Búsqueda de solapamiento en *clusters*, usando técnicas de computación evolutiva

María Andrea Cruz Blandón

Universidad del Valle
Facultad de ingeniería
Escuela de ingeniería de sistemas y computación
Santiago de Cali
2013

Búsqueda de solapamiento en *clusters*, usando técnicas de computación evolutiva

María Andrea Cruz Blandón
Código 0831816
andrea.cruz@correounivalle.edu.co

Documento presentado como requisito parcial para la obtención de grado de Ingeniero de Sistemas

Director
Ing. Angel García Baños, Ph.D.
angel.garcia@correounivalle.edu.co

Universidad del Valle
Facultad de ingeniería
Escuela de ingeniería de sistemas y computación
Santiago de Cali
2013

Trabajo de grado presentado por
María Andrea Cruz Blandón.
Como requisito parcial para la obtención del título de Ingeniero de Sistemas

---

Angel García Baños, Ph.D.
Director

No Asignado
Jurado

Agradecimientos

A Dios por darme la vida y la salud.
A mi familia por todo el apoyo y la confianza brindada durante este proceso
A la universidad por el espacio y las oportunidades brindadas.
A mis profesores por la guía en cada una de las actividades académicas.
A mis compañeros de estudio por compartir y disfrutar de este proceso.

Tabla de Contenido

Lista de Figuras

Glosario

Red dispersa [Newman, 2010] La densidad de una red ρ es:

$$\rho = \frac{c}{n} \tag{1}$$

donde c es la media de los grados de los nodos de la red. Cuando $\rho \to 0$ cuando $n \to \infty$ se dice que la red es dispersa y la cantidad de elementos que no son cero en la matriz de adyacencia también tiende a cero.

Redes libres de escala [Newman, 2010] Son aquellas cuya distribución de grados de los nodos sigue una ley de potencia. En éstas la mayoría de grados de los nodos son bajos. La Internet, las redes neuronales y las redes sociales son ejemplos de redes libres de escala.

Ley de potencia [Newman, 2010] Es una relación matemática que se expresa como:

$$y = ax^k \tag{2}$$

donde a es una constante. Visto desde la estadística, si x corresponde a una variable aleatoria y y a su frecuencia, si k cumple con $0 < k < 1$ entonces conforme x aumenta el valor, el valor de y va disminuyendo.

Dendrograma Es una representación gráfica de datos en forma de árbol, donde se va descomponiendo los datos hasta llegar al nivel de especificación que se desee.

Mating pool Corresponde a un arreglo con un tamaño menor al de la población, donde se realizan las operaciones de cruce y mutación. Generalmente se define su tamaño como un porcentaje del tamaño de la población.

Gran componente Las redes pueden estar compuestas por más de un componente o subgrafo conexo. El gran componente es aquel subgrafo conexo que contiene el mayor porcentaje de nodos en la red. Por ser conexo se cumple que:

$$\forall (i,j) \in N_c \; \exists P(i,j) \subseteq E_c \tag{3}$$

siendo N_c el conjunto de nodos del componente c, E_c el conjunto de aristas del componente c y $P(i,j)$ un camino entre los nodos i y j.

Community structure [Newman, 2006] Es la propiedad que exhiben algunas redes donde sus nodos pueden ser agrupados en *clusters* o grupos o comunidades y donde existen una densa conexión entre los nodos dentro de estas comunidades y una dispersa conexión entre comunidades.

Resumen

El estudio de redes ha generado gran movimiento en la comunidad científica en los últimos años. Las investigaciones han permitido avances en la comprensión de las redes. Por ejemplo se suele encontrar sinergia en las redes, ya que un nodo puede ser una entidad simple comparado con el comportamiento que emerge de toda la red, por ejemplo, la distribución de grados de los nodos que en redes libre de escala sigue una ley de potencias. Además la complejidad va aumentando conforme las relaciones entre nodos se hace más densa [Gog et al., 2007].

En el contexto de este trabajo de grado se amplía el concepto de búsqueda de *clusters* clásica [Han and Kamber, 2006], permitiendo el solapamiento de los *clusters*. La importancia de buscar *clusters* que se solapen en redes radica en que muchos problemas que se han resuelto con algoritmos tradicionales no encuentran los solapamientos, aunque en la vida real sí se dan. Ejemplo de ello son: en redes sociales encontrar perfiles de personas que se adapten a diferentes situaciones; en biología encontrar las proteínas que afectan varios procesos de metabolismo; en la red neuronal, donde una neurona pueden atender procesos de otros módulos del cerebro, etc.

En este trabajo de grado se diseña, modela y desarrolla un algoritmo que realiza la búsqueda de *clusters* con solapamiento en redes, haciendo uso de técnicas de computación evolutiva. Este algoritmo de *clustering* es de análisis estructural de la red y busca maximizar la modularidad. Puede ser aplicado a redes de aproximadamente 1000 nodos.

En este documento se describen el modelo y todas las especificaciones del algoritmo, como lo son: el tipo de redes con las que se puede usar el algoritmo (esto incluye todas las características que deben tener las redes); las métricas empleadas tanto para hallar los *clusters*, como para comparar entre soluciones encontradas y decidir cuál es la mejor; y por supuesto la técnica de computación evolutiva empleada.

Finalmente, con el algoritmo implementado, se presentan las pruebas realizadas. En éstas se compara el rendimiento respecto a la calidad de la solución del algoritmo, es decir, el valor de la modularidad en toda la red con la distribución de *clusters* y solapamientos encontrada, comparado con otros algoritmos encontrados en la bibliografía [Blondel et al., 2008], [Clauset et al., 2004], [Duch and Arenas, 2005], [Girvan and Newman, 2002] y [Newman, 2006].

Introducción

La técnica de *clustering* en *machine learning* corresponde a aprendizaje no-supervisado, es decir, que de antemano no se conoce a qué clase o *cluster* pertenece un dato que ingresa al algoritmo. Para determinar entonces qué dato pertenece a un *cluster*, se evalúa la similitud del dato con los datos que ya pertenecen al *cluster*. Esta similitud corresponde a funciones o métricas, por ejemplo, la distancia euclidiana ampliamente usada con datos numéricos. De acuerdo al resultado de la función se determina la pertenencia del dato al *cluster*. El *clustering* se puede aplicar además en datos discretos como lo son el dominio de las redes, y ahí se puede usar para encontrar comunidades, grupos de interacciones en las redes, entre otros. Éstas estructuras encontradas pueden representar comportamientos en el problema particular que representen las redes, entre varias otras alternativas de interpretación, todo depende de la semántica que se le dé a las redes. Por lo anterior, es importante describir a las redes de acuerdo a su fundamentación teórica matemática, la teoría de grafos.

Determinar si hay o no solapamiento, que un nodo pertenezca a más de un *cluster*, depende de cuánto éste mejore el agrupamiento, puesto que se busca maximizar la modularidad de la red total con todos los *clusters* definidos. En el capítulo 1 se encuentra el planteamiento del problema, los objetivos propuestos para resolverlo, la justificación de este trabajo de grado y se enumeran los antecedentes. En el capítulo 2 se describen los conceptos teóricos en los que está sustentado este trabajo de grado.

Para definir la solución planteada al problema, se específica el alcance de ésta en el capítulo 3. En el capítulo 4 se describe el modelo desarrollado, tanto las consideraciones matemáticas, como prácticas y finalmente en el capítulo 5 se presenta la implementación desarrollada.

En el capítulo 6 se listan todas las características de las pruebas, y se realiza el análisis respecto al desempeño de la solución implementada.

Finalmente se presentan las conclusiones y trabajos futuros de este trabajo de grado, en el capítulo 7.

Capítulo 1

Contexto

1.1. Planteamiento del problema

La búsqueda clásica de *cluster* evita por definición los solapamientos entre los *clusters* [Han and Kamber, 2006]. Sin embargo, en la vida real no siempre la intersección entre los *clusters* es vacía, por ejemplo, una red de colaboración de científicos donde cada uno puede investigar en más de un área o campo. Permitir los solapamientos entre *clusters* pudiera mejorar la distribución de *clusters* en una red y obtener resultados más parecidos a la realidad. Por otro lado, la forma de un *cluster* tradicional, en general es ovalada. Más aún, lograr obtener diferentes formas geométricas podría mejorar la composición de los *clusters* y por consecuencia optimizar la distribución en éstos en la red.

En este documento se desarrolla un algoritmo que permite el solapamiento entre *clusters*. Éste hace uso de las técnicas de computación evolutiva, siendo un algoritmo evolutivo, como método de exploración en la búsqueda de solapamientos en *clusters*. Con el desarrollo de este algoritmo se busca resolver las preguntas planteadas en el documento de anteproyecto de trabajo de grado, que son:

- ¿Es posible diseñar un algoritmo que encuentre solapamiento en clusters con distintas formas geométricas usando técnicas de computación evolutiva?
- ¿Esta propuesta representa una mejora respecto a eficiencia y tiempos en la búsqueda de *clusters* con solapamientos?

1.2. Objetivos

1.2.1. Objetivo general

Diseñar e implementar un algoritmo con técnicas de computación evolutiva, que realice la búsqueda de *clusters* con solapamiento en redes.

1.2.2. Objetivos específicos

1. Identificar y modelar el tipo de redes a analizar en el algoritmo, considerando la teoría matemática y los trabajos realizados.

2. Analizar y especificar las métricas a usar en el algoritmo que se plantee.

3. Analizar, establecer e implementar la técnica o técnicas de computación evolutiva, con las cuales se desarrollará el algoritmo.

4. Evaluar el modelo con un conjunto de redes del tipo establecido y realizar comparación respecto a otros trabajos ya realizados.

Los objetivos No. 1 y 2 se abordan en los capítulos 2 y 4, el objetivo No. 3 se aborda en los capítulos 2, 4 y 5. Finalmente el objetivo No. 4 se aborda en el capítulo 6.

1.3. Justificación

1.3.1. Justificación práctica

Identificar comunidades con solapamiento puede significar la mejora de procesos en los cuales interactúan las redes a analizar.

Este conocimiento, por ejemplo en el mundo de los negocios, significaría ahorrar significativamente en la inversión en campañas por segmentos de clientes, pues existe un conocimiento de clientes a los que se les puede ofrecer más de una opción, aumentando las posibilidades de que un cliente adquiera sus productos o servicios.

Además en una red social se podrían identificar los individuos que pertenecen a más de un grupo. Por ejemplo aquellos investigadores que participan en más de un laboratorio.

En lo biológico por su parte, hallar solapamientos en redes como por ejemplo, la red de interacción de proteínas, ayuda a identificar las proteínas que afectan a más de un proceso. Además se podría realizar pruebas del impacto que tendría un fallo en dichas proteínas.

1.3.2. Justificación académica

Si bien este problema ha sido explorado con distintas técnicas [Takaki et al., 2007], desarrollar una aproximación usando técnicas de computación evolutiva constituye un aporte considerando que este trabajo realiza una exploración de *clustering* en redes enfocándose en encontrar solapamientos, introduciéndolos con el uso de un algoritmo genético.

1.4. Antecedentes

El problema de la búsqueda de *clusters* ha sido ampliamente estudiado y es por ello que existen varias aproximaciones para resolverlo. Dichas aproximaciones usan diferentes tipos de técnicas incluidas las de computación evolutiva. Pero además, dadas las aplicaciones que tiene encontrar *clusters* con solapamiento, existen propuestas de solución para este problema. A continuación se describen algunos antecedentes encontrados en dos categorías, aquellos que permiten solapamiento y aquellos que no.

1.4.1. Algoritmos de *clustering* que no permiten solapamiento

Genetic Algorithms Applied to Multi-Class Clustering for Gene Expression Data

En [Pan et al., 2003] se describe un algoritmo genético híbrido que usa *simulated annealing*, el cual fue probado con datos de expresión de los genes, donde se busca maximizar la exploración de una distribución óptima o cercana a ésta en los *clusters* encontrados, y para ello se usa la medida interna del *cluster*, la cohesión, y el aislamiento entre *clusters*.

El número de *clusters* está dado, es decir, de antemano se conoce la cantidad de éstos que tendrá la solución, por lo que fue comparado con *K-Means* [Han and Kamber, 2006] y otros algoritmos que usan la misma metodología. El algoritmo da como resultado una solución óptima o cercana a ésta y gira en torno a la elección de los medoides, por lo que la codificación del cromosoma es un conjunto de **K** medoides, siendo **K** el número de *clusters*.

En la propuesta se señala que el cálculo de error cuadrático no siempre es una buena medida para evaluar los *clusters*, sobretodo si existe diferencias de tamaños entre ellos. Además no se permite el solapamiento entre *clusters*.

Document Clustering Using Differential Evolution

En [Ajith Abraham and Konar, 2006] se presenta un algoritmo modificado de evolución diferencial, que también es usado para realizar *clustering*. Se probó en grandes conjuntos de datos de documentos.

La modificación sobre la evolución diferencial clásica fue hecha sobre el operador de mutación. La evolución diferencial clásica es una adaptación de los algoritmos genéticos, donde se hace una modificación en los operadores de cruce y mutación que da lugar a un operador de vector diferencial. La evolución diferencial está basada en dicho operador. Es planteado como un problema de optimización, con el fin de mejorar los tiempos respecto a un algoritmo clásico como lo es *K-Means*. Finalmente se realizó un híbrido entre *K-Means* y evolución diferencial.

En las pruebas, dicha propuesta en comparación con el algoritmo *K-Means* clásico presenta mejor comportamiento. No se permiten solapamientos.

Evolutionary Multi-Objective Clustering for Overlapping Clusters Detection

En [Ripon and Siddique, 2009] se argumenta que los algoritmos evolutivos tienen dificultades para detectar solapamiento entre *clusters*, es decir, no minimizan los solapamientos. Es por ello que se presenta una aproximación que minimiza los solapamientos. Se describe

a EMCOC (*Evolutionary Multi-Objective Clustering for Detection Overlapping Clusters*) y se hace referencia a la importancia del *clustering* multi-objetivo. EMCOC tiene una codificación nueva de cromosoma, respecto a los algoritmos comparados, y un nuevo operador para realizar la asignación de *clusters*. Según los resultados expuestos, el algoritmo presenta un buen rendimiento respecto a otros algoritmos de *clustering* multi-objetivo evolutivos como RJGGA. Algo interesante es que el algoritmo determina la cantidad de *clusters* que optimizan el *clustering* automáticamente.

Community Detection in Complex Networks Using Collaborative Evolutionary Algorithms

En [Gog et al., 2007] se usa una técnica evolutiva para la detección de comunidades en redes. El algoritmo descrito es basado en la colaboración (compartir información entre nodos de la red).

Muchos de los algoritmos de *clustering* o relativos a este problema son NP-Hard, también existen aquellos que son NP-Completo y coinciden con el problema de Cliques [Fortunato, 2010], por lo que se ha discutido respecto a la aproximación del problema de detección de comunidades como un problema NP-Completo, sin embargo esto no es concluyente. Por lo anterior, presentan una propuesta con técnicas de programación evolutiva para resolver el problema como una aproximación al óptimo.

La codificación del cromosoma es un arreglo que contiene el total de nodos y relaciona a qué *cluster* pertenecen. Además, contiene la información sobre la mejor solución global que se haya encontrado en el algoritmo hasta el momento, y la mejor solución que se haya encontrado en su árbol genealógico. Si no tiene ancestros él mismo constituye la mejor solución encontrada. Con esta codificación se usa como función de aptitud la modularidad. Se concluye que la colaboración actúa como un acelerador, pues se conoce el mejor cromosoma relativo.

Finding community structure in very large networks

Este algoritmo de *clustering*, se propuso en [Clauset et al., 2004]. Es un algoritmo de *clustering* basado en el análisis estructural de la red, es decir, sólo se tiene en cuenta la información referente a cómo los nodos están relacionados con otros nodos (aristas); es un algoritmo jerárquico aglomerativo y voraz, que busca maximizar la modularidad Q.

En un principio cada nodo es un *cluster*, luego se buscan los valores de cambio en Q, resultado de la unión de *clusters* que incrementen el valor de la modularidad. El algoritmo finaliza cuando ya no hay uniones que la incrementen.

Para encontrar los valores de los cambios en Q, se plantea el uso de la matriz ΔQ, donde cada uno de los ΔQ_{ij} contiene el valor que afecta a la modularidad al unir los *clusters* i y j. Al inicio del algoritmo los valores de ΔQ_{ij} son obtenidos haciendo uso de la ecuación de la modularidad:

$Q = \sum_i (e_{ii} - a_i^2)$

$$\Delta Q_{ij} = \begin{cases} \frac{1}{2m} - \frac{k_i k_j}{(2m)^2} & \text{Si } i \text{ y } j \text{ están conectados} \\ 0 & \text{en otro caso} \end{cases}$$

Además de estos valores de la matriz ΔQ, se tiene: un montículo, donde se guardan los valores de ΔQ_{ij} ordenados de manera descendente, y un arreglo que contiene los valores de a_i. Los valores iniciales para a_i se calculan así:

$$a_i = \frac{k_i}{2m}$$

Una vez se realiza la primera unión de *clusters*, los valores de ΔQ_{ij}, a_i y del montículo deben ser actualizados según:

- Si el *cluster* k está conectado a los *clusters* i y j: $\Delta Q'_{jk} = \Delta Q_{ik} + \Delta Q_{jk}$
- Si k sólo está conectado a i: $\Delta Q'_{jk} = \Delta Q_{ik} - 2a_j a_k$
- Si k sólo está conectado a j: $\Delta Q'_{jk} = \Delta Q_{jk} - 2a_i a_k$
- $a'_j = a_j + a_i$
- Se extrae el valor del mayor ΔQ_{ij} del montículo y se reorganiza para tener la nueva referencia al mayor valor.

Este algoritmo tiene como complejidad $O(n \log^2 n)$ en redes dispersas y fue probado en un subgrafo de *amazon.com* de 400,000 nodos y 2 millones de aristas, revelando patrones en los hábitos adquisitivos de los clientes.

Community structure in social and biological networks

En [Girvan and Newman, 2002] se presenta un algoritmo jerárquico que explora la búsqueda de estructura de comunidad en las redes. Lo hace empleando una extensión de la métrica definida por Freeman *betweennes*, haciéndola pertinente para analizar las aristas y llamándola *edge betweenness*. La métrica *edge betweenness* es el número de los caminos más cortos entre nodos que pasan por una arista. Con el algoritmo se identifican aquellas aristas con alta *edge betweenness* bajo la premisa que estas aristas están entre dos comunidades y por eso conectan los nodos de una comunidad con la otra.

Los pasos del algoritmo son:

- Hallar el *edge betweenness* de todas las aristas en la red.
- Remover la arista con el más grande valor de la métrica.
- Actualizar el valor de la métrica para aquellas aristas que fueron afectadas por la arista que fue removida.
- Finalmente repetir desde el paso 2 hasta ya no tener más aristas.

Con el algoritmo se obtiene un dendrograma, con el cual se puede analizar la estructura de comunidad a diferentes niveles. Tiene un costo computacional de $O(m^2n)$ y en redes dispersas $O(n^3)$, por ello los autores descartan su uso en redes grandes, por ejemplo en redes de colaboración de científicos, más aun se presentan pruebas con redes de aproximadamente 100 tanto sintéticas como reales, donde el algoritmo presenta un buen resultado en términos de estructura de comunidad, según el análisis de los autores. Además las redes deben ser de aristas simples y no se permiten solapamientos entre las comunidades.

Modularity and community structure in networks

En [Newman, 2006] se presenta un algoritmo de *clustering* que busca maximizar la modularidad, pero planteada con vectores propios y una matriz que ellos llaman matriz de modularidad, la manera en que expresaron la modularidad es:

$Q = \frac{1}{4m} \sum_{ij} \left(A_{ij} - \frac{k_i k_j}{2m} \right) s_i s_j = \frac{1}{4m} s^T \mathbf{B} s$

Donde s representa un vector que contiene los valores s_i que pueden ser 1 si el nodo i pertenece al *cluster* uno o -1 si el nodo i pertenece al *cluster* dos, y $\mathbf{B}$ la matriz de modularidad que está definida como $B_{ij} = A_{ij} - \frac{k_i k_j}{2m}$. La matriz es simétrica con valores reales y tiene como valor propio 0 para el vector propio $\vec{1}$.

El algoritmos busca un valor para el vector s tal que se maximice el valor propio positivo de $\mathbf{B}$. Sin embargo como s contiene valores ± 1 no es posible realizar un cálculo exacto, y se realiza una aproximación describiendo a s como una combinación lineal, donde $s = \sum_1^n a_i \mathbf{u}_i$ $a_i = \mathbf{u}_i^T \cdot s$.

Para encontrar los *clusters*, el algoritmo haya el máximo valor propio de $\mathbf{B}$ y de acuerdo a este se obtiene s, una vez se tiene s se tiene la configuración de *clustering*, el proceso es repetido por cada uno de los *clusters* con la modificación que tiene en cuenta como se afectan los valores de los grados de los nodos y definiendo una nueva matriz de modularidad para el subgrafo. El algoritmo termina cuando ya no existe más subdivisiones que incrementen el valor de la modularidad. Tiene la particularidad que si el valor propio más grande que se encuentra es 0 se dice que el subgrafo no tiene más particiones.

El trabajo fue probado en en varias redes reales y comparado con otros algoritmos. También se probó la efectividad con dos redes que involucraban la representación de la política en Estados Unidos obteniendo buenos resultados según el autor. Además como aporte se muestra un corolario, a saber, *"Todas las comunidades en una red son por definición subgrafos indivisibles"*. El algoritmo no permite solapamientos y la complejidad computacional corresponde a $O(n^2 \log n)$, con el cual se pueden explorar redes de aproximadamente 100000 nodos. En una de las redes de pruebas con más nodos 27000 el algoritmo tuvo como tiempo de ejecución 20 minutos.

Community detection in complex networks using Extremal Optimization

En [Duch and Arenas, 2005] se presenta un algoritmo de *clustering* divisivo, que busca maximizar la modularidad empleando una heurística de búsqueda basada en optimización extrema. Los solapamientos no son permitidos.

En la optimizacion extrema se plantea una variable global que debe ser optimizada y variables locales que a su vez deben ser optimizadas. Partiendo de esto, en el trabajo ellos describen como la variable global la modularidad, y como las variables locales los aportes que realiza el nodo i a la configuración de *clustering*.

El algoritmo, sigue los siguientes pasos: se divide la red en dos *clusters* eligiendo los nodos aleatoriamente pero manteniendo cada *cluster* como un componente conexo. Una vez se tiene la división inicial, se procede a realizar la optimización de variables locales, moviendo un nodo de una comunidad a la otra hasta que se alcanza un máximo estable en la variable global. Luego se deben eliminar las aristas que conectan los componentes y realizar el proceso nueva nuevamente desde optimizar las variables locales. El algoritmo finaliza cuando ya no se incrementa la modularidad.

La complejidad del algoritmo corresponde a $O(N^2 \ln N)$. Se realizaron pruebas con redes artificiales y reales, donde ser verificó la efectividad del algoritmo y se comparó los resultados obtenidos en la modularidad con otros algoritmos respectivamente.

Fast unfolding of communities in large networks

En [Blondel et al., 2008] presentan un algoritmo de *clustering* aglomerativo jerárquico que usa una heurística para maximizar la modularidad.

El algoritmo propuesto consta de dos fases, ambas iterativas. La primera fase corresponde a considerar cada nodo como una comunidad o *cluster*, donde se evalúa el aporte que tiene a la modularidad mover el nodo a la comunidad de su vecino y esto por cada uno de los vecinos que tenga. Se mantiene la unión que corresponda al máximo aporte positivo a la modularidad. Este proceso se realiza con todos los nodos hasta que ya se haya alcanzado un máximo local de la modularidad, lo cual dependerá de cómo sean evaluados los nodos. Sin embargo se afirma que, mediante pruebas preliminares, estos cambios en el orden no afectan significativamente los valores de la modularidad. Para el cálculo de la modularidad usan ΔQ que son fáciles de calcular, a saber:

$$\Delta Q_C = \left[\frac{\sum_{in}+k_{i,in}}{2m} - \left(\frac{\sum_{tot}+k_i}{2m}\right)^2\right] - \left[\frac{\sum_{in}}{2m} - \left(\frac{\sum_{tot}}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)^2\right]$$

Donde $\sum_{in}$ es la suma de los pesos de las aristas dentro de la comunidad C, $\sum_{tot}$ es la suma de los pesos de las aristas que terminan en nodos de C, k_i es la suma de los pesos de las aristas que terminan en el nodo i, $k_{i,in}$ es la suma de los pesos de las aristas desde el nodo i hasta nodos en C y m es la suma de los pesos de todas las aristas de la red.

La segunda fase mapea todas las comunidades creadas en la primer fase como nodos y tiene en cuenta todas las aristas existentes tanto internamente como entre comunidades sumando todos los pesos y creando las nuevas aristas. Una vez se consigue la nueva red, se aplica de nuevo la primer fase hasta que no haya más cambios que aumenten la modularidad y se haya alcanzado un máximo de ésta.

Las dos fases son repetidas iterativamente hasta ya no conseguir aumentos en la modularidad. En las pruebas que se presentan en el documento se evidencia la eficiencia del algoritmo en

tiempo de ejecución, además también se realizaron pruebas con redes donde se conocía de antemano cómo debía ser la configuración de *clustering*, y se cotejaron los resultados obtenidos para probar la efectividad de dicho algoritmo obteniendo buenos resultados. El algoritmo fue pensado para ser aplicado en redes grandes, logrando aplicarlo en una red de 118 millones de nodos y aproximadamente 1 billón aristas, con un tiempo de respuesta de 152 minutos. El algoritmo no permite los solapamientos entre los *clusters*

1.4.2. Algoritmos de *clustering* que permiten el solapamiento

A Extraction Method of Overlapping Cluster based on Network Structure Analysis

En [Takaki et al., 2007] se presenta la detección de solapamiento en *clusters* con estructura de datos tipo red, pero con técnicas diferentes a las propuestas por la computación evolutiva.

Se presenta una técnica donde se usa el algoritmo de [Clauset et al., 2004] para hallar los *clusters*. Luego éstos son mapeados como nodos. Una vez se tiene esa contracción de nodos se aplica de nuevo el algoritmo de Clauset, Newman y Moore y se añaden los nodos que incrementan la métrica de modularidad. La modularidad es usada para optimizar la distribución de nodos en los *clusters*, como función objetivo a maximizar.

Usaron un algoritmo ya desarrollado y lo modificaron para que permitiera los solapamientos, bajo el argumento de la flexibilidad en la búsqueda de *clusters* y la posibilidad de encontrar nuevo conocimiento, por ejemplo, nodos importantes, como lo son los que envían información.

A Novel Genetic Algorithm for Overlapping Community Detection

En [Cai et al., 2011] es un algoritmo genético de *clustering* que a diferencia de los ya expuestos basa su análisis en las aristas de la red. No necesita conocer el número de *clusters*, puesto que el algoritmo encuentra este valor que maximiza la función objetivo.

La función objetivo que usan es llamada *densidad de partición* (D):

$D = \frac{2}{M} \sum_c m_c \frac{m_c - (n_c - 1)}{(n_c - 2)(n_c - 1)}$

Donde m_c es el número de aristas en el *cluster* c, y n_c es el número de nodos que están en las aristas que hay en el *cluster* c.

La codificación del cromosoma corresponde a un arreglo de m genes, $\{g_1, g_2, \cdots, g_m\}$, donde cada uno representa una arista y puede tomar los valores de los identificadores de las aristas adyacentes a esta. La interpretación es que tanto la arista g_m como el valor que esta contenga en el gen son aristas que pertenecen al mismo *cluster*. Como operador de cruce, hay dos cromosomas, se elige un gen aleatoriamente y se intercambian los valores contenidos. En el caso de la mutación se elige una arista adyacente al azar para reemplazar el valor actual del gen, esta debe ser diferente a la que tenga el gen.

Para verificar la eficacia y eficiencia del algoritmo se probó en redes de la vida real (aristas no dirigidas y la red de un solo componente) cuyos valores de nodos y aristas fueron

aproximadamente 1000 y fue comparado con otros algoritmos que también usan la densidad de partición como métrica. En los resultados de pruebas de comparación siempre obtuvo mejores resultados que los otros algoritmos.

Capítulo 2

Marco teórico

En este capítulo se encuentra la información respecto a los conceptos básicos que fueron necesarios para desarrollar este trabajo.

Se encuentra información respecto al análisis de redes (grafo), *clustering* en redes, y algoritmos genéticos, que corresponde a la técnica de computación evolutiva empleada en la solución del problema.

2.1. Redes

Las redes o grafos son estructuras de datos que representan relaciones entre objetos, como por ejemplo la interacción de proteínas, la amistad entre individuos de algún grupo, los científicos que publican juntos, las páginas web que dirigen a otras páginas web, entre muchas otras. Los grafos tienen su fundamentación matemática en lo que es conocido como teoría de grafos. En esta sección examinaremos algunos de los fundamentos que son relevantes para el problema del *clustering* en redes.

Para empezar, definimos una red $G(V, E)$ como el conjunto de nodos $V = \{v_1, v_2, \ldots, v_N\}$ y el conjunto de aristas $E = \{e_1, e_2, \ldots, e_M\}$, $\forall e\ e_k = (v_i, v_j),\ k \in \{1, \ldots, M\},\ i, j \in \{1, \ldots, N\}$ que corresponde a las relaciones entre nodos [Newman, 2010]. En este trabajo hablamos de N como el número total de nodos en la red, y de M como el número total de aristas en la red.

Las aristas de una red pueden ser, **dirigidas**, es decir, en la arista (v_i, v_j) existe un camino del nodo v_i al nodo v_j, o **no dirigidas**, donde existen dos caminos, uno en una dirección y otro en la otra dirección. Además éstas pueden tener un **peso**, cada arista tiene asociado un valor, por lo general un número real [Newman, 2010]. Este valor podría representar el costo de transportar algo de un nodo al otro y la semántica dependerá enteramente del fenómeno que represente la red. En una red se podría permitir que entre una misma pareja de nodos existan más de una arista, esto se conoce como **multi-arista** y la red que tenga multi-aristas es llamada multi-grafo. Finalmente en una red se podría permitir aristas **reflexivas**, esto quiere decir, que la arista es (v_i, v_i).

Las redes pueden ser representadas en el formato de matriz de adyacencia, en adelante A. A es una matriz cuadrada de $N \times N$. El valor de A_{ij} será 1 si existe la arista (v_i, v_j) y 0 si no. Sin embargo y como fue explicado en el párrafo anterior, las aristas podrían tener más

información como por ejemplo un valor de peso asociado. En este caso el valor de A_{ij} es el peso si la arista existe. En redes no dirigidas el valor de A_{ij} puede ser 2 cuando una arista existe, esto no es necesario, pero si es importante definirlo para algunos cálculos que involucren el número total de aristas, por ejemplo el cálculo de grados k_i de un nodo i, etc.

2.1.1. Modularidad

Se han desarrollado una serie de métricas que permiten medir ciertas características de las redes, incluso de obtener información cuando éstas no se pueden visualizar [Newman, 2010]. A continuación se describe la métrica usada en este trabajo.

La modularidad es una métrica que mide que tan bien se encuentra la configuración de *clustering*. Cuando una red tiene alto el valor de esta métrica se dice que la red es *assortative*. Una red *assortative* es aquella en la cual hay una fracción significativa de aristas que relacionan nodos del mismo tipo.[Newman, 2002]

Para calcular el valor de la modularidad Q se debe resolver la siguiente ecuación [Newman, 2010]

$$Q = \frac{1}{2m}\sum_{ij}\left(A_{ij} - \frac{k_i k_j}{2m}\right)\delta(c_i, c_j) \tag{2.1}$$

donde:

- c_i corresponde al número del *cluster* al que pertenece el nodo i.
- $\delta(c_i, c_j)$ corresponde a *delta de Kronecker*, y el valor que toma es de acuerdo a: $\delta(c_i, c_j) = \begin{cases} 1 & \text{si} \quad c_i = c_j \\ 0 & \text{si} \quad c_i \neq c_j \end{cases}$
- El término $\frac{k_i k_j}{2m}$ corresponde a la probabilidad que exista una arista entre el nodo i y el nodo j dados los grados de cada nodo y el total de aristas en la red.

En el trabajo presentado por Newman, se ha realizado una equivalencia de la ecuación 2.1 convirtiéndola en dos términos, cuyo orden de complejidad corresponde a $O(N + M)$ [Newman, 2010].
La equivalencia es:

La fracción de aristas que conectan nodos del *cluster* v con nodos del *cluster* w:

$$e_{vw} = \frac{1}{2m}\sum_{ij} A_{ij}\delta(c_i, v)\delta(c_j, w) \tag{2.2}$$

Y la fracción de aristas que terminan en nodos del *cluster* v:

$$a_v = \frac{1}{2m}\sum_{i} k_i\delta(c_i, v) \tag{2.3}$$

Finalmente la modularidad queda expresada como:

$$Q = \sum_{v} (e_{vv} - a_v^2) \tag{2.4}$$

Donde los indices i y j corresponden a los nodos, y los indices v y w corresponden a los *clusters*.

2.2. *Clustering* en redes

El *clustering* en redes es un problema que ya se encuentra explorado en varios trabajos ([Clauset et al., 2004];[Fortunato, 2010];[Gog et al., 2007];[Takaki et al., 2007]). Se ha discutido respecto a la complejidad del problema y una aproximación es problema NP-Completo, sin embargo, esto no es concluyente aún [Fortunato, 2010]. Además las propuestas de solución que hay son aproximaciones a la configuración óptima de *clustering*.

El *clustering* en redes sirve para analizar y entender los datos de la red, tiene aplicaciones en redes sociales, redes de páginas web, redes biológicas, etc. En las redes sociales sirve para entender la naturaleza de las interacciones de la comunidad que representa, en las redes de páginas web, encontrar grupos de páginas web que están relacionadas, y en redes biológicas encontrar los grupos de nodos que participan en más de un proceso funcional [Newman, 2010].

Con este tipo de *clustering* se trata de encontrar *clusters* que son permitidos por la topología de la red, es decir, debe existir al menos un camino entre todo par de nodos que pertenezca al *cluster* y el camino debe componerse sólo de aristas que pertenezcan al *cluster*.

El *clustering* en redes puede ser de dos tipos. El primero corresponde al *clustering* por contenido donde los nodos son agrupados de acuerdo a sus atributos. El segundo es el *clustering* estructural donde los nodos son agrupados de acuerdo a sus aristas [Takaki et al., 2007].

Para este trabajo se usó como algoritmo base el desarrollado en [Clauset et al., 2004] para conocer la configuración de *clustering* sin solapamientos. Este es un algoritmo voraz que usa la modularidad como métrica que evalúa qué tan bien se encuentra la configuración de *clustering*. El cálculo de la modularidad se hace incremental y conforme a un ΔQ_{vw} que corresponde al valor que se aporta a la modularidad de unir los *clusters* v y w.

El algoritmo de Clauset, Newman y Moore es de tipo estructural y jerárquico aglomerativo, donde en la primera iteración cada nodo es un *cluster* y conforme se va avanzando en el algoritmo se van uniendo los *clusters* hasta no tener más opciones para unir. La complejidad de este algoritmo es $O(md \log n)$ donde d es la profundidad del dendrograma que describe la configuración de *clustering*. En el caso de redes dispersas $m \sim n$ y $d \sim \log n$ la complejidad queda de orden: $O(n \log^2 n)$.

2.3. Algoritmos genéticos

La computación evolutiva es una rama de la inteligencia artificial que está inspirada en la evolución biológica. La evolución es vista como un algoritmo en el que se presentan las siguientes características [Baños, 2012]:

- Una población de entes.
- Reproducción (sexual: donde dos individuos participan en la creación de un nuevo individuo, o asexual: donde sólo un individuo participa en la creación de un nuevo individuo).
- Variación, puede ser incluida con la reproducción sexual o mutaciones.
- Presión selectiva.

Existen varias técnicas de computación evolutiva, entre ellas están: algoritmos genéticos, evolución gramatical, programación evolutiva, estrategias evolutivas entre otras. En este trabajo se usó como técnica los algoritmos genéticos. Los algoritmos genéticos son de búsqueda de óptimos, de búsqueda orientada. El algoritmo básico corresponde a:

- Población inicial de cromosomas
- Calcular la función de aptitud a todos los individuos
- Seleccionar probabilísticamente, en función de la aptitud, los individuos para el *mating pool*
- Aplicar operadores de cruce y mutación
- Continuar hasta cumplir una condición o hasta alcanzar un número determinado de generaciones.

Estos algoritmos son aplicados en problemas numéricos como: minimización de funciones multidimensionales, *clustering*, planeación de movimientos de robot, etc. Particularmente en *clustering* se usó en [Gog et al., 2007], donde además del algoritmo tradicional genético se comparte información entre cromosomas para acelerar el proceso de encontrar una solución.

Capítulo 3

Alcances de la propuesta

Se busca con este proyecto de trabajo de grado, obtener una propuesta de solución al problema de búsqueda de *clusters* con solapamiento, usando técnicas de computación evolutiva, y realizando un prototipo funcional.

Por ser un proyecto de investigación en pregrado, no se pretende obtener como resultado una teoría o hallazgos substanciales en la ciencia informática. Por el contrario se busca aportar con los resultados que se obtengan a la investigación en este problema y en las aplicaciones que tienen los algoritmos genéticos.

Las redes con las cuales se podrá usar y probar el algoritmo aquí desarrollado deben cumplir con las siguientes especificaciones:

- Las redes deben ser no dirigidas
- Las redes no deben tener multi-aristas ni aristas reflexivas
- Las redes pueden tener más de un componente, sin embargo el *clustering* sólo se realiza sobre el gran componente
- Las aristas no deben tener un peso asociado
- La cantidad de nodos de la red debe ser del orden de $N \approx 1000$
- La cantidad de aristas de la red debe cumplir con $M \sim N$

Capítulo 4

Modelo

Para plantear un modelo se revisaron las preguntas que debían responderse de acuerdo al planteamiento del problema (ver capítulo 1). Específicamente para la pregunta que hace referencia a las formas geométricas en las redes, se encontró que por la distribución de grados de los nodos, la forma tiende a ser ovalada y se define siempre por el nodo que se tome como centro, por consiguiente establecer una forma geométrica resultaba ser un procedimiento subjetivo. Por lo anterior, se decidió establecer un modelo sin buscar una forma geométrica. A continuación se describe el modelo desarrollado.

El algoritmo desarrollado en este trabajo, corresponde de tipo de *clustering* estructural, por lo que no es de nuestro interés analizar las etiquetas que tengan los nodos. Este algoritmo sólo debe conocer la información respecto a la topología de la red, es decir sólo la cantidad de nodos N, la cantidad de aristas M y el conjunto de aristas de toda la red E.

4.1. Redes y solapamiento

La especificación de redes definidas en el capítulo 3 son el resultado del análisis de la complejidad computacional del algoritmo desarrollado y de la complejidad espacial dada por las estructuras de datos empleadas. También se tuvo en cuenta la especificación del algoritmo de Clauset, Newman y Moore, en el cual establece que muchas de las redes reales son dispersas y jerárquicas y cumplen con $M \sim N$ [Clauset et al., 2004].

Se considera una red $G(V, E)$ como un conjunto de nodos V que interaccionan mediante un conjunto de aristas E.

$$E = \{e_1, e_2, \cdots, e_M\} \tag{4.1}$$
$$V = \{v_1, v_2, \cdots, v_N\} \tag{4.2}$$
$$|E| = M \tag{4.3}$$
$$|V| = N \tag{4.4}$$
$$\forall e \; e_k = (v_i, v_j) \tag{4.5}$$
$$k \in [1, M], \; i, j \in [1, N] \tag{4.6}$$

Además las aristas deben cumplir con:

$$\forall e\ e_k = (v_i, v_j) \leftrightarrow e_k = (v_j, v_i) \tag{4.7}$$
$$\nexists e\ e_k = (v_i, v_i) \tag{4.8}$$
$$\forall i, j\ \exists! e\ e_k = (v_i, v_j) \tag{4.9}$$
$$\forall e\ e_k = 1 \tag{4.10}$$
$$k \in [1, M],\ i, j \in [1, N] \tag{4.11}$$

Por otro lado el solapamiento en una configuración de *clustering* en este trabajo se da cuando un nodo pertenece a más de un *cluster*. Se tiene el conjunto de *clusters* C que describe cómo están distribuidos los nodos en el total de *clusters* $|C|$, además cada uno de los nodos tiene una variable de pertenencia y_{v_i} donde se guarda el número de *clusters* a los cuales el nodo pertenece. Esta variable de pertenencia debe tener como mínimo valor 1, puesto que en no se permite que un nodo no pertenezca al menos a un *cluster* (nodos islas).

$$C = \{c_1, c_2, \cdots, c_{|C|}\} \tag{4.12}$$
$$c_w = \{v_1, v_2, \cdots, v_i\} \tag{4.13}$$
$$c_w \subset V \tag{4.14}$$
$$c_w \in C \tag{4.15}$$
$$1 \leq y_{v_i} \leq |C| \tag{4.16}$$
$$i \in [1, N],\ w \in [1, |C|] \tag{4.17}$$

Para que halla un solapamiento en la configuración de *clustering* debe ocurrir que:

$$\exists v_i\ y_{v_i} > 1 \tag{4.18}$$
$$\exists c_w, c_z\ c_w \cap c_z \neq \emptyset \tag{4.19}$$
$$i \in [1, N],\ w, z \in [1, |C|] \tag{4.20}$$

Por como esta definido el solapamiento, éste puede ocurrir permitiendo que un nodo pertenezca a más de dos *clusters* incluso a todos los *clusters*. También es permitido que un solapamiento entre *clusters* contenga más de un nodo.

4.2. Parámetros de entrada

- $G(V, E)$**:** Corresponde a la topología de la red
- B**:** Cantidad de *bits* a mutar en la matriz X[1]. En este algoritmo se puede mutar 1 o 2 *bits* de la matriz X.
- G**:** Número de generaciones, es decir, la cantidad de veces que se debe repetir el proceso selección y mutación.
- I**:** Número de individuos, es decir, la cantidad de cromosomas con los que se inicia el algoritmo. En este algoritmo esta cantidad nunca cambia.
- K**:** Porcentaje del *mating pool*. Se debe establecer un porcentaje de la cantidad de individuos que entrarán en el *mating pool* para el proceso de mutación.

[1]Más adelante se revisará a qué corresponde esta matriz y su contenido.

4.3. Estructuras de datos

- **Topología de la red:** Se usa la estructura lista de adyacencia para almacenar la topología de la red. Esta estructura consiste en una lista de listas, cada indice corresponde a un nodo y cada lista de un nodo corresponde al conjunto de los nodos vecinos. Considere la siguiente red $G(V, E)$: $V = \{1, 2, 3, 4\}$ y $E = \{(1, 2), (3, 2), (4, 1)\}$ la lista de adyacencia es:

Nodo	Vecinos
1	2, 4
2	1, 3
3	2
4	1

- **Matriz X:** Esta matriz es descrita en [Takaki et al., 2007], corresponde a una matriz $C \times N$ donde C es el número de *clusters* que tiene la configuración de *clustering* inicial. X_{vi} es *true* si el nodo i pertenece al *cluster* v y es *false* si el nodo i no pertenece al *cluster* v. Entonces es una matriz que contiene la información de configuración de *clustering*.

- **Cromosoma:** Cada cromosoma es una estructura que contiene la matriz X de la configuración de *clustering*, y una variable que contiene la cantidad de *clusters* que tiene la solución.

4.4. Algoritmo

A continuación se presentan las funciones de aptitud empleadas en este trabajo:

- **Modularidad Q:** La explicación de esta medida puede encontrarse en el capítulo 2. El valor de la modularidad está en el intervalo $[-1, 1]$, y entre más cerca se esté de 1 se dice que la configuración de *clustering* es mejor. La función de aptitud modularidad es:

$$Q = \sum_{v} (e_{vv} - a_v^2) \tag{4.21}$$

- **Fracción de *clusters* C_{fra}:** Esta función, es alternativa a la modularidad y no siempre es calculada salvo que sea necesario[2].

  Se tiene un número inicial de *clusters* C_{ini}, sin embargo este puede cambiar C_{fin} (siempre disminuir nunca aumentar respecto al número inicial) por lo que se realiza una fracción entre estos dos valores. El valor de la fracción está en el intervalo $(0, 1]$ y conforme se acerque a 1 se considera mejor. Con esto estamos diciendo que se busca mantener el mismo número de *clusters* inicial y también se evita la solución trivial de todos los nodos que pertenecen al mismo *cluster*. La función de aptitud fracción de *clusters* queda:

$$C_{fra} = \frac{C_{fin}}{C_{ini}} \tag{4.22}$$

[2]Revisar el paso **Mutar** del algoritmo

Es importante aclarar que la modularidad no es una métrica que tenga en cuenta los solapamientos. Por lo tanto, en el caso que se presente un solapamiento donde intervenga un conjunto de *clusters* C_s y cuya intersección sea un conjunto de nodos V_s, es necesario verificar que éste sea coherente a la topología de la red, llevando a cabo los siguientes pasos:

- Seleccionar un *cluster* de C_s
- Obtener todos los nodos que pertenecen a él y todas las aristas internas
- Por cada nodo solapado de V_s, encontrar al menos un camino, con sólo las aristas internas, que conecte con cada uno de los demás nodos del *cluster*
- Repetir el proceso por cada uno de los *cluster* de C_s
- Si durante el proceso existe un par de nodos para los cuales no se puede encontrar al menos un camino, se concluye que el solapamiento no es coherente con la topología de la red

Un solapamiento válido con la topología de la red debe cumplir con:

$$C_s = \{c_1, \cdots, c_w\} \tag{4.23}$$

$$V_s = \{v_1, \cdots, v_i\} \tag{4.24}$$

$$\forall c_w \in C_S \; \forall v_i, v_j \in V_{c_w} \; \exists \texttt{path}(v_i, v_j) \in E_{c_w} \tag{4.25}$$

$$w \in [1, |C|], \; i, j \in [1, N] \tag{4.26}$$

Donde E_{c_w} es el conjunto de las aristas internas del cluster c_w, V_{c_w} es el conjunto de nodos que pertenecen al cluster c_w y $\texttt{path}(v_i, v_j)$ es un camino que une los nodos v_i y v_j.

Si la modularidad se evaluara sobre una configuración de *clustering* con solapamiento que no fuese coherente con la topología de la red, es posible obtener valores positivos y cercanos a 1 (valor máximo) y aun así corresponder a una mala configuracion de *clustering*. Un ejemplo de ello es ilustrado en la figura 4.2 y corresponde a una red de 12 nodos y 11 aristas que inicialmente fue dividida en 3 *clusters*, obtenidos con el algoritmo de [Clauset et al., 2004]. En la figura 4.1 se puede ver la configuración de *clustering* que obtuvo una modularidad de 0,458678. Mientras que la configuración de *clustering* mostrada en la figura 4.2 obtuvo una modularidad de 0,51447 con un solapamiento que no es coherente con la red (nodos 2 y 8). Por lo anterior es importante definir pasos que no admitan este tipo de solapamientos y por el contrario busquen aquellos que sean coherentes con la topologia.

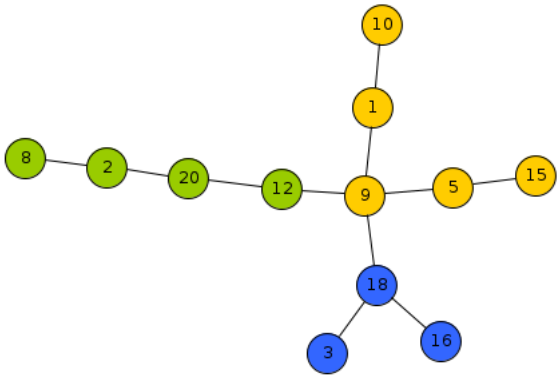

Figura 4.1: Red con la configuración de *clustering* encontrada por el algoritmo de Clauset, Newman y Moore. $Q = 0{,}458678$

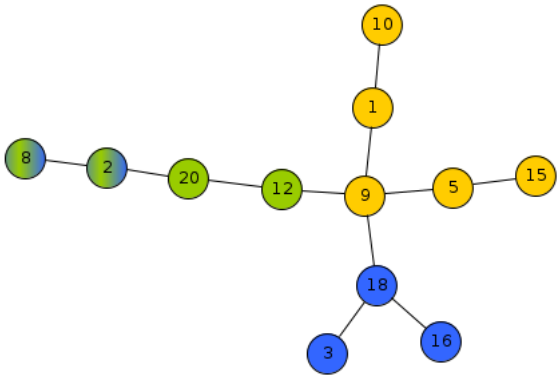

Figura 4.2: Red con solapamiento incoherente con la topologia de la red. $Q = 0{,}51447$

A continuación se explican los pasos que sigue el algoritmo, y cada uno de sus detalles.

1. **Aplicar algoritmo de Clauset, Newman y Moore[Clauset et al., 2004]:** Se usa este algoritmo como base de partida, se aplica a la red y se obtiene la configuración inicial de *clustering*, el número de *clusters* inicial C_{ini}, el vector que indica el tamaño de cada *cluster*, es decir, cuántos nodos pertenecen a un *cluster*, y se obtiene la modularidad.

2. **Inicializar matriz X:** Con la configuración de *clustering* obtenida en el paso anterior se construye la matriz X inicial, ésta no será modificada y sólo es usada de consulta.

3. **Inicializar población:** Para inicializar la población, por cada individuo se copia la matriz X inicial, aleatoriamente se elije una posición de la matriz X_{vi} cuyo valor sea *false* y se cambia a *true*. Lo que se busca es que cada individuo inicial tenga un solapamiento, y para garantizar que esto sea así se tienen las siguientes condiciones:

   - Elegir un *cluster* al azar.
   - Elegir un nodo al azar, y obtener los vecinos que no pertenezcan al *cluster* elegido.
   - Si no se obtiene ningún vecino, es decir, el nodo está en el centro del *cluster* se debe elegir otro nodo al azar, de nuevo.

- Si se encuentran vecinos posibles se elige uno al azar y se cambia el *bit* de pertenencia al *cluster* de *false* a *true*.

Sólo se pasa un cromosoma con la configuración obtenida en el paso **Aplicar algoritmo de Clauset, Newman y Moore** y se establece como el mejor cromosoma encontrado hasta ahora en el algoritmo, la idea es encontrar soluciones cuyos valores de la modularidad sean más altos que los encontrados con Clauset, Newman y Moore.

4. **Seleccionar:** Para el proceso de selección se usó el método de torneo, en el cual se eligen 2 cromosomas al azar y se evaluá su modularidad. El cromosoma que tenga mayor modularidad será el seleccionado para ingresar al *mating pool*. Si los 2 cromosomas obtienen la misma modularidad se usa como función de aptitud la fracción de *clusters*. El cromosoma con mayor valor en la fracción de *clusters* será el elegido para ingresar al *mating pool*. Finalmente si los dos cromosomas tienen igual fracción de *clusters* se elige cualquiera para ingresar al *mating pool*.

   Durante el proceso de selección se va guardando cuál ha sido la mejor modularidad obtenida hasta el momento y se guarda el cromosoma que obtuvo dicha modularidad.

5. **Mutar:** Por cada uno de los cromosomas en el *mating pool* excepto el cromosoma que sea el mejor hasta el momento en el algoritmo, se realizan los siguientes pasos: Primero se verifica cuántos *bits* se van a modificar, luego se eligen las coordenadas X_{vi} para la matriz X de acuerdo a la cantidad de *bits*, éstas se eligen aleatoriamente. Para elegir las coordenadas se empieza con un *cluster* (v), luego se elige un nodo que pertenezca a éste y se extraen sus vecinos para elegir uno de manera aleatoria (i), se realiza de esta manera para mantener la coherencia en la topología de la red y la configuración de *clustering*.

   Por cada cromosoma, se cambia el *bit* que indique la coordenada, teniendo en cuenta las siguientes situaciones:

   - Si el *bit* estaba en *true*, es decir, el nodo i pertenece al *cluster* v se cambia a *false*, pero se debe verificar que el nodo quede perteneciendo a al menos un *cluster*, si no es así entonces el *bit* no se cambia y se queda en *true*.
   - Si el *bit* estaba en *false* se cambia a *true*.

   En ambas situaciones se deben actualizar los valores de: tamaño del *cluster* y cantidad de *clusters*. Esta última variable depende de si el tamaño del *cluster* estaba a 0 o si por el contrario se actualiza a 0.

6. **Seleccionar solución:** Una vez se terminan las generaciones, se elige el cromosoma que haya sido marcado como el mejor cromosoma en todo el algoritmo y éste será la solución que retorna el algoritmo.

4.5. Complejidad computacional

Para calcular la complejidad se deben revisar cada uno de los pasos del algoritmo. Antes listaremos la nomemclatura que será usada:

N Número total de nodos

M Número total de aristas

C Número total de *clusters*

I Número de individuos

G Número de generaciones

$\frac{M}{N}$ Media de grado de un nodo

$\frac{N}{C}$ Media de tamaño de un *cluster*

K Tamaño del *mating pool*

1. **Algoritmo de Clauset, Newman y Moore:** Este algoritmo tiene como complejidad computacional $O(MD \log N)$ que en redes dispersas queda como $O(N \log^2 N)$ [Clauset et al., 2004].

2. **Inicializar matriz X:** Como ésta es una matriz de $C \times N$ entonces la complejidad es: $O(CN)$, el costo de consultar, editar un elemento es constante $O(1)$.

3. **Inicializar la población:** Este depende de la cantidad de individuos y de la cantidad de nodos que tiene un *cluster* elegido al azar, la complejidad es $O(\frac{IM}{C})$ en casos donde el nodo elegido no es un nodo central del *cluster*, es decir el caso promedio la complejidad queda $O(\frac{IM}{N})$

4. **Seleccionar:** Para este paso es importante saber que la complejidad de la modularidad es $O(N+M)$ y la complejidad de la fracción de *clusters* es $O(1)$, por lo que la complejidad de este paso queda en términos del tamaño del *mating pool* y la modularidad, lo que corresponde a $O(K(N + M))$

5. **Mutar:** La complejidad de este paso es $O(KC)$, pues en el peor caso, debe revisar que el nodo quede perteneciendo a al menos un *cluster*.

6. **Seleccionar solución:** Como en este proceso sólo se toma como solución el cromosoma que está marcado como el mejor encontrado en todo el algoritmo, la complejidad es constante $O(1)$.

Ahora bien para hallar la complejidad total del algoritmo, los pasos **Selección** y **Mutar** deben ser multiplicados por el número de generaciones. Finalmente la complejidad del algoritmo es:

$$O(N \log^2 N + CN + \frac{IM}{C} + GK(N + M + C) + 1) \tag{4.27}$$

Sin embargo las variables C y K pueden ser tomadas como constantes lo que daría lugar a tener una complejidad final de:

$$O(N \log^2 N + N + IM + GN + GM + G) \tag{4.28}$$

Capítulo 5

Implementación

En este capítulo se explicará la implementación del prototipo desarrollado, incluyendo los detalles técnicos como el lenguaje de programación, librerías y formatos usados.

A continuación se explican los parámetros del prototipo y luego se describen los detalles relevantes de la implementación del algoritmo.

5.1. Parámetros del prototipo

Archivo de red
Corresponde al archivo que contiene la red a la cual se le aplicará el algoritmo.

Población inicial
Es el tamaño de la población, es decir, la cantidad de individuos. Ésta puede ir desde 1 hasta 500.

Número de generaciones
El número de generaciones puede ir desde 10 hasta 100.

Porcentaje del *mating pool*
El porcentaje va desde 5 % hasta 100 %.

Cantidad de *bits* a mutar
Se puede elegir entre mutar un sólo *bit* o mutar dos *bits*.

5.2. Implementación algoritmo y prototipo

El prototipo fue desarrollado usando como lenguaje de programación *C++*. Para el procesamiento de la red se usó la librería *igraph*[Csardi and Nepusz, 2006] version 0.5.4, que cuenta con las opciones de procesar archivos que contengan una red en varios formatos, aplicar algoritmos de *clustering*, obtener datos como cantidad de nodos, aristas, etc de la red. En los próximos párrafos explicaremos los formatos usados. Para la interfaz gráfica del prototipo se usó la librería *QT5* version 5.0.2. La documentación del código desarrollado se realizó con *Doxygen* [van Heesch, 2008].

En la figura 5.1 se encuentra la interfaz gráfica del prototipo. Esta interfaz cuenta con 3

paneles: El primero es **Datos Entrada** donde se encuentra la información referente a la red (cantidad de nodos y aristas) a la cual se le aplicará el algoritmo. Es importante resaltar que una red puede tener más de un componente y como fue especificado en el capítulo 3 el algoritmo trabaja con una red conexa y por ello se extrae el componente que tenga más porcentaje de los nodos. Por lo anterior se muestra además los datos (cantidad de nodos y aristas) del subgrafo si éste llegase a necesitarse. El segundo panel es **Datos Salida** que indica la modularidad Q obtenida y la cantidad de nodos y aristas que tiene la red solución[1]. Finalmente el tercer panel es **Ajustes de parámetros** del algoritmo, ahí se ajustan los parámetros descritos en la sección anterior.

Figura 5.1: Prototipo desarrollado con la librería *QT5*

5.2.1. Formatos de entrada

Para ingresar las redes al prototipo, existen dos formatos de entrada posible *txt* y *graphml* [Brandes et al., 2002]. En ambos casos la librería *igraph* ofrece métodos para leer los archivos.

Formato *txt*

Este formato sólo permite ingresar la información respecto a la topología de la red, es decir todas las aristas. Por cada línea del archivo se escribe una arista, se escribe el id del nodo origen, un espacio en blanco y el id del nodo destino.

Para la red $G = \{(1,2),(2,3),(2,4),(3,4)\}$ el archivo en formato txt sería:

[1] siempre corresponde a los mismos valores en la información del subgrafo

```
1 2
2 3
2 4
3 4
```

En este formato los ids de nodos deben ser números, y por restricciones de *igraph* los nodos serán numerados desde el 0. Sin embargo en la solución que arroja el algoritmo, los ids ingresados en el archivo de entrada se mantienen.

Formato *graphml*

Este es un formato en *xml* que además de contener la topología de la red, contiene información como atributos de las aristas y nodos; en este formato es posible que el id de un nodo sea una combinación alfanumérica.

Para la red $G = \{(1,2),(2,3),(2,4),(3,4)\}$ el archivo en formato *graphml* sería[2]:

```
<?xml version="1.0" encoding="UTF-8"?><graphml xmlns="http://graphml.graphdrawing.org/xmlns">
<key attr.name="label" attr.type="string" for="node" id="label"/>
<key attr.name="Edge Label" attr.type="string" for="edge" id="edgelabel"/>
<key attr.name="weight" attr.type="double" for="edge" id="weight"/>
<key attr.name="Edge Id" attr.type="string" for="edge" id="edgeid"/>
<key attr.name="r" attr.type="int" for="node" id="r"/>
<key attr.name="g" attr.type="int" for="node" id="g"/>
<key attr.name="b" attr.type="int" for="node" id="b"/>
<key attr.name="x" attr.type="float" for="node" id="x"/>
<key attr.name="y" attr.type="float" for="node" id="y"/>
<key attr.name="size" attr.type="float" for="node" id="size"/>
<graph edgedefault="undirected">
<node id="1">
<data key="size">10.0</data>
<data key="r">153</data>
<data key="g">153</data>
<data key="b">153</data>
<data key="x">-1322.8407</data>
<data key="y">604.36487</data>
</node>
<node id="2">
<data key="size">10.0</data>
<data key="r">153</data>
<data key="g">153</data>
<data key="b">153</data>
<data key="x">-1280.5774</data>
<data key="y">519.8383</data>
</node>
<node id="3">
<data key="size">10.0</data>
<data key="r">153</data>
<data key="g">153</data>
<data key="b">153</data>
<data key="x">-1217.1825</data>
<data key="y">642.40186</data>
</node>
<node id="4">
<data key="size">10.0</data>
<data key="r">153</data>
<data key="g">153</data>
<data key="b">153</data>
<data key="x">-1187.5981</data>
<data key="y">532.51733</data>
</node>
<edge source="1" target="2">
<data key="weight">1.0</data>
</edge>
<edge source="2" target="3">
<data key="weight">1.0</data>
</edge>
<edge source="2" target="4">
<data key="weight">1.0</data>
</edge>
```

[2]Este archivo se obtuvo con la herramienta *gephi* [Bastian et al., 2009]

```
<edge source="3" target="4">
<data key="weight">1.0</data>
</edge>
</graph>
</graphml>
```

De nuestro interés son las etiquetas `node` y su atributo `id` y `edge` y sus atributos `source` y `target`, pues estas son las que describen la topología de la red, y el atributo `id` es el que se usa posteriormente para identificar el nodo en la solución.

5.2.2. Detalles técnicos del algoritmo

Antes de aplicar el algoritmo a la red, ésta debe pasar por un preprocesamiento donde se extraiga el gran componente si lo tiene, y si la red fue leída desde una archivo en formato *txt* se debe establecer los ids de los nodos. Luego de esto el algoritmo es aplicado. En esta sección nombraremos algunos de los métodos, y estructuras de datos usadas.

Para la etapa de preprocesamiento fue necesario usar el método *igraph_subgraph* de *igraph* para obtener el componente que tiene la mayor cantidad de nodos de la red. En el caso de los archivos con formato *txt* fue necesario agregar el atributo *"name"* a los nodos para guardar su identificador, esto se hizo usando el método *SETVAS(&grafo,"name",nodo,id)*.

Para el primer paso del algoritmo que corresponde a aplicar el algoritmo de Clauset, Newman y Moore se usó un método que provee la librería *igraph*, *igraph_community_fastgreedy*. Aunque éste está basado en el artículo [Clauset et al., 2004], tiene algunas optimizaciones basadas en otro articulo[3], aún así se mantiene la complejidad descrita en el capítulo 4.

Finalmente, para las estructuras listas de adyacencia se usó *unordered_set* de *C++11*, la justificación está dada en términos de la complejidad. Obtener el tamaño del arreglo es $O(1)$, eliminar e insertar un elemento que en el caso promedio es $O(1)$ y en el peor caso $O(d)$ donde d corresponde a la cantidad de elementos que haya en el arreglo.

5.2.3. Formato de salida

La solución encontrada por el algoritmo se exportará en formato *graphml* [Brandes et al., 2002] y no es competencia del prototipo realizar la visualización, sin embargo se recomiendan aplicaciones como yEd [yWorks GmbH,] y gephi [Bastian et al., 2009] para este propósito. A los nodos se les agrega un atributo referente al identificador del *cluster* al que pertenecen. A continuación un fragmento del formato de salida.

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
          http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">
<!-- Created by igraph -->
<key id="name" for="node" attr.name="name" attr.type="string"/>
<key id="modularity_class_1" for="node" attr.name="modularity_class_1" attr.type="double"/>
<graph id="G" edgedefault="undirected">
        <node id="n0">
                <data key="name">1</data>
                <data key="modularity_class_1">0</data>
        </node>
        <node id="n1">
```

[3]Según la página web de documentación `http://igraph.sourceforge.net/doc/html/ch22s06.html#igraph_community_fastgreedy`

```
            <data key="name">2</data>
            <data key="modularity-class-1">1</data>
        </node>
        <!-- ETC -->
        <edge source="n1" target="n4">
        </edge>
        <edge source="n0" target="n5">
        </edge>
        <!-- ETC -->
</graph>
</graphml>
```

El nuevo atributo corresponde a *modularity-class* y contiene un entero (id del *cluster*). La solución puede visualizarse con cualquier herramienta que grafique redes desde archivos con formato *graphml* un ejemplo de estas es *gephi* [Bastian et al., 2009].

Capítulo 6

Pruebas y discusión de resultados

En este capítulo se describe el proceso de pruebas que se llevó a cabo para verificar la correctitud del proceso de aplicar el algoritmo de Clauset, Newman y Moore a la red y el rendimiento del algoritmo genético en cuanto a las soluciones que encuentra para una misma red. También se realiza una comparación del algoritmo con otros algoritmos de la bibliografía.

A continuación se explica cada una de las pruebas, sus resultados y el análisis de éstos.

6.1. Pruebas unitarias sobre el procedimiento aplicar algoritmo de Clauset, Newman y Moore

Con esta prueba se busca comprobar la correctitud del procedimiento. Se usó una red ampliamente usada en la bibliografía y se evaluó que los resultados para un número de ejecuciones fueran siempre los mismos.

La red seleccionada para esta prueba fue la red del club de karate de Zachary, que es una red clásica en la bibliografía [Takaki et al., 2007], [Gog et al., 2007], [Newman, 2006] y [Clauset et al., 2004].

Tabla 6.1: Datos de la red de club de karate de Zachary

Nodos	**Aristas**
34	78

El proceso de aplicar algoritmo de Clauset, Newman y Moore se repitió 1000 veces.

6.2. Prueba algoritmo genético

El objetivo de esta prueba es medir qué tan efectivo es el algoritmo genético concretamente, obtener el porcentaje de veces que el algoritmo obtuvo una mejor modularidad que la incial (la modularidad obtenida con el algoritmo de Clauset, Newman y Moore). Se revisó la desviación estándar de las modularidades obtenidas.

Para la prueba se usó la red del club de karate de Zachary [Zachary, 1977], se ejecutó el algoritmo 1000 veces con los siguientes parámetros:

Tabla 6.2: Parámetros algoritmo genético

Población (I)	**Generaciones (G)**	***Mating pool* (%) (K)**	**No. *bits* a mutar (B)**
100	100	10	1

6.3. Prueba de comparación del algoritmo con otros algoritmos de *clustering*

Esta prueba evaluó el rendimiento del algoritmo en términos de la modularidad. Esto se consigue comparando la modularidad obtenida con el algoritmo con las modularidades obtenidas en otros trabajos de *clustering* en redes. Los algoritmos de comparación no permiten los solapamientos.

En [Newman, 2006] se presenta una tabla de comparación del algoritmo desarrollado con otros algoritmos. Se comparan cuatro algoritmos con seis redes de las cuales logramos obtener dos, esto es porque dos de las redes presentadas tienen más de 5.000 nodos, una red es dirigida, y finalmente no se pudo extraer una de las redes y al no tener los mismos datos quedó por fuera de comparación. Los algoritmos que se compararan en el trabajo [Newman, 2006] son el algoritmo de Clauset, Newman y Moore [Clauset et al., 2004] (CNM), el algoritmo de Duch y Arenas (DA) que es un algoritmo de optimiazación, el algoritmo de Girvan y Newman (GN) que es un algoritmo basado en la métrica *betweenness* y el algoritmo desarrollado en el artículo (MC) que es un algoritmo basado en vectores propios. Las redes que logramos obtener son la red del club de karate de Zachary [Zachary, 1977], y la red de músicos de jazz [Gleiser and Danon, 2003]. En la tabla 6.3 se puede ver la cantidad de nodos y aristas que tiene cada una de estas redes.

Además de estas dos redes se probó el algoritmo con cuatro redes más que se comparan con el algoritmo de Clauset, Newman y Moore [Clauset et al., 2004] y el algorimto presentado por Blondel, Guillaume, Lambiotte, y Lefebvre (BGLL) que es un algoritmo de optimización de la modularidad aplicado en redes extensas. Estos dos algoritmos se encuentran implementados en la librería igraph [Csardi and Nepusz, 2006]. Las cuatro redes corresponden a: una red de colaboración de científicos [Leskovec et al., 2007] que han publicado juntos en *arXiv*[1], una red de interacción de proteínas [Jeong et al., 2001], una red eléctrica de los Estados Unidos [Watts and Strogatz, 1998] y finalmente una red social (subgrafo) de facebook [McAuley and Leskovec, 2012]. Se puede ver la cantidad de nodos y aristas en la tabla 6.3

Tabla 6.3: Redes para comparación de algoritmos

[1]Para conocer más ver `http://arxiv.org`

Red	**Nodos (N)**	**Aristas (M)**	**Algoritmos para comparar**
Zachary	38	78	(GN), (CNM), (DA), (MC) y (BGLL)
Músicos de jazz	198	2742	(GN), (CNM), (DA), (MC) y (BGLL)
Interacción de proteínas	1458	1948	(CNM) y (BGLL)
Red eléctrica	4941	6594	(CNM) y (BGLL)
Colaboración de científicos	4158	13422	(CNM) y (BGLL)
Red (subgrafo) de facebook	4039	88234	(CNM) y (BGLL)

Respecto a los archivos de datos encontrados para estas redes, los archivos de la red de músicos de jazz, interacción de proteínas y colaboración entre científicos requirieron un paso adicional. Este paso consistió en eliminar las multi-aristas y las aristas reflexivas, de esta manera nuestro algoritmo ya podía ser probado con estas redes.

Para la ejecución del algoritmo evolutivo se establecieron los siguientes parámetros:

Tabla 6.4: Parámetros algoritmo genético para pruebas de comparación

Población (I)	**Generaciones (G)**	***Mating pool* (%) (K)**	**No. *bits* a mutar (B)**
100	100	10	2

El algoritmo genético se ejecutó 200 veces por red.

6.4. Resultados

Para todas las pruebas realizadas se usó la red del club de karate de Zachary [Zachary, 1977]. En la figura 6.1 podemos ver la topología de la red[2]. Esta red corresponde a un ejemplo de la vida real, cuyos nodos representan personas que pertenecían al club y las aristas representan la amistad entre ellas. Esta red tiene dos *clusters*. Mientras se estudiaba la red hubo una discusión interna entre el administrador y el instructor del club y ésta provocó que la red se dividiera en dos *clusters*. Un *cluster* agrupa a quienes apoyaron al administrador del club y el otro a los que apoyaron al instructor del club. En la figura 6.2 se puede observar esta configuración de *clustering*, sin embargo esta configuración, que es la natural, no maximiza el valor de la modularidad [Gog et al., 2007].

[2]Las imagenes se realizaron con la herramienta *yED* [yWorks GmbH,]

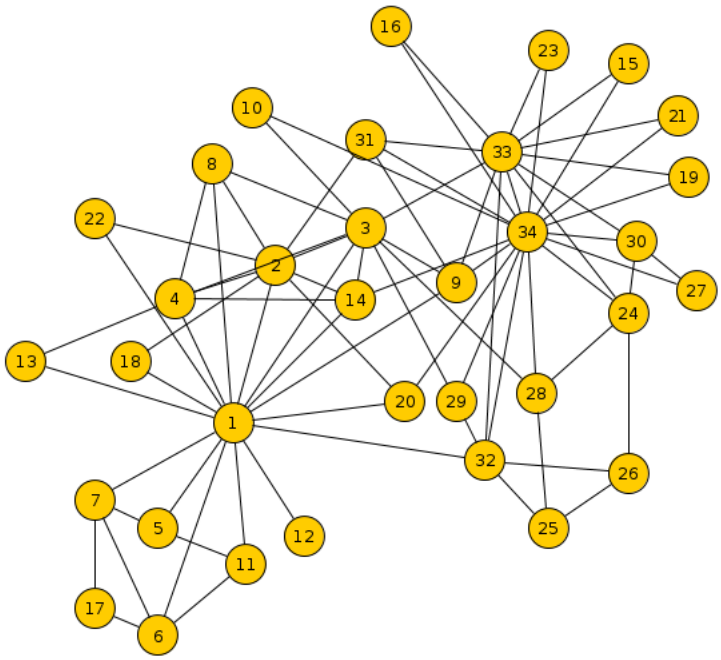

Figura 6.1: Red del club de karate de Zachary

6.4.1. Resultados pruebas unitarias sobre el procedimiento aplicar algoritmo de Clauset, Newman y Moore

En las 1000 ejecuciones que se realizaron, por cada iteración se guardó el valor de la modularidad, la cantidad de *clusters*, el tamaño de cada *cluster* y la configuración de *clustering*. Se calculó la varianza para cada una de las columnas obteniendo para todas varianza igual a 0, lo que quiere decir que los resultados siempre fueron los mismo en todas las ejecuciones.

A continuación se describe los resultados obtenidos:

Tabla 6.5: Resultados al aplicar el algoritmo de Clauset, Newman y Moore a la red de club de karate de Zachary

Propiedad	**Resultado**
Modularidad	0,380671
Cantidad de *clusters*	3
Tamaño de cada *cluster*	C1=8; C2=17; C3=9
Configuración C1 (color azul)	$\{1, 5, 6, 7, 11, 12, 17, 20\}$
Configuración C2 (color amarillo)	$\{9, 15, 16, 19, 21, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34\}$
Configuración C3 (color verde)	$\{2, 3, 4, 8, 10, 13, 14, 18, 22\}$

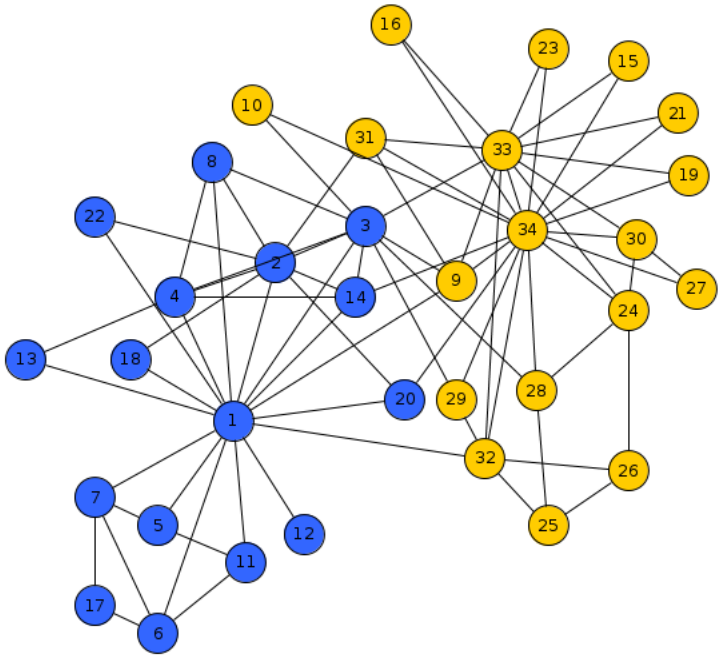

Figura 6.2: Red del club de karate de Zachary con la configuración de *clustering* natural

En la figura 6.3 se puede observar gráficamente la configuración de *clustering* que encontró el algoritmo de Clauset, Newman y Moore.

Podemos concluir que la prueba unitaria para el procedimiento de aplicar el algoritmo de Clauset, Newman y Moore ha sido satisfactoria y se evidencia que no se encuentran errores.

6.4.2. Resultado prueba algoritmo genético

Esta prueba se realizó para verificar el propósito del algoritmo que es: "maximizar la modularidad permitiendo el solapamiento entre *clusters*". A continuación los resultados obtenidos:

- En las 1000 ejecuciones se logró obtener una modularidad superior a la obtenida por el algoritmo de Clauset, Newman y Moore $0,380671$. La modularidad mínima que se encontró fue $0,424227$
- La desviación estándar obtenida en la muestra fue de $0,005831$ lo que nos indica que la mayoría de los valores de la modularidad en la muestra, no presentaron grandes variaciones.
- Se obtuvieron 64 soluciones diferentes en toda la muestra. En la tabla 8.1 se resume toda la información de las diferentes soluciones, modularidad, configuración de *clustering* y cantidad de soluciones de ese tipo en la muestra.
- En todas las soluciones encontradas, la configuración de *clustering* tuvo solapamientos.
- Los datos se organizaron en cuartiles. La tabla 6.6 ilustra la distribución de los datos:

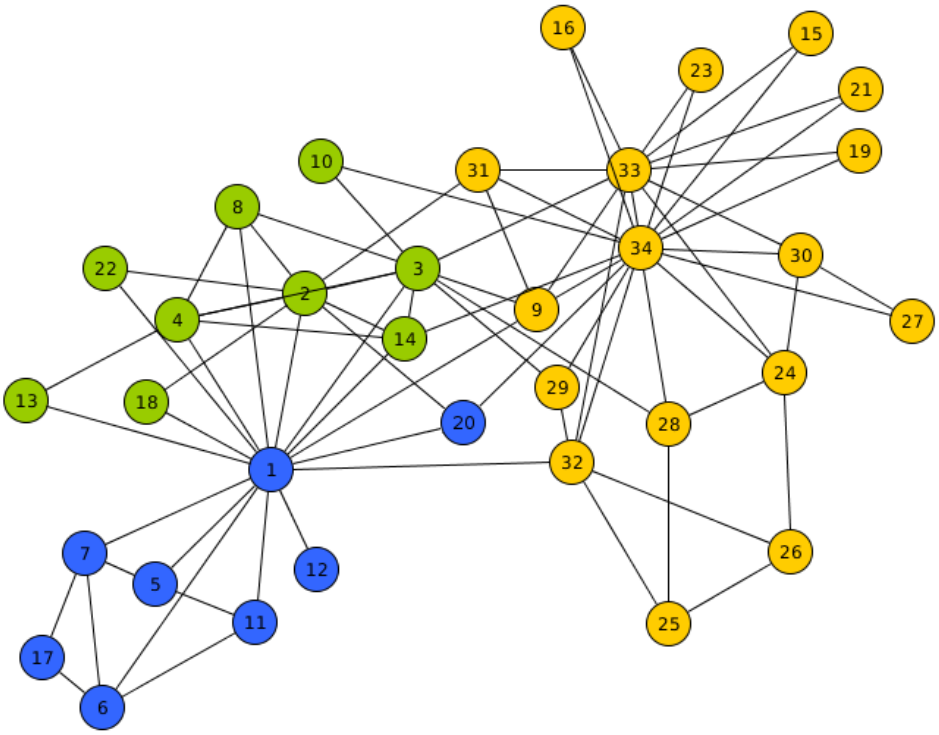

Figura 6.3: Red del club de karate de Zachary con la configuración de *clustering* que encontró el algoritmo de Clauset, Newman y Moore. Q = 0,380671

Tabla 6.6: Cuartiles de la muestra. Tamaño 1000 datos

Modularidad (límite inferior)	**Modularidad (límite superior)**	**Total datos**	**Total datos (%)**
0,424227	0,42846	285	28,5
0,42846	0,431254	307	30,7
0,431254	0,435199	207	20,7
0,435199	0,461703	201	20,1

La mejor solución encontrada corresponde a la descrita en la tabla 6.7, cuya modularidad fue de 0,461703 que supera por 0,081032 la modularidad obtenida con el algoritmo de Clauset, Newman y Moore. En la figura 6.4 se puede ver gráficamente la configuración de *clustering* encontrada, los nodos que tienen dos colores (gradiente) son los que conforman el solapamiento encontrado. Realizamos una comparación similar a la que se hizo en el articulo [Newman, 2006] en la página 4, y si revisamos los nodos solapados y comparamos con la configuración de *clustering* real de la red, este solapamiento tiene sentido. Si no se pudiera comparar con la configuración de *clustering* real este solapamiento da indicios de cómo los nodos se relacionan y de un posible colapso entre las comunidades C1 y C3.

Tabla 6.7: Resultados al aplicar el algoritmo genético a la red de club de karate de Zachary (Mejor solución encontrada)

Propiedad	**Resultado**
Modularidad	0,461703
Cantidad de *clusters*	3
Tamaño de cada *cluster*	C1=13; C2=17; C3=10
Configuración C1 (color azul)	$\{1, 2, 3, 4, 5, 6, 7, 8, 11, 12, 14, 17, 20\}$
Configuracion C2 (color amarillo)	$\{9, 15, 16, 19, 21, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34\}$
Configuracion C3 (color verde)	$\{1, 2, 3, 4, 8, 10, 13, 14, 18, 22\}$

Los resultados de esta prueba permiten concluir que el algoritmo alcanza su propósito de maximizar la modularidad permitiendo los solapamientos entre *clusters*, ya que en toda la muestra, la modularidad presenta un valor superior al valor inicial (el obtenido con el algoritmo de Clauset, Newman y Moore). Sin embargo si observamos la distribución de los valores de la modularidad en la muestra, el último cuartil sólo contiene el 20 % de los datos. Lo anterior evidencia que aunque se supera la modularidad obtenida con el algoritmo de Clauset, Newman y Moore, el algoritmo genético converge a óptimos locales.

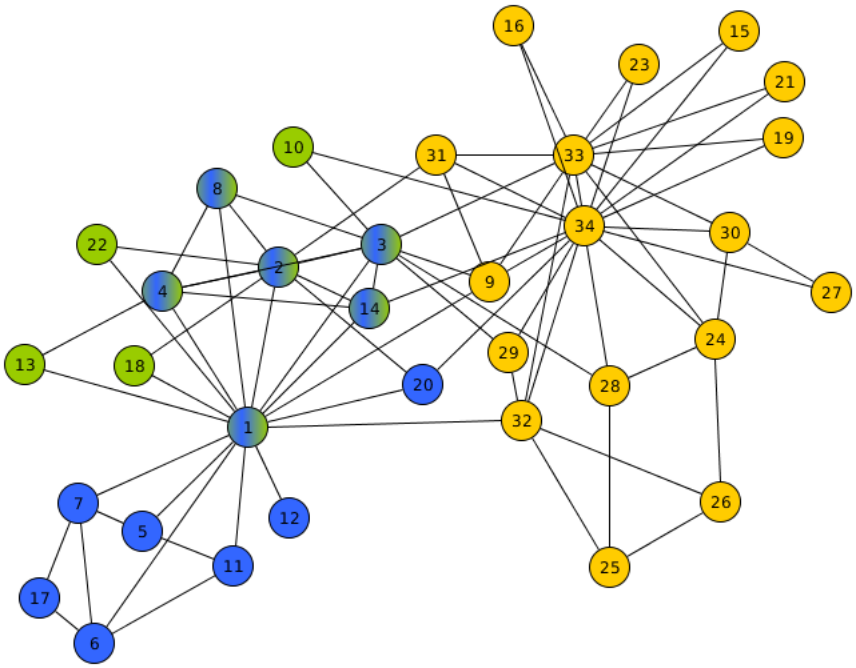

Figura 6.4: Configuración de *clustering* encontrada por el algoritmo genético para la red de club de karate de Zachary. Q = 0,461703

6.4.3. Resultado prueba de comparación del algoritmo con otros algoritmos de *clustering*

Los resultados obtenidos para las redes: club de karate de Zachary y músicos de jazz se pueden ver en la tabla 6.8 y los resultados para las redes: colaboración de científicos, subgrafo de facebook, interacción de proteínas y red eléctrica se pueden ver en la tabla 6.9. AG significa los resultados de este algoritmo genético.

Tabla 6.8: Modularidad obtenida para las redes: club de karate de Zachary y músicos de jazz

Red	**Nodos (N)**	**Aristas (M)**	**GN**	**CNM**	**DA**	**MC**	**BGLL**	**AG**
Zachary	38	78	0.401	0.381	0.419	0.419	0.419	0.469
Músicos de jazz	198	2742	0.405	0.439	0.445	0.442	0.419	0.453

Tabla 6.9: Modularidad obtenida para las redes: colaboración de científicos, subgrafo de facebook, interacción de proteínas y red eléctrica

Red	**Nodos (N)**	**Aristas (M)**	**CNM**	**BGLL**	**AG**
Interacción de proteínas	1458	1948	0.819	0.844	0.823
Red eléctrica	4941	6594	0.934	0.936	0.935
Colaboración de científicos	4158	13422	0.800	0.860	0.803
Red (subgrafo) de facebook	4039	88234	0.777	0.835	0.782

Como podemos observar en las tablas, el algoritmo genético desarrollado presenta los siguientes resultados en términos de la modularidad. Con las redes: club de karate de Zachary y músicos de jazz nuestro algoritmo obtuvo mayor modularidad que el resto de los algoritmos. Mientras que con las redes: colaboración de científicos, subgrafo de facebook, interacción de proteínas y red eléctrica, el algoritmo genético supera el algoritmo de Clauset, Newman y Moore pero no supera el algoritmo de Blondel, Guillaume, Lambiotte, y Lefebvre.

6.5. Discusión de resultados obtenidos

La primer prueba consistió en verificar que el procedimiento de aplicar el algoritmo de Clauset, Newman y Moore fuese correcto en la implementación de la librería *igraph*. Una vez obtenidos los resultados, sólo se evidencia que no se encuentran errores y que se puede confiar en las configuraciones de *clustering* que retorna, para usarlas en los pasos siguientes del algoritmo genético.

En la segunda prueba, prueba algoritmo genético, los resultados muestran que el algoritmo alcanza una modularidad más alta que la obtenida con el algoritmo base, y lo hace siempre manteniendo el número de *clusters* inicial, introduciendo solapamientos entre éstos. Esto es debido a que, de acuerdo a la codificación de la configuración de *clustering*, la matriz X, para eliminar un *cluster* debe primero introducir un solapamiento por cada uno de los nodos y luego sí eliminar el *cluster*, es decir, poner en 0 los valores de la fila correspondiente al *cluster* en la matriz X. Sin embargo, dada la combinatoria y la cantidad de experimentos que se realizan (número de generaciones y porcentaje de mutaciones), son los solapamientos los que van ganando lugar en la población, pues aumentan el valor de la modularidad inicial, mientras que eliminar un solapamiento que incrementaba el valor de la modularidad supone un cromosoma que pierde valor en su función de aptitud. Por tanto si éste se enfrentara contra otro cromosoma, es posible que pierda su lugar en la población, y esto a su vez va limitando la posibilidad de eliminar un *cluster* de la solución.

Por otro lado, la preferencia por los solapamientos está justificada en la ecuación de la modularidad:

$$Q = \sum_{v} (e_{vv} - a_v^2) \tag{6.1}$$

Aunque no todos los solapamientos contribuyen a aumentar la modularidad [Takaki et al., 2007], los que lo hacen incrementan uno de los términos de la ecuación y disminuyen otros, o aumentan todos los términos que están involucrados en el solapamiento en el mejor de los casos. Con "términos"nos referimos a los ΔQ_v, donde su valor corresponde al aporte que le hace el *cluster* v a la modularidad. En el caso donde sólo un término aumenta su valor después del solapamiento, éste debe ser lo suficientemente significativo para mantener o incrementar el valor de la modularidad.

Al inicializar la población de cromosomas sólo uno no contiene solapamiento, el que tiene la configuración de *clustering* retornada por el algoritmo de Clauset, Newman y Moore. Esto condiciona a que la presión selectiva beneficie aquellos solapamientos que incrementan los términos de la ecuación de la modularidad, que es el propósito del algoritmo.

Finalmente, en la prueba de comparación del algoritmo con otros algoritmos de *clustering*, el algoritmo presenta mejores resultados en el primer conjunto de redes que en el segundo, es decir, aumenta significativamente[3] los valores de las modularidades iniciales. Esto se debe a que, en el primer conjunto donde las redes tienen menos de 1000 nodos y menos de 3000 aristas, estos valores describen un espacio de búsqueda mucho más reducido que las redes del segundo conjunto ($1458 \leq N \leq 4039$ y $1948 \leq M \leq 88234$), entonces el algoritmo logra explorar más las posibles configuraciones de *clustering* con solapamiento en el número dado de generaciones. Con el segundo conjunto de redes de prueba es recomendable realizar un ajuste en los parámetros del algoritmo y ejecutar más veces el algoritmo para obtener mejores resultados.

[3] los valores iniciales de la modularidad fueron superados en más de 0,006 puntos

Capítulo 7

Conclusiones y trabajos futuros

7.1. Conclusiones

1. Identificar solapamientos permite obtener información que con el *clustering* clásico (sin permitir solapamientos) no se obtendría. Un ejemplo fue mostrado en la prueba del algoritmo genético, donde para la red del club de karate de Zachary se obtuvo un solapamiento entre dos *clusters* de los tres que constituyen la configuracion de *clustering*, que involucra los nodos $\{1, 2, 3, 4, 8, 14\}$. Este solapamiento da indicios de una amistad entre las personas identificadas con esos números, y si se revisa la configuración de *clustering* real (la red está dividida en dos grupos) de esta red se observa que estas personas pertenecen al mismo grupo.

2. Realizar una descripción y especificación de las redes permitió establecer un subconjunto en el cual el algoritmo puede ser aplicado, puesto que, se indican los límites en el tamaño de la red y la naturaleza de las aristas. Con estas especificaciones quedaron por fuera de consideración, entre algunas, redes a gran escala con $N \gg 1000$ y redes dirigidas como las redes de citación de trabajos.

3. Usar la modularidad como métrica que evalúa la configuración de *clustering* ofrece como ventaja que, para hallar su valor se usa información que está disponible en las estructuras de datos, con una complejidad promedio de $\Theta(1)$ al ser consultadas, lo cual es importante en este trabajo ya que al ser un algoritmo genético y la modularidad ser su función de aptitud, ésta debe calcularse $K \times G$ veces, siendo K el tamaño del *mating pool* y G número de generaciones.

4. La modularidad es una métrica que no identifica los solapamientos, es decir, no penaliza cuando un solapamiento no es coherente con la topología de la red. Por lo anterior, se debe verificar que los solapamientos sean coherentes con la topología de la red, lo cual incrementa la complejidad del algoritmo. En este algoritmo esta situación afecta la mutación, pues no se pueden elegir coordenadas de la matriz X de manera aleatoria sin antes revisar que el cambio del *bit* mantenga la configuración coherente con la topología de la red.

5. Este algoritmo genético se presenta como una buena estrategia de búsqueda de solapamiento en *clusters* cuando más que el tiempo de ejecución del algoritmo lo importante sea obtener una buena solución en términos de la modularidad y cuando el tamaño de las redes sea pequeño. Puesto que dada la forma en que está planteada la mutación se

permite explorar varias configuraciones. Sin embargo, en términos de eficiencia y tiempo de ejecución la complejidad computacional del algoritmo no representa una mejora comparado con otras propuestas existentes.

6. No se logró definir una forma geométrica diferente al óvalo a partir de la distribución de los nodos en los *clusters*, pues resulta subjetivo, ya que dependerá de la visualización y el nodo que se tome como nodo central. Con esto se responde la primera pregunta del planteamiento del problema.

7. Con el algoritmo desarrollado se alcanza el propósito de maximizar la modularidad respecto a la inicial (la obtenida con el algoritmo de Clauset, Newman y Moore), permitiendo los solapamientos entre *clusters*, ya que como se muestra en los resultados de las pruebas del algoritmo genético, en toda la muestra obtiene un valor de modularidad superior al inicial.

8. Aunque se logra obtener mejores valores de modularidad respecto a los que se obtienen con el algoritmo de Clauset, Newman y Moore, el algoritmo genético converge a óptimos locales y esto se observa en la tabla 6.6 donde sólo el 20,1 % de la muestra obtuvo las modularidades más altas.

9. Respecto a la modularidad, el algoritmo presenta mejores resultados en el primer conjunto de redes que en el segundo, ya que el primer conjunto de redes describe un espacio de búsqueda (tamaño de las redes) mucho más pequeño que el segundo, y en este sentido explora más solapamientos con un mayor número de nodos, lo cual le permite obtener mejores valores para la modularidad.

10. En las pruebas, el algoritmo genético muestra una tendencia a no cambiar el número de *clusters* inicial, y a introducir los solapamientos entre los *clusters*. Esto es debido a que para eliminar un *cluster* es necesario introducir solapamientos para aquellos nodos que pertenecen solamente a dicho *cluster*, sin embargo, los casos que corresponden a solapamientos que incrementen la modularidad le darán al cromosoma más probabilidad de mantenerse en la población, conservando el solapamiento y limitando la posibilidad de eliminar un *cluster* de la solución.

7.2. Trabajos futuros

1. Investigar o modificar la métrica que se está empleando para medir la calidad de los *clusters*, puesto que aunque la modularidad es una métrica que permite evaluar qué tan bien configurados están los *clusters*, no identifica los solapamientos. Se debe investigar una métrica que penalice los solapamientos incoherentes con la topología de la red.

2. Adaptar el modelo realizando los ajustes necesarios, tanto en la métrica, como en los pasos del algoritmo para ampliar los tipos de redes a las cuales se les puede aplicar. Un cambio es permitir aristas dirigidas en la red.

3. Con el fin de reducir la complejidad computacional del algoritmo, se debe analizar algoritmos alternativos como el propuesto en [Blondel et al., 2008], para establecerlo como reemplazo del algoritmo de [Clauset et al., 2004]. También realizar el cálculo de la modularidad incremental, es decir, reformular las ecuaciones para que se pueda calcular teniendo el valor de la modularidad inicial y el incremento o decremento que signifique cambiar un *bit* de la matriz X.

4. Sería interesante realizar un análisis donde se verifique el impacto que tiene la cantidad de *bits* que se mutan en la matriz X.

5. Para seguir explorando el problema de la búsqueda de solapamiento en *clusters* usando un algoritmo evolutivo, se debe adaptar el modelo para no usar un algoritmo de *clustering* base y modificar la configuración del cromosoma para que permita explorar el número de *clusters* que maximizan la modularidad.

Capítulo 8

Bibliografía

[Ajith Abraham and Konar, 2006] Ajith Abraham, S. D. and Konar, A. (2006). Document clustering using differential evolution. *IEEE Congress on Evolutionary Computation*.

[Bastian et al., 2009] Bastian, M., Heymann, S., and Jacomy, M. (2009). Gephi: An open source software for exploring and manipulating networks. In Adar, E., Hurst, M., Finin, T., Glance, N. S., Nicolov, N., and Tseng, B. L., editors, *ICWSM*. The AAAI Press.

[Baños, 2012] Baños, A. G. (2012). Computación evolutiva: Notas de clase. Technical report, Universidad del Valle, Colombia.

[Blondel et al., 2008] Blondel, V., Guillaume, J., Lambiotte, R., and Mech, E. (2008). Fast unfolding of communities in large networks. *J. Stat. Mech*, page P10008.

[Brandes et al., 2002] Brandes, U., Eiglsperger, M., Herman, I., Himsolt, M., and Marshall, M. S. (2002). GraphML Progress Report: Structural Layer Proposal. In *Proceedings of the 9th International Symposium Graph Drawing (GD '01) LNCS 2265*, pages 501–512. Springer-Verlag.

[Cai et al., 2011] Cai, Y., Shi, C., Dong, Y., Ke, Q., and Wu, B. (2011). A novel genetic algorithm for overlapping community detection. In Tang, J., King, I., Chen, L., and Wang, J., editors, *ADMA (1)*, volume 7120 of *Lecture Notes in Computer Science*, pages 97–108. Springer.

[Clauset et al., 2004] Clauset, A., Newman, M. E. J., and Moore, C. (2004). Finding community structure in very large networks. *Physical Review E*, 70:066111.

[Csardi and Nepusz, 2006] Csardi, G. and Nepusz, T. (2006). The igraph software package for complex network research. *InterJournal*, Complex Systems:1695.

[Duch and Arenas, 2005] Duch, J. and Arenas, A. (2005). Community detection in complex networks using extremal optimization. *Physical Review E*, 72:027104.

[Fortunato, 2010] Fortunato, S. (2010). Community detection in graphs. *Physics Reports*, 486:75–174.

[Girvan and Newman, 2002] Girvan, M. and Newman, M. E. J. (2002). Community structure in social and biological networks. *PNAS*, 99(12):7821–7826.

[Gleiser and Danon, 2003] Gleiser, P. M. and Danon, L. (2003). Community structure in jazz. *Advances in Complex Systems*, 6(4):565–574.

[Gog et al., 2007] Gog, A., Dumitrescu, D., and Hirsbrunner, B. (2007). Community detection in complex networks using collaborative evolutionary algorithms. In e Costa, F. A., Rocha, L. M., Costa, E., Harvey, I., and Coutinho, A., editors, *ECAL*, volume 4648 of *Lecture Notes in Computer Science*, pages 886–894. Springer.

[Han and Kamber, 2006] Han, J. and Kamber, M. (2006). *Data Mining. Concepts and Techniques*. Morgan Kaufmann, 2nd ed. edition.

[Jeong et al., 2001] Jeong, H., Mason, S., Barabasi, A., and Oltvai, Z. (2001). Lethality and centrality in protein networks. *Nature*, 411:41–42.

[Leskovec et al., 2007] Leskovec, J., Kleinberg, J. M., and Faloutsos, C. (2007). Graph evolution: Densification and shrinking diameters. *TKDD*, 1(1).

[McAuley and Leskovec, 2012] McAuley, J. J. and Leskovec, J. (2012). Learning to discover social circles in ego networks. In Bartlett, P. L., Pereira, F. C. N., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *NIPS*, pages 548–556.

[Newman, 2006] Newman, M. (2006). Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582.

[Newman, 2002] Newman, M. E. J. (2002). Assortative mixing in networks. *Phys. Rev. Lett.*, 89:208701.

[Newman, 2010] Newman, M. E. J. (2010). *Networks: An Introduction*. Oxford University Press, New York.

[Pan et al., 2003] Pan, H., Zhu, J., and Han, D. (2003). Genetic Algorithms Applied to Multi-Class Clustering for Gene Expression Data. *Genomics Proteomics Bioinformatics*, 1(4):279–287.

[Ripon and Siddique, 2009] Ripon, K. S. N. and Siddique, M. N. H. (2009). Evolutionary multi-objective clustering for overlapping clusters detection. In *IEEE Congress on Evolutionary Computation*, pages 976–982. IEEE.

[Takaki et al., 2007] Takaki, M., Tamura, K., Mori, Y., and Kitakami, H. (2007). A extraction method of overlapping cluster based on network structure analysis. In *IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology - Workshops*, pages 212–216. IEEE.

[van Heesch, 2008] van Heesch, D. (2008). Doxygen: Source code documentation generator tool.

[Watts and Strogatz, 1998] Watts, D. J. and Strogatz, S. H. (1998). Collective dynamics of "small-world" networks. *Nature*, 393:440–442.

[yWorks GmbH,] yWorks GmbH. yed software. zipped yed jar file. for experts only. requires oracle's jre 6 or higher, version 3.11.1.

[Zachary, 1977] Zachary, W. (1977). An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33:452–473.

Anexos

Tabla de soluciones encontradas con el algoritmo evolutivo para la red de karate de Zachary

Tabla 8.1: Soluciones encontradas en la muestra

Modularidad	**Configuración de *clustering***	**Total**
0,424227	C1 = $\{1, 5, 6, 7, 11, 12, 13, 17, 20\}$ C2 = $\{9, 15, 16, 19, 21, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34\}$ C3 = $\{1, 2, 3, 4, 8, 10, 13, 14, 18, 22\}$	22
0,424227	C1 = $\{1, 5, 6, 7, 11, 12, 17, 18, 20\}$ C2 = $\{9, 15, 16, 19, 21, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34\}$ C3 = $\{1, 2, 3, 4, 8, 10, 13, 14, 18, 22\}$	22
0,424227	C1 = $\{1, 5, 6, 7, 11, 12, 17, 20, 22\}$ C2 = $\{9, 15, 16, 19, 21, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34\}$ C3 = $\{1, 2, 3, 4, 8, 10, 13, 14, 18, 22\}$	16
0,425008	C1 = $\{1, 5, 6, 7, 11, 12, 13, 17, 20\}$ C2 = $\{9, 15, 16, 19, 21, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34\}$ C3 = $\{1, 2, 3, 4, 8, 9, 10, 13, 14, 18, 22\}$	4
0,425296	C1 = $\{1, 2, 5, 6, 7, 11, 12, 17, 18, 20\}$ C2 = $\{3, 9, 15, 16, 19, 21, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34\}$ C3 = $\{1, 2, 3, 4, 8, 10, 13, 14, 18, 20, 22\}$	3
0,42546	C1 = $\{1, 5, 6, 7, 11, 12, 14, 17, 20\}$ C2 = $\{9, 15, 16, 19, 21, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34\}$ C3 = $\{1, 2, 3, 4, 8, 10, 13, 14, 18, 20, 22\}$	6
0,425501	C1 = $\{1, 5, 6, 7, 11, 12, 17, 20\}$ C2 = $\{9, 15, 16, 19, 21, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34\}$ C3 = $\{1, 2, 3, 4, 8, 10, 12, 13, 14, 18, 22\}$	1
0,427309	C1 = $\{1, 2, 5, 6, 7, 11, 12, 14, 17, 20\}$ C2 = $\{9, 15, 16, 19, 21, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34\}$ C3 = $\{1, 2, 3, 4, 8, 10, 12, 13, 14, 18, 22\}$	6
0,427844	C1 = $\{1, 2, 5, 6, 7, 8, 11, 12, 14, 17, 20\}$ C2 = $\{9, 15, 16, 19, 21, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34\}$ C3 = $\{1, 2, 3, 4, 8, 10, 13, 14, 18, 22\}$	12
0,428008	C1 = $\{1, 5, 6, 7, 11, 12, 17, 20\}$ C2 = $\{9, 15, 16, 19, 21, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34\}$	16
Continua en la página siguiente		

Tabla 8.1 – Viene de la página anterior

Modularidad	Configuración de *clustering*	Total
	C3 = {1, 2, 3, 4, 8, 9, 10, 13, 14, 18, 20, 22}	
0,428296	C1 = {1, 5, 6, 7, 11, 12, 17, 20} C2 = {9, 10, 15, 16, 19, 21, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34} C3 = {1, 2, 3, 4, 8, 10, 13, 14, 18, 20, 22}	5
0,42846	C1 = {1, 5, 6, 7, 11, 12, 17, 20} C2 = {9, 15, 16, 19, 21, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34} C3 = {1, 2, 3, 4, 8, 10, 13, 14, 18, 20, 22}	172
0,428789	C1 = {1, 5, 6, 7, 8, 11, 12, 17, 20} C2 = {9, 15, 16, 19, 21, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34} C3 = {1, 2, 3, 4, 8, 10, 13, 14, 18, 20, 22}	6
0,429282	C1 = {12, 1, 5, 17, 6, 18, 7, 20, 11} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 2, 3, 4, 31, 8, 9, 10, 13, 14, 18, 22}	4
0,429405	C1 = {12, 1, 14, 4, 5, 17, 6, 7, 20, 11} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 13, 2, 14, 3, 4, 18, 8, 20, 10, 22}	6
0,429405	C1 = {12, 1, 2, 5, 17, 6, 7, 20, 22, 11} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 13, 2, 14, 3, 4, 29, 18, 8, 10, 22}	6
0,429405	C1 = {12, 1, 2, 13, 5, 17, 6, 18, 7, 20, 11} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 20, 21, 23} C3 = {1, 13, 2, 14, 3, 4, 18, 8, 10, 22}	6
0,429652	C1 = {12, 1, 13, 4, 5, 17, 6, 7, 20, 11} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 13, 2, 14, 3, 4, 18, 8, 10, 22}	23
0,430473	C1 = {1, 12, 13, 5, 17, 6, 7, 20, 22, 11} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 10, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 13, 2, 14, 3, 4, 18, 8, 10, 22}	12
0,430638	C1 = {12, 1, 13, 5, 17, 6, 7, 20, 22, 11} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 13, 2, 14, 3, 4, 18, 8, 10, 22}	17
0,430638	C1 = {12, 1, 5, 17, 6, 18, 7, 20, 22, 11} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 13, 2, 14, 3, 4, 18, 8, 10, 22}	6
0,430638	C1 = {12, 1, 13, 5, 17, 6, 18, 7, 20, 11} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 13, 2, 14, 3, 4, 18, 8, 10, 22}	5
0,431254	C1 = {12, 1, 2, 5, 17, 6, 7, 20, 22, 11} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 13, 2, 14, 3, 4, 18, 8, 10, 22}	105
0,431254	C1 = {12, 1, 2, 5, 17, 6, 18, 7, 20, 11} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 13, 2, 14, 3, 4, 18, 8, 10, 22}	111
0,43146	C1 = {1, 2, 5, 6, 7, 8, 9, 11, 12, 17, 18, 20}	6

Continua en la página siguiente

Tabla 8.1 – Viene de la página anterior

Modularidad	Configuración de *clustering*	Total
	C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 10, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 13, 2, 14, 3, 4, 18, 8, 10, 22}	
0,431624	C1 = {12, 1, 2, 14, 4, 5, 17, 6, 7, 20, 11} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 13, 2, 14, 3, 4, 18, 8, 10, 22}	5
0,432035	C1 = {12, 1, 2, 5, 17, 6, 7, 20, 22, 11} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 13, 2, 14, 3, 4, 18, 8, 9, 10, 22}	17
0,432035	C1 = {12, 1, 2, 5, 17, 6, 18, 7, 20, 11} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 13, 2, 14, 3, 4, 18, 8, 9, 10, 22}	7
0,43224	C1 = {1, 12, 13, 5, 17, 6, 7, 20, 11} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {12, 1, 13, 2, 14, 3, 4, 18, 8, 10, 22}	6
0,43224	C1 = {12, 1, 13, 2, 4, 5, 17, 6, 7, 20, 11} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 2, 13, 14, 3, 4, 18, 8, 10, 22}	10
0,43224	C1 = {1, 12, 2, 4, 5, 17, 6, 18, 7, 20, 11} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 13, 2, 14, 3, 4, 18, 8, 10, 22}	5
0,43224	C1 = {12, 1, 5, 17, 6, 18, 7, 20, 11} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {12, 1, 13, 2, 14, 3, 4, 18, 8, 10, 22}	4
0,432405	C1 = {12, 1, 2, 14, 4, 5, 17, 6, 7, 20, 11} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 13, 2, 14, 3, 4, 18, 8, 9, 10, 22}	7
0,43261	C1 = {112, 13, 5, 17, 6, 7, 20, 11} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 2, 3, 4, 29, 8, 10, 13, 14, 18, 20, 22}	6
0,432857	C1 = {12, 1, 2, 14, 5, 17, 6, 7, 20, 22, 11} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 20, 21, 23} C3 = {1, 2, 3, 4, 6, 8, 10, 11, 13, 14, 18}	3
0,433226	C1 = {1, 2, 3, 5, 6, 7, 8, 11, 12, 17, 18, 20} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 13, 2, 3, 14, 4, 18, 8, 10, 22}	6
0,434336	C1 = {1, 12, 13, 14, 4, 5, 17, 6, 7, 20, 11} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 13, 2, 14, 3, 4, 5, 18, 8, 10, 22}	6
0,434336	C1 = {12, 1, 2, 5, 17, 6, 7, 8, 20, 11} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 13, 2, 14, 3, 4, 18, 8, 20, 10, 22}	16
0,434747	C1 = {12, 1, 13, 5, 17, 6, 7, 20, 11} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 2, 3, 4, 8, 9, 10, 13, 14, 18, 20, 22}	5

Continua en la página siguiente

Tabla 8.1 – Viene de la página anterior

Modularidad	Configuración de *clustering*	Total
0,435035	C1 = {12, 1, 13, 5, 17, 6, 7, 20, 11} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 10, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 13, 2, 14, 3, 4, 18, 8, 20, 10, 22}	4
0,435076	C1 = {12, 1, 13, 4, 5, 17, 6, 7, 20, 22, 11} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 13, 2, 14, 3, 4, 18, 8, 10, 22}	18
0,435076	C1 = {12, 1, 13, 4, 5, 17, 6, 18, 7, 20, 11} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 13, 2, 14, 3, 4, 18, 8, 10, 22}	9
0,435199	C1 = {12, 1, 5, 17, 6, 18, 7, 20, 11} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 13, 2, 14, 3, 4, 18, 8, 20, 10, 22}	49
0,435199	C1 = {12, 1, 13, 5, 17, 6, 7, 20, 11} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 13, 2, 14, 3, 4, 18, 8, 20, 10, 22}	18
0,436185	C1 = {1, 12, 2, 4, 5, 17, 6, 7, 8, 20, 11} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 13, 2, 14, 3, 4, 18, 8, 10, 22}	16
0,436185	C1 = {12, 1, 13, 14, 4, 5, 17, 6, 7, 20, 11} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 13, 2, 14, 3, 4, 18, 8, 10, 22}	6
0,436185	C1 = {12, 1, 13, 2, 5, 17, 6, 18, 7, 20, 11} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 13, 2, 14, 3, 4, 18, 8, 10, 22}	10
0,436555	C1 = {12, 1, 2, 14, 5, 17, 6, 18, 7, 20, 11} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 13, 2, 14, 3, 4, 18, 8, 10, 22}	18
0,436555	C1 = {12, 1, 2, 14, 5, 17, 6, 7, 20, 22, 11} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 13, 2, 14, 3, 4, 18, 8, 10, 22}	10
0,438034	C1 = {12, 1, 13, 4, 5, 17, 6, 7, 20, 11} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 2, 3, 4, 5, 8, 10, 11, 13, 14, 18, 22}	6
0,438034	C1 = {12, 1, 5, 17, 6, 18, 7, 20, 11} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 2, 3, 4, 31, 8, 9, 10, 13, 14, 18, 20, 22}	6
0,440171	C1 = {12, 1, 13, 4, 5, 17, 6, 7, 8, 20, 11} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 13, 2, 14, 3, 4, 18, 8, 10, 22}	12
0,440623	C1 = {12, 1, 13, 4, 5, 17, 6, 7, 20, 11} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 13, 2, 14, 3, 4, 18, 8, 20, 10, 22}	4
0,440787	C1 = {12, 1, 2, 5, 17, 6, 7, 8, 20, 22, 11} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23}	12

Continua en la página siguiente

Tabla 8.1 – Viene de la página anterior

Modularidad	Configuración de *clustering*	Total
	C3 = {1, 13, 2, 14, 3, 4, 18, 8, 10, 22}	
0,440787	C1 = {12, 1, 2, 5, 17, 6, 18, 7, 8, 20, 11} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 13, 2, 14, 3, 4, 18, 8, 10, 22}	24
0,441609	C1 = {12, 1, 13, 5, 17, 6, 7, 20, 22, 11} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 13, 2, 14, 3, 4, 18, 8, 20, 10, 22}	5
0,442226	C1 = {12, 1, 2, 5, 17, 6, 7, 20, 22, 11} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 13, 2, 14, 3, 4, 18, 8, 20, 10, 22}	27
0,444444	C1 = {1, 2, 5, 6, 7, 8, 11, 12, 14, 17, 18, 20} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 13, 2, 14, 3, 4, 18, 8, 10, 22}	5
0,447896	C1 = {1, 2, 4, 5, 6, 7, 11, 12, 13, 14, 17, 20} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 13, 2, 14, 3, 4, 18, 8, 10, 22}	6
0,449006	C1 = {12, 1, 2, 5, 17, 6, 18, 7, 20, 22, 11} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 13, 2, 14, 3, 4, 18, 8, 10, 22}	11
0,451019	C1 = {1, 2, 4, 5, 6, 7, 8, 11, 12, 14, 17, 20} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 13, 2, 14, 3, 4, 18, 8, 10, 22}	6
0,451759	C1 = {12, 1, 2, 5, 17, 6, 18, 7, 8, 20, 11} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 13, 2, 14, 3, 4, 18, 8, 20, 10, 22}	7
0,452622	C1 = {1, 2, 4, 5, 6, 7, 8, 11, 12, 17, 18, 20} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 13, 2, 14, 3, 4, 18, 8, 10, 22}	5
0,461703	C1 = {1, 2, 3, 4, 5, 6, 7, 8, 11, 12, 14, 17, 20} C2 = {24, 25, 26, 27, 28, 29, 30, 31, 32, 9, 33, 34, 15, 16, 19, 21, 23} C3 = {1, 2, 13, 14, 3, 4, 18, 8, 10, 22}	5

Diagrama de clases del prototipo desarrollado

Como se describió en los capítulos 5 y 3, se desarrolló un prototipo de aplicación con el fin de realizar las pruebas. En la figura 8.1 se muestra el diagrama de clases del prototipo desarrollado. En la figura 8.2 se muestra la clase Algoritmo que contiene todos los procedimientos para llevar a cabo el algoritmo descrito en el capítulo 4. En la figura 8.3 se muestra la clase Cromosoma que contiene la configuración del cromosoma y los procedimientos necesarios para mutar y obtener el valor de la función de aptitud descritos en el capítulo 4. En la figura 8.4 se muestra la clase LectoraArchivo, esta clase se encarga de leer el archivo de entrada de grafo, y de crear el archivo de salida con la solución encontrada. En la figura 8.5 se muestra la clase MainWindow, la clase de interfaz gráfica que despliega la interfaz mostrada en la figura 5.1.

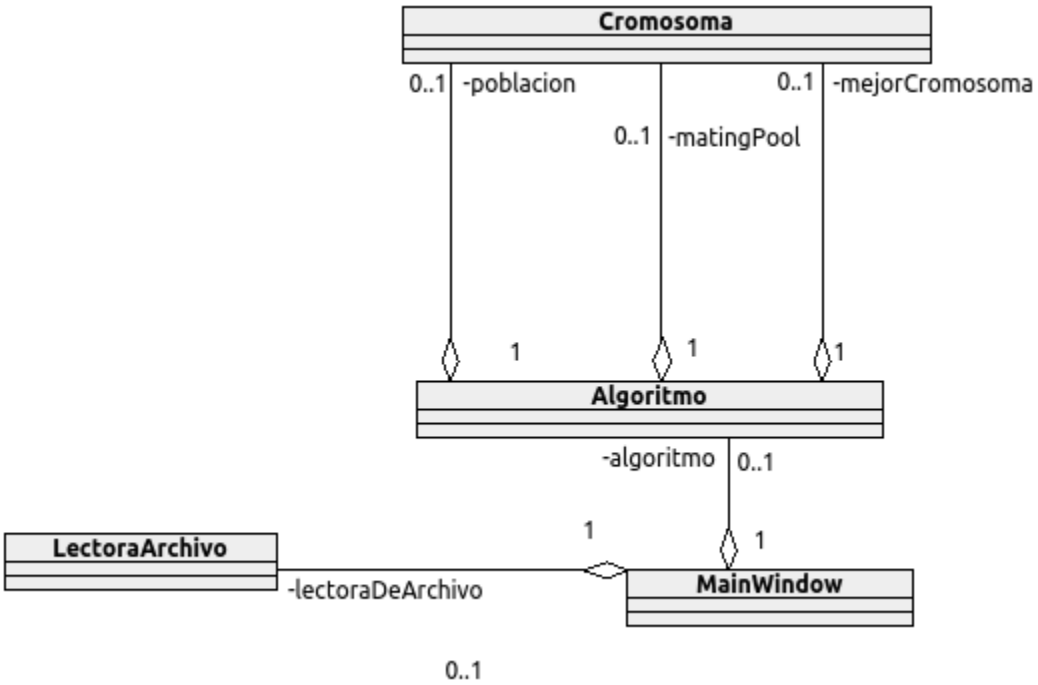

Figura 8.1: Diagrama de clases del prototipo desarrollado

Algoritmo
- nodos : int - aristas : int - individuos : int - generaciones : int - poblacion : Cromosoma** - matingPool : Cromosoma** - mejorIndicePoblacion : int - mejorModularidad : double - mejorCromosoma : Cromosoma* - tamMatingPool : int - porcentajeMatingPool : double - mutarDosBits : bool - grafo : igraph_t - modularidadInicial : double - modularidadFinal : double - cantClustersInicial : int - cantCambios : int - tamanioClustersInicial : igraph_vector_t - membresiaInicial : igraph_vector_t - matrizXInicial : vector< vector < bool > > - configuracionInicial : vector< unordered_set < int > > - listaAdyacencia : vector< unordered_set < int > >
- inicializarPoblacion() - seleccionar() - reproducir() - clusteringNewman() - inicializarMatrizX() - inicializarConfiguracionInicial() - construirListaAdyacencia() + Algoritmo() + Algoritmo(grafo : igraph_t&, nodos : int, aristas : int, individuos : int, generaciones : int, porcentajeMatingPool : double, mutarDosBits : bool) + ~ Algoritmo() + algoritmo() + getModularidad() : double + getConfiguracionFinal() : vector< unordered_set < int > > + getCantCambios() : int

Figura 8.2: Clase Algoritmo

Figura 8.3: Clase Cromosoma

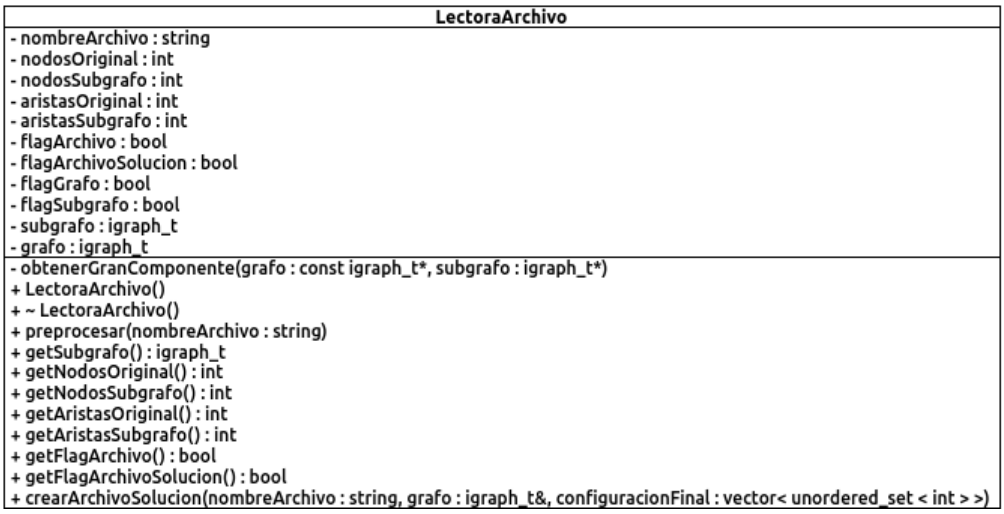

Figura 8.4: Clase LectoraArchivo

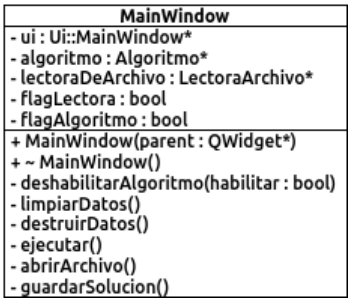

Figura 8.5: Clase MainWindow