Sourceforge[1] bajo licencia *AFL 3.0 (Academic Free License)*. Puede descargarse y modificarse libremente.

Por otra parte LaNet-vi2 ha sido incorporado a Network Workbench [28], en un trabajo conjunto con la Universidad de Indiana. Network Workbench es un framework para el estudio de redes que presenta varias cualidades:

1. Soporte para la mayoría de los formatos de descripción de redes empleados actualmente, como Pajek, GraphML y XGMML[2] (estos dos últimos basados en XML[3]).
2. Preprocesamiento de los datos.
3. Extracción de estadísticas e información de la red como distribución de grados, caminos mínimos, distancia promedio, diámetro, etc.
4. Múltiples algoritmos de visualización.
5. Modelado de redes de variados tipos.

[1] http://sourceforge.net/projects/lanet-vi/.

[2] eXtensible Graph Markup and Modeling Language.

[3] eXtensible Markup Language (XML) es una especificación general para la creación de lenguajes de marcado.

5.2. Trabajo futuro

Se vislumbra como trabajo futuro:

1. La incorporación de *grafos dirigidos (digrafos)* a la visualización. Un grafo dirigido es aquel en que sus aristas tienen una orientación o sentido: uno de sus extremos es el *origen* y el opuesto es el *destino*. Son ejemplos de estas redes las páginas Web (ya que sus enlaces son unidireccionales), las cadenas alimentarias, etc.
2. La incorporación de información estadística en la visualización, por ejemplo respecto a la distribución de grados, capas, etc.
3. La utilización de *renderers* más interactivos, por ejemplo empleando el estándar *OpenGL*[4]. Con *OpenGL* se podrían obtener representaciones realistas e interactivas, y sobre todo aprovechar la aceleración por hardware.

[4] www.opengl.org/.

Apéndice A

Descripción de clases

A.1. Module `Circular_average`

This class implements a weighted circular average calculator

```
class circular_average_class :
  object

    method average : float -> int -> float -> int -> float
  end
```

Definition of class circular_average

A.2. Module `Clique`

Gets a graph and returns a list of lists

```
class type graph =
  object

    method getMaxVertex : int
    method getVertex : int -> Host.host_class
    method printGraph : unit
```

```
  end
class clique_class :
  object
    val mutable cliques : Host.host_class list list
    method buildCliques : 'a. (#Clique.graph as 'a) -> int -> unit
```

Gets a graph and returns a list of lists

```
    method getCliques : Host.host_class list list
  end
val clique : clique_class
```

A.3. Module Component

```
val degree_squares : float Pervasives.ref
val central_core_ratio : float Pervasives.ref
type coordinate = {
  r : float ;
  alfa : float ;
}
val clique : Clique.clique_class
```

If this is the maximum shell index component, it's structured in cliques

```
val all_clusters : Host_list.host_list_class array Pervasives.ref
```

Vector with all clusters, by id_cluster

```
val pi : float
```

This class implements a component

```
class component_class : int ->
  object
    val mutable hosts : Host_list.host_list_class array
```

All hosts belonging to this component, with shell index equal to this component's shell index. That's why they don't belong to a child component.

```
val mutable hostListsCardinal : int
```

Amount of lists in previous variable hosts. It's needed because lists are appended in groups of 1000 for performance, so there must be some way to determine which of them are useless

```
method getHosts : Host_list.host_list_class array
val mutable clustersConnectivity : int array
```

For k-connectivity. Is computed as min(d+,d-)

```
method getClusterConnectivity : int -> int
method getIdClusterConnectivity : Host.IntMap.key -> int
method setClusterConnectivity : int -> int -> unit
val mutable parentComponent : Component.component_class list
method setParentComponent : Component.component_class -> unit
method hasParentComponent : int
method getParentComponent : Component.component_class
val mutable childrenComponents : Component.component_class list
```

All hosts belonging to this component children belong to this component as well.

```
method getChildrenComponents : Component.component_class list
val mutable componentCardinal : int
```

Amount of hosts inside this component

```
method getComponentCardinal : int
method getPreviousComponentsCardinal : int
```

Sumatory of previous components cardinals

```
method getBrotherComponentsCardinal : int
```

Sumatory of previous components cardinals

```
val shellIndex : int
```

Component's associated shell index

```
val mutable shellCardinal : int
```

Amount of hosts with this shell index, in this component

```
method getShellIndex : int
method getShellCardinal : int
val mutable rho : float
```

Component's rho

```
method getRho : float
method setRho : float -> unit
val mutable phi : float
```

Component's phi

```
method getPhi : float
method setPhi : float -> unit
val mutable xCoord : float
```

Component's center X coordinate

```
method getX : float
method setX : float -> unit
val mutable yCoord : float
```

Component's center Y coordinate

```
method getY : float
method setY : float -> unit
val mutable u : float
```

Component's unit length

```
method getU : float
method setU : float -> unit
method getClique : Clique.clique_class
val mutable actualComponent : int
val mutable h : Host.host_class
method addHost : Host.host_class -> unit
method setHosts : Host_list.host_list_class -> unit
method computeComponents : int -> int Pervasives.ref -> unit
```

Builds child components. A breadth search is done

```
    method printComponents : unit
    method printCoordinates : unit
    method placeInCircularSector :
      float -> float -> int -> int -> float -> Component.coordinate
    method findCoordinates : int -> int -> int -> unit
```

This method calculates coordinates for all hosts in this connected component and, recursively, in its children

```
    method bfs_with_frontier :
      Host.host_class Queue.t Pervasives.ref ->
      int -> (Host.host_class * float * int) list ->
    Host.host_class Pervasives.ref
```

Computes diameter, but considering that all vertices in the cluster's frontier are connected to a virtual vertex

```
    method bfs :
      Host.host_class Queue.t Pervasives.ref ->
      int -> Host.host_class Pervasives.ref
```

Computes diameter

```
    method computeCentralCoreDiameterRecursive :
      int Pervasives.ref -> Host.host_class Pervasives.ref
    method computeCentralCoreDiameter : int -> int
    method determineConnectivity : int -> unit
  end
```

Definition of class component

A.4. Module Files

```
val readLine : Unix.file_descr -> string Pervasives.ref -> int
```

A.5. Module Graph

This module defines a graph as a set of vertices and edges G=(V,E) and allows to add vertices and edges

```
class < addNeighbour : 'a -> float -> 'b;
applyPFunction : float list -> 'c;   getDegree : int; getNeighbours :
('a * 'd * 'e) list; getNumber : int;   getP : int;
getWeight : int -> float; isVertex : bool;

  printVertex : string -> unit; removeNeighbour :
'a * 'd * 'e -> unit;   setName : 'f -> 'g;
setNumber : int -> 'h; .. >

as 'a graph_class : 'a ->
  object
```

`val arraySize : int`

Implementation parameter. It defines a vertex array size. When this size is covered, a new vertex array of the same size is created and appended to the previous one.

`val mutable maxVertex : int`

`method getMaxVertex : int`

`val vertexPrototype : 'a`

A prototype used to construct vertices

`val mutable maxDegree : int`

Maximum degree of a vertex. Initially set to -1 (unknown)

`method getMaxDegree : int`

Returns maximum degree

`val mutable maxPValue : int`

Maximum pValue. (For weighted graphs)

`method getMaxPValue : int`

Returns maximum pValue

`val mutable weights : float list`

`val mutable pFunction : float list`

Table containing the p-function. 'p(n) = h' iff 'n is between the (h-1) and the h element in this list'

```
  val mutable amountOfEdges : int
  method getAmountOfEdges : int
  method setVertexName : int -> 'f -> 'g
  val mutable vertices : 'a array list
```

Set of vertices. This corresponds to V. E is given by vertices' neighbours. For efficiency, V was implemented as a list of preallocated arrays. Each array has a fixed size, when more vertices are needed, a new array is created and appended at the end of the list.

```
  method addVertexNumber : int -> unit
  method addVertex : 'a -> unit
```

Adds the graph a new vertex. Receives the vertex. Only the neighbours in the graph are left as vertex neighbours

```
  method getVertex : int -> 'a
```

Returns a vertex given its number

```
  method addEdge : int -> int -> float -> unit
```

Adds the graph a new edge. If any or both of the extreme vertices don't exist, they're created. Receives both vertices' numbers

```
  method addName : int -> 'f -> 'g
  method printGraph : unit
```

This is a test function. It prints the complete graph

```
  method buildPFunction : unit
  method endGraph : unit
end
```

Definition of class graph G=(V,E)

A.6. Module Graph_builder

This module reads a file with edges and adds them to an empty graph

```
class type graph =
  object

    method addEdge : int -> int -> float -> unit
    method addVertexNumber : int -> unit
    method setVertexName : int -> string -> unit
  end

class graph_builder_class :
  object

    method read_file : 'a. (#Graph_builder.graph as 'a) ->
    string -> unit
```

Reads an input file with format: file := `set of {line} EOF line := [ { vertex1 vertex 2 EOL} vertex1: int vertex2: int ]` and builds the graph. Receives the name of the file and the graph

```
  end
```

Definition of class graph_builder G=(V,E)

A.7. Module `Graph_builder_nwb`

This module reads a file with edges and adds them to an empty graph

```
class type graph =
  object

    method addEdge : int -> int -> float -> unit
    method addVertexNumber : int -> unit
    method setVertexName : int -> string -> unit
  end

class graph_builder_nwb_class :
  object
```

```
    method read_file : 'a. (#Graph_builder_nwb.graph as 'a) ->
    string -> unit
```

Reads an input file with format: file := set of {line} EOF line := [{ vertex1 vertex 2 EOL} vertex1: int vertex2: int] and builds the graph. Receives the name of the file and the graph

```
  end
```

Definition of class graph_builder G=(V,E)

A.8. Module Graph_node_names

This module reads a file with nodes and their names, and saves it in the graph

```
class type graph =
  object

    method addName : int -> string -> unit
  end

class graph_node_names_class :
  object

    method read_file : 'a. (#Graph_node_names.graph as 'a) ->
    string -> unit
```

Reads an input file with format: file := set of {line} EOF line := [{ vertex1 vertex 2 EOL} vertex1: int vertex2: int] and builds the graph. Receives the name of the file and the graph

```
  end
```

Definition of class graph_node_names

A.9. Module Graphics

```
type imagePosition = {
  xi : float ;
  yi : float ;
  dirxi : int ;
  diryi : int ;
}
class graphics_class : < addBlock : float -> float ->
float -> Types.color -> unit;

    addCylinder : float ->

                  float ->

                  float ->

                  float -> float -> float -> Types.color ->
float -> unit;

    addHeaders : float -> float -> float -> 'a;

    addHost : float -> float -> float -> Types.color -> unit;

    addText : string ->

              Types.color -> float -> float -> float ->
string -> unit;

    close : unit; .. > ->
  object

    val mutable namesMesh : int array array
    val mutable xMeshSize : int
    val mutable yMeshSize : int
    method private interpolate :
      Types.color -> float -> Types.color -> Types.color
    method private computeHostColor : Network.network_class ->
    int -> Types.color
```

```
    method private computeHostRatio : Network.network_class ->
    int -> float
    method generateNetworkFile : Network.network_class -> unit
  end
```

A.10. Module Host

This class extends the functionality of a vertex

This class extends the functionality of a vertex

```
module IntMap :
Map.Make(  sig
    type t = int
    val compare : 'a -> 'a -> int
  end )
class host_class : int ->
  object
    inherit Vertex.vertex_class [A.23]
    val mutable component : int
        This host's subcomponent inside its parent component
    method getComponent : int
    method setComponent : int -> unit
    val mutable prevHost : Host.host_class list
        Previous host in the component
    val mutable nextHost : Host.host_class list
        Next host in the component
    val mutable lastComponentComputed : int
        This atribute keeps the last shellIndex for which the
        component number of this host was computed. It's
        initially 0, and will equal the shellIndex at the end.
    val mutable clusterComputed : int
    val mutable id_cluster : int
```

This host's unique cluster number

```
val mutable cluster : int
```

This host's cluster inside its component

```
method isClusterComputed : int
method setCluster : int -> int -> unit
method getCluster : int
method getClusterId : int
method setPrev : Host.host_class -> unit
method setNext : Host.host_class -> unit
method clearPrev : unit
method clearNext : unit
method hasPrev : int
method hasNext : int
method getPrev : Host.host_class
method getNext : Host.host_class
val mutable shellIndex : int
method getShellIndex : int
method setShellIndex : int -> unit
method setLastComponentComputed : int -> unit
method getLastComponentComputed : int
val mutable rho : float
method setRho : float -> unit
method getRho : float
val mutable phi : float
method setPhi : float -> unit
method getPhi : float
val mutable xCoord : float
method setX : float -> unit
method getX : float
val mutable yCoord : float
method setY : float -> unit
method getY : float
val mutable color : Types.color
```

```
    method setColor : Types.color -> unit
    method getColor : Types.color
    val mutable usedToComputeDiameter : int
    method setUsedToComputeDiameter : int -> unit
    method getUsedToComputeDiameter : int
    val mutable diameter : int
    method getDiameter : int
    method setDiameter : int -> unit
    val mutable dplus : int Host.IntMap.t
    method getdplus : int Host.IntMap.t
```

List of (cliqueNumber, d+)

```
    method adddplus : Host.IntMap.key -> int Host.IntMap.t
    method setdplus : int Host.IntMap.t -> unit
    val mutable dminus : int
    method getdminus : int
    method setdminus : int -> unit
    val mutable m : int
    method getM : int
    method setM : int -> unit
    val mutable isKConnected : bool
    method getIsKConnected : bool
    method setIsKConnected : bool -> unit
  end
```

Definition of class host

A.11. Module Host_list

This class implements a list of hosts

```
class host_list_class :
  object

    val mutable cardinal : int
```

Amount of hosts in the list

```
method getCardinal : int
```
Returns the amount of hosts in the list

```
val mutable firstHost : Host.host_class list
val mutable lastHost : Host.host_class list
method hasFirstHost : int
```
Returns 1 if the list is not empty, 0 if it's empty

```
method getFirstHost : Host.host_class
method hasLastHost : int
method getLastHost : Host.host_class
method add : Host.host_class -> unit
```
Adds a host to the list

```
method remove : Host.host_class -> unit
```
Removes a host from the list

```
method concat : Host_list.host_list_class -> unit
method printList : unit
```
Prints the list from first to last, and from last to first

```
end
```

Definition of class host_list

A.12. Module Log

This class manages the program logs through stdout or log files

```
val null : string
class log_class :
  object
    val mutable isLogStdout : bool
```
Logs must be send to stdout

```
    val mutable isLogFile : bool
```

Logs must be send to files

```
val mutable vertexInsertionFile : Pervasives.out_channel
```
Names of logs files. They'll be assigned when isLogFile is set to true

```
val mutable graphFile : Pervasives.out_channel
val mutable coresAlgorithmFile : Pervasives.out_channel
val mutable degreesFile : Pervasives.out_channel
val mutable coresFile : Pervasives.out_channel
val mutable componentsFile : Pervasives.out_channel
val mutable componentsAlgorithmFile : Pervasives.out_channel
val formatstd : Format.formatter
```
Formatter to send to stdout

```
val mutable vertexInsertionFormat : Format.formatter
```
Formatters to send to log files

```
val mutable graphFormat : Format.formatter
val mutable coresAlgorithmFormat : Format.formatter
val mutable degreesFormat : Format.formatter
val mutable coresFormat : Format.formatter
val mutable componentsFormat : Format.formatter
val mutable componentsAlgorithmFormat : Format.formatter
method setLogStdout : unit
```
Sets log to stdout

```
method getLogStdout : bool
method setLogFile : string -> unit
```
Sets log to files

```
method getLogFile : bool
method print : string -> string -> unit
```
Prints information in logs. Section indicates one among the possible log sections. See match for examples

```
method closeFiles : unit
end
val log : log_class
```
The log object

A.13. Module `Main`

Main class

References: See "Large Scale Networks fingerprinting and visualization using the k-core decomposition, NIPS 2005". This paper will be referenced in this program as `NIPS2005`

```
val main : unit -> 'a
```

A.14. Module `Network`

This module extends a graph functionality to a network. Besides defining a graph as a set of vertices and edges G=(V,E) and offering the graph class services, it maintains lists of vertices with the same degree, thus allowing the computation of the vertices' shell index (k-core decomposition). It also uses an extended version of a vertex: a host, which also has a shell index and references to the previous and next host with the same shell index.

```
val pi : float
class network_class :
  object
    inherit Graph.graph_class [A.5]
    val mutable isShellIndexComputed : bool
        Indicates if shellIndex is already calculated
    val mutable maxShellIndex : int
    val mutable minShellIndex : int
    method getMaxShellIndex : int
    method setMaxShellIndex : int -> unit
    method getMinShellIndex : int
    method setMinShellIndex : int -> unit
    val mutable component : Component.component_class
    val mutable centralCoreDiameter : int
    method getComponent : Component.component_class
    val mutable lists : Host_list.host_list_class array
```

An array of lists, containing vertices initially grouped by their degree, and finally grouped by their shell index. Initially, 1 list are built.

`method buildInitialLists : unit`
Buils the list of lists, grouping vertices by their degree

`method printLists : unit`
Prints groups of vertices. If shellIndex is already computed, vertices are grouped by their degree. Otherway they're grouped by their shell index. Must not be called after computeComponents

`method generateCoresFile : string -> unit`
Generates a file with (node, core) Must be called after shellIndex is already computed, and before computeComponents

`method computeShellIndex : unit`
This method contains the algorithm to compute the shell index

`method computeComponents : unit`
This method contains the algorithm to compute the components. computeShellIndex must have been invoked previously.

`method findCoordinates : unit`
computeComponents must have been invoked before

`method computeCentralCoreDiameter : int`
`method getCentralCoreDiameter : int`
`method determineConnectivity : unit`
`method printComponents : unit`
computeComponents must have been invoked before

`method printCoordinates : unit`
findCoordinates must have been invoked before

`method getShellIndex : int -> int`
Receives the vertex number

`end`

Definition of class network

A.15. Module Normal

This class implements a normal distribution number generator

```
class normal_class : float -> float ->
  object

    val mutable mean : float
    val mutable dev : float
    method setMean : float -> unit
    method setDev : float -> unit
    method getValue : float
  end
```

Definition of class normal

A.16. Module Parameters

This class validates input parameters and offers help

```
class parameters_class :
  object

    val mutable bckGnd : string
    method getBckGnd : string
    val mutable color : string
    method getColor : string
    val mutable percentOfEdges : float
    method getPercentOfEdges : float
    val mutable minEdges : int
    method getMinEdges : int
    val mutable widthResolution : int
    method getWidthResolution : int
    val mutable heightResolution : int
    method getHeightResolution : int
```

```
val mutable epsilon : float
method getEpsilon : float
val mutable delta : float
method getDelta : float
val mutable gamma : float
method getGamma : float
val mutable hstart : float
method getHStart : float
val mutable hend : float
method getHEnd : float
val mutable vstart : float
method getVStart : float
val mutable vend : float
method getVEnd : float
val mutable u : float
method getU : float
val mutable format : string
```

Options: ”lanet.or ”nwb”

```
method getFormat : string
val mutable show : string
```

Options: or ”java”

```
method getShow : string
val mutable outputFile : string
val mutable outputRendFile : string
method getOutputFile : string
val mutable outputPngFile : string
method getOutputPngFile : string
val mutable inputFile : string
method getInputFile : string
val mutable inputNodeNamesFile : string
method getInputNodeNamesFile : string
val mutable coresFile : string
method getCoresFile : string
```

```
    val mutable multigraph : bool
    method getMultigraph : bool
    val mutable weighted : bool
    method getWeighted : bool
    val mutable granularity : int
```

For weighted graphs

```
    method getGranularity : int
    val mutable renderer : string
    method getRenderer : string
    val mutable noCliques : bool
    method getNoCliques : bool
    val mutable kConn : bool
    method getKConn : bool
    val mutable onlyGraphic : bool
    method getOnlyGraphic : bool
    val mutable opacity : float
    method getOpacity : float
    val mutable isLogFile : bool
    method printHeader : unit
    method private printHelp : unit
    method validateParameters : unit
  end
val parameters : parameters_class
```

The parameters object

A.17. Module Povray

This class generates the .pov file, drawing every host as a sphere and some edges as lines, and adding references

```
class povray_class : string ->
  object
```

```
    val out_file : Pervasives.out_channel
    val mutable r_saved : float
    val mutable u_saved : float
    method addHeaders : float -> float -> float -> unit
    method addHost : float -> float -> float -> Types.color -> unit
    method addBlock : float -> float -> float -> Types.color -> unit
    method addCylinder :
      float ->
      float -> float -> float -> float -> float ->
    Types.color -> float -> unit
    method addText :
      string -> Types.color -> float -> float -> float -> string -> unit
    method close : unit
  end
```

Definition of class povray

A.18. Module Povray_renderer

This class renders the .png image by calling povray

```
val null : string
```

This class renders the .png image by calling povray

```
class povray_renderer_class :
  object
```

```
    method render : string -> unit
```

Returns exit code (int)

```
  end
```

Definition of class povray_renderer

```
val povray_renderer : povray_renderer_class
```

A.19. Module Svg

Definition of class svg

```
class svg_class : string ->
  object
    val out_file : Pervasives.out_channel
    method addHeaders : float -> float -> float -> unit
    method addHost : float -> float -> float -> Types.color -> unit
    method addBlock : float -> float -> float -> Types.color -> unit
    method addCylinder :
      float ->
      float -> float -> float -> float ->
    float -> Types.color -> float -> unit
    method addText :
      string -> Types.color -> float ->
    float -> float -> string -> unit
    method close : unit
  end
```

Definition of class svg

A.20. Module Svg_renderer

```
val null : string
```

This class renders the .svg image by calling the svg renderer

```
class svg_renderer_class :
  object
    method render : string -> unit
```

Returns exit code (int)

```
  end
```

Definition of class svg_renderer

```
val svg_renderer : svg_renderer_class
```

A.21. Module Types

```
type color = {
  r : float ;
  g : float ;
  b : float ;
}
val magenta : color
```

Rainbow colors

```
val blue : color
val cyan : color
val green : color
val yellow : color
val red : color
val rainbow : (color * float) list
```

Colors list for color image

```
val white : color
```

B&W scale

```
val gray : color
val black : color
val blackwhite : (color * float) list
```

Colors list for B&W image

A.22. Module Uniform

This class implements a uniform distribution number generator

```
class uniform_class : float -> float ->
  object
```

```
    val mutable min : float
    val mutable max : float
    method setMin : float -> unit
    method setMax : float -> unit
    method getValue : float
  end
```

Definition of class uniform

```
val uniform2Pi : uniform_class
```

A.23. Module `Vertex`

This module defines a vertex as a node with a number and a set of neighbours, and allows to create vertices and append neighbours to them

```
class < getNumber : int; .. > vertex_class : int ->
  object
```

`val mutable number : int`

Vertex's number. A value of -1 means this object is empty (it does not contain a real vertex). This allows to preallocate an array of vertex_class and realize which of them are real vertices and which of them are empty.

`val mutable cc : float`

Vertex's clustering coeficient

`val mutable pValue : int`

Vertex's p-value. Computed from degree or edges weight, depending on the graph being weighted or not.

`method getPValue : int`

`val mutable amountNeighbours : int`

Vertex's amount of neighbours

`method getAmountNeighbours : int`

Gets the vertex's amount of neighbours (degree)

`method getDegree : int`

Gets the vertex's degree

`val mutable neighbours : ((< getNumber : int; .. > as 'a) float * int) list`

List of neighbour vertices references, weights and p-values

`method getWeight : int -> float`

`val mutable p : int`

`method getP : int`

Returns sum of p-values of all edges

`val mutable name : string`

`method isVertex : bool`

True if this object contains a vertex. False if it's empty

`method getNumber : int`

Gets the vertex number

`method setName : string -> unit`

`method getName : string`

`method setNumber : int -> unit`

Sets the vertex number

`method setClusteringCoeficient : float -> unit`

Sets the clustering coeficient

`method getClusteringCoeficient : float`

Returns the clustering coeficient

`method addNeighbour : 'a -> float -> unit`

Adds a new neighbour to this vertex

`method removeNeighbour : 'a * float * int -> unit`

Removes a neighbour to this vertex

`method getNeighbours : ('a * float * int) list`

Gets a reference to the list of neighbours

`method isNeighbourOf : 'a -> bool`

Returns true if vertex is v's neighbour

`method applyPFunction : float list -> unit`

`method printVertex : string -> unit`

This is a test function. It prints vertex number and its neighbours

`end`

Definition of class vertex

Apéndice B

Artículo NJP

M G Beiró, J I Alvarez-Hamelin, and J R Busch. A low complexity visualisation tool that helps to perform complex systems analysis. *New Journal of Physics, Focus on Visualization in Physics*, 2008. *En prensa*.

Ver `http://www.iop.org/EJ/journal/-page=forthart/1367-2630`

El artículo será publicado en Diciembre de 2008. A partir de ese momento, se incluirá también por completo en este apéndice de la presente versión electrónica.

Bibliografía

[1] Domain Counts & Internet Statistics, Junio 2008. `http://www.domaintools.com/internet-statistics/`.

[2] Internet World Stats, Junio 2008. `http://www.internetworldstats.com/stats.htm`.

[3] R Albert, J Hawoong, and A L Barabási. Diameter of the World Wide Web. *Nature*, 401:130, 1999.

[4] J I Alvarez-Hamelin and J R Busch. Edge connectivity in graphs: an expansion theorem. *[math.GM]*, arXiv:0803.3057v1, 2008.

[5] J I Alvarez-Hamelin, L Dall'Asta, A Barrat, and A Vespignani. k-core decomposition: a tool for the visualization of large scale networks. *CoRR*, cs.NI/0504107, 2005.

[6] J I Alvarez-Hamelin, L Dall'Asta, A Barrat, and A Vespignani. Large scale networks fingerprinting and visualization using the k-core decomposition. In Y Weiss, B Schölkopf, and J Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 41–50, Cambridge, MA, 2006. MIT Press.

[7] J I Alvarez-Hamelin, M Gaertler, R Görke, and D Wagner. Halfmoon - A new Paradigm for Complex Network Visualization. Technical Report TR-2005-29, Faculty of Informatics, Universität Karlsruhe (TH), 2005.

[8] A L Barabási and R Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.

[9] Marc Barthelemy, Alain Barrat, Romualdo Pastor-Satorras, and Alessandro Vespignani. Characterization and modeling of weighted networks. *PHYSICA A*, 346:34–43, 2005.

[10] V Batagelj and M Zaversnik. Generalized Cores. *CoRR*, arXiv.org/cs.DS/0202039, 2002.

[11] M Baur, U Brandes, M Gaertler, and D Wagner. Drawing the AS Graph in 2.5 Dimensions. In *12th International Symposium on Graph Drawing, Springer-Verlag editor*, pages 43–48, 2004.

[12] M G Beiró, J I Alvarez-Hamelin, and J R Busch. A low complexity visualisation tool that helps to perform complex systems analysis. *New Journal of Physics, Focus on Visualization in Physics*, 2008. *In press.*

[13] C Bron and J Kerbosch. Finding All Cliques of an Undirected Graph. *Communications of the ACM*, 16(9):575–577, 1973.

[14] S K Card, J D Mackinlay, and B Shneiderman. Readings in Information Visualization: Using Vision to Think, 1999.

[15] S A Cook. The complexity of theorem proving procedures. In *Third Annual ACM Symposium on Theory of Computing*, pages 121–158, 1971.

[16] P Eades. A heuristic for graph drawing. *Congressus Nutnerantiunt*, 42:149–160, 1984.

[17] A Fabrikant, E Koutsoupias, and C H Papadimitriou. Heuristically Optimized Trade-Offs: A New Paradigm for Power Laws in the Internet. *LNCS*, 2380:110–122, Jun 2002.

[18] M Faloutsos, P Faloutsos, and C Faloutsos. On power-law relationships of the internet topology. In *SIGCOMM*, pages 251–262, 1999.

[19] T M J Fruchterman and E M Reingold. Graph Drawing by Force-directed Placement. *Software-Practice and Experience*, 21:1129–1164, 1991.

[20] C Gkantsidis, M Mihail, and E W Zegura. Spectral analysis of internet topologies. In *INFOCOM*, 2003.

[21] R Görke, M Gaertler, and D Wagner. LunarVis - Analytic Visualizations of Large Graphs. In *Graph Drawing 2007*, pages 352–364, 2007.

[22] N. Har'El. Finding the largest eigenvalues of a real symmetric matrix, and corresponding eigenvectors, 1992. Research Report, Department of Mathematics, Israel Institute of Technology.

[23] J A Hartigan and M A Wong. A K-means clustering algorithm. *Applied Statistics*, 28:100–108, 1979.

[24] T Kamada and S Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31:7–15, 1989.

[25] M E J Newman. Assortative Mixing in Networks. *Physical Review Letters*, 89(20):208701, 2002.

[26] M E J Newman. The structure and function of complex networks. *SIAM Review*, 45(2):167–256, 2003.

[27] M E J Newman and M Girvan. Finding and evaluating community structure in networks. *Phys. Rev. E*, 69:026113, 2004.

[28] NWB Team. (2006-2008). Network Workbench Tool. Indiana University and Northeastern University. `http://nwb.slis.indiana.edu/` .

[29] J Plesník. Critical graphs of a given diameter. *Acta Fac. Rerum Natur. Univ. Comenian. Math.*, 30:71–93, 1975.

[30] D J Price. A general theory of bibliometric and other cumulative advantage processes. *J. Amer. Soc. Inform. Sci.*, 27:292–306, 1976.

[31] S B Seidman. Network structure and minimum degree. *Social Networks*, 5:269–287, 1983.

[32] Ben Shneiderman. Why not make interfaces better than 3d reality? *IEEE Computer Graphics and Applications*, 23(6):12–15, November/December 2003.

[33] J D Watson, T A Baker, S P Bell, A Gann, M Levine, and R Losick. *Molecular Biology of the Gene*. CSHL Press, 2004.

[34] D J Watts and S H Strogatz. Collective Dynamics of Small-World Networks. *Nature*, 393:440–442, 1998.