UNIVERSIDAD
DE LA REPUBLICA
URUGUAY

FACULTAD DE INGENIERIA

A fast Genetic Algorithm for the Maximum Cut-Clique Problem

Giovanna Elizabeth Fortez Hitateguy

Programa de Posgrado en Investigación de Operaciones
Facultad de Ingeniería
Universidad de la República

Montevideo – Uruguay
Noviembre de 2019

UNIVERSIDAD
DE LA REPUBLICA
URUGUAY

FACULTAD DE INGENIERIA

A fast Genetic Algorithm for the Maximum Cut-Clique Problem

Giovanna Elizabeth Fortez Hitateguy

Tesis de Maestría presentada al Programa de Posgrado en Investigación de Operaciones, Facultad de Ingeniería de la Universidad de la República, como parte de los requisitos necesarios para la obtención del título de Magíster en Investigación de Operaciones.

Directores:
Dr. Ing. Franco Robledo
Dr. Ing. Pablo Romero

Director académico:
Prof. Ing. Omar Viera

Montevideo – Uruguay
Noviembre de 2019

Fortez Hitateguy, Giovanna Elizabeth

A fast Genetic Algorithm for the Maximum Cut-Clique Problem / Giovanna Elizabeth Fortez Hitateguy. - Montevideo: Universidad de la República, Facultad de Ingeniería, 2019.

XII, 63 p.: il.; 29, 7cm.

Directores:

Franco Robledo

Pablo Romero

Director académico:

Omar Viera

Tesis de Maestría – Universidad de la República, Programa en Investigación de Operaciones, 2019.

Referencias bibliográficas: p. 57 – 61.

1. Optimización, 2. Máximo Clique-Corte, 3. Metaheurísticas, 4. Algoritmos Genéticos. I. Robledo, Franco, Romero, Pablo, . II. Universidad de la República, Programa de Posgrado en Investigación de Operaciones. III. Título.

INTEGRANTES DEL TRIBUNAL DE DEFENSA DE TESIS

_____

Dr. Ing. Jorge Pérez Zerpa IET-UdelaR

_____

Dr. Ing. Santiago Iturriaga INCO-UdelaR

_____

Dr. Ing. Juan Kalemkerián IMERL-UdelaR

Montevideo – Uruguay
Noviembre de 2019

A mi padres: Livia y Héctor.

Agradecimientos

En un emprendimiento de esta magnitud, existe mucha gente que aporta de diversas formas y con diferentes intensidades. En primer lugar, quisiera expresar mi gratitud hacia mis tutores: Omar Viera, Franco Robledo y Pablo Romero por sus oportunos consejos, paciencia y dedicación. Por darme ánimo para continuar por el camino de la investigación y facilitarme las herramientas para lograr un buen trabajo.

A su vez, quisiera destacar el aporte del Prof. Sergio Nesmachnow, quien me enseñó la técnica utilizada en la resolución, exigiendo mis capacidades analíticas al límite de forma de priorizar la calidad de los resultados obtenidos.

Finalmente, me gustaría destacar que ésta tesis no hubiera sido posible sin una estructura familiar de sustento. Gracias a Andrés, por su valentía, compromiso y empatía. Infinitas gracias a nuestros pequeños niños: Candelaria, Eugenia y Nicolás, quienes han sido engranajes fundamentales durante todo el proceso creativo, aún sin tener conciencia de ello.

RESUMEN

En esta tesis se presenta un problema reciente de teoría de grafos conocido como *Máximo Clique-Corte* o *MCC*. Dado un grafo simple, se desea hallar un subgrafo completo tal que el corte inducido por sus nodos tenga máximo cardinal. Este problema combinatorio fue introducido por P. Martins en 2012, y encuentra aplicaciones en el Análisis de Mercados, donde interesa la correlación de artículos de venta. La correspondiente versión de decisión del *MCC* pertenece a la categoría de problemas $\mathcal{NP}$-Completos. Como consecuencia, los métodos exactos resultan computacionalmente prohibitivos para grafos de gran tamaño. En la literatura científica se dispone de heurísticas previas que explotan aleatoriedad con distintas estructuras de vecindad, como GRASP/VND o ILS. En esta tesis se presenta una nueva solución basada en Algoritmos Genéticos. Un estudio comparativo comprueba que la propuesta es altamente competitiva con trabajos anteriores, siendo su mayor fortaleza la eficiencia computacional.

Palabras claves:
Optimización, Máximo Clique-Corte, Metaheurísticas, Algoritmos Genéticos.

ABSTRACT

In this work, we present a recently introduced problem, enmarked in graph theory, and catalogued as $Maximum\ Cut\text{-}Clique$, MCC. Given a simple graph $\mathcal{G}$, the objective is to find a complete subgraph where the cut induced by its nodes has the maximum cardinality. This combinatorial problem was introduced by P. Martins in 2012 and find its application in Market Basket Analysis, where data mining technique is concentrated on correlations between products. The corresponding decision problem of MCC belongs to $\mathcal{NP}$-Complete class. For that, the exact methods are computationally time prohibitive for large graphs. In the scientific literature, there are metaheuristics that combine ramdomness with distinct neighborhood structures as GRASP/VND or ILS. In this thesis a new solution based on Genetic Algorithms is developed while a comparative analysis shows that this work is highly competitive with state-of-art resolution strategies, being its greatest strength computational efficiency.

Keywords:
Optimization, Maximum Cut-Clique, Metaheuristics, Genetic Algorithms.

Lista de figuras

Lista de tablas

Tabla de contenidos

Capítulo 1

Introducción

1.1. Motivación

El problema combinatorio correspondiente a hallar el clique máximo en un grafo es de gran interés para la comunidad científica, dadas sus diversas aplicaciones [5]. Su correspondiente versión de decisión pertenece a la clase de problemas $\mathcal{NP}$-Completo, y por lo tanto posee gran relevancia teórica. Más precisamente, no solo es $\mathcal{NP}$-Completo sino que existe una cota de inaproximabilidad. De hecho figura entre los primeros 21 problemas de decisión de la lista elaborada por Richard Karp [28].

Esta es una de las razones por la cual existe una extensa bibliografía donde se han presentado varios modelos, métodos exactos y métodos guiados por heurísticas para su resolución [41]. Otra razón de peso es su aplicabilidad en áreas operativas tan diversas como teoría de códigos, diagnóstico de fallas, reconocimiento de patrones [5] y *análisis de afinidad* [1], entre otros. Se ha dedicado tanto esfuerzo a su estudio que hasta se cuenta, desde hace varios años, con instancias de grafos DIMACS propias, sobre las cuales verificar el rendimiento de los algoritmos exactos o heurísticas desarrolladas para encontrar su solución.

Un problema combinatorio relacionado, llamado $Máximo\ Clique - Corte$ o $Max\ Cut - Clique$, MCC, es tratado en esta tesis. En él, dado un grafo $\mathcal{G} = (V, E)$, donde V es el conjunto de vértices y E el de aristas respectivamente, se busca el Clique $\mathcal{C} \subset V$ que induzca al corte con mayor número de aristas entre los conjuntos $\mathcal{C}$ y V - $\mathcal{C}$. A diferencia del problema anterior, $Max\ Cut - Clique$ es un problema introducido recientemente por P. Martins [30] en el año

2012. Hasta lo mejor de nuestro conocimiento existen solamente las siguientes publicaciones sobre el mismo [30, 31, 22, 7, 8, 9] y dos equipos de investigadores (locales y externos) trabajando sobre él:

- con base en Portugal[1] [2] y España [3] compuesto por Pedro Martins, Helena Ramalhinho, Luis Gouveia y Antonio Ladrón [30, 31, 22]
- con base en Uruguay[4] compuesto por los doctores Franco Robledo, Eduardo Canale, Pablo Romero, Mathias Bourel y Luis Stábile [7, 8, 9].

Uno de los motivos para estudiar este problema es su aplicabilidad en áreas operativas en constante desarrollo, como lo son el marketing y el comercio electrónico [34]. En la literatura científica también se indican aplicaciones en Análisis de Canasta de Mercado o *Carrito de Compras*, Market Basket Analysis o MBA por sus siglas en inglés [12].

Esta área de Análisis de Datos o Data Mining originada en el campo del marketing, tiene aplicaciones tanto en bioinformática [3, 10], como en la operativa de redes financieras [27], de Internet [24] y hasta en las redes criminales [11]. El análisis desarrollado mediante MBA consiste en encontrar relaciones subyacentes entre grupos de productos o categorías, aprovechando la enorme cantidad de información originada en las ventas con la que cuentan las grandes cadenas de retail [5].

Estas relaciones son un insumo de gran incidencia en la estrategia de negocio definida por este tipo de empresas [34], por ejemplo, determinan el lanzamiento de promociones de productos o modifican las ubicaciones de los mismos en una superficie de venta o generan sugerencias de listas de productos relacionados en una compra en línea.

Es oportuno aclarar que los trabajos [30, 22] generalizan el MCC a una versión con pesos en las aristas, los cuales representan una medida de cuán relacionados están un par de artículos o productos. Aquí, se busca el clique en $\mathcal{G} = (V, E)$ que induzca al corte de mayor peso, $Weighted\ Max\ Cut - Clique$ o $WMCC$ por sus siglas en inglés.

[1] Instituto Superior de Contabilidad y Administración (ISCAC), Instituto Politécnico de Coimbra, Coimbra, Portugal

[2] Facultad de Ciencias, Centro de Investigación de Operaciones, Universidad de Lisboa, Lisboa, Portugal.

[3] Departamento de Economía y Empresa, Grupo de Análisis de Negocio, Universidad Pompeu Fabra, Barcelona, España.

[4] FIng, UdelaR, Facultad de Ingeniería, Universidad de la República, Montevideo, Uruguay.

[5] Una cadena de Retail es un comercio donde el público objetivo es el consumidor final.

En concreto, esta tesis recopila información reciente sobre la formalización, determinación de complejidad computacional del problema, a la vez que, introduce una solución basada en Algoritmos Genéticos. Una técnica conocida y exitosa en diversos problemas combinatorios y de búsqueda, cuya aplicabilidad ha crecido exponencialmente en los últimos 30 años. Los Algoritmos Genéticos pertenecen a una familia más amplia de metaheurísticas inspirada en la naturaleza, denominada Computación Evolutiva [21], cuyo funcionamiento está basado en mecanismos análogos a los principios que rigen la evolución natural de las especies biológicas.

1.2. Estructura de la tesis

Esta tesis se estructura de la siguiente manera. El Capítulo 2 contiene el estado del arte en lo que refiere a este joven problema. El Capítulo 3 contiene algunas definiciones consideradas relevantes para la comprensión del problema abordado en este trabajo. El Capítulo 4 describe formalmente el objeto de estudio, el *Máximo Clique Corte* o MCC. Asimismo, se demuestra que el MCC pertenece a la clase de problemas $\mathcal{NP}$-Completos, tomando el trabajo presentado en [8]. Este resultado promueve el desarrollo de metaheurísticas, y sobre esta metodología, hasta ahora se encuentran solamente 2 soluciones disponibles en la literatura, una desarrollada sobre ILS y otra sobre GRASP/VND. Ambas propuestas se pueden consultar en el Capítulo 5.

El Capítulo 6 presenta las principales contribuciones de esta tesis, donde se introduce un Algoritmo Genético para la resolución del objeto de estudio. En el Capítulo 7 se presenta una validación del algoritmo, los resultados obtenidos y un estudio comparativo de la propuesta actual versus el GRASP previamente desarrollado. Las principales conclusiones de esta tesis y líneas para trabajo futuro se presentan en el Capítulo 8 sobre una temática que aún tiene mucho por investigar y descubrir.

Capítulo 2

Estado del Arte

2.1. Introducción

En este capítulo se hace una recorrida por los avances logrados en las investigaciones relacionadas a ésta temática desde su introducción en el 2012 hasta nuestros días.

2.2. Reseña

En el trabajo original de Pedro Martins [30] se abordan problemas combinatorios relativos a la densidad del conjunto de vecinos de un clique $\mathcal{C}$, considerado como el cociente entre el cardinal del conjunto de vecindad y el producto de cardinales del clique y su complemento $\frac{|N(\mathcal{C})|}{(|\mathcal{C}|\times|V-\mathcal{C}|)}$. Se estudian casos extremales de este objetivo, tanto para su maximización como su minimización. Asimismo, introduce una variante de los problemas extremales anteriores, que consiste en maximizar el cardinal de vecinos de un clique fijo a encontrar, que los autores han denominado $MaxENC$ por sus siglas en inglés *Max Edge-Neighborhood Clique*.

Los autores motivan el problema a partir de aplicaciones en *Análisis de Afinidad*, y desarrollan formulaciones exactas tanto del $MaxENC$ como sus respectivas variantes citadas, mediante programación lineal entera. Asimismo, los autores presumen de la complejidad computacional del $MaxENC$, dada su similitud estructural con respecto tanto a $Max\text{-}Clique$ como a $Max\text{-}Cut$, reconocidos problemas de la clase $\mathcal{NP}$-Completos, que pertenecen a la lista de Karp [28]. En abstracto, los autores muestran potenciales aplicaciones a su

estudio, que procura determinar en general la relevancia de cliques, y discuten aplicaciones en redes de comunicación, terrorismo y aislación de moléculas proteicas en sistemas biológicos. Para la resolución de las instancias de prueba, redes reales asociadas a las aplicaciones discutidas en [30], se ha utilizado la herramienta de optimización ILOG/CPLEX 9.0, que provee soluciones exactas como también cotas mediante relajaciones. Se observa que ante instancias del orden de 10 mil nodos, la solución exacta no es encontrada, y la herramienta se detiene por límite de tiempo.

Posteriormente, en un trabajo de cooperación entre Luis Gouveia y Pedro Martins, desarrollan una formulación compacta eficiente para el $MaxENC$ y generalizaciones, considerando pesos en los nodos [22]. Los autores denominan al problema introducido por Martins en 2012 como *Maximun Edge-Weight Neighborhood Clique*, $MEWNC$, en asociación con un nombre que describe mejor las variantes respecto a un problema histórico, conocido en literatura como *Maximum Edge-Weighted Clique* o $MEWC$. A su vez, realizan una comparación de tiempos de ejecución utilizando la herramienta CPLEX entre 6 formulaciones exactas vía programación lineal entera, ejecutados luego para grafos con cientos de nodos. Las comparaciones efectuadas se enfocan en grafos dispersos, dadas sus aplicaciones en la interacción entre proteínas en un sistema biológico. Es oportuno mencionar que los autores no presentan un estudio de complejidad computacional del problema. Si bien no existe una formulación que supere a las restantes, el rendimiento de los nuevos modelos presentados supera a los anteriores para grafos dispersos, como una ventaja a la propuesta dada por los autores. Además de que sugieren como exploración a futuro el uso de planos de corte para reducir tiempos de cómputo.

Hasta lo mejor de nuestro conocimiento, el último trabajo realizado por el mismo equipo de investigadores fue publicado en el año 2015, e incluye al mismo Pedro Martins como primer autor. Fue realizado de manera conjunta por Antonio Ladrón y Helena Ramalhinho, creadora de la metaheurística Iterated Local Search, ILS [29]. En este trabajo se introduce la primera metaheurística para el problema, utilizando justamente la metodología ILS. Es en éste trabajo donde además se le denomina *Maximum Cut-Clique* o MCC al problema de estudio de esta tesis. En particular, adoptamos este nombre en lugar del original de Martins, dado su carácter descriptivo.

Es en este mismo trabajo donde los autores destacan que, hasta la fecha, solo se han desarrollado métodos de resolución exacta. Se puede apreciar

por primera vez, un compromiso de la calidad de la solución con el objetivo de premiar el tiempo de cómputo, mediante un algoritmo aproximado. La propuesta incluye un conjunto de 3 búsquedas locales, que apuntan a agregar/aspirar/intercambiar nodos del clique de manera individual (nodos singleton). La diversificación de la metaheurística se logra mediante la construcción aleatoria de cliques, y una función de perturbación a los cliques construidos, que no es tan agresiva como las técnicas de *shaking* utilizadas en VND.

Los métodos desarrollados en trabajos previos fueron ejecutados en computadoras con prestaciones similares, utilizando la herramienta de optimización ILOG/CPLEX 11.2. Los autores demuestran la alta competitividad de la metaheurística desarrollada en términos de calidad, pues logran igualar la optimalidad de los métodos antes construidos para más del 90 % de las instancias de estudio, mientras que en el resto se alcanzan cotas inferiores conocidas del problema.

Es importante remarcar que nuevamente se prescinde de un estudio de complejidad computacional del problema, es decir, la dureza del problema no fue demostrada por los autores. En efecto, no hay un fundamento teórico de una resolución aproximada, a menos que el problema sea $\mathcal{NP}$-Difícil.

A partir del año 2017, un equipo de investigadores uruguayos conformado por el Dr. Franco Robledo, Dr. Pablo Romero (tutores de esta tesis), Dr. Eduardo Canale, Dr. Mathias Bourel y el tesista, MSc. Ing. Luis Stábile, ha continuado con la línea de investigación relativa al *Max Cut-Clique*. En el artículo[8], los autores demuestran formalmente por primera vez en la literatura que el MCC pertenece a la clase de problemas $\mathcal{NP}$-Completos, hecho que fundamenta el uso de metaheurísticas. En este mismo artículo se desarrolla una metodología híbrida GRASP/VND enriquecida con Tabú Search para el mismo. Las comparaciones con metaheurística ILS, utilizada previamente, demuestra competitividad en lo relativo a calidad de las soluciones, encontrando la optimalidad en las mismas instancias encontradas anteriormente [30].

En el trabajo [7], los mismos autores introducen un modelo exacto de programación lineal entera para el MCC. Basados en resultados novedosos de teoría de grafos que aplican sobre el $Clique - Corte$ óptimo, se introduce una condición necesaria que debe cumplir el óptimo global. Esto permite incorporar una restricción sobre el tamaño del clique que alcanza el máximo corte, e incorporarlo como una restricción, tanto en la búsqueda del GRASP/VND como en una formulación exacta que se introduce en el mismo artículo. Cabe

destacar que ésta restricción reduce significativamente el espacio de soluciones, y por tanto los tiempos computacionales que tendría la ejecución del modelo sin ésta misma cota.

Finalmente, en un trabajo que se encuentra bajo referato científico, los mismos autores extienden el análisis, considerando pesos o importancia en los nodos. El mismo estudio de cotas aplica en éste contexto con leves modificaciones, tanto para el modelo exacto como para mejorar el GRASP/VND implementado. El problema en su versión de pesos es $\mathcal{NP}$-Difícil por inclusión del MCC, o también a partir de la última reducibilidad de Karp para el problema de la partición [28]. Para la versión con pesos o $WMCC$, se compara un GRASP/VND a medida con los algoritmos exactos del trabajo [22] desarrollado por Pedro Martins y Luis Gouveia. Los resultados computacionales para el $WMCC$ son contundentes, se alcanza la optimalidad en todos los casos anteriormente hallados, y además se supera en calidad las instancias de mayor tamaño en las cuales las formulaciones exactas finalizan por límite de tiempo.

Capítulo 3

Conceptos Preliminares

En este capítulo se listan los conceptos preliminares necesarios para introducir el problema de interés abordado en esta tesis y se realiza un breve resumen referente a la Complejidad Computacional.

3.1. Teoría de Grafos

Los grafos son básicamente una abstracción que permite representar de forma intuitiva aquellos problemas de la realidad donde entidades (nodos) y las interacciones entre ellas (aristas) son datos relevantes. En el problema que nos compete de Análisis de la Canasta de Mercado o Carrito de Compra, los nodos del grafo representan productos y las aristas que los enlazan son relaciones de coexistencia en la misma canasta o carrito de compra.

Definición 1. Grafo.
Un grafo es un par ordenado $\mathcal{G} = (V, E)$, comprendiendo un conjunto finito de vértices o nodos denotados por $V \neq 0$, junto con un conjunto de aristas o enlaces que son pares de vértices denotado por $E \subseteq V \mathrm{x} V$.

Definición 2. Grafo dirigido.
Es un grafo $\mathcal{G} = (V, E)$, donde sus aristas conforman un par ordenado. Es decir, dados dos nodos u y v la arista (u, v) es diferente a la arista (v, u) y pueden no existir las 2 a la vez.

Definición 3. Grafo no dirigido.
Es un grafo $\mathcal{G} = (V, E)$, donde tanto el par (u, v) como el (v, u) se corresponde con la misma arista del grafo, por lo cual no se distinguen direcciones en sus arcos.

Definición 4. Multigrafo.
Es un grafo $\mathcal{G} = (V, E)$, donde se pueden repetir aristas entre 2 nodos.

En la Figura 3.1 se visualizan los diferentes tipos de grafos tratados. A la izquierda un *grafo dirigido* con un lazo, en el centro un *grafo simple no dirigido* y a la derecha un *multigrafo*. Un ***lazo*** es una arista, cuyos extremos son el mismo vértice. Cuando estamos en presencia de un grafo sin lazos, decimos que es un ***grafo simple***. Para evitar confusiones, cuando no se especifica cual es el tipo de grafo, se asume que el grafo es ***no dirigido y simple***.

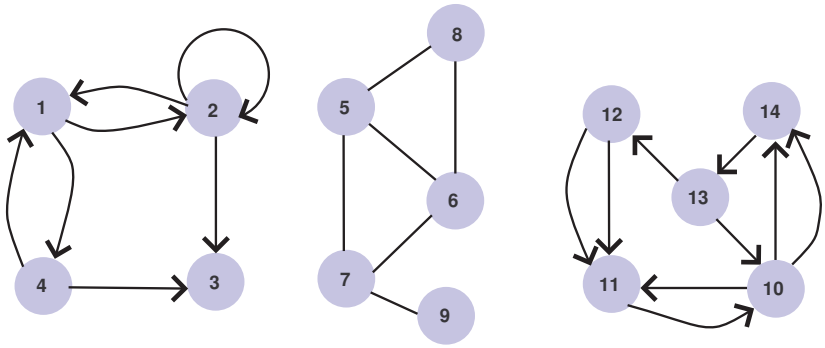

Figura 3.1: Diferentes tipos de grafos.

Definición 5. Grafo Conexo.
Dado un grafo $\mathcal{G} = (V, E)$, para cualquier par de nodos $u, v \in V$ existe al menos una secuencia de aristas que permite alcanzar el nodo v desde u. Ver Figura 3.2.

Definición 6. Grafo Completo.
Dado un grafo conexo $\mathcal{G} = (V, E)$ es completo cuando sus n vértices son adyacentes entre sí. Si es simple se denota por K_n. Ver Figura 3.2.

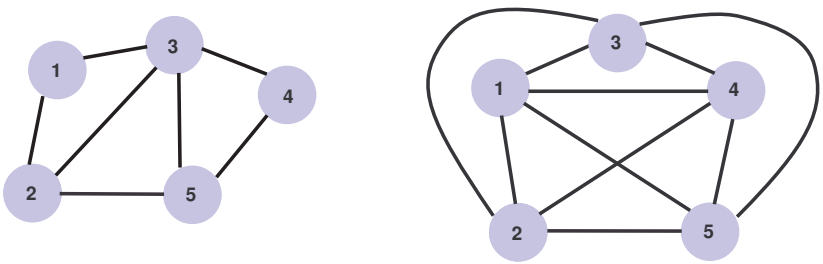

Figura 3.2: Grafo conexo, grafo completo.

Definición 7. Orden y Tamaño.
El orden de un grafo $\mathcal{G} = (V, E)$ es la cantidad de nodos $|V|$, y el tamaño la cantidad de aristas $|E|$.

Definición 8. Subgrafo de $\mathcal{G}$.
Dado un grafo $\mathcal{G} = (V, E)$, un subgrafo de $\mathcal{G}$, es cualquier grafo $\mathcal{G}' = (V', E')$ incluido en él. Esto es, $\mathcal{G}' = (V', E')$ donde $V' \subseteq V$ y $E' \subseteq E$.

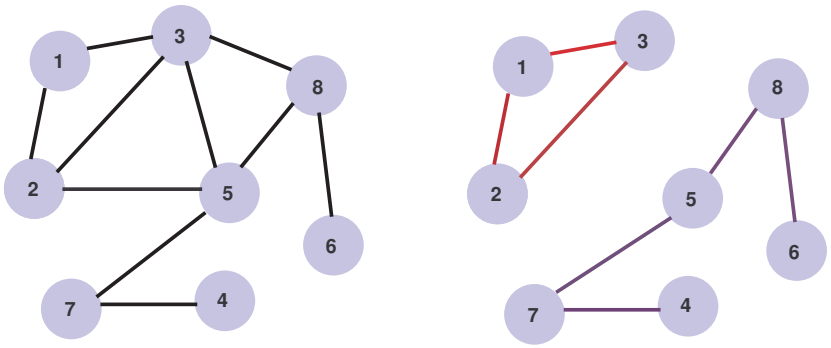

Figura 3.3: Grafo $\mathcal{G}$, subgrafos de $\mathcal{G}$.

Definición 9. Vecindad de un nodo.
Dado un grafo $\mathcal{G} = (V, E)$ y un vértice $v \in V$, el conjunto de vecinos del nodo v en G es denotado como $N_{\mathcal{G}}(v)$ o $N(v)$, donde $N(v) = \{w \in V : (v, w) \in E\}$.

Definición 10. Grado de un nodo.
Dado un grafo $\mathcal{G} = (V, E)$ y un vértice $v \in V$, el grado o valencia de un nodo v es el número $d(v) = |N(v)|$. Este valor coincide con la cardinalidad del número de vecinos de v.

Definición 11. Grado mínimo de $\mathcal{G}$.
Dado un grafo $\mathcal{G} = (V, E)$, se define su grado mínimo como $\delta(G) = min\{d(v)|v \in V\}$.

Definición 12. Grado máximo de $\mathcal{G}$.
Dado un grafo $\mathcal{G} = (V, E)$, se define su grado máximo como $\Delta(G) = max\{d(v)|v \in V\}$.

Definición 13. Grado promedio de $\mathcal{G}$.
Dado un grafo $\mathcal{G} = (V, E)$, se define su grado promedio como $d(G) = \frac{1}{|V|}\Sigma_{v \in V} d(v)$. Claramente, $\delta(G) \leqslant d(G) \leqslant \Delta(G)$.

En la Figura 3.4, sobre el margen izquierdo tenemos el grafo simple $\mathcal{G}$, no dirigido, de orden $|V| = 5$ y tamaño $|E| = 6$. A la derecha, se puede apreciar el concepto de vecindad de un nodo, tomando como ejemplo al nodo 5, para ello se hace foco en él y sus nodos adyacentes.

A su vez, el grado mínimo del grafo lo aporta el nodo 9 con $d(9) = 1$ y el grado máximo se da en los nodos 5, 6 y 7, $d(5) = d(6) = d(7) = 3$. El grado promedio de este grafo es $d(G) = \lfloor (3 \times 3 + 2 + 1)/5 \rfloor = 2$.

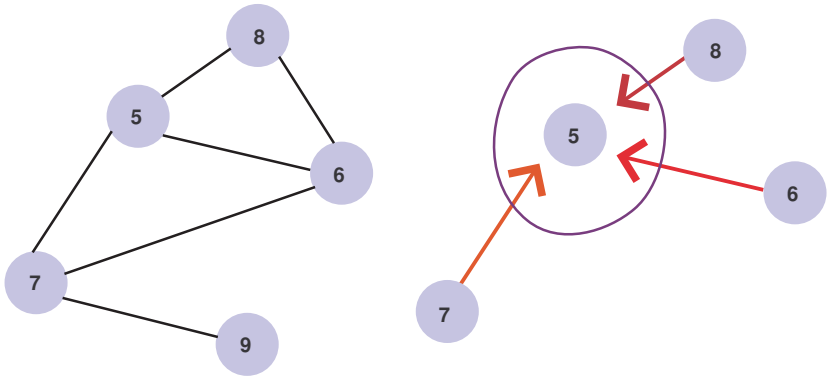

Figura 3.4: Grafo $\mathcal{G}$, vecindad y grado de un vértice.

Definición 14. Grafo Regular
Es un grafo $\mathcal{G} = (V, E)$, donde todos sus nodos tienen el mismo grado.

Definición 15. Lema de Handshaking.
Dado un grafo $\mathcal{G} = (V, E)$, la suma de los grados de todos sus nodos es igual a $2 \times |E|$.

Definición 16. Conjunto de corte.
Dado un grafo $\mathcal{G} = (V, E)$ y un conjunto de nodos $\mathcal{C}$ / $\mathcal{C} \subseteq V$, $\mathcal{C}$ es un conjunto de corte si el grafo $\mathcal{G} - \mathcal{C}$ es disconexo.

Definición 17. Clique
Sea $\mathcal{G} = (V, E)$ un grafo simple, un ***Clique*** es un subconjunto de nodos $\mathcal{C} \subseteq V$ donde todos y cada uno de sus elementos forman pares de adyacentes con los demás nodos del conjunto. En otras palabras, el subgrafo inducido por $\mathcal{C}$ sobre $\mathcal{G}$ es un grafo completo.

3.2. Complejidad Computacional

La teoría de la complejidad computacional busca formalizar la complejidad de un problema, permitiendo clasificarlo según su dificultad inherente. El lector puede consultar un libro autorizado sobre el tema como lo es *Computers and Intractability: A Guide to the Theory of NP-Completeness* de los autores Garey and Johnson [18]. Se trata de encontrar la respuesta a la siguiente pregunta: *¿qué hace que algunos problemas sean computacionalmente difíciles y otros sencillos?* Su finalidad es la creación de una teoría capaz de describir y analizar la complejidad de un algoritmo y la dificultad inherente a un problema.

Las medidas utilizadas para determinar la dificultad de una problema generalmente son los recursos de tiempo y memoria empleados por el algoritmo utilizado para su resolución. Siendo el principal indicador de eficiencia de un algoritmo, el tiempo de ejecución. Para comparar los tiempos computacionales de dos algoritmos es necesario definir una función $f(n)$ que indique la cantidad de operaciones necesarias para la ejecución. A su vez, esta función depende del largo de la configuración de entrada del algoritmo, n. Se dice que la función $f(n)$ es de orden $O(g(n))$ si existe una constante M, tal que $f(n) \leq M \times g(n)$. Los algoritmos se pueden distinguir por los tiempos empleados; entre polinomiales y exponenciales, esto es, según la función que acote superiormente a $f(n)$. Si un problema requiere un algoritmo que ejecute en forma exponencial entonces estamos en presencia de un problema intratable [18].

Alan Turing fue quien obtuvo los primeros resultados sobre intratabilidad en 1936, demostrando que ciertos problemas son *indecidibles*. Para ello, definió un dispositivo de cómputo universal, que luego fue conocido como la máquina de Turing, capaz de brindar la solución de cualquier problema siempre que la misma sea computable. La premisa utilizada en su célebre artículo *On Computable Numbers, with an Application to the Entscheidungs Problem* [37], es que todo problema que no es resoluble por este dispositivo no es resoluble por ningún otro. El problema utilizado para demostrar la intratabilidad es conocido como *el problema de la parada* en inglés (*The Halting Problem*). Aquí, dada una máquina de Turing y una serie de parámetros de entrada, se determina si la máquina termina o no en un número finito de operaciones, por lo que se espera que retorne *Sí/No* en un tiempo finito.

Formalmente, basados en este concepto es que se definen los *Problemas de Decisión* donde el objetivo es determinar si una configuración de entrada satis-

face una propiedad específica, buscando una respuesta de tipo *Sí/No*. Por otro lado, tenemos que los problemas se clasifican en clases según su complejidad. Los problemas de decisión para los cuales existe un algoritmo que encuentra una solución en tiempo polinomial pertenecen a la clase $\mathcal{P}$. Aquellos problemas de decisión para los cuales se puede verificar una solución en tiempo polinomial pertenecen a la clase $\mathcal{NP}$. De aquí, que si un problema puede resolverse en tiempo polinomial, también puede verificarse en tiempo polinomial, es decir $\mathcal{P} \subseteq \mathcal{NP}$. Más aún se sospecha que $\mathcal{P} \nsupseteq \mathcal{NP}$, afirmación base para uno de los problemas abiertos más importantes que a la fecha de hoy no ha podido ser demostrado.

Un problema $\mathcal{H}$ es $\mathcal{NP}$-Difícil si todo otro problema de la clase $\mathcal{NP}$ admite una reducción polinomial al primero. Si además $\mathcal{H}$ pertenece a $\mathcal{NP}$ entonces es $\mathcal{NP}$-Completo.

Los *Problemas de Conteo* son aquellos problemas donde la solución es un entero no negativo. Estos problemas representan otra clase de complejidad de interés para el análisis de la complejidad computacional. Lo interesante es que cada problema de conteo, tiene un problema de decisión asociado. Por ejemplo, encontrar los n subgrafos cliques en un grafo $\mathcal{G} = (V, E)$ cuyo tamaño sea menor que un entero dado k, tiene como problema de decisión asociado verificar que cada uno de esos subgrafos sea un clique. Para la clase de problema de decisión en $\mathcal{NP}$ se define la clase $\#\mathcal{P}$ para clasificar los problemas de conteo o enumeración relacionados. La clase de problemas $\#\mathcal{P}$-Completo refiere a los problemas de conteo cuyos problemas de decisión subyacentes son de la clase $\mathcal{NP}$-Completo [38].

En síntesis y como procedimiento para demostrar la complejidad computacional de un problema, se debe utilizar el teorema de Stephen Cook [13]. Esto es, para demostrar la complejidad de un problema es suficiente con probar que el problema de decisión asociado pertenece al conjunto $\mathcal{NP}$, y que es por lo menos tan difícil de resolver como un problema $\mathcal{NP}$-Difícil. Este teorema es uno de los pilares de la teoría de la Complejidad Computacional [18] y fue utilizado por Richard Karp en [28] para presentar una lista con 21 problemas a los cuales les llamó *problemas completos*. La complejidad de los mismos fue demostrada valiéndose del teorema de Cook. En esa lista inicial de problemas $\mathcal{NP}$-Completos encontramos a:

- *Clique Máximo* - ¿dado un grafo, se encuentra algún subgrafo completo de tamaño k o menor?

- *Partición* - ¿es posible partir la suma de un conjunto de naturales a la mitad?
- *Corte Máximo* - ¿existe un corte con capacidad no menor a k?

Capítulo 4

Máximo Clique-Corte

En este capítulo se detalla el problema de estudio con el fin de lograr un entendimiento del mismo que permita la lectura fluida del documento. Primero se muestra de forma gráfica las interrogantes que llevaron a la génesis del problema. Luego se brindan definiciones formales junto con la argumentación de la dificultad del mismo, acompañada de una ingeniosa demostración de complejidad.

4.1. Introducción

Es un problema reciente de teoría de grafos conocido como *Máximo Clique − Corte*, *Max Cut − Clique* o *MCC* por sus siglas en inglés. Dado un grafo simple, se desea hallar un subgrafo completo tal que el corte inducido por sus nodos tenga máximo cardinal. Este problema combinatorio fue introducido por P. Martins en 2012, y encuentra aplicaciones en áreas de *Minería de Datos* como lo es *Análisis de la Canasta de Mercado o Carrito de Compra* (MBA), donde interesa la correlación de artículos de venta.

A modo intuitivo, en la Figura 4.1 se visualiza que desde el clique que genera el corte máximo se pueden alcanzar nodos ubicados en diferentes zonas de la red. Además, en este caso en particular, el tamaño del clique máximo y el clique del máximo corte son muy similares. Estas observaciones son el fundamento para reorientar la búsqueda hacia los cliques con vecindades más densas, en lugar de los cliques más grandes [31].

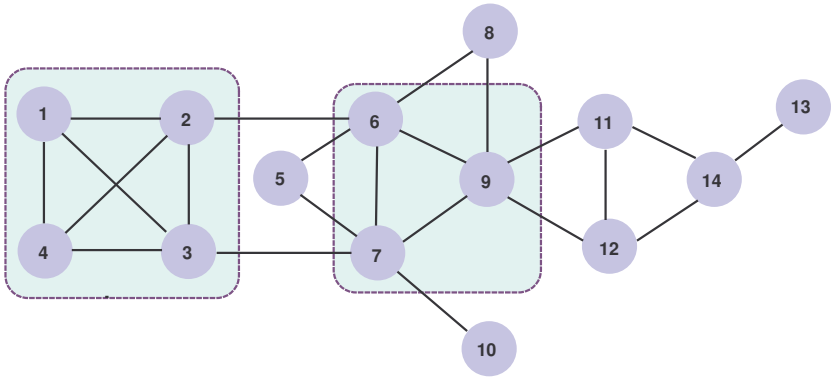

Figura 4.1: Un Max Clique y un Max Clique-Corte en un grafo de 14 nodos.

4.2. Definición del Problema

Para continuar con las definiciones formales pero haciendo énfasis en el objeto de estudio, veremos un $Clique - Corte$ $\{\mathcal{C}, V - \mathcal{C}\}$ y finalmente el problema del $Máximo\ Clique - Corte$

Definición 18. Clique-Corte
Dado un grafo simple $\mathcal{G} = (V, E)$ y un clique $\mathcal{C}$ sobre G, el ***Clique-Corte*** es el conjunto de aristas $(x, y) \in E$, donde $x \in \mathcal{C}$, $y \in (V - \mathcal{C})$, estableciendo el corte $\{\mathcal{C}, V - \mathcal{C}\}$ sobre $\mathcal{G}$. El conjunto de aristas inducido por $\mathcal{C}$ es denotado por $\delta(\mathcal{C})$.

Definición 19. Máximo Clique-Corte MCC
Es el $Clique - Corte$ $\{\mathcal{C}, V - \mathcal{C}\}$, sobre un grafo simple $\mathcal{G} = (V, E)$, con la **máxima** cantidad de aristas en la vecindad del Clique, $max|\delta(\mathcal{C})|$.

En la Figura 4.2 se visualiza un grafo de ejemplo, donde verificar las definiciones brindadas. Se tiene

- un ***máximo clique*** definido sobre $\{1, 2, 3, 4\} = \mathcal{C}_1$,
- un ***máximo clique-corte*** definido sobre $\{6, 7, 9\} = \mathcal{C}_2$,
- un ***clique*** definido sobre el conjunto $\{11, 12, 14\} = \mathcal{C}_3$.

Para el ***máximo clique*** $\mathcal{C}_1$, las aristas sobre el corte inducido son $\delta(\mathcal{C}_1) = \{(2, 6), (3, 7)\}$, entonces $|\delta(\mathcal{C}_1)| = 2$ mientras el tamaño del clique es $|\mathcal{C}_1| = 4$. Para el ***máximo clique-corte*** $\mathcal{C}_2$, el conjunto de aristas es

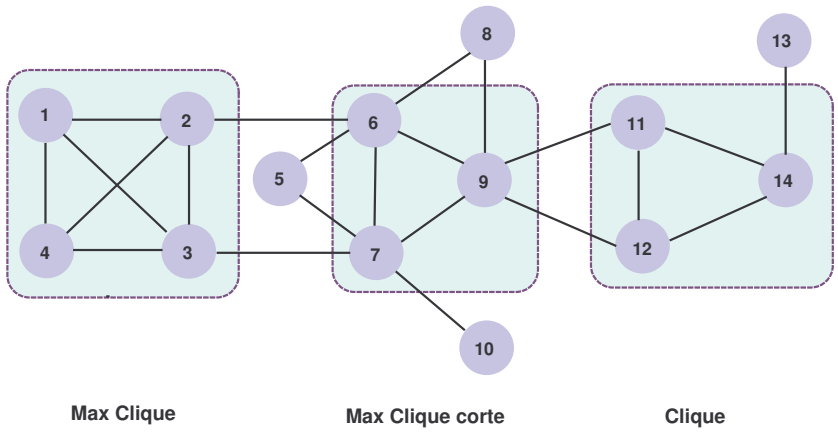

Figura 4.2: Diferentes cliques en un Grafo ejemplo de 14 nodos.

$\delta(\mathcal{C}_2) = \{(6,2),(6,5),(6,8),(7,3),(7,5),(7,10),(9,8),(9,11),(9,12)\}$, el cardinal del corte máximo es $|\delta(\mathcal{C}_2)| = 9$ y el tamaño del clique es $|\mathcal{C}_2| = 3$. Finalmente, tenemos para $\mathcal{C}_3$, $\delta(\mathcal{C}_3) = \{(11,9),(12,9),(14,13)\}$, $|\delta(C_3)| = 3$ y $|\mathcal{C}_3| = 3$.

Martins en su trabajo inicial [30] observa que el óptimo global se puede alcanzar en varios cliques a la vez. Más aún, sabemos que la cantidad de cliques de un tamaño k determinado, en un grafo denso es inclusive mayor que la cantidad de cliques máximos que se puedan registrar. Estas observaciones sugieren que el espacio de soluciones factibles para este problema puede tener dimensiones considerables si el grafo es particularmente denso.

4.3. Demostración de Complejidad

Hasta donde se ha investigado Martins presupone que el *MCC* pertenece a la familia de problemas $\mathcal{NP}$-Completos dada su similitud con problemas de esta clase como *Max Cut* o *Max Clique*. Sin embargo, no presenta reducción polinomial a ninguno de los problemas, ni tampoco prueba formal de complejidad. Una elegante prueba de $\mathcal{NP}$-Completitud de este problema se presenta en el artículo [7]. A modo de completitud y dada la importancia para esta tesis es que se incluye la demostración en esta sección.

Para establecer la complejidad computacional del *MCC*, se realizarán algunas puntualizaciones previas que conducirán a su demostración.

El problema de encontrar el $Max-Clique$ en un grafo es un problema $\mathcal{NP}$-Completo, está incluido en la lista original de Karp [28], con los 21 problemas

para los cuales se demostró su pertenencia a la clase $\mathcal{NP}$-Completo. En el año 1998, se demostró que es inaproximable en un factor de $n^{\frac{1}{4}-\varepsilon}$ para cualquier $\varepsilon > 0$ [4].

La prueba se basa en una reducción al problema *Max-Clique*, es por esto que las definiciones de los problemas serán hechas como problemas de decisión. Formalmente se probará que *MCC* es al menos tan difícil como *Max-Clique*. Para ello, primero describiremos la versión de decisión para ambos problemas.

Definición 20. Max-Clique *MC*
DADO: un grafo $G = (V, E)$ y un número real K.
PREGUNTA: existe un clique $\mathcal{C} \subseteq V$ tal que $|\mathcal{C}| \geq K$?

Sin pérdida de generalidad, denotemos como $\delta(\mathcal{C})$ el corte inducido por el conjunto de nodos $\mathcal{C}$, o el valor objetivo para *MCC* siempre que $\mathcal{C}$ sea un clique.

Definición 21. Max Clique-Corte *MCC*
DADO: un grafo $G = (V, E)$ y un número real K.
PREGUNTA: existe un clique $\mathcal{C} \subset V$ tal que $|\delta(\mathcal{C})| \geq K$?

Teorema 1. *El MCC pertenece a la clase de problemas $\mathcal{NP}$-Completo.*

Demostración. La idea de la prueba es mostrar que el problema *MCC* es por lo menos tan difícil como el problema *Max-Clique*. Considerar un grafo simple $\mathcal{G} = (V, E)$ con orden $n = |V|$ y tamaño $m = |E|$. Agregar m nodos terminales colgando de cada nodo simple $v \in V$ (observar que ahora tenemos $m \times n$ de esos nodos). Al grafo resultante de esta modificación lo llamamos H, ver Figura 4.3. Si encontramos un algoritmo que ejecute en tiempos polinomiales para *MCC*, entonces podemos producir el *Máximo Clique-Corte* en H.

Sin embargo, observar que el clique $\mathcal{C}$ que alcanza el máximo clique corte en H debe pertenecer a $\mathcal{G}$. Si $\mathcal{C}$ tiene cardinalidad c, entonces el clique-corte tiene precisamente $c \times m$ nodos colgantes. Por construcción, el clique-corte debe maximizar el número de nodos colgantes, entonces el tamaño $|E| = m$ es agregado al corte cada vez que un nodo se suma al clique. Como consecuencia, c debe ser el *Max-Clique*.

Esto es, se ha probado que el *MCC* es al menos tan difícil como el *Max Clique*, como se pretendía. *MCC* pertenece al conjunto $\mathcal{NP}$ de los problemas de decisión y por lo tanto es $\mathcal{NP}$-Completo. □

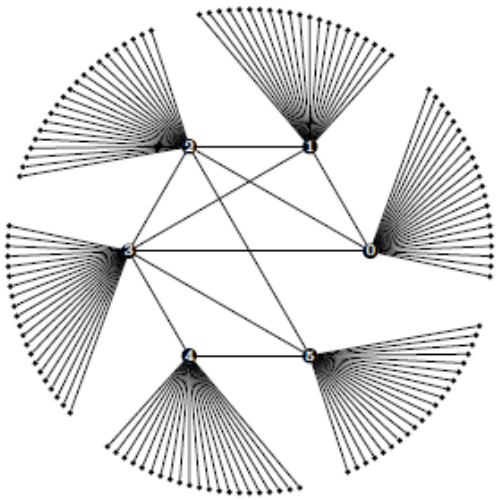

Figura 4.3: Construcción de H con $M = 21$ nodos colgantes [7].

4.4. Formulación Matemática

En esta sección, se presenta la formulación matemática del problema en términos de programación entera ya que es un problema de optimización combinatoria. La formulación matemática del problema fue definida en [7] de la siguiente forma:

$$w_i = \begin{cases} 1 & \text{si } i \in \mathcal{C} \\ 0 & \text{sino} \end{cases}, \forall i \in V$$

$$w_{(i,j)} = \begin{cases} 1 & \text{si } (i,j) \in \delta(\mathcal{C}) \\ 0 & \text{en otro caso} \end{cases}, \forall (i,j) \in E$$

En el modelo de programación entera que se presenta a continuación, se puede apreciar que las restricciones (1) y (2) refieren a que ambos nodos i, j pertenecen al clique $\mathcal{C}$ si y solo si $(i,j) \in \delta(\mathcal{C})$.

A su vez, las restricciones (3) y (4) determinan una cota inferior LB y superior UB, definiendo un intervalo factible para el tamaño del clique, c_{min}. Mientras que las restricciones (5) y (6) aseveran que w_i y $w_{(i,j)}$ son variables binarias. El objetivo es maximizar el clique-corte, que es precisamente la diferencia entre la suma de los grados de los nodos pertenecientes al clique menos

el grado interno de los mismos.

$$
\begin{array}{lll}
\max & \sum_{i \in V} d_i \times w_i - 2 \times \sum_{(i,j) \in E} w_{(i,j)} & \\
\text{s.a.} & 2w_{(i,j)} \leq w_i + w_j & \forall (i,j) \in E \quad (1) \\
& w_i + w_j - 1 \leq w_{(i,j)} & \forall i,j \in V \quad (2) \\
& & \\
& \sum_{i \in V} w_i \geq LB & (3) \\
& \sum_{i \in V} w_i \leq UB & (4) \\
& & \\
& w_{(i,j)} \in \{0,1\} & \forall (i,j) \in E \quad (5) \\
& w_i \in \{0,1\} & \forall i \in V \quad (6)
\end{array}
$$

Las cotas LB y UB introducidas en [9] mejoran notoriamente la eficiencia computacional tanto de la solución exacta como de las búsquedas locales de la metaheurística GRASP/VND, ya que contribuyen a reducir el espacio de soluciones. El limite superior fue propuesto por Martins y Gouveia en [22]. Mientras que el inferior fue deducido en [9] en la sección de *Analisis y Complejidad*, por el equipo local de investigadores.

A partir del Lema de Handshaking (15) aplicado a un clique cualquiera, se vincula el tamaño del clique $|\mathcal{C}|$ con el valor objetivo $|\delta(\mathcal{C})|$.

Es conveniente aclarar que los métodos exactos mediante formulación matemática fueron expuestos en el trabajo previo de P.Martins [31], donde se realizan 3 formulaciones diferentes específicas para 3 familias de grafos con características bien distinguidas; muy densos, espaciados y el resto que no pertenece a ninguna de estas dos categorías.

Capítulo 5

Soluciones Previas

Las soluciones propuestas hasta la fecha para el *MCC* incluyen tanto métodos exactos como aproximados. Para los aproximados se han utilizado Búsqueda Local Iterada (ILS) y la metodología Ávida, Aleatorizada y Adaptativa (GRASP/VND) que combina diversas estructuras de búsquedas locales. En esta tesis no detallaremos los métodos exactos, cuya formulación matemática fue realizada en el Capítulo 4, sino que profundizaremos en las metaheurísticas empleadas como método de resolución.

5.1. Introducción

Ahora sabemos que *MCC* pertenece a la familia de problemas $\mathcal{NP}$-Completos, y esta afirmación ha sido demostrada. Razón por la cual, las aproximaciones a su solución han sido mayormente mediante heurísticas. En este sentido, es relevante aclarar que las heurísticas son métodos de resolución basados en procedimientos conceptualmente simples, que encuentran soluciones de buena calidad, no necesariamente óptima a problemas difíciles de un modo sencillo y eficiente a la vez.

Las metaheurísticas, según [19] en su definición original, son métodos de resolución que orquestan la interacción entre procedimientos locales de mejora y estrategias de más alto nivel para crear procesos capaces de escapar del óptimo local y desempeñar una búsqueda robusta en el espacio de soluciones. Muy a menudo los procedimientos locales utilizados son guiados por heurísticas, por lo que se produce una combinación de ambas técnicas, sobre todo cuando se está tratando con espacios de soluciones muy complejos. Se caracterizan por

admitir una descripción abstracta de alto nivel, tener una aplicación genérica, esto es, pueden aplicar para cualquier clase de problema siempre y cuando sean instanciadas para la especificidad.

5.2. ILS

El método Búsqueda Local Iterada, Iterated Local Search ILS por sus siglas en inglés tiene muchas de las características deseables de una metaheurística; es simple, fácil de implementar, robusto y altamente efectivo. La idea esencial de ILS es hacer foco en un subespacio de soluciones definido por las soluciones que son óptimos locales para una estrategia subordinada que guía un búsqueda más inteligente.

Es oportuno remarcar que el éxito que pueda tener esta técnica en encontrar soluciones de calidad, radica en la elección del algoritmo de búsqueda local, la perturbación definida sobre los óptimos locales y el criterio de aceptación de una solución. Para profundizar en los detalles del ILS se puede leer una guía especializada en el tema como lo es el Handbook of Metaheuristics [29].

El esqueleto algorítmico de esta técnica se muestra a continuación.

Algorithm 1 ITERATED LOCAL SEARCH

1: $s_0 = GenerateInitialSolution()$;
2: $s^* = LocalSearch(s_0)$
3: **repeat**
4: $\quad s' \quad = Perturbation(s^*, history)$;
5: $\quad s^{*'} = LocalSearch(s')$
6: $\quad s^* \quad = AcceptanceCriterion(s^*, s^{*'}, history)$
7: **until** *termination condition met*

5.3. ILS para MCC

Por su parte P. Martins [31], presentó una solución basada en esta metaheurística donde no solo se cubrieron varias clases de grafos de las instancias DIMACS, sino que se encontraron los mejores valores óptimos que siguen vigentes al día de hoy.

El algoritmo ILS sigue las búsquedas propuestas en [23] para el problema del *Max Clique*. A su vez, las búsquedas empleadas son derivadas de las búsquedas

locales dinámicas descritas en [33]. La elección de esta técnica sienta sus bases en los reportes realizados por [23]; precisión, rapidez y baja dependencia en parametrización.

Este algoritmo se basa en dos movimientos principales, los que corresponden a un proceso de construcción incremental; ***add/aspiration*** y los que corresponden a la corrección de una solución buscando un clique vecino de igual tamaño que permita expandir el corte generado; ***swap***.

Uno de los procesos responsables del éxito de este algoritmo es ***plateau search***, donde el nodo que entra al clique es seleccionado del conjunto de candidatos que tienen una arista en común con todos los nodos del clique excepto con uno de ellos. Luego cuando el procedimiento no admite más movimientos de ***add*** o ***swap***, se implementa el procedimiento que reinicia el algoritmo. Este procedimiento inicia con una versión distorsionada del mejor clique retornado en la etapa anterior.

Para comprender el funcionamiento de la búsqueda, el entendimiento de los siguientes conjuntos es fundamental:

- $\mathcal{C}$, es el conjunto de nodos en el Clique que esta bajo construcción
- $K_0(\mathcal{C})$, es el conjunto de nodos candidatos para aumentar el tamaño del Clique $\mathcal{C}$, $i \in K_0(\mathcal{C})$ si $(i,j) \in E\ \forall j \in \mathcal{C}$
- $K_1(\mathcal{C})$, es el conjunto de nodos a distancia 1 del conjunto $\mathcal{C}$, $i \in K_1(\mathcal{C})$ si $(i,j) \in E$, $\forall j \in \mathcal{C} \setminus \{r\}$ y $(i,r) \notin E$
- U es el conjunto de nodos Tabú. Aquí van los nodos que son excluidos del conjunto $\mathcal{C}$, cuando se realiza un movimiento de ***swap***.

Las líneas 6-19, corresponden a una búsqueda local completa. Un ciclo de búsqueda completa termina cuando $K_0(\mathcal{C}) = \emptyset$ y $K_1(\mathcal{C}) \setminus U = \emptyset$ y $\mathcal{C}$ es un clique-corte máximal y no quedan más candidatos para movimientos de ***swap***, excluyendo los nodos de U, o cuando $\mathcal{C} \cap \mathcal{C}' = \emptyset$, esto es, cuando el clique actual no tiene nodos en común con el primer clique-corte maximal encontrado en la búsqueda actual, o porque se alcanzó el numero máximo de selecciones max_sel.

Algorithm 2 ILS: MAX CUT CLIQUE

Input: $\mathcal{G}$
Output: $\mathcal{C}^*$

1: $\mathcal{C}^* \leftarrow \emptyset$, $\mathcal{C} \leftarrow \emptyset$, $sel \leftarrow 0$;
2: **while** $(sel < max\_sel)$ **do**
3: *Randomly select a node* $i \in (V \setminus C)$; /*Perturbación o nuevo comienzo*/
4: $\mathcal{C} \leftarrow [\mathcal{C} \cap N(i)] \cup \{i\}$;
5: $\mathcal{U} \leftarrow \emptyset$, $\mathcal{C}^* \leftarrow \mathcal{C}$; /*Local/plateau search*/
6: **while** $(K_0(\mathcal{C}) \neq \emptyset)$ **or** $(K_1(\mathcal{C}) \setminus U) \neq \emptyset)$**and**$(\mathcal{C} \cap \mathcal{C}^* \neq \emptyset)$**and**$(sel < max\_sel)$ **do**
7: **if** $(K_0(\mathcal{C}) \setminus U) \neq \emptyset)$ **then** /*mov: Add*/
8: *select a node* $i \in (K_0(\mathcal{C}) \setminus U)$;
9: $\mathcal{C} \leftarrow \mathcal{C} \cup \{i\}$;
10: **if** $(U = \emptyset)$ **then**
11: $\mathcal{C}^* \leftarrow \mathcal{C}$;
12: **if** $(K_1(\mathcal{C}) \setminus U) \neq \emptyset)$ **then** /*mov: Swap*/
13: *select a node* $i \in (K_1(\mathcal{C}) \setminus U)$;
14: $\mathcal{C} \leftarrow [\mathcal{C} \cup \{i\}] \setminus \{j\},\ where\{j\} = \mathcal{C} \setminus N(i)$;
15: $U \leftarrow U \cup \{j\}$;
16: **if** $(K_0(\mathcal{C}) \cap U) \neq \emptyset)$ **then** /*mov: Aspiration*/
17: *select a node* $i \in (K_0(\mathcal{C}) \cap U)$;
18: $\mathcal{C} \leftarrow \mathcal{C} \cup \{i\}$;
19: $sel \leftarrow sel + 1$;
20: **if** $|E'(\mathcal{C})| > |E'(\mathcal{C}^*)|$ **then**
21: $\mathcal{C}^* \leftarrow \mathcal{C}$;
22: **return** $\mathcal{C}^*$

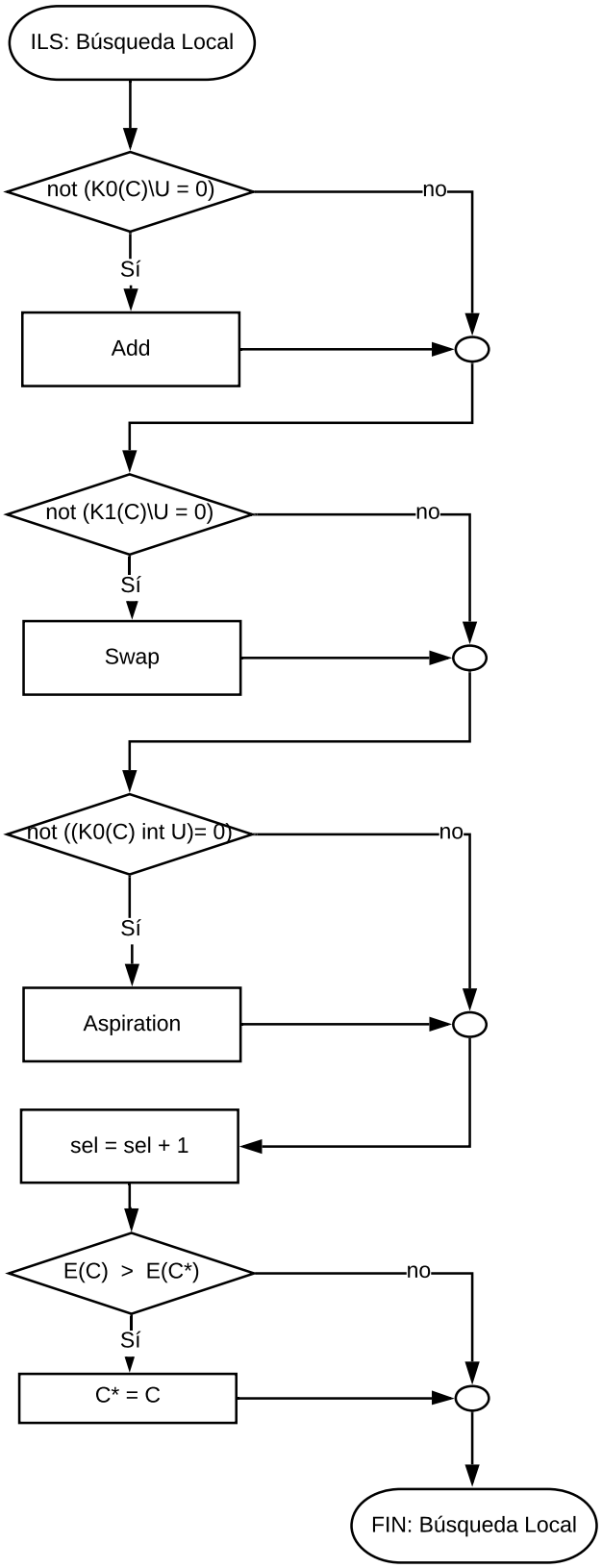

Figura 5.1: ILS: Diagrama de Flujo para la fase de búsqueda local.

5.4. GRASP/VND

El método GRASP, es un procedimiento de búsqueda; ávida, aleatorizada y adaptativa (Greedy Randomized Adaptive Search procedure). Es un proceso iterativo [36], donde en cada iteración se desarrollan dos fases; una de construcción y otra de búsqueda local. En la primera se construye una solución factible, mientras que en la segunda se explora su vecindario en busca de una mejor solución. El resultado es la mejor solución encontrada sobre la base de todas las iteraciones realizadas por el proceso.

Una característica determinante de esta metodología es su fácil implementación, ya que requiere poca parametrización. Lo ingenioso del método radica en implementar las estructuras de datos adecuadas para acelerar las iteraciones; mediante algoritmos de construcción y búsquedas eficientes. Los parámetros principales son, el **criterio de parada** y la calidad de los elementos en la lista restringida de candidatos **RCL**.

Algorithm 3 GRASP

Input: $max\_iterations$, $seed$
Output: $bestSolution$

```
1: readInput();
2: for (i = 1 to max_iterations) do
3:     solution ← GRConstruction(seed);
4:     if (not(feasible(solution))) then
5:         solution ← repair(solution);
6:     solution ← localSearch(solution);
7:     updateSolution (solution, bestSolution);
   return bestSolution;
```

El éxito de la fase de búsqueda local esta altamente relacionada: con la calidad de la solución inicial, la estructura de vecindad elegida, técnicas de búsquedas eficientes, fácil evaluación de la función de costos.

Dependiendo del problema que se esté resolviendo, existen dos formas de explorar un vecindario, investigar todos los vecindarios y la solución actual es reemplazada por **la mejor de todas** o reemplazar la solución actual por **la primera mejor** solución encontrada.

La fase de construcción juega un rol muy importante con respecto a la fase de evaluación porque la producción de soluciones iniciales de calidad, es un insumo importante para el procedimiento de búsqueda local. Se puede consultar el siguiente código genérico para apreciar la lógica con la cual opera.

Algorithm 4 GRConstruction

Input: *seed*
Output: *solution*

```
1: solution ← ∅;
2: candidateList ← initial();
3: incrementalCostEvaluation(candidateList);
4: while not(feasible(solution)) do
5:     RCL ← constructionRCL(candidateList);
6:     s ← selectRandom(RCL);
7:     solution ← solution ∪ {s};
8:     candidateList ← update();
9:     incrementalCostEvaluation(candidateList);
   return solution;
```

5.5. GRASP/VND para MCC

En la resolución presentada por el equipo local de investigadores [8], se utilizó un procedimiento GRASP enriquecido con VND Variable Neighborhood Descent y Tabu Search y además se limitó la lista de candidatos RCL de la fase de construcción con las cotas máximas y mínimas establecidas en [7], mostrada en el Capítulo 4.

Veremos que la estructura básica de un GRASP se ve enriquecida por la construcción y evaluación del VND, mientras que la construcción y mantenimiento de una Lista Tabú trata de potenciar las soluciones factibles. Esto es, la Búsqueda Tabú o Tabu Search [20, 2] es usada para prevenir que las búsquedas locales vuelvan a soluciones que ya fueron consideradas y descartadas.

Más precisamente, lo que se hace es definir un parámetro en el algoritmo θ^{max}, luego de la fase de VND, los nodos más frecuentes en todas las soluciones no son considerados para futuras soluciones durante θ iteraciones, siempre que se alcanza la cantidad de θ^{max} iteraciones consecutivas sin mejora en la solución. Estos nodos son elegidos, sólo si aparecen más de ϕ veces desde la última vez que la Lista Tabú fue actualizada.

Algorithm 5 GRASP/VND con Tabu Search

Input: α, θ^{max}, $maxIter$, $\mathcal{G}$
Output: $\mathcal{C}^*$

$\mathcal{C}^* \leftarrow \emptyset$
$\mathcal{T} \leftarrow \emptyset$
for $iter = 1$ **to** $maxIter$ **do**
  $\mathcal{C} \leftarrow$ Clique$(\alpha, \mathcal{T}, \mathcal{G})$
  $\mathcal{C} \leftarrow$ VND$(\mathcal{C}, \mathcal{T}, \mathcal{G})$
  $\mathcal{T} \leftarrow$ Update$(\mathcal{T}, \theta^{max}, \mathcal{C})$ /*Lista Tabú*/
  if $|E'(\mathcal{C})| > |E'(\mathcal{C}^*)|$ **then**
    $\mathcal{C}^* \leftarrow \mathcal{C}$
return $\mathcal{C}^*$

El algoritmo de la fase de construcción del GRASP se detalla a continuación. Se denota con la letra $\mathcal{C}$, al clique que está siendo construido. El grado mínimo y máximo de los nodos del conjunto U, se denota con $\delta(U)$ y $\Delta(U)$ respectivamente. En la definición de la lista restricta de candidatos RCL, se incluyen los nodos con los grados máximos y mínimos así como también un parámetro α que inyecta aleatoriedad sobre los nodos elegidos de forma ávida.

Algorithm 6 GRASP Construction: Clique

Input: α, $\mathcal{T}$, $\mathcal{G}$
Output: $\mathcal{C}$

$\mathcal{C} \leftarrow \emptyset$
$improving =$ MAX_ATTEMPTS
$RCL \leftarrow \{v \in V - \mathcal{C} : |E'(v)| \geq \Delta(V - \mathcal{C}) - \alpha(\Delta(V - \mathcal{C}) - \delta(V - \mathcal{C}))\}$
while $improving > 0$ **do**
  $i \leftarrow selectRandom(RCL)$
  $\mathcal{C}' \leftarrow [\mathcal{C} \cap N(i)] \cup \{i\}$
  if $|E'(\mathcal{C}')| > |E'(\mathcal{C})|$ **then**
    $\mathcal{C} \leftarrow \mathcal{C}'$
    $improving \leftarrow$ MAX_ATTEMPTS
  else
    $improving \leftarrow improving - 1$
return $\mathcal{C}$

El VND[15] se construye sobre cinco estructuras de vecindad, que son resultado de aplicar los movimientos: ***add***, ***aspiration*** y ***swap*** definidos en la solución ILS, más dos que son resultado de la propuesta de nuevos movimientos, a saber: ***cone*** y ***remove***. El movimiento de ***cone*** es una generalización de ***swap*** para múltiples nodos mientras que el movimiento de ***remove*** es cuando

un singleton $\{i\}$ es removido de un Clique $\mathcal{C}$.

Las búsquedas locales en el procedimiento VND se realizan en el siguiente orden: ***Remove***, ***Add***, ***Swap***, ***Cone***, ***Aspiration***, un óptimo local en cada una de las estructuras de vecindades es alcanzado y retornado como resultado de la búsqueda local.

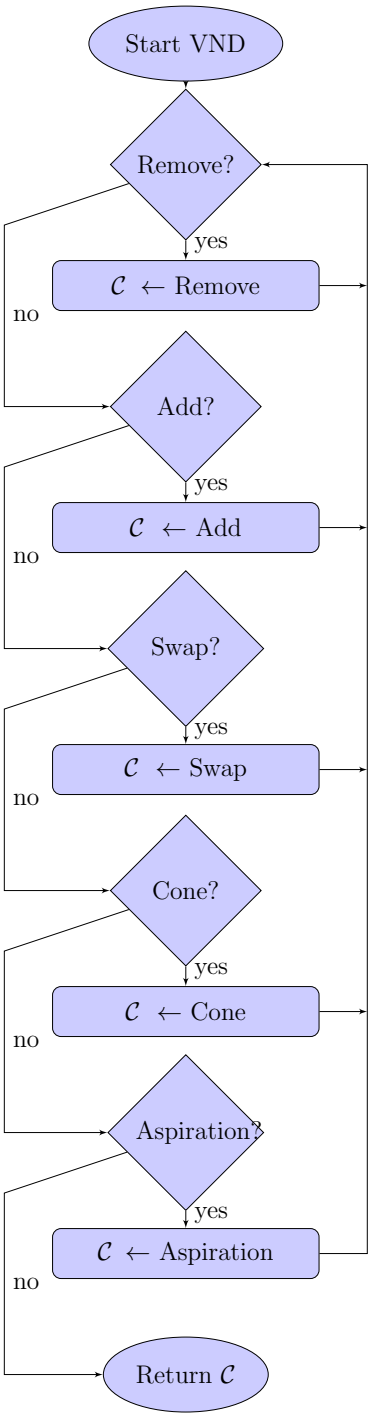

Figura 5.2: VND: Diagrama de Flujo resumido para la fase de búsqueda local [9].

Capítulo 6

Solución basada en GA

6.1. Introducción

En este capítulo se brinda una breve reseña de la técnica metaheurística utilizada, pasando por los hitos que han marcado el rumbo de su desarrollo en las últimas décadas.

Luego, se describe detalladamente la solución implementada desde la elección de su representación hasta el tratamiento de las soluciones no factibles, pasando por los operadores evolutivos entre otros. Finalmente, se brindan detalles de las piezas de código desarrolladas para instanciar y resolver este problema en particular.

6.2. Algoritmos Genéticos

6.2.1. Reseña

Los Algoritmos Genéticos, Genetic Algorithms o GA por su siglas en inglés, son una técnica de Computación Evolutiva que ha logrado grandes avances, dada su aplicabilidad a una amplia gama de problemas de optimización y búsqueda difíciles de resolver. Las metaheurísticas no garantizan encontrar la solución óptima a un problema, pero encuentran soluciones aproximadas de buena calidad en tiempos razonables.

El término de *Algoritmo Genético* fue utilizado por primera vez por John Holland [26], en su libro *Adaptation in Natural and Artificial Systems* publicado en el año 1975. Este libro es la génesis de lo que hoy conocemos como

un floreciente campo de investigación y aplicaciones que reúne varias técnicas, además de los GA originales.

Si bien éste trabajo se centra en la línea de desarrollo de John Holland, hubo otros investigadores con diversas formaciones en otras áreas científicas que estuvieron trabajando en ideas similares en el mismo momento, pero fue él quien publicó el compendio de sus observaciones y experimentos [35]. Aquí podemos encontrar en la década de los 60, el concepto de *evolution strategy* desarrollado por Ingo Rechenberg [39], Hans-Paul Schwefel [6] en Alemania. Mientras que en Estados Unidos se gestaba la idea de *evolutionary programming* desarrollada por Bremermann y Fogel [17]. El hilo conductor de estas ideas fue el uso de la mutación y selección de las soluciones más aptas, operadores que forman parte del concepto de evolución de la teoría neo Darwiniana.

Por su parte, la tesis de doctorado de Ken DeJong, uno de los estudiantes de Holland en la universidad, fue otro hito que acompañó la publicación del libro de Holland en la década de los 70. En este trabajo se realiza un análisis profundo de la aplicabilidad de los algoritmos genéticos en el campo de la *optimización*. [14].

Sin lugar a dudas, la década de despegue de los GA fue la del 80. En ella, otro alumno de doctorado de Holland, David Goldberg obtuvo sus primeros resultados exitosos con GA y puso en el centro de discusión científica este nuevo paradigma. Luego en 1989, Goldberg publica un libro fundamental llamado *Genetic Algorithms in Search, Optimization, and Machine Learning* [21]. Desde ese entonces, los GA han generado un desarrollo realmente sorprendente pasando por varias ramas de la ciencia y la ingeniería, como medio para resolver problemas de optimización combinatoria difíciles o $\mathcal{NP}$-Completos.

Es de relevancia acotar que éstas ideas no pudieron someterse a pruebas exhaustivas en el momento mismo de su desarrollo porque el poder de cálculo y la memoria con la que contaban las computadoras o máquinas de calcular de la época era limitado. Esta es la razón por la cual, la historia del desarrollo de la metaheurística, tiene un lapso temporal entre la idea y la materialización de la misma. Es sabido que la idea de una máquina que aprenda a medida que resuelve un problema es manejado desde la década del 40 por Alan Turing [25], quien detectó la relación entre los procesos de aprendizaje y la evolución natural. En 1948, Turing proponía desarrollar programas *automodificables* que fueran capaces de jugar ajedrez y copiar otras actividades inteligentes realizadas por los seres humanos, utilizando técnicas evolutivas.

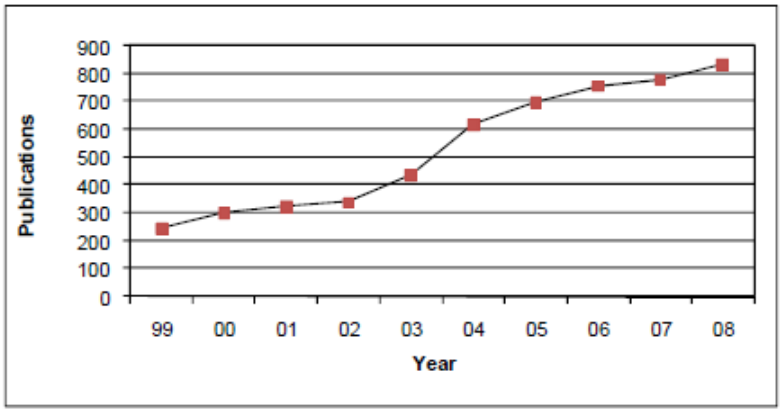

Figura 6.1: Publicaciones sobre GA entre 1999 y 2008.[42]

En los años subsiguientes científicos de la talla de Von Neumann entre otros continuaron desarrollando ideas similares. John Von Neumann propuso mecanismos evolutivos basados en la programación para implementar autómatas con poder computacional equivalente a una máquina universal de Turing. Su teoría quedó inconclusa y fue publicada de forma póstuma por su colega David Burke, *Theory of self-reproducing automata* [40]. Por más detalle, sobre la historia del desarrollo de las técnicas evolutivas, consultar el compendio realizado en la tesis de maestría de Sergio Nesmachnow[32]. Aquí solo presentamos una breve reseña para contextualizar el trabajo de ésta tesis.

De modo gráfico se muestra en la Figura 6.1, la evolución de las publicaciones que incluyen algún algoritmo genético [42]. Ésto es solo para la década del 2000 y fue realizado por la Universidad de Waterloo Canadá.

6.2.2. Definición

En concreto, es una técnica de optimización, no determinística que trabaja sobre una población de soluciones que evolucionan por medio de la selección y la construcción de soluciones factibles por recombinación de características de soluciones previas o mutación de la propia solución. La idea de la superviviencia del individuo más apto se materializa mediante la función de selección. La adaptabilidad de cada individuo al problema está medida por la función de adecuación o evaluación del individuo. La evolución propiamente dicha se da en la reproducción de los individuos, por ejemplo, se toman dos soluciones, se recombinan sus características y se obtienen dos o más soluciones nuevas

a partir de ellas. La población se mantiene con un tamaño fijo mediante el criterio de recambio generacional y va evolucionando como consecuencia de la aplicación de los operadores evolutivos en los individuos más aptos, hasta alcanzar la cantidad máxima de generaciones o un valor de adecuación definido como aceptable.

La técnica tiene una estructura genérica, bien definida, para utilizarla debemos instanciar nuestro problema en forma de algoritmo genético. Luego de obtener los resultados, se establece una correspondencia entre la representación interna de la solución y su análogo en la realidad del problema que se está resolviendo. Conviene que ésta correspondencia, sea una función biyectiva ya que precisamos de su función inversa para realizar la conversión de la solución.

La estructura algorítmica se detalla a continuación.

Algorithm 7 GA ALGORITHM

```
1: Initialize(P_0);
2: generation = 0;
3: while (notstopCriteria) do
4:     evaluate(P(generation));
5:     parents ← selection(P(generation));
6:     offspring ← evolutiveOperators(parents);
7:     newpop ← replacement(offspring, P(generation));
8:     generation++;
9:     P(generation) ← newpop;
   return BestSolutionEverFound;
```

Una solución basada en un algoritmo genético consiste en un ciclo de cuatro etapas:

- evaluación
- selección
- aplicación de operadores evolutivos
- reemplazo

Se trabaja sobre una población de individuos que representan soluciones potenciales al problema a resolver. La representación (individuo) es el genotipo, la solución es el fenotipo. La función de fitness evalúa a los individuos de acuerdo a su adecuación para la resolución del problema. Mientras que la población es inicializada mediante un mecanismo aleatorio, o guiado por heurísticas específicas dependiendo del problema a resolver. Luego se han desarrollado diversas

políticas de selección y reemplazo, como privilegiar al individuo más adaptado, aumentar la presión selectiva o incrementar la diversidad genética, tratando de hacer un balance entre estos dos últimos de forma de evitar la convergencia temprana y/o caer en una exploración exhaustiva del espacio de soluciones.

Si bien esta técnica es muy versátil a lo largo de los años diversos estudios han indicado de que no es aplicable a todos los problemas $\mathcal{NP}$-Completos que se conozcan. En general, no funciona bien cuando se debe incorporar mucho conocimiento de la realidad para llegar a una solución. Por la naturaleza de éstos algoritmos no se puede implementar una mejora fina (fine tuning) como en otros.

6.3. Estrategia de Resolución

Para instanciar el problema en términos de algoritmos evolutivos se define el tamaño del individuo o la solución, como la cantidad de vértices $|V| = n$ que tiene el grafo $\mathcal{G} = (V, E)$. La información de la topología del grafo es un parámetro del problema. A partir de ella se conoce el grado de cada nodo, dato necesario para realizar el cálculo de la función de adecuación de la solución, así como para determinar si una solución es factible. Es por ello que al momento de cargar los datos, se construye una estructura auxiliar que contiene información referente al grado de cada vértice. Esta construcción optimiza el tiempo computacional cada vez que se evalúa una solución.

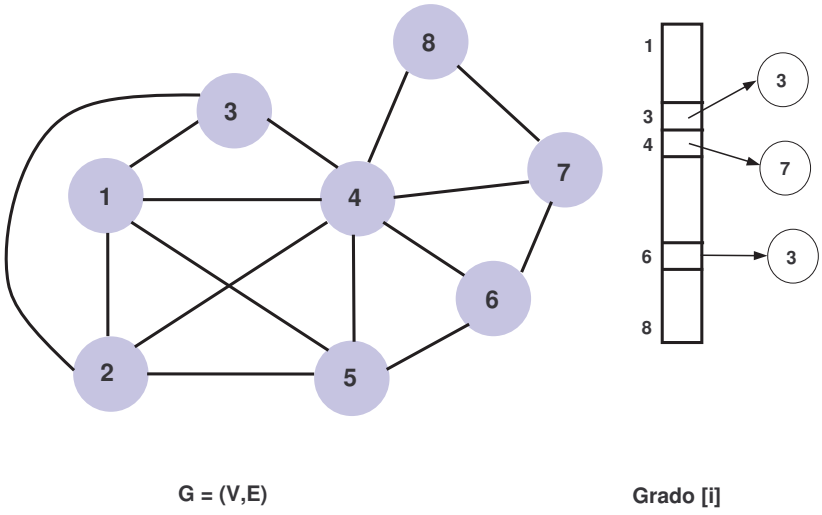

Figura 6.2: Un grafo $\mathcal{G}$ y su vector de grados de nodos.

Se puede apreciar en la Figura 6.2, que el $grado(3) = 3$, el $grado(4) = 7$ y el $grado(8) = 2$ esto es a modo de ejemplo para comprender la estructura auxiliar.

A continuación se detallan las áreas de interés; representación de la solución, función de evaluación, operadores evolutivos, características de la población, criterio de parada y algoritmo propuesto para resolver el problema.

6.3.1. Representación de la Solución

La solución está representada por una tupla de largo igual a $|V| = n$, la cantidad de nodos del grafo $\mathcal{G} = (V, E)$, conformada únicamente por 0s y 1s. Donde un 1 en la posición i, significa que el nodo i pertenece a la solución y un 0 representa el caso contrario.

$$X_i = \begin{cases} 1 & \text{si nodo } i \in \mathcal{C} \\ 0 & \text{sino} \end{cases}, \forall i \in V$$

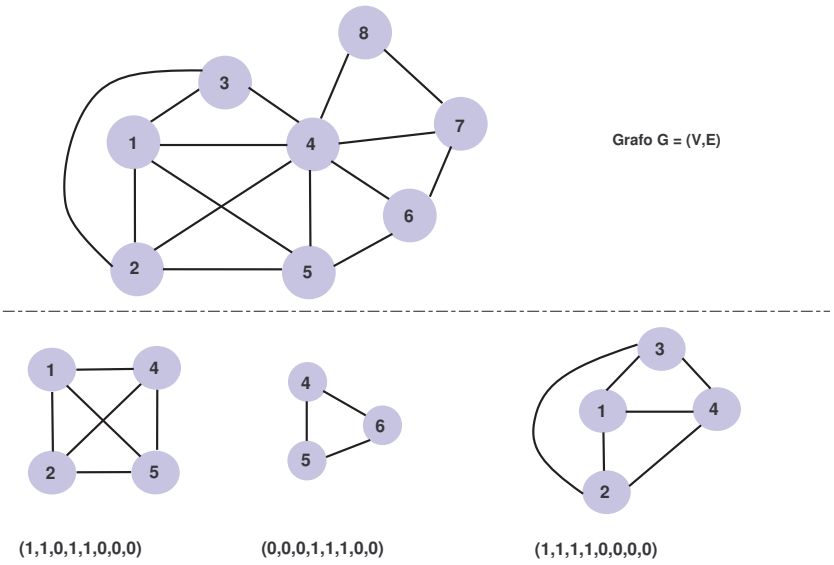

Figura 6.3: Representación de potenciales soluciones de $\mathcal{G}$.

La tira binaria de tamaño n, representa más soluciones que las aceptadas por el problema, ya que una tupla binaria no necesariamente representa un grafo. Es una representación sencilla que sólo hace referencia a los nodos del grafo sin tomar en cuenta la información de las adyacencias de los mismos. En este caso, ese dato no es necesario por la sencilla razón que una solución es un

subgrafo completo.

En la Figura 6.3, se muestran algunas tiras binarias que representan potenciales soluciones junto a los grafos resultado de su decodificación.

Si realizamos un paralelismo con la biología, la representación también es denominada *cromosoma* o *individuo*. Mientras que cada posición de la tira es denominada *gen* y al valor dentro de la posición *alelo*.

6.3.2. Función de Evaluación

La función de evaluación o fitness coincide con la función objetivo y busca maximizar la cantidad de aristas en el corte generado por el clique $\mathcal{C}$, $max|E(\mathcal{C})|$. Para ello, se deben sumar los grados externos de cada vértice perteneciente al clique, esto es, sumar los grados de los vértices que forman el clique y restarle los grados internos de cada uno de ellos. Este razonamiento se corresponde con la siguiente fórmula.

$$|E(C)| = \sum grado(i) - |C| * |C-1|, \forall \ nodo \ \ i \in \mathcal{C} \tag{6.1}$$

Al momento de cargar la información del grafo $\mathcal{G} = (V, E)$ del problema, se construye una estructura donde se almacena la información del grado de cada vértice. Esto hace que los accesos se realicen en $O(1)$ cada vez que se calcula cuán apta es una solución para resolver el problema. En el peor de los casos: $n * O(1)$, con $n = |V|$. Como se mencionó anteriormente, esto logra un cuidado de los tiempos empleados en el cómputo que se traduce en la eficiencia computacional en la obtención de los resultados.

En este caso, la función de evaluación está claramente formulada por una expresión matemática y está eficientemente implementada gracias a la estructura auxiliar de los grados de cada nodo. Como es utilizada constantemente, suele ser determinante en el desempeño computacional del GA.

6.3.3. Operadores Evolutivos

Los operadores evolutivos elegidos para esta solución fueron:

- Selección por torneo de tamaño 2
- Cruzamiento de 2 puntos
- Mutación simple

El mecanismo de *selección* elegido determina fuertemente la operativa del procedimiento de búsqueda, es responsable de dirigir la exploración a secciones relevantes del espacio. La presión de la selección es crítica para el buen funcionamiento del algoritmo. Sin embargo, una presión selectiva alta puede ocasionar pérdida de diversidad haciendo que la búsqueda termine de forma precipitada en un óptimo local, *convergencia prematura*. Por otro lado, si la presión es baja, la búsqueda empleará tiempo computacional en exceso. Lo razonable es mantener un compromiso entre la *exploración* de un espacio de búsqueda y la *explotación* de buenas soluciones.

Para salvar este problema y según recomendaciones tomadas del curso de Sergio Nesmachnow[32], se realiza un muestreo mixto, incluyendo características determinísticas y aleatorias, mediante la *Selección por Torneo de Goldberg*. Este operador selecciona el mejor individuo de un conjunto aleatorio de tamaño 2, es decir, se toman de forma aleatoria 2 soluciones que pasan a conformar el conjunto y luego se elige a la mejor de ellas para pasar a la siguiente generación.

El operador de *cruzamiento* tiene como objetivo recombinar las características de los individuos de la población en nuevos individuos para garantizar la *explotación* de buenas regiones del espacio de búsqueda. El cruzamiento elegido define dos puntos de corte en el cromosoma que marca que porción de material genético pasa a sus respectivos descendientes. En la Figura 6.4, se puede ver gráficamente como funciona esta operación. Luego del cruzamiento contamos con dos nuevos individuos, que fueron generados a partir de dos individuos existentes.

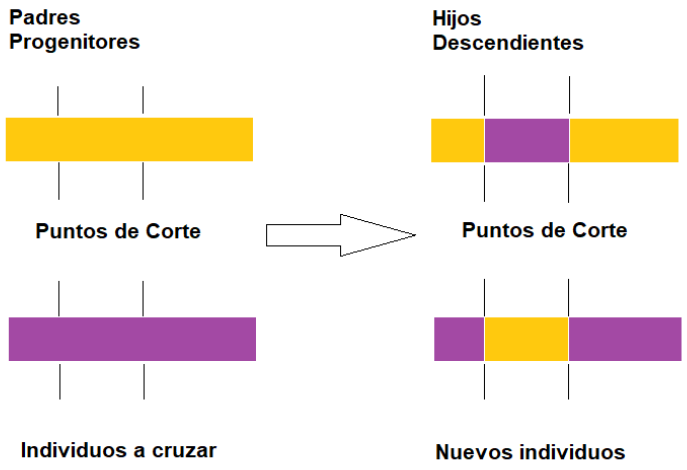

Figura 6.4: Operador de Cruzamiento.

Por su parte, el operador de *mutación* tiene como objetivo introducir diversidad de forma aleatoria en los individuos de la población, lo cual posibilita la *exploración* de diferentes secciones del espacio de búsqueda. La mutación elegida es la que modifica el valor de una posición determinada de forma aleatoria. Es una operación análoga a la mutación de la evolución natural, ya que copiando el material genético de una tira de ADN a otra existe cierta probabilidad de error en la copia de un alelo. Esta operación se puede visualizar en la Figura 6.5.

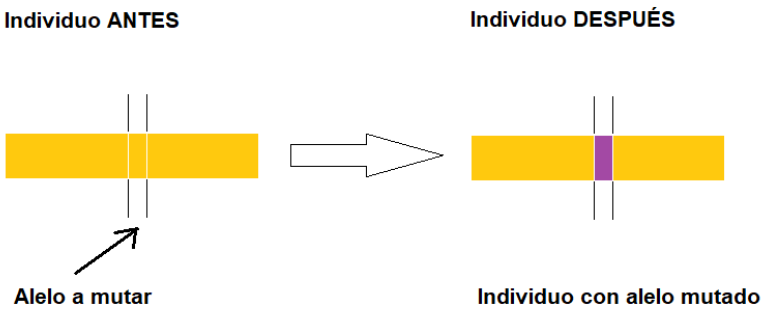

Figura 6.5: Operador de Mutación.

En este punto es acertado mencionar que los Algoritmos Genéticos priorizan la explotación del espacio de búsqueda sobre la exploración del mismo. Ésto es, el operador de cruzamiento tiene mayor probabilidad de ser aplicado que el operador de mutación.

6.3.4. Población y Criterio de Parada

La población es inicializada de forma randómica para otorgar diversidad a la población inicial. Sobre esta generación de individuos y antes de ser evaluadas, se construyen soluciones factibles que luego serán valoradas para determinar su adecuación a la solución y continuar con la mecánica del GA.

Se determina como criterio de parada del algoritmo, la cantidad de generaciones por las cuales pasa el mismo. En este caso, el algoritmo termina cuando la evolución de la población ha llegado a 1.000, constante determinada durante la etapa de validación del algoritmo. Se realizaron ejecuciones con 10.000 generaciones y se detectó que se llegaba a la mejor solución en las primeras

1.000 generaciones. Información valiosa para acelerar los tiempos de ejecución.

6.3.5. Solución propuesta

En esta sección las piezas relevantes del código, las que fueron desarrolladas para instanciar este problema, son descritas en detalle. Es importante mencionar que el algoritmo fue implementado sobre MALVA, un entorno de desarrollo para inteligencia artificial construido en *C++*, colaborativo y de código abierto. Por más información consultar *The Malva Project*[16], el Centro de Cálculo de Facultad de Ingeniería [1] colabora activamente en el desarrollo y mantenimiento de este proyecto.

La razón principal de ésta elección es el rendimiento en los tiempos de ejecución ya que esta tesis utiliza instancias de grafos tomadas de las DIMACS [2], muchas de ellas son muy densas conteniendo más de 1000 nodos. La investigación será aplicada a una instancia real del campo del Análisis de Canasta de Mercado, será tomada sobre enormes grafos que modelarán la relación entre productos de consumo y la pertenencia a una misma canasta de compra. Como las cadenas de venta al público final, cuentan con una vasta información de las compras realizadas ya en orden de miles de millones de registros, es necesario que este algoritmo contemple esos casos.

MALVA trae implementado la estructura algorítmica básica de un GA con el paradigma de orientación a objetos lo cual hace que sea sencillo instanciar un problema y pasar a obtener resultados mediante una configuración por defecto.

El arte de una buena solución radica, en una representación adecuada, una función de fitness definida sin ambigüedades y un cuidado manejo de la memoria ya que construyen objetos en *C++*. Además de la correcta elección de los operadores, el ajuste de los parámetros entre otros. En concreto, para instanciar un problema en MALVA se deben desarrollar completamente las clases *Problem* y *Solution*, del diagrama de la Figura 6.6.

[1] Centro de Cálculo, grupo de trabajo que se enfoca a las áreas de: Computación de Alta Performance y Computación Gráfica e Interacción

[2] Instancias disponibles la siguiente url, último acceso Octubre 2019, http://www.dcs.gla.ac.uk/~pat/maxClique/distribution/DIMACS_cliques/

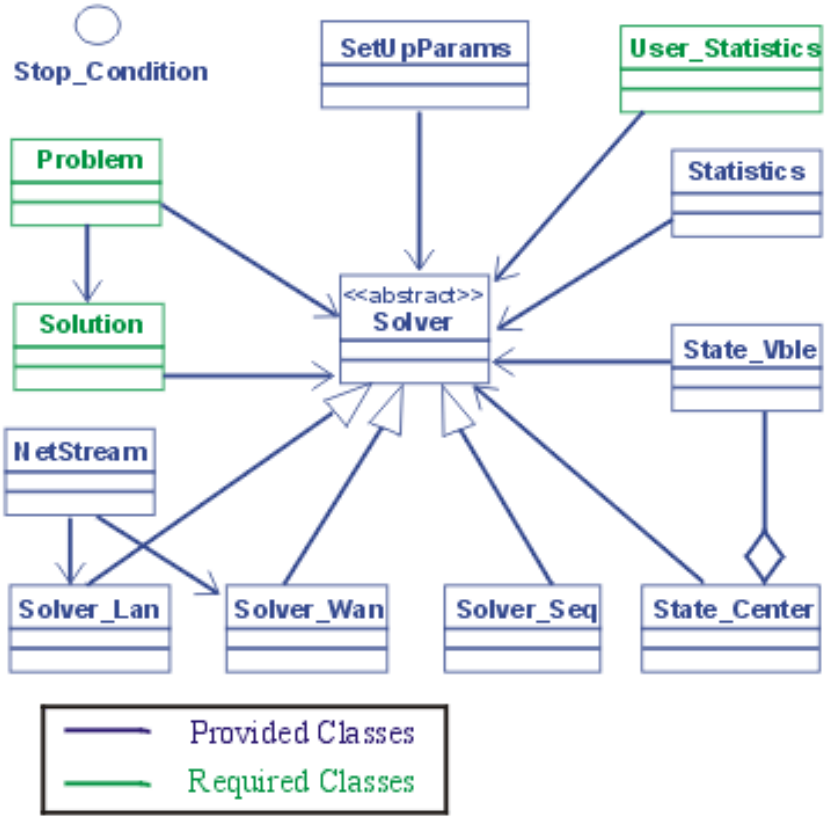

Figura 6.6: Diagrama de componentes de clases de Arquitectura[16].

6.3.6. Construcción del Clique

Éste es un bloque constructivo simple pero altamente importante ya que el algoritmo sólo trabaja con soluciones factibles. Es apropiado mencionar que la estrategia de la heurística no radica en agrandar el clique sino que agranda sólo si éste incrementa la cantidad de aristas en el nuevo corte.

La generación de soluciones factibles es prioridad en esta solución, sino fuera así, no habría clique, y menos un Clique-Corte. Entonces cada vez que el algoritmo GA genera un nuevo individuo $\mathcal{X}$, éste es verificado por la función $isClique(\mathcal{X})$. Si no pasa la verificación, un clique es construido tomando como conjunto inicial de nodos, los que ese individuo trae como solución. El resultado de esta construcción es un nuevo individuo $\mathcal{X}$ que aplica a la premisa $isClique(\mathcal{X})$.

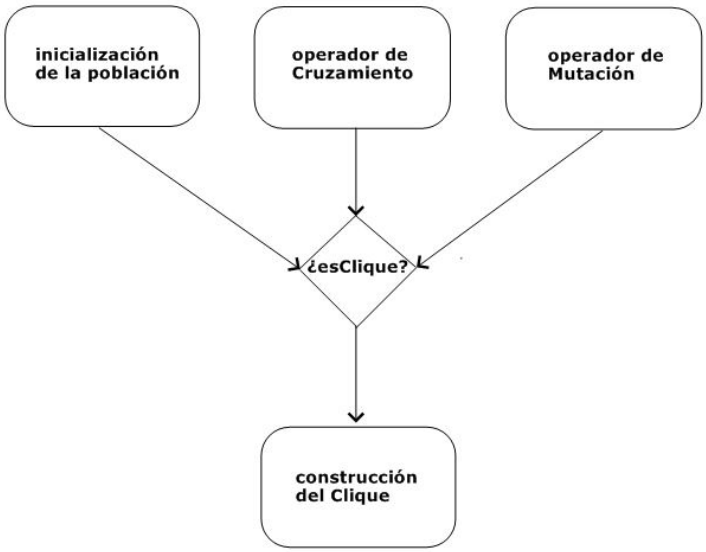

Figura 6.7: Diagrama de corrección de las soluciones.

Un individuo es generado en dos diferentes momentos de la evolución, ver Figura 6.7. Primero, cuando la población es creada e inicializada y luego cuando se recombinan las características de 2 individuos para crear uno nuevo o cuando se aplica mutación sobre un individuo obteniendo uno nuevo a partir de la transformación. Cada vez que un nuevo individuo es creado o generado, este debe ser una solución factible para el problema. Entonces para asegurar esta propiedad, contamos con el siguiente procedimiento de construcción, que es

una adaptación del procedimiento de la etapa de construcción del GRASP visto con anterioridad.

Como el desarrollo se ha realizado en el marco del paradigma de orientación a objetos, reutilizando las definiciones y el encapsulado de conceptos y comportamientos definido por el proyecto MALVA. El Algoritmo 8 es una transformación sobre el individuo $\mathcal{X}$, es decir, una operación de la propia clase *Solution*. Razón por la cual no utiliza mas información que la almacenada en la instancia de clase, como lo es el grafo $\mathcal{G}$, con sus nodos, aristas y adyacencias. El resultado de aplicar esta operación sobre el individuo $\mathcal{X}$, es un individuo transformado en una solución factible del problema, es decir en un clique.

Como se puede observar en la línea 4 del algoritmo 8 se selecciona de forma aleatoria un alelo del individuo para analizar. En la línea 6 se lo incluye en el clique en construcción $\mathcal{C}'$ y se evalúa si el corte generado por este nuevo clique $\mathcal{C}'$ es mayor al corte generado por $\mathcal{C}$. Si éste es el caso, se sustituye el clique $\mathcal{C}$ por $\mathcal{C}'$ y se actualiza la variable que registra la cantidad de intentos de mejoras, asignándole el valor máximo. En caso que el nuevo clique no mejore el corte se decrementa la variable de mejora y el clique $\mathcal{C}$ no cambia. Al finalizar la construcción del clique en la línea 11, se actualiza al individuo $\mathcal{X}$ quien ahora es una solución factible del problema.

Algorithm 8 CLIQUE CONSTRUCTION

Input: $\mathcal{X}$
Output: $\mathcal{X}$

1: $\mathcal{C} \leftarrow \emptyset$
2: $improving =$ MAX_ATTEMPTS
3: **while** $improving > 0$ **do**
4: $\quad i \leftarrow selectRandom(\mathcal{X})$
5: $\quad \mathcal{C}' \leftarrow [\mathcal{C} \cap N(i)] \cup \{i\}$
6: $\quad$ **if** $|E'(\mathcal{C}')| > |E'(\mathcal{C})|$ **then**
7: $\quad\quad \mathcal{C} \leftarrow \mathcal{C}'$
8: $\quad\quad improving \leftarrow$ MAX_ATTEMPTS
9: $\quad$ **else**
10: $\quad\quad improving \leftarrow improving - 1$
11: $X \leftarrow C$

6.3.7. Inicialización de Soluciones

En la etapa de inicialización de la población, cada individuo es generado de forma aleatoria y luego es ajustado a un subgrafo completo, sobre la base de sus nodos iniciales.

Notar que, la característica aleatoria de este procedimiento más la representación de la solución posibilitan la generación de individuos que no se correspondan con ningún grafo, incluso un *individuo vacío* para ese problema.

En la línea 3 del Algoritmo 9 presentado a continuación, se subsana el caso de la solución vacía por medio una función que sortea n nodos y luego les asigna el valor 1. En la línea 4, se realiza la validación de la estructura del individuo. Una vez más, sino corresponde con una solución factible se construye un clique en la línea 5 a partir de los nodos de la solución $\mathcal{X}$. Al igual que en la construcción del clique, este algoritmo es una transformación sobre el individuo $\mathcal{X}$, un método de instancia invocado sobre un individuo particular.

Algorithm 9 FEASIBLE SOLUTION INIT

Input: $\mathcal{X}$
Output: $\mathcal{X}$

1: $\mathcal{X} \leftarrow randomInit(\mathcal{X})$
2: **while** $isEmpty(\mathcal{X})$ **do**
3: $\quad \mathcal{X} \leftarrow initYRandomNodes(\mathcal{X})$
4: **if** $not(isClique(\mathcal{X}))$ **then**
5: $\quad \mathcal{X} \leftarrow cliqueConstruction(\mathcal{X})$

6.3.8. Evaluación de Soluciones

Cuando se diseña una solución en el marco de un GA, una decisión que incide fuertemente el éxito del algoritmo implementado, es el procedimiento a aplicar cuando una solución es *NO factible*. En general, los procedimientos adoptados son: descartar, penalizar y corregir. Descartar, lleva tiempo de evaluación para luego desechar; y para penalizar, al tiempo de evaluación se le debe sumar algún algoritmo que determine cuán mala es la solución para determinar el valor de su pena.

En el caso que nos compete, dadas las características inherentes del problema que sólo trabaja con subgrafos cliques, el procedimiento acertado es corregir y para ello se utiliza el algoritmo detallado en la subsección 6.3.6.

El tiempo empleado es similar a la opción de penalizar, pero con un enfoque constructivo ya que se evalúa y se corrige dándole una nueva oportunidad al individuo, enriqueciendo así la población del problema.

Por lo cual, antes de calcular el fitness o la adecuación al problema de una solución en particular, el procedimiento se asegurará de que la solución es factible y si esto no ocurre, la corregirá, construyendo una sobre ella.

Seguidamente, se presenta el algoritmo 10 de evaluación de fitness sobre una solución factible. Allí se puede apreciar que en las líneas 5-7 se realiza el cálculo de la función de fitness propiamente dicho. Primero se realiza la suma de todos los grados de los vértices que pertenecen a la solución y luego a ésta suma total se le resta el grado interno de los nodos pertenecientes al clique $\mathcal{C}$.

Algorithm 10 FITNESS CALCULATION OVER FEASIBLE SOLUTION

Input: $\mathcal{X}$, $\mathcal{G}$
Output: ***fitness***

1: **if** $not(isClique(\mathcal{X}))$ **then**
2: **while** $isEmpty(\mathcal{X})$ **do**
3: $\mathcal{X} \leftarrow initYRandomNodes(\mathcal{X})$
4: $\mathcal{X} \leftarrow cliqueConstruction(\mathcal{X}, \mathcal{G})$
5: **for** $i = 1$ **to** $|\mathcal{X}|$ **do** $\forall i \in clique(\mathcal{X})$
6: $fitness+ = grado(i)$
7: ***fitness*** $= fitness - |\mathcal{C}| * |\mathcal{C} - 1|$

Capítulo 7

Resultados Obtenidos

Este capítulo describe el procedimiento de validación del resultado brindado por el algoritmo y el ajuste de los parámetros utilizados por el mismo. Así como los resultados obtenidos sobre las instancias de prueba seleccionadas.

7.1. Validación del Algoritmo

Tanto para realizar la validación del algoritmo diseñado, como el ajuste de parámetros y las pruebas, se tomaron las instancias de grafos de las DIMACS de Clique. Para éstas instancias, se conoce la mejor solución factible hallada en el trabajo realizado por P. Martins y H. Ramalhinho [31]. La mejor solución factible refiere a la cantidad de aristas inducida por el corte del conjunto de nodos que conforman un clique. De aquí en más cuando se hable de corte máximo se estará haciendo referencia a la mejor solución factible reportada por éstos investigadores.

En una primera etapa, se utilizó una instancia pequeña y conocida, un grafo de 14 nodos para validar el algoritmo, Figura 7.1. Sobre este grafo se verificó que se alcanzara la solución conocida. Luego se tomaron algunos grafos con mayor cantidad de nodos alrededor de 100, para verificar que administrara correctamente los recursos de tiempo y memoria, ya que fue implementado en MALVA sobre *C++* y en una PC portátil con solo 3GB de memoria.

Una vez validado su funcionamiento con los operadores evolutivos y los valores de probabilidad de los parámetros por defecto, se procede a realizar el ajuste de los mismos.

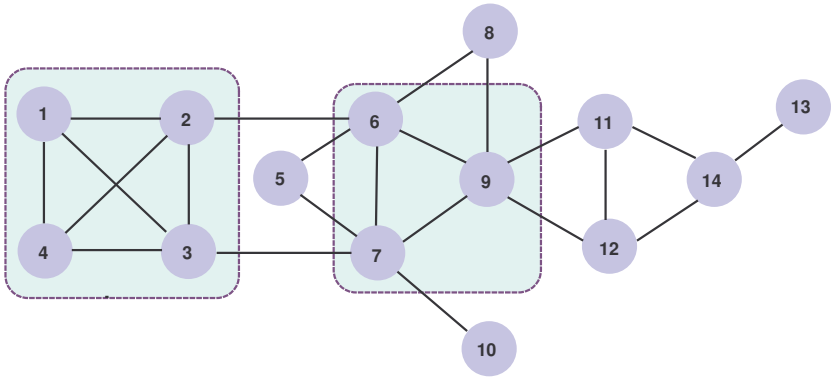

Figura 7.1: Grafo de 14 nodos con solución conocida.

7.2. Ajuste de Parámetros

Por ser una técnica estocástica, la etapa de ajuste de parámetros es un problema en sí mismo. Se deben realizar corridas atendiendo a cada posible configuración de ellos y repetirlas por cada instancia. Es un arduo trabajo que lleva horas de registro, preparación de datos, ejecución del algoritmo n veces,y luego aplicar un Test Estadístico para determinar cual es la mejor disposición de parámetros.

La elección de las instancias de grafos es un balance entre la cantidad de nodos, la cantidad de aristas y la densidad de los mismos. Con esta combinación se busca que sean lo más representativas posibles del escenario real donde operarán, ya que en Análisis de Canasta de Mercado o Análisis del Carrito de Compra (MBA), se trabaja con grafos potencialmente grandes y densos simulando escenarios cercanos a los reales.

La caracterización de las instancias seleccionadas para la calibración se puede apreciar en la siguiente tabla, donde presentamos el nombre de la instancia, la cantidad de vértices $|V|$, la cantidad de aristas $|E|$, la densidad y la cantidad de aristas del conjunto de corte generado por el clique $\mathcal{C}$.

Instancia	$\|V\|$	$\|E\|$	**Densidad**	$\|E(C)\|$
p_hat300-1	300	10933	0.244	789
MANN_a9	45	918	0.9273	412
keller4	171	9435	0.649	1140

Tabla 7.1: Características de instancias para ajuste de parámetros.

Los parámetros sobre los cuales trabajar son; probabilidad del operador de cruzamiento, probabilidad del operador de mutación y tamaño de la población. Los valores utilizados para cada parámetro, se lista a continuación. Es recomendado, visualizar la subsección de operadores del Capítulo 5 de solución 6.3.3 donde se justifica la prioridad del cruzamiento sobre la mutación.

- operador de cruzamiento $\{0.6, 0.8, 0.9\}$
- operador de mutación de $\{0.1, 0.01, 0.005\}$
- tamaño de la población $\{50, 100, 200\}$

La calibración de los parámetros se realizó de la siguiente manera: primero se fijaron los parámetros de los operadores de cruzamiento y mutación al valor medio, y se realizaron 30 ejecuciones para cada una de las 3 instancias, variando el parámetro de tamaño de la población entre $\{50, 100, 200\}$.

Al aplicar el test estadístico, denominado *Test de Rangos* el valor que corresponde a tamaño de población 50, queda descartado. Los 2 restantes valores registran rangos similares, 1.4 para tamaño 100 y 1.0 para tamaño 200; por lo cual se decide mantener ambos valores para calibrar los parámetros de probabilidad de ocurrencia de los operadores evolutivos.

El segundo parámetro tratado fue el operador de cruzamiento y la evaluación mediante *Test de Rangos* descartó el valor 0.6, por lo cual siguieron con chance las probabilidades 0.8 y 0.9. El tercer y último parámetro fue el operador de mutación. Para ello, se aplicó el test sobre todas las combinaciones de los 3 parámetros; exceptuando las combinaciones con valores que ya habían sido descartados como: tamaño de *población* $= 50$ y probabilidad de *cruzamiento* $= 0.6$. Se obtuvieron como resultado los siguientes valores: tamaño de la población $pop = 200$, probabilidad de cruzamiento $pc = 0.8$ y probabilidad de mutación $pm = 0.1$.

Parámetro	**Valor**
tamaño población	200
prob. cruzamiento	0.8
prob. mutación	0.1

Tabla 7.2: Resultado de la calibración.

7.3. Resultados del Algoritmo Genético

Antes de presentar los resultados, debemos puntualizar que la naturaleza no determinística de ésta técnica hace que 2 ejecuciones idénticas sobre la misma instancia, puedan retornar como resultado óptimo diferentes valores. Es por ello que para reportar resultados se deben realizar por lo menos 30 ejecuciones sobre la misma instancia y misma configuración de parámetros para luego reportar como solución el promedio de los valores óptimos obtenidos y el promedio de los tiempos de ejecución empleados para obtener esos resultados.

Con este procedimiento, se trabajó en cada una de las instancias de grafos que fueron tomadas como casos de prueba, listadas en la Tabla 7.3. Allí se muestra, la cantidad de nodos, aristas y la densidad del grafo para contextualizar la prueba, además del mejor valor del corte conocido hasta ahora. Como resultado del GA se reportan el valor *óptimo promedio* y el *mejor valor óptimo* alcanzado en dichas ejecuciones. Así como el tiempo promedio utilizado para alcanzar los resultados.

Características Grafo					**Resultado Obtenido**			GAP
Instancias	$\|V\|$	$\|E\|$	Densidad	$\|E(C)\|$	Mejor	Prom.	T(s) prom.	(%)
c-fat200-1	200	1534	0.071	**81**	**81**	81	6.4	0.0
c-fat200-2	200	3235	0.163	**306**	**306**	306	7.5	0.0
c-fat200-5	200	8473	0.426	**1892**	**1892**	1892	12.5	0.0
c-fat500-1	500	4459	0.036	**110**	**110**	110	16.15	0.0
c-fat500-2	500	9139	0.073	**380**	**380**	380	14.3	0.0
c-fat500-5	500	23191	0.186	**2304**	**2304**	2304	20.36	0.0
c-fat500-10	500	46627	0.374	**8930**	**8930**	8930	32.59	0.0
p_hat300-2	300	21928	0.489	**4637**	**4637**	4633.40	171.9	0.0
p_hat300-3	300	33390	0.744	**7740**	**7740**	7387.27	279.8	0.0
c125_9	125	69632	0.899	**2766**	**2766**	2737.2	5.0	0.0
keller5	776	225990	0.752	**15184**	**13120**	12382	50.57	0.16
MANN_a27	378	70551	0.990	**31284**	**30570**	30405	46.49	0.02

Tabla 7.3: Resultados obtenidos para el Máximo Clique-Corte.

Se puede observar en la Tabla 7.3 que el algoritmo encuentra el mejor resultado conocido en 10 de los 12 casos y que sus tiempos computacionales son estables, es decir se mantienen dentro de los 4.5 minutos a medida que el tamaño y la densidad de los grafos crecen. Por ejemplo, en el caso de *keller5* un grafo de densidad 0.752, le toma menos de un minuto encontrar una solución *bastante cercana* a la mejor solución conocida, siendo éste GAP (0.16 %) el

más notorio de las pruebas realizadas. La fórmula de cálculo del GAP es la siguiente: $GAP = (\frac{|Mejor - E(\mathcal{C})|}{|E(\mathcal{C})|}) \times 100$.

7.4. Comparación con otras soluciones

La solución implementada en este trabajo, encuentra los mejores resultados existentes en 10 de los 12 casos de prueba y los tiempos computacionales cuando las instancias empiezan a crecer en nodos o en densidad son menores a los obtenidos hasta ahora por cualquier otra solución. Por ejemplo, para el caso del grafo *keller5*, el tiempo es menor a 1 minuto mientras que en la solución existente hasta ahora, GRASP/VND es de 20 minutos. Si bien es menos precisa, se demora menos tiempo en obtener un valor óptimo aproximado al global. Las ejecuciones fueron realizadas en una PC portátil personal (Intel Core i5, 2.26 Ghz, 3GB RAM); un hardware claramente inferior al utilizado por las demás soluciones (Intel Core i7, 2.88 Ghz, 8GB RAM). Sin embargo, los tiempos obtenidos son sorprendentes, por lo que podemos afirmar que los resultados son muy alentadores.

		GRASP/VND		**Algoritmo Genético**		GAP
Instancias	$\|E(C)\|$	$\|E(C)\|$ prom.	$T(s)$ prom.	$\|E(C)\|$ prom.	$T(s)$ prom.	(%)
c-fat200-1	81	81	0.37	81	6.4	0.0
c-fat200-2	306	306	0.81	306	7.5	0.0
c-fat200-5	1892	1892	4.94	1892	12.5	0.0
c-fat500-1	110	110	2.46	110	16.15	0.0
c-fat500-2	380	380	5.83	380	14.3	0.0
c-fat500-5	2304	2304	10.85	2304	20.36	0.0
c-fat500-10	8930	8930	65.74	8930	32.59	0.0
p_hat300-2	4637	4636.2	3659.39	4633.40	171.9	≈0.0
p_hat300-3	7740	7726.8	3992.42	7387.27	279.8	0.04
c125_9	2766	2766	253.25	2737.2	5.0	0.01
keller5	15184	15183.24	1167.64	12382	50.57	0.18
MANN_a27	31284	31244.10	548.54	30405	46.49	0.03

Tabla 7.4: Comparación con otras soluciones.

La comparación numérica de los resultados se refleja en la Tabla 7.4, tomando en cuenta los valores promedio para hacerla más precisa. Ésto es por la forma en que operan los GA, donde para validar una solución se deben realizar entre 30 y 50 corridas de la misma instancia y la misma parametrización.

Allí para cada instancia se muestra el mejor valor conocido, los resultados promedios para la solución basada en GRASP/VND y la solución basada en GA. Además de una columna GAP que visualiza de manera contundente el rendimiento del algoritmo desarrollado. Este valor fue construido de la siguiente manera:
$GAP = (\frac{|AG - GRASP/VND|}{|GRASP/VND|}) \times 100.$

7.5. Comparación por dimensiones

Para visualizar los resultados de forma diferente, utilizaremos las dimensiones del *valor óptimo* y el *tiempo* por separado. Para ello, se mostrarán gráficas elaboradas sobre los datos recortados por la dimensión que se esté tratando.

En primer lugar, para la dimensión de los valores óptimos obtenidos, es decir el mejor valor de corte hallado, se utilizó la siguiente Tabla 7.5, abreviación de 7.4.

	ILS	**GRASP/VND**	**GA**
Instancias	$\|E(C)\|$	$\|E(C)\|$ prom.	$\|E(C)\|$ prom.
c-fat200-1	81	81	81
c-fat200-2	306	306	306
c-fat200-5	1892	1892	1892
c-fat500-1	110	110	110
c-fat500-2	380	380	380
c-fat500-5	2304	2304	2304
c-fat500-10	8930	8930	8930
p_hat300-2	4637	4636.2	4633.40
p_hat300-3	7740	7726.8	7387.27
c125_9	2766	2766	2737.2
keller5	15184	15183.24	12382
MANN_a27	31284	31244.10	30405

Tabla 7.5: Comparación con otras soluciones, dimensión valor óptimo.

Sobre ella se construyó la gráfica que se puede apreciar en la Figura 7.2.

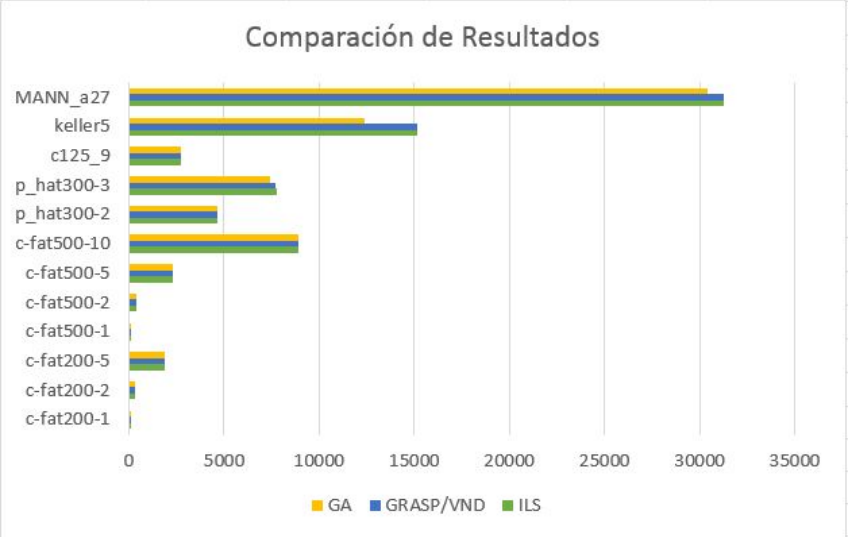

Figura 7.2: Comparación de Mejores Valores obtenidos entre soluciones.

En segundo lugar, para la dimensión del tiempo empleado por la solución, se utilizó la siguiente Tabla 7.6, abreviación de 7.4.

	GRASP/VND	**GA**
Instancias	$T(s)$ prom.	$T(s)$ prom.
c-fat200-1	0.37	6.4
c-fat200-2	0.81	7.5
c-fat200-5	4.94	12.5
c-fat500-1	2.46	16.15
c-fat500-2	5.83	14.3
c-fat500-5	10.85	20.36
c-fat500-10	65.74	32.59
p_hat300-2	3659.39	171.9
p_hat300-3	3992.42	279.8
c125_9	253.25	5.0
keller5	1167.64	50.57
MANN_a27	548.54	46.49

Tabla 7.6: Comparación con otras soluciones, dimensión tiempo empleado.

Luego, sobre ella se construyó la gráfica que se puede apreciar en la Figura 7.3.

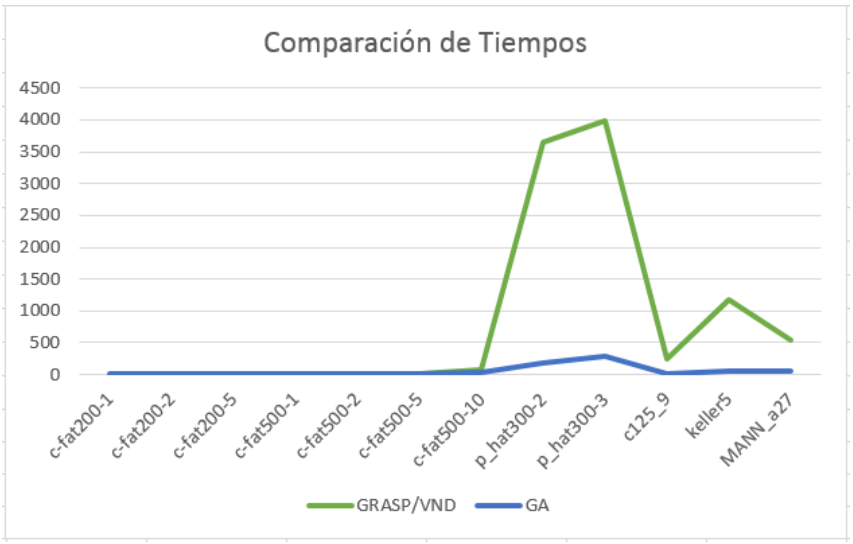

Figura 7.3: Comparación de tiempos empleados entre soluciones.

Capítulo 8

Conclusiones y Trabajo Futuro

Este capítulo detalla las conclusiones a las cuales fue decantando el trabajo realizado en ésta tesis. Se realizó un trabajo comprometido con el uso óptimo de los recursos de cómputo buscando una solución con un buen rendimiento. A la vez que se fue delimitando el alcance de la solución e identificando áreas donde se debería profundizar la temática.

8.1. Conclusiones

En este trabajo se presentó y formalizó un problema recientemente introducido en la comunidad científica, denominado $Maximum\ Cut - Clique\ MCC$, se brindó una prueba de su complejidad y se describieron las heurísticas existentes al día de hoy para su resolución. A la vez, que se desarrolló la primera solución basada en Algoritmos Genéticos que resultó ser competitiva con las soluciones existentes hasta el momento de realizada esta investigación, también basadas en metaheurísticas.

Como primera medición de la efectividad de éste algoritmo, podemos afirmar empíricamente que se alcanzan las mejores soluciones factibles conocidas, en tiempos extraordinariamente pequeños (menor a medio minuto), en grafos de hasta 500 nodos con una densidad inferior al 0.5. Por otro lado, si consideramos grafos más densos, es decir, con mayor cantidad de aristas, se alcanzan resultados cercanos al óptimo. Es por ello que, podemos afirmar que la solución desarrollada es estable y competitiva. Ver comparación realizada anteriormente en 7.5.

Sin embargo, en algunos casos se observó que la mejor solución se alcanza-

ba en generaciones tempranas, muy inferiores a las 1000. Este comportamiento es indicio de una convergencia prematura del algoritmo. Es decir, el algoritmo queda atrapado en óptimos locales y no logra explorar nuevos espacios de búsqueda. La principal razón de este comportamiento es que rápidamente se filtran los individuos más aptos y los operadores evolutivos no logran generar individuos con nuevas características, haciendo que el algoritmo se quede *escalando la colina* por las 1000 generaciones estipuladas.

8.2. Trabajo Futuro

El algoritmo alcanza las mejores soluciones factibles conocidas en un porcentaje alto de ejecuciones y su eficiencia computacional es bastante buena comparada con otras soluciones existentes, ver 7.4. Sin embargo, se identifican mejoras en lo que respecta a la performance en general. Se abren dos líneas de trabajo; una interna al algoritmo y otra externa referente a la plataforma de hardware empleado.

En la línea interna, podemos considerar las siguientes alternativas:

- evaluación de otros operadores de cruzamiento y mutación: Cruzamiento Uniforme y Mutación Invertida con Desplazamiento
- ajuste de parámetros del algoritmo según características de las distintas familias de grafos, considerar los más densos separados de los más dispersos.

El primer punto hace referencia a la utilización de operadores más disruptivos que logren preservar la diversidad en la población, como los utilizados en [43] [1]. Esta medida contribuye a que el algoritmo explore nuevos espacios de búsqueda salvando el problema actual de quedarse atrapado en óptimos locales. En consecuencia, lograría mayor precisión en los resultados.

Mientras que el segundo punto sugiere tratar a cada familia de grafos con parámetros diferentes. Ésto es, realizar un ajuste específico para instancias de grafos muy densos, otro para grafos espaciados y un tercer ajuste para los grafos que no caen en las categorías anteriores. Es un análisis similar al realizado por Martins en [31] en su formulación exacta al problema y tiene sustento en los resultados obtenidos por el test de Rangos aplicados en el Ajuste realizado

[1] El GA más rápido al día de hoy, para calcular el Máximo Clique, según los investigadores Zhang, Wang y Zhan en 2014.

en la sección 7.2. Para las tres instancias seleccionadas que intentaron ser lo mas representativas posibles de todos los casos a considerar, se detectaron valores similares que dificultaban la selección. En este sentido es que creemos que un tratamiento específico puede ser beneficio para lograr precisión en los resultados.

En lo que respecta a la línea externa de trabajo, podríamos considerar la utilización de PC portátiles con mejores prestaciones, como lo es una pc con un procesador Intel i7 y 8GB de RAM ya que disponer de más gigas de memoria permitiría extender la prueba a instancias con más nodos y más aristas. Otra alternativa sería paralelizar la ejecución del algoritmo y con esto lograríamos mejorar la capacidad de cómputo y atacar instancias con más de 1500 nodos y grafos con densidad cercana a 1.0. Además, sería esperable que una solución paralelizada, debería ejecutar en la plataforma de cómputo de alto desempeño como la que disponemos en *ClusterUY*, el centro Nacional de Supercomputación (ClusterUY).

Referencias bibliográficas

[1] Aguinis, H., Forcum, L. E., and Joo, H. (2013). Using market basket analysis in management research. *Journal of Management*, 39(7):1799–1824.

[2] Amuthan, A. and Thilak, K. D. (2016). Survey on tabu search meta-heuristic optimization. In *2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES)*, pages 1539–1543.

[3] Bader, G. D. and Hogue, C. W. V. (2003). An automated method for finding molecular complexes in large protein interaction networks. *BMC Bioinformatics*, 4:2.

[4] Bellare, M., Goldreich, O., and Sudan, M. (1998). Free bits, pcps, and nonapproximability—towards tight results. *SIAM Journal on Computing*, 27(3):804–915.

[5] Bomze, I. M., Budinich, M., Pardalos, P. M., and Pelillo, M. (1999). The maximum clique problem. In *Handbook of combinatorial optimization*, pages 1–74. Springer.

[6] Born, J. (1980). Schwefel, h.-p., numerische optimierung von computer-modellen mittels der evolutionsstrategie. mit einer vergleichenden einführung in die hill-climbing- und zufallsstrategien. (isr 26) basel-stuttgart, birkhäuser verlag 1977. 390 s., sfr. 48,–. *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik*, 60(5):272–272.

[7] Bourel, M., Canale, E., Robledo, F., Romero, P., and Stábile, L. (2018a). Complexity and heuristics for the max cut-clique problem. In *International Conference on Variable Neighborhood Search*, pages 28–40. Springer.

[8] Bourel, M., Canale, E., Robledo, F., Romero, P., and Stábile, L. (2018b). A grasp/vnd heuristic for the max cut-clique problem. In *International Conference on Machine Learning, Optimization, and Data Science*, pages 357–367. Springer.

[9] Bourel, M., Canale, E., Robledo, F., Romero, P., and Stábile, L. (2019). Complexity and Heuristics for the Weighted Max Cut-Clique Problem. *International Transactions in Operational Research*. Unpublished.

[10] Brohée, S. and van Helden, J. (2006). Evaluation of clustering algorithms for protein-protein interaction networks. *BMC Bioinformatics*, 7(1):488.

[11] Bruinsma, G. and Bernasco, W. (2004). Criminal groups and transnational illegal markets. *Crime, Law and Social Change*, 41(1):79–94.

[12] Cavique, L. (2007). A scalable algorithm for the market basket analysis. *Journal of Retailing and Consumer Services*, 14(6):400–407.

[13] Cook, S. A. (1971). The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, STOC '71, pages 151–158, New York, NY, USA. ACM.

[14] De Jong, K. A. (1975). Analysis of the behavior of a class of genetic adaptive systems.

[15] Duarte, A., Mladenović, N., Sánchez-Oro, J., and Todosijević, R. (2016). *Variable Neighborhood Descent*, pages 1–27. Springer International Publishing, Cham.

[16] Fagundez, G. (último acceso Octubre 2019). The malva project. a framework of artificial intelligence en c++. GitHub Inc. https://themalvaproject.github.io.

[17] Fogel, D. (1998). *Evolutionary Computation: The Fossil Record*. Wiley.

[18] Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, NY, USA.

[19] Gendreau, M., Potvin, J.-Y., et al. (2010). *Handbook of metaheuristics*, volume 2. Springer.

[20] Glover, F. and Laguna, M. (1997). *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, USA.

[21] Goldberg, D. E. (1989). *Genetic algorithms in search*. Addison Wesley Publishing Co. Inc.

[22] Gouveia, L. and Martins, P. (2015). Solving the maximum edge-weight clique problem in sparse graphs with compact formulations. *EURO Journal on Computational Optimization*, 3(1):1–30.

[23] Grosso, A., Locatelli, M., and Pullan, W. (2008). Simple ingredients leading to very efficient heuristics for the maximum clique problem. *Journal of Heuristics*, 14(6):587–612.

[24] Henzinger, M. and Lawrence, S. (2004). Extracting knowledge from the world wide web. *Proceedings of the National Academy of Sciences*, 101(suppl 1):5186–5191.

[25] Hodges, A. (2013). Alan turing. In Zalta, E. N., editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, winter 2013 edition.

[26] Holland, J. H. (1975). Adaption in natural and artificial systems.

[27] Hüffner, F., Komusiewicz, C., Moser, H., and Niedermeier, R. (2008). Enumerating isolated cliques in synthetic and financial networks. In Yang, B., Du, D.-Z., and Wang, C. A., editors, *Combinatorial Optimization and Applications*, pages 405–416, Berlin, Heidelberg. Springer Berlin Heidelberg.

[28] Karp, R. M. (1972). Reducibility among combinatorial problems. In Miller, R. E. and Thatcher, J. W., editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press.

[29] Lourenço, H. R., Martin, O. C., and Stützle, T. (2019). Iterated local search: Framework and applications. In *Handbook of metaheuristics*, pages 129–168. Springer.

[30] Martins, P. (2012). Cliques with maximum/minimum edge neighborhood and neighborhood density. *Computers & Operations Research*, 39(3):594–608.

[31] Martins, P., Ladrón, A., and Ramalhinho, H. (2014). Maximum cut-clique problem: ILS heuristics and a data analysis application. *International Transactions in Operational Research*, 22(5):775–809.

[32] Nesmachnow, S. (2004). Algoritmos genéticos paralelos y su aplicación al diseño de redes de comunicaciones confiables. Master's thesis, Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Julio Herrera y Reissig 565. Montevideo. Uruguay.

[33] Pullan, W. and Hoos, H. H. (2006). Dynamic local search for the maximum clique problem. *Journal of Artificial Intelligence Research*, 25:159–185.

[34] Raeder, T. and Chawla, N. V. (2011). Market basket analysis with networks. *Social network analysis and mining*, 1(2):97–113.

[35] Reeves, C. R. (2010). Genetic algorithms. In *Handbook of metaheuristics*, pages 109–139. Springer.

[36] Resende, M. G. and Ribeiro, C. C. (2016). *Optimization by GRASP - Greedy Randomized Adaptive Search Procedures.* Computational Science and Engineering. Springer-Verlag New York.

[37] Turing, A. (1936). On computable numbers with an application to the entscheidungsproblem. *Proceeding of the London Mathematical Society.*

[38] Valiant, L. G. (1979). Completeness classes in algebra. In *Proceedings of the eleventh annual ACM symposium on Theory of computing*, pages 249–261. ACM.

[39] Vent, W. (1975). Rechenberg, ingo, evolutionsstrategie — optimierung technischer systeme nach prinzipien der biologischen evolution. 170 s. mit 36 abb. frommann-holzboog-verlag. stuttgart 1973. broschiert. *Feddes Repertorium*, 86(5):337–337.

[40] Von, N. J. In Burks, A. (1966). *Goldstine, H. Theory of self-reproducing automata.* University of Illinois Press.

[41] Wu, Q. and Hao, J.-K. (2015). A review on algorithms for maximum clique problems. *European Journal of Operational Research*, 242(3):693–709.

[42] Younes, A., Elkamel, A., Areibi, S., and Lohi, A. (2009). Evolutionary algorithms: What, when, and how? *U.A.E.*

[43] Zhang, S., Wang, J., Wu, Q., and Zhan, J. (2014). A fast genetic algorithm for solving the maximum clique problem. In *2014 10th International Conference on Natural Computation (ICNC)*, pages 764–768. IEEE.

ANEXOS

Anexo 1

Instancias de Prueba

El siguiente listado contiene las características de las instancias de ajuste y de prueba que fueron utilizadas en esta tesis.

Instancias	Características de las instancias			
	$\|V\|$	$\|E\|$	Densidad	$\|E(C)\|$
c-fat200-1	200	1534	0.071	81
c-fat200-2	200	3235	0.163	306
c-fat200-5	200	8473	0.426	1892
c-fat500-1	500	4459	0.036	110
c-fat500-2	500	9139	0.073	380
c-fat500-5	500	23191	0.186	2304
c-fat500-10	500	46627	0.374	8930
p_hat300-1	300	10933	0.244	789
p_hat300-2	300	21928	0.489	4637
p_hat300-3	300	33390	0.744	7740
keller4	171	9435	0.649	1140
keller5	776	225990	0.752	15184
MANN_a9	45	918	0.9273	412
MANN_a27	378	70551	0.990	31284
c125_9	125	69632	0.899	236406

Tabla 1.1: Caracterización de las instancias de prueba.