



Universidad  
Rey Juan Carlos

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA  
INFORMÁTICA

GRADO EN INGENIERÍA DEL SOFTWARE

Curso académico 2019/2020

TRABAJO FIN DE GRADO

**BÚSQUEDA DEL CLIQUE DE RATIO MÁXIMO  
MEDIANTE EL ALGORITMO GRASP**

Autor:

José Miguel García Benayas

Tutores:

Dr. Jesús Sánchez-Oro Calvo

Dr. Alfonso Fernández Timón



*«Victory is always possible for the person who refuses to stop fighting»,  
Napoleon Hill*



## *Resumen*

En la presente memoria de este Trabajo Fin de Grado se trata el problema del clique de ratio máximo también llamado Maximal Ratio Clique Problem (MRCP), el cual consiste en encontrar un subgrafo completo o clique, de ratio máximo.

A través de esta memoria se intenta explicar, de una manera clara, como se ha abordado el problema desde el diseño hasta la implementación, pasando por el estudio y comprensión de la metaheurística empleada, para la búsqueda optima de una solución factible. La implementación se ha realizado mediante el algoritmo Greedy Randomized Adaptive Search Procedure (GRASP) o Procedimiento de Búsqueda Voráz Aleatoria y Adaptativa, el cual ofrece buenas soluciones en un tiempo de computo razonable.

Este problema es una derivación del clásico Problema del Clique Máximo o Maximal Clique Problem (MCP) que consiste en encontrar el clique máximo en un grafo.

El MRCP tiene aplicaciones tan actuales como el machine-learning o las redes sociales, ya que un clique es un grupo de personas con intereses afines y por ello se ha intentado implementar un algoritmo que trate los datos de la manera más eficiente posible, de forma que se agilice el tratamiento de estos posteriormente.



## *Agradecimientos*

Todo este trabajo no habría sido posible sin mi familia, sin su apoyo y cariño, en especial a Rebeca, la cual ha sido mi compañera de viaje y la luz que me ilumina.

Sin olvidar a mis amigos en la universidad, con quienes he disfrutado cada día en clase, y a mis tutores, Jesús y Alfonso, por darme la oportunidad de realizar este trabajo y de los que he aprendido mucho.





# Índice general

<b>Resumen</b>	<b>v</b>
<b>Agradecimientos</b>	<b>vii</b>
<b>1. Introducción</b>	<b>1</b>
1.1. Estructura de la memoria . . . . .	1
1.2. Motivación del problema . . . . .	2
1.3. Conceptos previos . . . . .	4
1.3.1. Optimización combinatoria . . . . .	4
1.3.2. Heurística y Metaheurística . . . . .	4
1.3.3. Clique . . . . .	7
1.4. Definición del problema . . . . .	9
1.4.1. Problema del clique de ratio máximo . . . . .	9
1.5. Estado del arte . . . . .	11
<b>2. Objetivos</b>	<b>13</b>
2.1. Objetivos principales . . . . .	13
2.2. Objetivos secundarios . . . . .	13
<b>3. Descripción algorítmica</b>	<b>15</b>
3.1. Metaheurística GRASP . . . . .	15
3.1.1. Fase constructiva . . . . .	16
3.1.2. Fase de búsqueda . . . . .	18
<b>4. Descripción informática</b>	<b>21</b>
4.1. Diseño . . . . .	21
4.2. Implementación . . . . .	24
4.3. Metodología empleada . . . . .	26

<b>5. Resultados</b>	<b>29</b>
5.1. Recursos utilizados . . . . .	29
5.1.1. Descripción de la máquina utilizada . . . . .	29
5.1.2. Instancias utilizadas . . . . .	30
5.2. Análisis de los resultados . . . . .	30
5.2.1. Experimentos preliminares . . . . .	31
5.2.2. Experimento final . . . . .	33
<b>6. Conclusiones</b>	<b>39</b>
6.1. Consecución de los objetivos . . . . .	39
6.2. Conocimientos adquiridos . . . . .	40
6.3. Líneas de desarrollo futuras . . . . .	40
<b>Bibliografía</b>	<b>43</b>

# Índice de figuras

1.1. Grafo relaciones en Facebook. . . . .	2
1.2. Grafo de telecomunicaciones. . . . .	3
1.3. Grafo sección del metro de Madrid. . . . .	3
1.4. Representación estándar y en grafo de la molécula del Tolueno. . . . .	3
1.5. Clasificación de metaheurísticas . . . . .	6
1.6. Diagrama de un grafo. . . . .	7
1.7. Cliques de cardinalidad 2 del grafo. . . . .	8
1.8. Cliques de cardinalidad 3 del grafo. . . . .	8
1.9. Clique máximo del grafo. . . . .	9
1.10. Diagrama de problemas NP. . . . .	10
1.11. Diagrama de problemas NP y NP-difícil. . . . .	11
4.1. Diagrama de clases del proyecto. . . . .	23
4.2. Ejemplo del sistema de tarjetas de Trello. . . . .	27



## Índice de tablas

5.1. Resultados de los experimentos preliminares con constructivo adyacentes.	31
5.2. Resultados de los experimentos preliminares con constructivo ratio. . . .	32
5.3. Resultados del experimento final. . . . .	34



# Listado de abreviaciones

**ACO** Ant Colony Optimization. 6

**AMS** Adaptive Multi-Start. 6

**API** Application Programming Interface. 2

**AT** Asynchronous Teams. 6

**CA** Cultural Algorithms. 6

**EDA** Estimation Distribution Algorithms). 6

**FANS** Fuzzy Adaptive Neighborhood Search. 6

**GA** Genetic Algorithms. 6

**GIL** Global Interpreter Lock. 41

**GLS** Guided Local Search. 6

**GRASP** Greedy Randomized Adaptive Search Procedure. 6, 15, 21, 25, 32, 33, 39, 40

**HC** Heuristic Concentration. 6

**IDE** Integrated Development Environment. 29

**ILS** Iterated Local Search. 6

**MA** Memetic Algorithms. 6

**MCP** Maximum Clique Problem. 9

**MRCP** Maximum Ratio Clique Problem. 10

**MSM** Multi-Start Methods. 6, 12

**MWCP** Maximum Weight Clique Problem. 10

**NM** Noising Methods. 6

**POPMUSIC** Partial OPTimization Metaheuristic Under Special Intensification Conditions. 6

**PR** Path Relinking. 6

**SA** Simulated Annealing. 6

**SI** Swarm Intelligence. 6

**SS** Scatter Search. 6

**TAM** Threshold Accepting Methods. 6

**TS** Tabu Search. 6

**VNS** Variable Neighborhood Search. 6, 12



# Capítulo 1

## Introducción

En este capítulo se presenta la motivación del problema, los conceptos previos que ayudarán al lector a entender mejor el desarrollo, siguiendo con la definición del problema, y por último, se muestra una revisión del estado del arte relacionado con este problema.

### 1.1. Estructura de la memoria

La memoria de este trabajo final de grado se estructura de la siguiente manera:

- Capítulo 1, en este capítulo se introduce el tema a tratar, describiendo el problema y el estado del arte del mismo, así como conceptos previos a tener en cuenta.
- Capítulo 2, en este capítulo se muestran los objetivos que se esperan alcanzar con este trabajo final de grado.
- Capítulo 3, en este capítulo se describe de manera algorítmica la metaheurística empleada para abordar el problema tratado.
- Capítulo 4, en este capítulo se explica el desarrollo de las funciones para obtener una solución al problema.
- Capítulo 5, en este capítulo se exponen los resultados recopilados durante el procesamiento del problema, así como el análisis de los mismos.
- Capítulo 6, en este capítulo se muestran las conclusiones finales obtenidas durante todo este trabajo final de grado.

## 1.2. Motivación del problema

En la actualidad, los datos se han convertido en una pieza fundamental en la vida cotidiana de las personas. Tanto su obtención como su posterior tratamiento son grandes retos que deben ser estudiados y, por supuesto, realizar esto de una manera eficiente y en el menor tiempo posible, es un reto aún mayor. A partir de esto se obtiene el término grafo, el cual es fundamental en este ámbito y que se define como la relación entre nodos o vértices mediante uniones llamadas aristas, y esto, es lo realmente interesante a estudiar, las relaciones existentes entre los distintos nodos.

Existen muchos casos en los que se puede realizar la obtención y tratamiento de estos datos, como por ejemplo, la red social Facebook<sup>1</sup>, que como se observa en la figura 1.1 define su grafo como las relaciones entre las personas y aficiones que les unen. Además ofrece una API<sup>2</sup> con la que poder interactuar con los datos que son recopilados en ella [9].

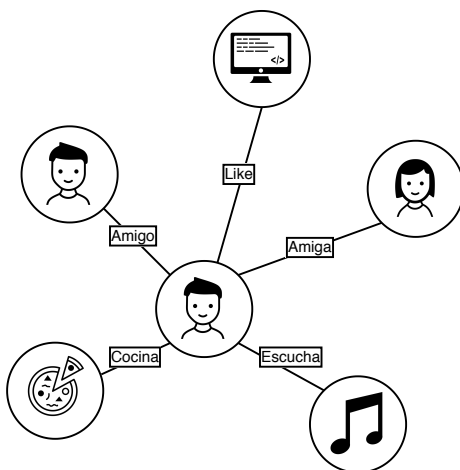


FIGURA 1.1: Grafo relaciones en Facebook.

Por otro lado, también muy cercano a la vida cotidiana de las personas, se encuentran las redes de telecomunicaciones, figura 1.2, que interconectan todos los aparatos electrónicos, que de una manera u otra se comunican entre ellos, y las redes de metro, figura 1.3, que permiten el movimiento de las personas en la ciudad y pueden ser estudiadas por uso, localización, etcétera.

<sup>1</sup><https://es-es.facebook.com/>

<sup>2</sup>Conjunto de métodos que forman parte de una librería y que a modo de capa de abstracción son publicados para que puedan ser usados en otros desarrollos software.

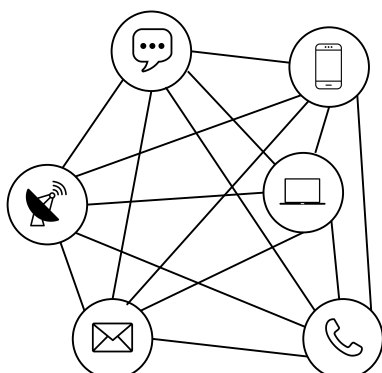


FIGURA 1.2: Grafo de telecomunicaciones.

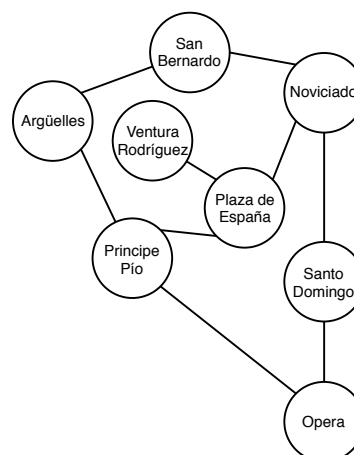


FIGURA 1.3: Grafo sección del metro de Madrid.

Y por último, una representación de grafo que es básico son las moléculas y los enlaces químicos [11], que se pueden representar como en la figura 1.4.

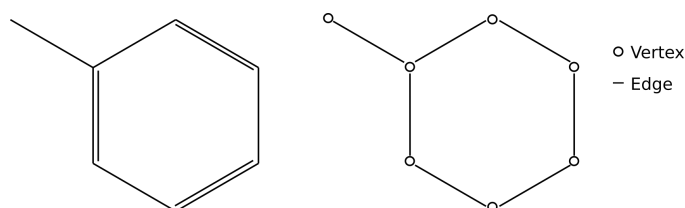


FIGURA 1.4: Representación estándar y en grafo de la molécula del Tolueno.

Fuente: <https://www.blopig.com/blog/2019/01/graph-based-methods-for-cheminformatics>

Estos ejemplos, tienen en común las conexiones entre todos los nodos que forman su estructura de grafo, y son muchas las aplicaciones e información que se obtiene de ellos como, nuevas terapias [2], visión computacional y reconocimiento de patrones [6], análisis del mercado bursátil para conocer nuevas formas de predicción de éxito en inversiones [18].

Por todo esto, es muy importante tratar los datos de forma adecuada y responsable, así como las relaciones entre ellos eficientemente. Para ello se han creado incluso base de

datos orientadas a grafos, como es el caso de Neo4j<sup>3</sup>, con el fin de entender las relaciones y así poder hallar nuevas técnicas y más posibilidades para afrontar problemas y resolverlos de una manera mucho más ágil.

## 1.3. Conceptos previos

Para comprender mejor todas las explicaciones que se van a exponer a lo largo del documento, se describen a continuación los conceptos más importantes.

### 1.3.1. Optimización combinatoria

La optimización combinatoria es el área, dentro de las matemáticas aplicadas, que se encarga de maximizar o minimizar una función en un espacio de soluciones, el cuál debe ser finito. Los problemas que pertenecen a este área tienen en común la dificultad de encontrar soluciones factibles, puesto que existen muchas posibles y alguna de estas es óptima.

Algunos casos conocidos son el problema de la mochila<sup>4</sup> o el problema del vendedor viajero<sup>5</sup>. Estos problemas tienen un planteamiento sencillo pero son difíciles de solucionar, y mediante ciertos algoritmos, como se explicará en la sección 1.3.2, se pueden obtener soluciones factibles en tiempos de cómputo reducidos.

En la actualidad es un campo con un gran crecimiento ya que “muchos de los problemas de la vida real pueden ser formulados mediante optimización combinatoria” [14].

### 1.3.2. Heurística y Metaheurística

Se define heurística, según el Diccionario de la Real Academia de la Lengua Española [8], como “técnica de la indagación y del descubrimiento” y “en algunas ciencias, manera de buscar la solución de un problema mediante métodos no rigurosos, como por tanteo, reglas empíricas, etc.”. Aplicándolo a temas científicos se define como el proceso de creación de medios, estrategias y principios para alcanzar un objetivo eficaz al problema dado [7]. El término heurística fue acuñado por George Polya en su libro

---

<sup>3</sup><https://neo4j.com/>

<sup>4</sup><https://www.sciencedirect.com/topics/computer-science/knapsack-problem>

<sup>5</sup><https://www.sciencedirect.com/topics/computer-science/travelling-salesman-problem>

“How to Solve It” [22], más tarde traducido a “Cómo plantear y resolver problemas” [21].

Añadiendo al término heurística el prefijo “meta”, procedente del griego que significa “más allá” o “nivel superior”, se puede definir metaheurística como el conjunto de procedimientos heurísticos combinados para obtener una solución a un problema que no tiene un algoritmo heurístico específico o su aplicación es ineficiente [16]. Este término lo acuñó Fred Glover en sus trabajos sobre búsqueda tabú en 1986 [12].

Existen una gran variedad de algoritmo metaheurísticos que se pueden clasificar de diferentes maneras según el enfoque, en la figura 1.5 se muestra una posible clasificación.

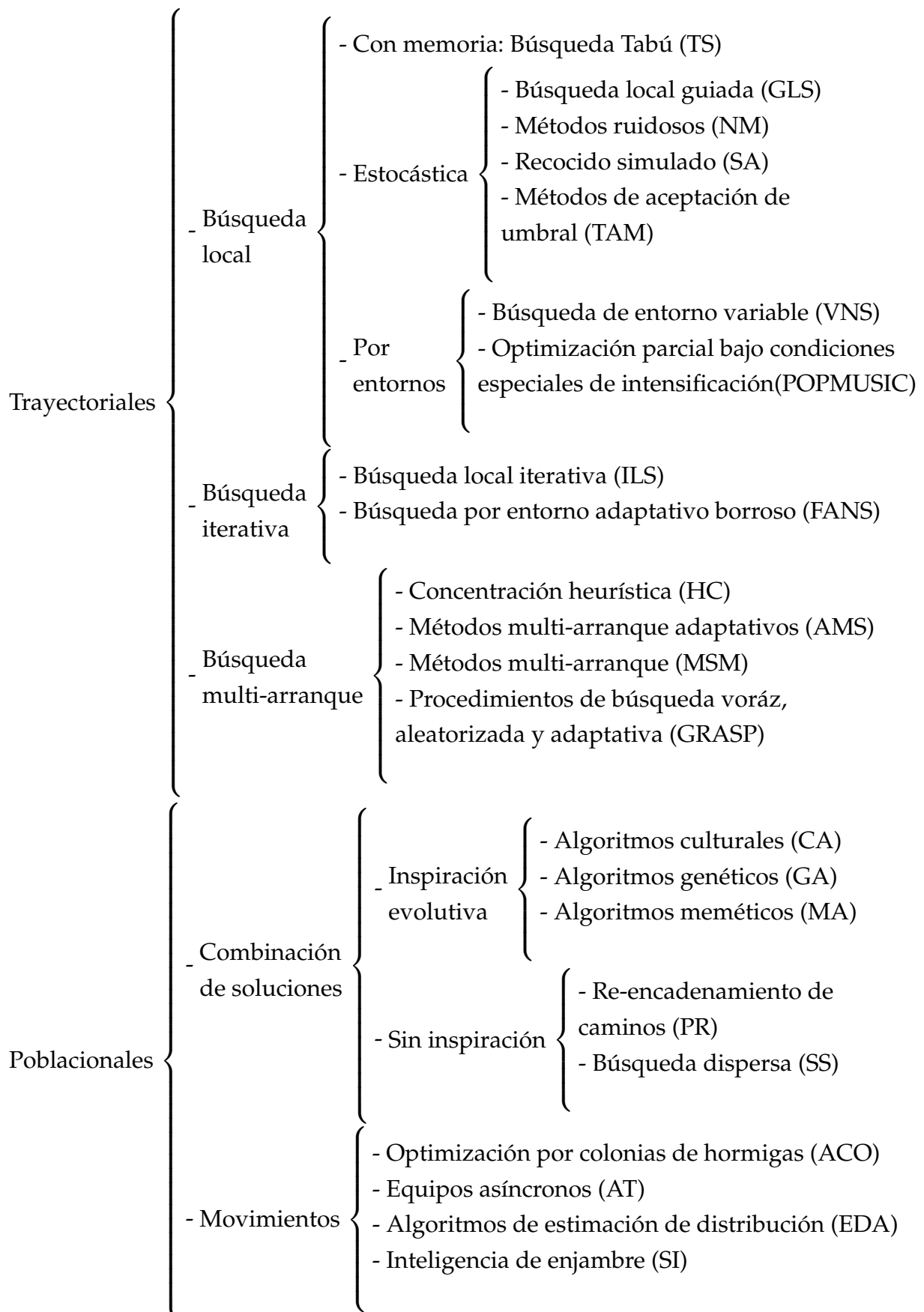


FIGURA 1.5: Clasificación de metaheurísticas

Dos conceptos a tener en cuenta en un algoritmo metaheurístico son la intensificación, que indica la exhaustividad con la que el algoritmo explora una región, y la diversificación, que indica como el algoritmo es capaz de explorar nuevas regiones del espacio de soluciones [1].

### 1.3.3. Clique

El término clique se puede asemejar a un grupo de personas, las cuales serían los nodos o vértices de un grafo, a las que les unen los mismos intereses por algún tema en concreto, estas uniones serían las aristas que unen los nodos del grafo [15].

En teoría de grafos, un clique, es el subgrafo perteneciente a un grafo, en el cuál todos sus nodos o vértices son adyacentes entre sí, es decir, todo par de nodos o vértices están conectados mediante una arista. En términos matemáticos se describe como, dado un grafo  $G = (V, E)$  donde  $V$  indica el conjunto de vértices del grafo y  $E$  indica el conjunto de aristas del grafo [31], un clique  $C$  se define como:

$$C \subseteq V(G) \wedge u, v \in C \wedge u \neq v \Rightarrow u, v \in E(G)$$

Mostrandolo de una manera gráfica se indica en la figura 1.6 un grafo compuesto por el conjunto de nodos  $V = \{A, B, C, D, E\}$ .

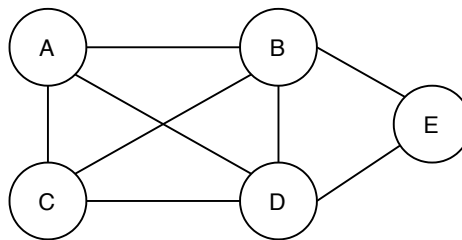


FIGURA 1.6: Diagrama de un grafo.

En este grafo de ejemplo se encuentran los cliques con cardinalidad igual a 2 como se indican resaltado en negro en la figura 1.7.

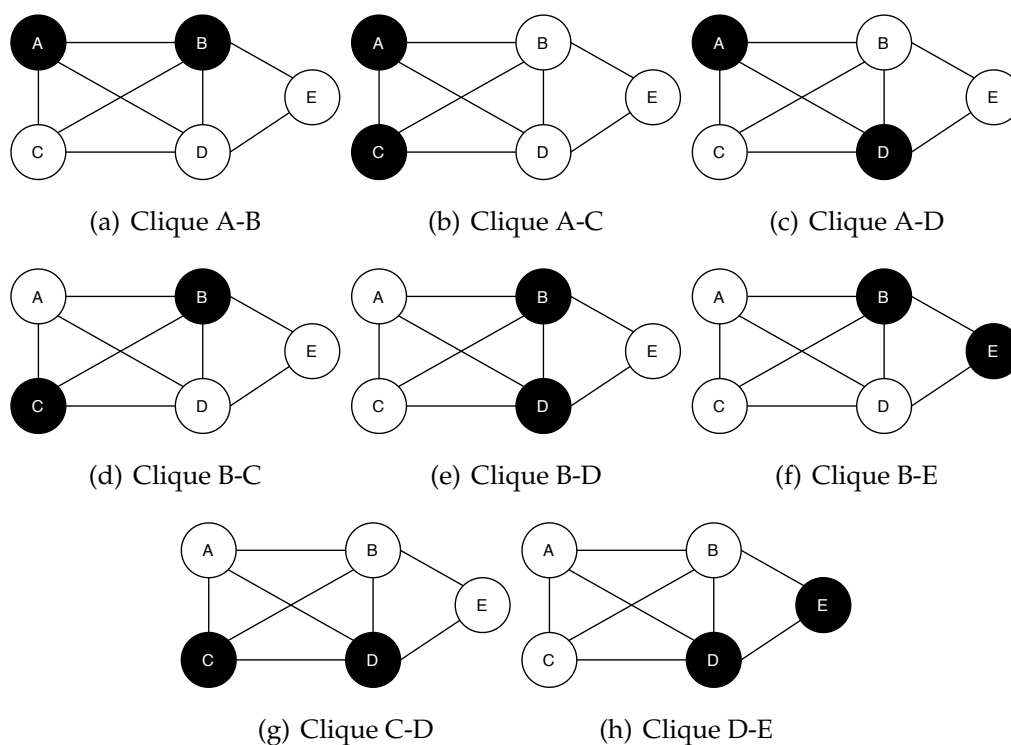


FIGURA 1.7: Cliques de cardinalidad 2 del grafo.

También se encuentran, como indica la figura 1.8, los cliques con cardinalidad igual a 3.

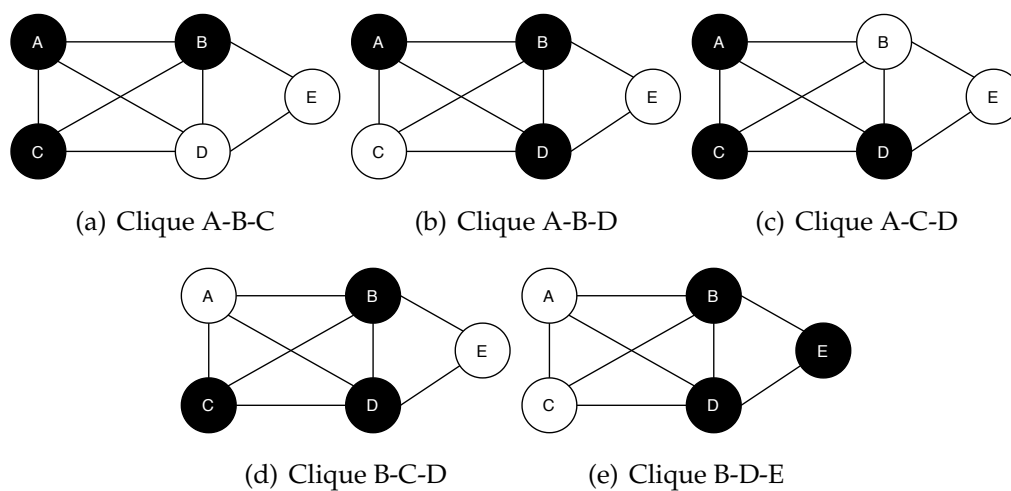


FIGURA 1.8: Cliques de cardinalidad 3 del grafo.

Cabe destacar que los nodos, por si solos, también forman clique.



Una característica importante de un clique es que este puede ser máximo [33]. Esto quiere decir que no se pueden añadir más nodos adyacentes que cumplan las restricciones necesarias para formar un clique, y a su vez, es el de mayor tamaño del grafo, como se muestra en la figura 1.9, a diferencia de un clique que se podría denominar simple [32].

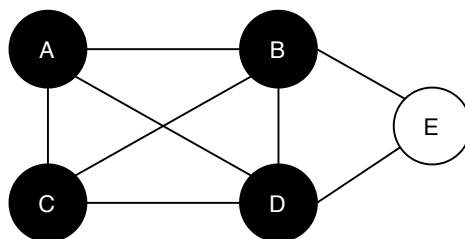


FIGURA 1.9: Clique máximo del grafo.

En este caso, se obtiene  $\omega(G) = 4$ , donde  $\omega$  denota el número de vértices del clique, su cardinalidad.

## 1.4. Definición del problema

### 1.4.1. Problema del clique de ratio máximo

Como se introdujo en la sección 1.3.3, el concepto clique es fundamental y muy estudiado, en concreto, la búsqueda del clique máximo dentro de un grafo. A este problema se le conoce como el problema del clique máximo o MCP (Maximum clique problem), y es catalogado como un NP-completo.

En teoría de complejidad computacional, a los problemas denominados como NP, acrónimo de non-deterministic polynomial time o tiempo polinomial no determinista, se les conoce como el conjunto de problemas que se pueden resolver en un tiempo polinómico por una máquina de Turing no determinista. Esta clasificación, como se muestra en la figura 1.10, también contiene todos los problemas de tipo P y de tipo NP-completos, como es el caso del problema del clique máximo.

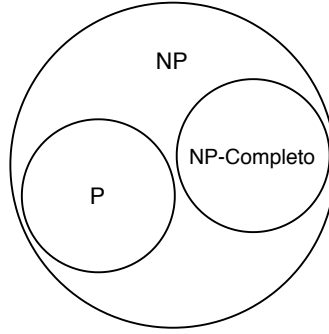


FIGURA 1.10: Diagrama de problemas NP.

Una variante de este problema es el llamado problema del clique de peso máximo o MWCP (Maximum Weight Clique Problem), en el que se asocia un peso no negativo a cada vértice, y cuyo objetivo es encontrar un clique con el máximo valor en la suma de los pesos de sus vértices. Este problema está estrechamente ligado al problema tratado en este trabajo.

Otra variante, es la tratada en este trabajo y que se corresponde con el problema del clique de ratio máximo o MRCP (Maximum Ratio Clique Problem) que trata de encontrar el subgrafo completo de ratio máximo. Este ratio se define como la proporción de la suma de los pesos no negativos asociados a los vértices del grafo como se indica a continuación:

$$\frac{\sum_{i=1}^n p_i x_i}{\sum_{i=1}^n q_i x_i}$$

donde  $p$  y  $q$  son pesos no negativos asociados a cada vértice  $i$ , y  $x$  se determina como:

$$x_i = \begin{cases} 1 & : \text{si el vértice } i \text{ forma parte de la solución.} \\ 0 & : \text{en otro caso} \end{cases}$$

Como se expone en el modelo de ecuación 1.1, partiendo de un grafo simple no dirigido  $G = (V, E)$ , donde  $V$  se asocia al conjunto de vértices pertenecientes al grafo,  $\{v_1, \dots, v_n\}$ , y  $E$  es el conjunto de aristas que conectan los vértices del grafo,  $\{(v_i, v_j)\}$  tal que  $i \neq j$  y  $v_i, v_j \in V$ , y suponiendo que los pesos asociados a cada vértice son positivos, se obtiene un clique máximo  $\hat{S}$ , siempre y cuando se cumplan las restricciones 1.2 - 1.4.

$$\text{maximizar} \quad f = \frac{\sum_{i=1}^n p_i x_i}{\sum_{i=1}^n q_i x_i} \quad (1.1)$$

sujeto a :

$$x_i + x_j \leq 1 : \forall (v_i, v_j) \notin E, i \neq j, \quad (1.2)$$

$$\sum_{i=1}^n (a_{ij}) x_i \geq 1 : \forall v_j \in V, \quad (1.3)$$

$$x_i \in 0, 1 : \forall v_i \in V. \quad (1.4)$$

El problema del clique de ratio máximo ha sido catalogado como un problema NP-difícil o NP-complejo como se indica en la figura 1.11, por lo que no es posible obtener una solución factible por métodos heurísticos o exactos.

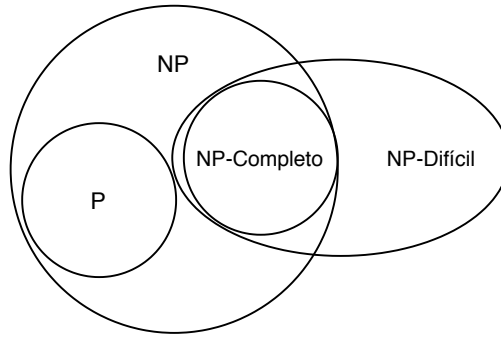


FIGURA 1.11: Diagrama de problemas NP y NP-difícil.

## 1.5. Estado del arte

Si bien es cierto que no existe demasiada literatura al respecto, ya que se trata de un problema poco estudiado en comparación con el clásico problema de búsqueda de la clique máxima, [3], [19], [34], o [23], este análisis, y todo lo que rodea al mismo, ha sido documentado mediante la literatura que se explica a continuación.

Un estudio muy cercano al tratado en este trabajo es el problema de la búsqueda del clique de peso máximo, en [30] y [28] se implementan algoritmos de búsqueda local, llamados SCCWalk y SCCWalk4L, en los que se hace uso del muestreo no determinista y el machine-learning para abordar el problema, eliminando variables de decisión que no formarían parte de una solución óptima.

En el documento desarrollado por Samyukta Sethuraman y Sergiy Butenko [26] se trata el problema desde tres puntos de vista, mediante el IBM CPLEX, para resolver el problema de manera lineal, la aplicación de la búsqueda binaria y el método de Newton.

En [17] se trata el problema basándose en el enfoque eficiente que dan las funciones de diferencia de convexos (DC) y sus algoritmos (DCA), que proveen resultados competitivos y a su vez introducen las desigualdades válidas, que ayudan a mejorar el tiempo computacional en la obtención de resultados de calidad al problema.

Cabe destacar entre todos los trabajos el de Dominik Goeke, Mahdi Moeini y David Poganiuch [13] el cuál se ha tomado como referencia para realizar este trabajo. En este artículo, los autores parten de un punto de vista multi-arranque MSM, añadiendo una búsqueda de vecindario variable VNS, lo que permite obtener soluciones de gran calidad con un tiempo de computo menor.

## Capítulo 2

# Objetivos

En este capítulo se indican los objetivos, tanto principales como secundarios, establecidos al inicio de este trabajo fin de grado.

### 2.1. Objetivos principales

- Estudio y comprensión del problema de la búsqueda del clique de ratio máximo.
- Estudio, diseño e implementación del algoritmo metaheurístico GRASP para la resolución del problema.

### 2.2. Objetivos secundarios

- Conocer los distintos algoritmos metaheurísticos existentes.
- Comprender la complejidad computacional relacionada con la búsqueda del clique de ratio máximo.
- Aprendizaje del lenguaje de programación Python para el desarrollo del algoritmo metaheurístico GRASP.
- Profundización y mejora en técnicas algorítmicas, de programación y de estructuras de datos para la realización de este trabajo fin de grado.



## Capítulo 3

# Descripción algorítmica

En este capítulo se describe el algoritmo metaheurístico utilizado, exponiendo todas sus características para la obtención de una solución al problema.

### 3.1. Metaheurística GRASP

El acrónimo GRASP se forma a partir de sus siglas en inglés Greedy Randomized Adaptive Search Procedure o que en castellano es Procedimiento de búsqueda voráz aleatorizado y adaptativo, el término fue introducido por primera vez por Feo y Resende en 1995 en su artículo con el mismo nombre [10].

Este algoritmo se basa en el multi-arranque, dónde cada arranque es una iteración de un procedimiento que está constituido por dos partes bien diferenciadas, la fase constructiva, en la que se obtiene una solución de buena calidad, y una fase de mejora, en la que partiendo de la solución obtenida en la fase anterior, se intenta mejorar localmente [1]. En [27] [25] [20] [4] [29] [24] se pueden encontrar diversos documentos en los que se tratan problemas aplicando la metaheurística GRASP.

En el algoritmo 1 se muestra el pseudocódigo de la metaheurística GRASP que se ha empleado para el desarrollo y obtención de una solución preliminar para este problema, y posteriormente, se muestra el algoritmo 5, con el cual se ha refinado esta solución para obtener una mejor. Dichos algoritmos se puede explicar, aplicándolo al problema tratado, de la siguiente manera:

Partiendo de un grafo  $G = (V, E)$  donde  $V$  son los vértices o nodos del grafo, y  $E$  las aristas que unen estos nodos. Se toma un vértice  $v$  aleatorio de entre los vértices del

grafo.  $v$  se incluye en la solución  $S$  ya que cumple con las restricciones del problema, descritas en la sección 1.4.1, a partir de  $v$  se construye la lista de candidatos  $CL$ , definida como todos los nodos adyacentes a  $v$  que forman parte de la lista de nodos del grafo de partida. A partir de este momento se toma un elemento de la lista de candidatos y con este se obtiene, mediante una función voráz el nodo con valor máximo y mínimo de esta función, con estos valores y un valor dado  $\alpha$ , que oscilará entre 0 y 1 y marca la aleatoriedad de la función de manera que si  $\alpha$  es igual a 1 la función será menos aleatoria y se obtendría solo el mejor candidato, mientras que si es 0 la función es más aleatoria y se obtendrán todos los candidatos posibles, teniendo esto en cuenta, se obtiene el valor de  $\mu$ , el cuál pondrá el límite para la lista de candidatos restringida  $RCL$ , obtenida mediante la lista de candidatos  $CL$  ordenada de mayor valor a menor valor tomando los primeros valores hasta alcanzar el valor de  $\mu$ . Tomando esta lista, se elegirá de manera aleatoria un nodo de la misma,  $u$ , el cuál se añadirá a la lista solución  $S$  y eliminándolo de la lista de candidatos junto con los nodos de la lista de candidatos que no son adyacentes a ese nodo. Este procedimiento será repetido hasta que la lista de candidatos este vacía, obteniéndose en ese momento la lista  $S$  que conformará la solución preliminar. A partir de este momento, se obtendrán todos los nodos adyacentes a los nodos que conforman la solución, de una manera ordenada mediante el valor de su ratio calculado y se irán incluyendo en la solución, comprobando si forman o no un clique, como indican las restricciones del problema, si no es así, se eliminarán los todos que causen el conflicto para obtener un clique, y posteriormente se añadirán todos los nodos adyacentes a los del clique, obteniendo de esta manera un clique máximo y solución factible al problema.

### 3.1.1. Fase constructiva

En esta fase se ha usado el algoritmo 2 el cuál es común a ambos constructivos y sólo se diferencian en como obtiene cada uno su mejor nodo a escoger para incluir en su solución, los cuales se definen a continuación.

En primer lugar se define el algoritmo 3 para el constructivo que obtiene una solución mediante un algoritmo voraz, buscando el mayor ratio de cada nodo adyacente al de partida.



**Algoritmo 1:** Pseudocódigo algoritmo GRASP.

---

```

(1)  $v \leftarrow \text{rnd}(V)$ 
(2)  $S \leftarrow \{v\}$ 
(3)  $CL \leftarrow \{u \in V : (u, v) \in E\}$ 
(4) mientras  $|CL| \neq 0$  hacer
(5)    $g_{\min} \leftarrow \min_{c \in CL} g(c)$ 
(6)    $g_{\max} \leftarrow \max_{c \in CL} g(c)$ 
(7)    $\mu \leftarrow g_{\max} - \alpha(g_{\max} - g_{\min})$ 
(8)    $RCL \leftarrow \{c \in CL : g(c) \geq \mu\}$ 
(9)    $u \leftarrow \text{rnd}(RCL)$ 
(10)   $S \leftarrow S \cup \{u\}$ 
(11)   $CL \leftarrow CL \setminus \{u\} \setminus \{w : (u, w) \notin E\}$ 
(12) fin
(13) devolver  $S$ 

```

---

**Algoritmo 2:** Constructivo voráz.

---

```

(1)  $S \leftarrow \emptyset$ 
(2)  $Adyacentes \leftarrow \text{SeleccionAdyacentes}(\text{nodo})$ 
(3) mientras  $|Adyacentes| \neq 0$  hacer
(4)    $\text{candidato} \leftarrow \text{buscarMejor}(adyacentes)$ 
(5)   si  $\text{formaClique}(\text{candidato})$  entonces
(6)      $Adyacentes \leftarrow Adyacentes \cap \text{SeleccionAdyacentes}(\text{candidato})$ 
(7)      $S \leftarrow S \cup \{\text{candidato}\}$ 
(8)   fin
(9)   en otro caso
(10)     $Adyacentes \leftarrow Adyacentes \setminus \{\text{candidato}\}$ 
(11)  fin
(12) fin
(13) devolver  $S$ 

```

---

---

**Algoritmo 3:** Pseudocódigo método buscarMejor de tipo ratio.

---

```

(1)  $ratio \leftarrow -1$ 
(2)  $nodoElegido \leftarrow NULO$ 
(3) para  $nodo \in adyacentes$  hacer
(4)    $ratioNodo \leftarrow calcularRatio(nodo)$ 
(5)   si  $ratioNodo > ratio$  entonces
(6)      $ratio \leftarrow ratioNodo$ 
(7)      $nodoElegido \leftarrow nodo$ 
(8) devolver  $nodoElegido$ 

```

---

Y en segundo lugar, se define el algoritmo 4 para el constructivo que obtiene una solución mediante un algoritmo voraz buscando el mayor número de vecinos de cada nodo adyacente al de partida.

---

**Algoritmo 4:** Pseudocódigo método buscarMejor de tipo adyacentes.

---

```

(1)  $vecinos \leftarrow -1$ 
(2)  $nodoElegido \leftarrow NULO$ 
(3) para  $nodo \in adyacentes$  hacer
(4)    $vecinosNodo \leftarrow SeleccionAdyacentes(nodo)$ 
(5)   si  $|vecinosNodo| > vecinos$  entonces
(6)      $vecinos \leftarrow |vecinosNodo|$ 
(7)      $nodoElegido \leftarrow nodo$ 
(8) devolver  $nodoElegido$ 

```

---

### 3.1.2. Fase de búsqueda

Para esta segunda fase, se define el siguiente algoritmo 5, el cuál parte de la solución obtenida previamente en la fase constructiva.

Con esta solución, se obtienen todos los nodos vecinos de cada nodo que forma parte de la solución, ordenados de mayor a menor ratio. Esta ordenación se realiza con el fin de aumentar las posibilidades de obtener un mejor valor de ratio, ya que, por cada uno

de estos nodos, se va a añadir a la solución, comprobando si forma o no una solución, es decir, un clique.

En el caso de que no formará clique, se descartan todos aquellos nodos de la solución que impiden que se forme un clique.

Un vez se tiene un clique, se añaden todos los nodos posibles, es decir, que sigan formando clique, con el fin de obtener el máximo clique, como marca la restricción del problema.

---

**Algoritmo 5:** Pseudocódigo algoritmo búsqueda local.

---

```

(1)  $vecinos \leftarrow \emptyset$ 
(2)  $vecinos \leftarrow obtenerVecinos(solucion)$ 
(3)  $vecinosOrdenados \leftarrow ordenarVecinos(vecinos)$ 
(4) para  $nodo \in vecinosOrdenados$  hacer
(5)    $solucion \leftarrow incluirNodo(solucion, nodo)$ 
(6)   si  $no esClique(solucion)$  entonces
(7)      $solucion \leftarrow excluirNodos(solucion, nodo)$ 
(8)    $solucion \leftarrow incluirAdyacentes(solucion, nodo)$ 
(9) devolver  $solucion$ 

```

---



## Capítulo 4

# Descripción informática

En este capítulo se describe el desarrollo completo del proyecto, desde el diseño hasta la implementación de este, así como la metodología utilizada para la correcta evolución del mismo.

### 4.1. Diseño

Para la realización del desarrollo del proyecto se ha optado por seguir un paradigma de programación orientada a objetos y se ha seguido la arquitectura que se plantea en el diagrama de clases de la figura 4.1, en el que se parte de un documento principal, en este caso se trata de un script ejecutable, `grasp_main.py`, que contiene la información inicial y las llamadas a las clases y métodos de estas necesarios para obtener los resultados al problema.

A continuación de este parten las clases:

- **Instance:** Esta clase contendrá toda la información relacionada con una instancia o definición de grafo del problema, la cuál se compone de la clase **Node**, que contiene toda la información relativa a un nodo, así como operaciones relacionadas con el mismo.
- **SolutionGrasp:** Esta clase contendrá los métodos necesarios para la implementación del algoritmo GRASP y de la búsqueda local.
- **GraphUtils:** Esta clase está formada por métodos estáticos en su mayoría, y contendrá todos los métodos de utilidad relacionadas con los grafos, así como la posterior exportación a fichero de los resultados obtenidos durante el proceso. También se compone de la clase **Logger**, la cuál ayuda a configurar el sistema de

registros del proyecto, con el cuál poder trazar errores y ver el estado del progreso más fácilmente.

También se encuentran las clases:

- **SolutionGreedy**: Esta clase abstracta contiene lo necesario para construir una solución completa mediante un algoritmo voráz. El término completa se refiere a los atributos *density*, el cuál indica la densidad del grafo del que parte la solución; *clique*, el cuál contiene el listado de nodos que forman parte de la solución; *sol\_value*, el cuál indica el valor de la solución o función objetivo; *cardinality*, que contiene el valor del número de nodos de la solución y *compute\_time*, que indica el tiempo total de computo en hallar esa solución.
- **SolutionGreedyRatio**: Clase especializada en la obtención del mejor candidato posible por la condición de ratio.
- **SolutionGreedyNeighbors**: Al igual que **SolutionGreedyRatio**, esta clase se especializa en encontrar el mejor nodo candidato pero con mayor número de nodos vecinos.

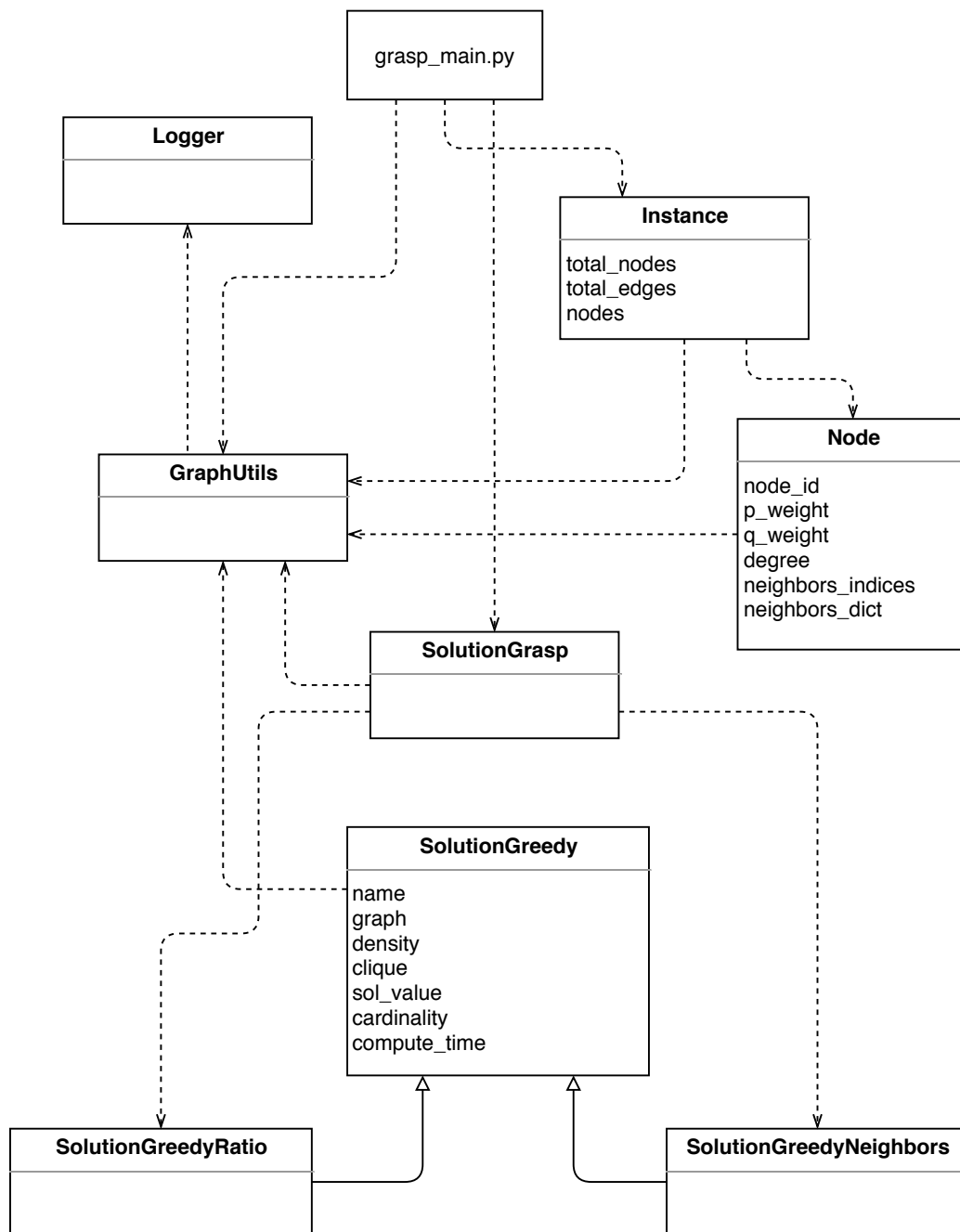


FIGURA 4.1: Diagrama de clases del proyecto.

## 4.2. Implementación

La implementación de este proyecto se basa en la ejecución de un script<sup>1</sup> escrito en lenguaje Python el cual parte de una clase en la que se encuentra la siguiente sentencia de código:

```
if __name__ == "__main__"
```

la cual posibilita la ejecución del script mediante el comando:

```
python grasp_main.py
```

Este script, en adelante Grasp Main, tiene la configuración necesaria para ajustar los parámetros del programa:

- Número de iteraciones a realizar por cada fichero.
- Ruta donde se encuentran los ficheros de definición de los grafos a procesar o instancias.
- Ruta de los ficheros de resultados generados por el programa.
- Valores de  $\alpha$  para ajustar la aleatoriedad del algoritmo.

Grasp Main se encarga de recorrer recursivamente los ficheros que se encuentran en la ruta de recursos definida, y, por cada uno de los ficheros encontrados crea un objeto de tipo Instance, añadiendo la información del grafo:

- Número de nodos.
- Número de aristas.
- Estructura de datos con los nodos del grafo.

Esta estructura de datos contiene tantos objetos de tipo Node como tengo el grafo, cada uno de ellos con la siguiente información:

- Identificador del nodo.
- Valor del peso p.
- Valor de peso q.
- Grado del nodo.

---

<sup>1</sup>Secuencia de ordenes o instrucciones que serán intepretadas para su ejecución.



- Estructura de datos con las relaciones de este nodo con otros nodos del grafo.

Tras terminar esta operación, realiza el procesado un número  $N$  de veces, según se haya definido previamente en la configuración de la aplicación, y por cada tipo de constructivo de los que dispone la aplicación, en este caso, constructivo en base al ratio de los nodos y constructivo en base al número de nodos adyacentes, los cuales serán explicados más adelante. A partir de este punto se llama al algoritmo GRASP, el cual se encuentra en la clase `SolutionGrasp` y contiene los métodos necesarios para la obtención de una solución, partiendo de la función *find\_grasp\_solution*, la cual inicializa los siguientes datos:

- `vertex`, el cuál es obtenido de manera aleatoria entre todos los nodos del grafo.
- `solution`, conjunto inicializado con el vértice obtenido anteriormente.
- `cl`, lista de candidatos posibles para encontrar una solución.

Para la fase constructiva del algoritmo, la implementación se apoya en la función *get\_g* y según el tipo elegido en la configuración inicial, se creará un objeto del constructivo específico, `SolutionGreedyRatio` o `SolutionGreedyAdjacent`. Estos heredan de la clase abstracta<sup>2</sup> `SolutionGreedy`, la cuál tiene la información compartida por ambos tipos, y delega la implementación de la función *find\_better* en cada una de las clases específicas, quienes mediante un algoritmo de tipo voraz buscan una solución factible en un tiempo muy reducido. Un vez se obtienen los resultados de los candidatos, esta lista es procesada mediante la función *get\_rcl*, la cual mediante el valor de  $\mu$ , calculado como se mostró en el algoritmo 1 con el valor configurado  $\alpha$ , permite obtener la lista de candidatos restringida. Con esta lista restringida, el siguiente paso es escoger de manera aleatoria un nodo de esta e incluirlo en el conjunto solución, eliminando los nodos de la lista de candidatos que no son adyacentes a este, ya que no formarían una solución factible. Esta operación es realizada hasta que la lista de candidatos esté vacía.

Para mantener cierta información en un único lugar se ha implementado la clase `GraphUtils`, la cuál contiene información necesaria para los grafos y métodos útiles para usarse durante el procesado de los ficheros, así como su posterior exportación a ficheros de tipo CSV en este caso para mostrar los resultados obtenidos.

---

<sup>2</sup>En programación orientada a objetos, es un tipo de clase que no puede ser instanciada, y por lo general sirve para definir otras clases de este tipo.

Para probar las diferentes funciones del proyecto se han escrito casos de prueba mediante la librería de Python unittest<sup>3</sup>, para conseguir un código tolerable a posibles fallos y mantenible.

### 4.3. Metadología empleada

Para el desarrollo de este proyecto se ha optado por seguir una metodología de tipo iterativa e incremental, lo que permite evolucionar el proyecto progresivamente e ir adaptando los requisitos del cliente, en este caso los tutores del proyecto, en el menor tiempo posible, mejorando así la calidad del producto final con el menor esfuerzo.

Estas iteraciones e incrementos de funcionalidad se han realizado durante todo el desarrollo del proyecto, mediante reuniones, con un lapso de aproximadamente 3 a 4 semanas entre ellas, corrigiendo errores de la iteración anterior si los hubiera y aumentando la funcionalidad del producto a entregar tanto en funcionalidad como en calidad, de esta manera se consigue una evolución progresiva y segura sobre el producto y los requerimientos que el proyecto exige.

Para mantener el control y administrar lo correspondiente sobre las tareas a realizar, las que se han realizado y las realizadas del proyecto, se ha usado el administrador de proyectos Trello<sup>4</sup>, el cuál mediante tarjetas sobre el tablero del proyecto permite conocer las tareas del proyecto, así como su estado, detalles de la misma y añadir nuevas si fuera necesario, en la figura 4.2 se muestra un ejemplo de uso del sistema de tarjetas que ofrece la herramienta.

---

<sup>3</sup><https://docs.python.org/3/library/unittest.html>

<sup>4</sup><https://trello.com/es>

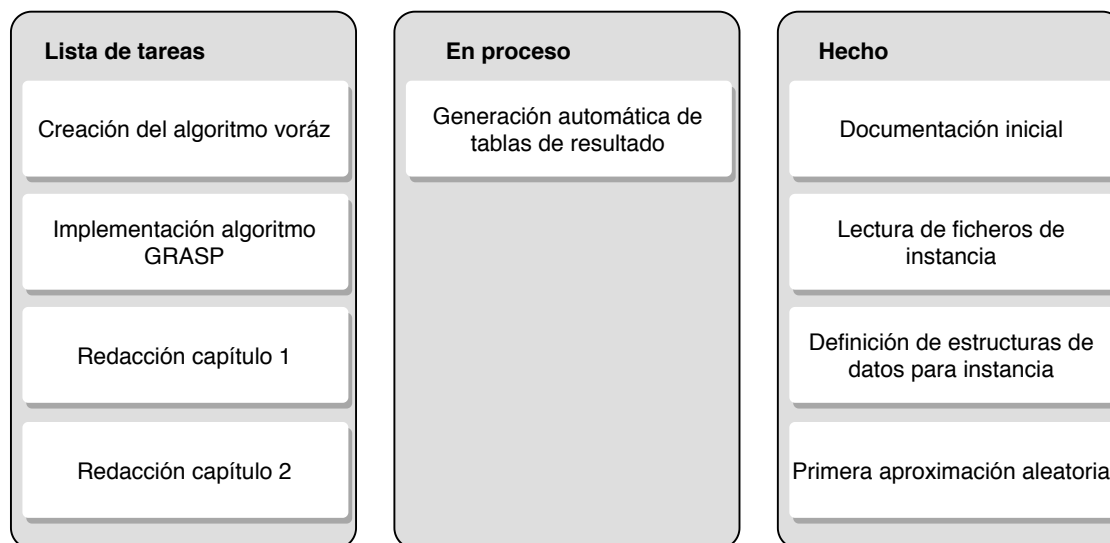


FIGURA 4.2: Ejemplo del sistema de tarjetas de Trello.

En cuanto al mantenimiento de versiones del proyecto se ha usado el sistema de control de versiones Git<sup>5</sup>. Este control de versiones se ha gestionado a su vez a través del portal de alojamiento de repositorios GitHub<sup>6</sup>. Para interactuar entre el repositorio local y el repositorio remoto se ha optado por hacer uso tanto de la terminal mediante los comandos del propio sistema de control de versiones Git como del cliente para tal propósito GitKraken<sup>7</sup>, el cuál permite mediante su sencilla e intuitiva interfaz mantener un control exhaustivo sobre las ramas y versionado de las distintas piezas de código del proyecto en el que se trabaja, así como revisar posibles conflictos que se produzcan.

---

<sup>5</sup><https://git-scm.com/>

<sup>6</sup><https://github.com/>

<sup>7</sup><https://www.gitkraken.com/>



## Capítulo 5

# Resultados

En este capítulo se exponen los diferentes recursos tanto hardware como software para la realización de este trabajo fin de grado y como, a partir de estos, se han obtenido los resultados para su posterior análisis.

### 5.1. Recursos utilizados

A continuación se detallará la máquina y software para el desarrollo y procesado de las instancias para la comprobación del algoritmo, y las instancias utilizadas que describen diferentes casos y situaciones reales.

#### 5.1.1. Descripción de la máquina utilizada

Para la realización de las diversas pruebas y procesado de las instancias de este problema, se ha utilizado una máquina con las siguientes características:

- **Procesador:** Intel Core i5 2,7 GHz
- **Memoria RAM:** 8 GB 1867 Mhz DDR3

El desarrollo del código se ha realizado mediante el lenguaje de programación Python<sup>1</sup>, en su versión 3.7.4, a través del IDE <sup>2</sup>, PyCharm<sup>3</sup> de JetBrains en su versión 2019.3.1.

---

<sup>1</sup><https://www.python.org/>

<sup>2</sup>Entorno de desarrollo integrado.

<sup>3</sup><https://www.jetbrains.com/es-es/pycharm/>

### 5.1.2. Instancias utilizadas

Las instancias con las que se ha contado para comprobar la eficiencia del algoritmo desarrollado han sido proporcionadas por los estudios previos en los que se basa y compara este trabajo final de grado, estas, hacen referencia a diferentes conjuntos de grafos, tanto generados de manera aleatoria como obtenidos de diferentes fuentes de datos como precios del mercado de valores y turbinas de viento. A continuación se detallan los diferentes conjuntos:

- **Conjuntos de tipo A y B:** Estos conjuntos son instancias de grafos generadas mediante una distribución de probabilidad uniforme, variando entre los 100 y los 500 nodos, así como la densidad del mismo que varía entre 45,78 % y 53,64 %.
- **Conjunto de tipo C:** Los datos pertenecientes a las instancias de este tipo hacen referencia a datos de los precios del mercado de valores.
- **Conjunto de tipo D:** Las instancias de este conjunto son datos para la construcción de turbinas de viento, donde cada nodo representa una localización de estas turbinas y sus pesos son la media de la velocidad del viento y el coste de construcción de una turbina en ese punto.
- **Conjunto de tipo E:** Estas instancias están extraídas del segundo y décimo DIMACS Implementation Challenge<sup>4</sup>, donde cada nodo tiene un peso  $p_i = 1$  y un peso  $q_i = 2$ . Adicionalmente se ha añadido un nodo más a cada instancia de este conjunto, el cual está conectado al resto de nodos de la misma, con un peso  $p_i = 1$  y un peso  $q_i = 1$ , donde  $i$  es el número del nodo dentro de esa instancia.
- **Conjunto de tipo F:** Estas instancias son las mismas que en el conjunto E pero en este caso los pesos de cada nodo son  $p_i = i$  y  $q_i = |V| - i + 1$ , donde  $i$  es el número del nodo dentro de esa instancia.

## 5.2. Análisis de los resultados

En esta sección se muestran los resultados obtenidos tras el procesamiento de las instancias descritas en la sección 5.1.2 y empleando la máquina descrita en la sección 5.1.1.

---

<sup>4</sup><http://dimacs.rutgers.edu/programs/challenge/>

Estos resultados se han obtenido en dos fases, una primera fase preliminar, con un conjunto reducido de las instancias y una segunda fase final con todas las instancias del problema para su posterior comparación con los resultados de estudios previos.

### 5.2.1. Experimentos preliminares

En estas primeras pruebas se han seleccionado un subconjunto de 22 instancias del problema, con el fin de estudiar la calidad del algoritmo y comprobar los resultados obtenidos mediante distintos valores para  $\alpha$  en varias iteraciones.

Para la realización de las pruebas de esta sección se han elegido las instancias de todos los conjuntos para ampliar el rango de las pruebas. El número de nodos en estas instancias varía desde los 30 hasta los 1000 nodos, con densidades del 1 al 50 por ciento aproximadamente. A esta selección de instancias se les ha aplicado los dos constructivos desarrollados para la resolución a este problema, descritos en la sección 4.2, y a su vez se ha aplicado los valores de  $\alpha \in \{0.25, 0.5, 0.75\}$  y adicionalmente un valor de  $\alpha$  aleatorio. Este experimento se ha realizado procesando cada instancia 100 veces, que unido a lo anteriormente descrito, hace un total de 800 iteraciones por fichero.

TABLA 5.1: Resultados de los experimentos preliminares con constructivo adyacentes.

Instancias	GRASP				GRASP y búsqueda local				Mejor
	$f$	c	t (sec)	$\alpha$	$f$	c	t (sec)	$\alpha$	
random-1	1.2421	15	0.0620	0.8377	1.2421	15	0.1430	0.5000	No
random-4	1.0168	26	1.6731	0.7500	1.4650	27	5.7613	0.5000	No
random-5	0.9130	31	3.6512	0.7500	1.4592	32	3.4072	0.5000	No
random-6	0.9814	19	0.0990	0.0631	0.9672	18	0.2590	0.5000	No
random-8	0.8551	22	0.7140	0.7500	0.9229	23	1.9691	0.7500	No
random-10	0.8989	31	3.3232	0.7500	1.0688	30	2.5841	0.4817	No
market-1	1.5434	8	0.0300	0.0111	2.1429	8	0.2550	0.0111	Si
market-2	1.2310	8	0.0510	0.7500	1.3260	8	0.3060	0.5000	Si
market-5	1.4725	8	0.0230	0.2500	2.8116	8	0.3880	0.2053	Si
wind-2005	90 391.7500	4	0.0030	0.2500	90 391.7500	4	0.0160	0.5000	No
wind-2006	87 219.5122	5	0.0050	0.5000	90 034.2000	5	0.0520	0.5000	No
brock200-2	0.5455	12	0.0620	0.7500	0.5455	12	0.8240	0.7500	Si
football	0.5263	10	0.0040	0.2500	0.5263	10	0.1650	0.7500	Si
johnson16-2-4	0.5556	10	0.0640	0.2500	0.5556	10	0.4110	0.5000	Si
karate	0.5455	6	0.0010	0.2500	0.5455	6	0.0100	0.2500	No
keller4	0.5417	13	0.0810	0.7500	0.5417	13	0.5060	0.5000	Si
c-fat500-10	1.0080	126	84.0878	0.5000	1.0080	126	98.4496	0.2500	Si
email	1.4532	12	0.0040	0.7500	1.4532	12	0.0040	0.0981	No
jazz	0.8886	30	0.2160	0.8065	0.8886	30	0.0630	0.5000	No

Continúa en la siguiente página

Tabla 5.1 Continuación de la página anterior

Instancias	GRASP				GRASP y búsqueda local				Mejor
	$f$	$c$	$t$ (sec)	$\alpha$	$f$	$c$	$t$ (sec)	$\alpha$	
p-hat1000-1	0.8058	9	0.1470	0.7500	2.0753	10	0.6540	0.3496	Si
polbooks	2.9750	6	0.0010	0.7500	2.9750	6	0.0040	0.5000	No
sanr400-0-7	1.3352	18	0.5860	0.2500	2.0469	18	3.1032	0.5000	No
Concluido									

TABLA 5.2: Resultados de los experimentos preliminares con constructivo ratio.

Instancias	GRASP				GRASP y búsqueda local				Mejor
	$f$	$c$	$t$ (sec)	$\alpha$	$f$	$c$	$t$ (sec)	$\alpha$	
random-1	1.2421	15	0.0460	0.8377	1.2421	15	0.1170	0.5000	Si
random-4	0.9821	28	1.3561	0.7500	1.1858	28	2.5681	0.7500	Si
random-5	0.8088	31	1.3531	0.7500	1.4592	32	2.9242	0.7500	Si
random-6	0.9514	18	0.0790	0.5000	0.9672	18	0.2570	0.0631	Si
random-8	0.9073	22	0.2110	0.7500	0.9229	23	0.8310	0.7500	Si
random-10	0.9014	30	1.3901	0.7500	1.0688	30	2.5391	0.7500	Si
market-1	1.8981	8	0.0260	0.7500	2.1429	8	0.2730	0.0111	No
market-2	0.8790	8	0.0470	0.4401	1.3260	8	0.3230	0.7500	No
market-5	1.2501	8	0.0580	0.5000	3.8616	7	0.3630	0.7500	No
wind-2005	90 391.7500	4	0.0030	0.2500	90 391.7500	4	0.0150	0.7500	Si
wind-2006	88 663.4000	5	0.0040	0.6243	90 034.2000	5	0.0220	0.5000	Si
brock200-2	0.5238	11	0.0540	0.1726	0.5238	11	0.4620	0.5000	No
football	0.5263	10	0.0040	0.2500	0.5263	10	0.1720	0.7500	No
johnson16-2-4	0.5294	9	0.0730	0.2500	0.5294	9	0.2820	0.7500	No
karate	0.5455	6	0.0010	0.2500	0.5455	6	0.0100	0.7500	Si
keller4	0.5217	12	0.0760	0.2500	0.5217	12	0.3130	0.7500	No
c-fat500-10	1.0080	126	99.9827	0.2500	1.0080	126	113.8295	0.7500	No
email	1.4532	12	0.0080	0.5000	1.4532	12	0.0020	0.7500	Si
jazz	0.8886	30	0.1700	0.7500	0.8886	30	0.0240	0.7500	Si
p-hat1000-1	2.9810	9	0.1580	0.3496	2.0753	10	0.7830	0.7500	No
polbooks	2.9750	6	0.0010	0.5000	2.9750	6	0.0030	0.7500	Si
sanr400-0-7	2.9186	18	0.7960	0.2500	2.3953	19	3.5592	0.5000	Si

Durante los experimentos preliminares se han recopilado los datos mostrados en las tablas 5.1 y 5.2, las cuales están divididas en el uso del algoritmo GRASP y su posterior mejora mediante la búsqueda local y donde:

- $f$ : Indica el valor de la función objetivo calculado.
- $c$ : Indica la cardinalidad del clique encontrado.
- $t$ : Indica el tiempo total empleado en encontrar la solución en segundos.
- $\alpha$ : Indica el valor de  $\alpha$  con el que se ha obtenido esa solución.



- Mejor: Indica si la solución obtenida mediante este constructivo es mejor respecto al otro.

En ellas se encuentran los mejores resultados tras todas las iteraciones de ambos constructivos y la posterior mejor del algoritmo con la búsqueda local. Aunque se puede observar que añadir al algoritmo la búsqueda local no parece suponer una mejor en algunos casos, en muchos otros si, por lo que finalmente se añadirá para la experimentación final. Además, tras estas iteraciones se ha comprobado que el constructivo mediante el cálculo por ratio ha obtenido mejores resultados en relación ratio-cardinalidad, es decir, este constructivo permite obtener valores de ratio mayores y cliques con mayor número de nodos en tiempos de computo razonables, a esto hay que añadir que el  $\alpha$  que ha permitido obtener mejores resultado ha sido el valor 0.75.

### 5.2.2. Experimento final

Como se ha comentado en la anterior sección, este experimento final se ha realizado mediante la siguiente configuración:

- Algoritmo GRASP.
- Búsqueda local.
- Constructivo calculado por ratio.
- Valor de  $\alpha = 0,75$ .
- 100 iteraciones por instancia.

Con los que tras el procesado de todas las instancias, se han obtenido los datos mostrador en la tabla 5.3, donde:

- $f$ : Indica el valor de la función objetivo calculado.
- $c$ : Indica la cardinalidad del clique encontrado.
- $t$ : Indica el tiempo total empleado en encontrar la solución en segundos.
- $\Delta f(\%)$ : Indica la variación respecto al valor de la función objetivo en la que se ha basado este trabajo.
- Mejor: Indica si la solución obtenida mediante el algoritmo es mejor o no que la del trabajo anterior.

TABLA 5.3: Resultados del experimento final.

Instancias	$f$	c	t (sec)	$\Delta f$ (%)	Mejor
random 1	1.2421	15	0.1320	62,13	No
random 2	1.0816	20	0.3720	76,94	No
random 3	0.9475	21	0.4400	77,49	No
random-4	1.5226	27	2.4531	68,48	No
random-5	1.4592	32	3.4452	59,91	No
random-6	0.9672	18	0.2740	15,89	No
random-7	1.0473	19	0.3310	12,73	No
random-8	0.9504	23	0.7030	20,14	No
random-9	0.9670	28	1.4321	26,74	No
random-10	0.9905	31	3.6612	27,70	No
market-1	2.1429	8	0.2810	89,90	No
market-2	1.3260	8	0.3240	94,40	No
market-3	2.0168	8	0.3500	85,51	No
market-4	2.3678	8	0.3520	87,36	No
market-5	2.8116	8	0.3140	80,19	No
wind-2004	91 286.0000	4	0.0260	0	Si
wind-2005	90 391.7500	4	0.0170	4,54	No
wind-2006	90 034.2000	5	0.0240	8,57	No
Set E					
brock200-1	0.5135	19	1.7311	3,11	No
brock200-2	0.5263	10	0.4600	4,31	No
brock200-3	0.5185	14	0.7350	3,98	No
brock200-4	0.5172	15	1.0301	2,41	No
c-fat200-1	0.5200	13	0.5160	0	Si
c-fat200-2	0.5102	25	0.8730	0,04	Si
c-fat200-5	0.5043	59	8.4095	0,85	Si
C-fat500-1	0.5172	15	5.5273	0,53	No
c-fat500-2	0.5094	27	4.5143	0,11	No
c-fat500-5	0.5039	65	19.1571	0,78	Si
c-fat500-10	0.5020	127	146.8904	0,4	Si
hamming6-2	0.5077	33	1.1881	2,37	No
hamming6-4	0.5556	5	0.0280	7,41	No
hamming8-4	0.5152	17	0.9801	8,01	No
johnson8-2-4	0.5556	5	0.0080	0,79	No
johnson8-4-4	0.5172	15	0.1380	2,41	No
johnson16-2-4	0.5294	9	0.2810	0,11	No
johnson32-2-4	0.5152	17	15.6079	0,93	No

Continúa en la siguiente página

Tabla 5.3 Continuación de la página anterior

Instancias	$f$	$c$	$t$ (sec)	$\Delta f$ (%)	Mejor
keller4	0.5217	12	0.2870	5,14	No
p-hat300-1	0.5333	8	0.6900	6,43	No
p-hat300-2	0.5111	23	1.3121	5,35	No
p-hat300-3	0.5082	31	5.5273	2,27	No
p-hat500-1	0.5294	9	2.2275	7,12	No
p-hat700-1	0.5263	10	3.3702	7,66	No
p-hat1000-1	0.5263	10	7.5684	7,66	No
san200-0-7-1	0.5111	23	1.5341	3,56	No
san200-0-7-2	0.5161	16	1.4451	4,42	No
san200-0-9-1	0.5039	64	16.6310	1,19	No
san200-0-9-2	0.5057	44	11.3847	0,83	No
san200-0-9-3	0.5075	34	9.2055	0,50	No
san400-0-5-1	0.5263	10	2.6161	6,02	No
san400-0-7-1	0.5111	23	7.3954	0	Si
san400-0-7-2	0.5135	19	6.3104	3,11	No
san400-0-7-3	0.5152	17	5.1413	0	Si
san400-0-9-1	0.5043	59	49.7058	0	SI
san1000	0.5238	11	23.8044	4,76	No
sanr200-0-7	0.5143	18	1.3771	2,96	No
sanr200-0-9	0.5070	36	11.1576	0,58	No
sanr400-0-5	0.5200	13	1.7641	5,45	No
sanr400-0-7	0.5135	19	4.8443	3,11	No
karate	0.5455	6	0.0100	9,09	No
dolphins	0.5455	6	0.0290	9,09	No
polbooks	0.5385	7	0.1730	10,26	No
adjnoun	0.5455	6	0.1370	9,09	No
football	0.5263	10	0.1810	12,28	No
jazz	0.5082	31	0.5830	15,30	No
celegans-metabolic	0.5263	10	9.5435	12,28	No
email	0.5200	13	94.5984	13,33	No
Set F					
brock200-1	1.5021	18	1.1171	78,17	No
brock200-2	1.6833	11	0.2330	89,00	No
brock200-3	3.3226	12	0.3790	76,08	No
brock200-4	2.1094	14	0.6250	75,56	No
c-fat200-1	1.1383	12	0.0210	6,70	No
c-fat200-2	1.0000	24	0.2150	13,04	No
c-fat200-5	1.0201	58	6.1404	0,01	Si

Continúa en la siguiente página

Tabla 5.3 Continuación de la página anterior

Instancias	$f$	$c$	$t$ (sec)	$\Delta f$ (%)	Mejor
c-fat500-1	1.0745	14	0.0370	14,72	No
c-fat500-2	1.0745	26	0.4100	0,42	Si
c-fat500-5	1.0080	64	9.2605	3,08	No
c-fat500-10	1.0080	126	116.2877	0,20	No
hamming6-2	1.0000	32	0.6660	39,39	No
hamming6-4	1.0000	4	0.0040	65,99	No
hamming8-4	1.0000	16	0.4560	49,75	No
johnson8-2-4	1.0714	4	0.0030	6,83	No
johnson8-4-4	1.0000	14	0.0990	29,08	No
johnson16-2-4	1.0209	8	0.1720	18,98	No
johnson32-2-4	1.1268	16	9.2425	0	Si
keller4	1.5126	11	0.1570	63,55	No
p-hat300-1	6.2907	7	0.0590	75,58	No
p-hat300-2	1.0048	21	1.0941	86,62	No
p-hat300-3	1.6018	32	4.7473	77,84	No
p-hat500-1	4.3871	8	0.2310	77,69	No
p-hat700-1	4.5360	8	0.4420	78,48	No
p-hat1000-1	2.0753	10	0.8270	94,07	No
san200-0-7-1	1.1401	25	0.9961	89,60	No
san200-0-7-2	2.7520	14	1.2351	84,83	No
san200-0-9-1	1.1930	55	11.8537	75,85	No
san200-0-9-2	1.3278	43	10.2146	73,44	No
san200-0-9-3	1.1145	35	9.7036	74,95	No
san400-0-5-1	1.7034	9	1.2581	97,05	No
san400-0-7-1	3.8660	22	5.2073	69,15	No
san400-0-7-2	3.3273	18	4.6083	79,45	No
san400-0-7-3	3.4127	16	3.1572	80,98	No
san400-0-9-1	1.7565	55	50.8769	0	Si
san1000	2.9117	10	9.4345	96,51	No
sanr200-0-7	1.3582	17	0.7650	86,50	No
sanr200-0-9	1.4446	36	10.0596	70,28	No
sanr400-0-5	1.0058	12	0.7040	93,40	No
sanr400-0-7	4.0658	19	3.4942	71,35	No
karate	0.1589	5	0.0010	99,04	No
dolphins	1.2028	5	0.0010	89,63	No
polbooks	2.9750	6	0.0040	75,71	No
adjnoun	0.4637	5	0.0200	97,63	No
football	1.9659	9	0.0020	79,41	No

Continúa en la siguiente página

Tabla 5.3 Continuación de la página anterior

Instancias	$f$	$c$	$t$ (sec)	$\Delta f$ (%)	Mejor
jazz	0.8886	30	0.0510	87,81	No
celegans-metabolic	1.2098	9	0.0260	80,86	No
email	1.4532	12	0.0020	97,46	No

Se puede observar que en todas las instancias se han obtenido unos valores muy cercanos al trabajo anterior y por lo tanto el algoritmo cumple ciertos mínimos de calidad, aún así, sólo en 10 casos se han obtenido valores mejores. También cabe destacar que los resultados obtenidos se han realizado en un tiempo de computo significativamente bajo, por lo que aunque no supere al trabajo anteriormente realizado por Dominik Goeke, Mahdi Moeini y David Poganiuch [13] si mejora los resultados obtenidos mediante la linealización y el método de Newton con el solucionador Gurobi con los que se compara dicho trabajo.



## Capítulo 6

# Conclusiones

En este capítulo se describen las conclusiones finales alcanzadas tras el desarrollo del proyecto, así como las lecciones aprendidas durante el mismo.

### 6.1. Consecución de los objetivos

Los objetivos establecidos al comienzo del proyecto son:

- Estudio y comprensión del problema de la búsqueda del clique de ratio máximo.
- Estudio e implementación del algoritmo metaheurístico GRASP para la resolución del problema.

Y por otro lado los objetivos secundario:

- Estudio de la importancia de los grafos en la vida real.
- Estudio de los distintos algoritmos metaheurísticos existentes.
- Estudio y comprensión de la complejidad computacional relacionada con la búsqueda de clique de ratio máximo.
- Aprendizaje del lenguaje de programación Python para el desarrollo del algoritmo metaheurístico GRASP.
- Profundización y mejora en técnicas algorítmicas, de programación y de estructuras de datos para la realización de este trabajo fin de grado.

Estos objetivos se han cumplido de manera satisfactoria, puesto que se ha estudiado el estado del arte del problema del clique de ratio máximo para poder abordarlo, así como problemas relacionados como son el problema del clique de peso máximo y el

clásico problema del clique máximo. También se han estudiado las distintas familias en las que se categorizan los algoritmos metaheurísticos y en especial el algoritmo GRASP. Todo esto ha ayudado en la comprensión del problema y su posterior diseño e implementación.

## 6.2. Conocimientos adquiridos

El proceso de realización de este trabajo fin de grado ha supuesto la superación de diferentes retos los cuales han permitido adquirir muchos conocimientos sobre el desarrollo de software y la gestión de un proyecto. También se han adquirido y profundizado en conceptos sobre algoritmia y estructuras de datos para obtener soluciones mejores y más eficientes al planteamiento de problemas. Por lo tanto se destacan:

- El aumento y mejora de conocimientos en el lenguaje de programación Python, usado para la implementación de este proyecto.
- La comprensión sobre conceptos de algoritmia y estructuras de datos, así como la mejora continua del código, enfocados en la resolución de problemas de optimización.
- Adquisición de conocimientos sobre heurísticas y metaheurísticas aplicadas a la resolución de problemas.
- Profundización en el uso de grafos en programación, así como la relación con el mundo real.
- El aprendizaje sobre  $\text{\LaTeX}$ , al utilizarlo para documentar el trabajo fin de grado.

## 6.3. Líneas de desarrollo futuras

En cuanto a las líneas de desarrollo futuras para este problema se podrían seguir las siguientes posibilidades:

En primer lugar algo que ofrece un considerable aumento en el rendimiento del código es la utilización de Cython<sup>1</sup>, mediante pequeñas modificaciones en el código para

---

<sup>1</sup><https://cython.org/>



ajustarlo a su lenguaje propio, este compilador optimizado permite aumentar el rendimiento de las funciones escritas en Python, algunas de sus características son la unificación de la mayor legibilidad del código escrito en Python con el rendimiento de C y la interacción eficientemente con grandes conjuntos de datos mediante NumPy<sup>2</sup>.

Y otra buena mejora en el propio código Python podría ser la implementación del procesamiento paralelo aprovechando el módulo `concurrent.futures` que ofrece Python el cual hace uso del módulo `multiprocessing`, por lo tanto no se ve afectado por el GIL<sup>3</sup> y permite crear un pool de procesos, mediante la función `ProcessPoolExecutor`, para ejecutar llamadas asíncronas. Una buena opción a priori sería separar los ficheros de las instancias y que cada proceso las vaya procesando paralelamente.

---

<sup>2</sup><https://numpy.org/>

<sup>3</sup><https://docs.python.org/3/glossary.html?highlight=gil#term-global-interpretor-lock>



# Bibliografía

- [1] Micael Gallego Carrillo Abraham Duarte Muñoz Juan José Pantrigo Fernández. *Metaheurísticas*. Ed. por Servicio de Publicaciones Universidad Rey Juan Carlos. 2010.
- [2] Vicente. Falcó Montesinos Antonio. Amigó J. M. Villar Amigó. «Topología molecular / J. M. Amigó ... [et al.].» En: *Sociedad Española de Matemática Aplicada (SeMA)* 39 (2007), págs. 135-149.
- [3] Mikhail Batsyn y col. «Improvements to MCS algorithm for the maximum clique problem». eng. En: *Journal of Combinatorial Optimization* 27.2 (2014), págs. 397-416. ISSN: 1382-6905.
- [4] J. Beltran y col. «Procedimientos constructivos adaptativos (GRASP) para el problema del empaquetado bidimensional». En: *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial* 6 (ene. de 2002), págs. 26-33. DOI: 10.4114/ia.v6i15.755.
- [5] Vladimir Boginski, Sergiy Butenko y Panos M. Pardalos. «Mining market data: A network approach». En: *Computers and Operations Research* 33.11 (2006). Part Special Issue: Operations Research and Data Mining, págs. 3171 -3184. ISSN: 0305-0548. DOI: <https://doi.org/10.1016/j.cor.2005.01.027>. URL: <http://www.sciencedirect.com/science/article/pii/S0305054805000286>.
- [6] Pardalos P.M. Pelillo M. Bomze I.M. Budinich M. «The Maximum Clique Problem». En: *Pardalos P.M. (eds) Handbook of Combinatorial Optimization. Springer, Boston, MA* (1999).
- [7] Conceptodefinicion.de. *Heurística*. Jul. de 2019. URL: <https://conceptodefinicion.de/heuristica/>.
- [8] Real Academia Española. *Heurística*. URL: <https://dle.rae.es/heur%C3%ADstico>.
- [9] Facebook. *API Graph*. Mar. de 2020. URL: <https://developers.facebook.com/docs/graph-api>.

- [10] Thomas Feo y Mauricio Resende. «Greedy Randomized Adaptive Search Procedures». En: *Journal of Global Optimization* 6 (mar. de 1995), págs. 109-133. DOI: 10.1007/BF01096763.
- [11] L.R. Foulds. *Graph Theory Applications*. Universitext 7. Springer New York, 2012. ISBN: 9781461209331. URL: <https://books.google.es/books?id=5G4QBwAAQBAJ>.
- [12] Fred Glover. «Future paths for integer programming and links to artificial intelligence». En: *Computers operations research* 13 (1986), págs. 533-549.
- [13] Dominik Goeke, Mahdi Moeini y David Poganiuch. «A Variable Neighborhood Search heuristic for the maximum ratio clique problem». En: *Computers y Operations Research* 87 (2017), págs. 283 -291. ISSN: 0305-0548. DOI: <https://doi.org/10.1016/j.cor.2017.01.010>. URL: <http://www.sciencedirect.com/science/article/pii/S0305054817300102>.
- [14] Mauricio Granada-Echeverri y Jhon Santa. *Optimización combinatoria - de la teoría a la práctica*. Mayo de 2018. ISBN: 978-958-8859-43-9.
- [15] R D LUCE y A D PERRY. «A method of matrix analysis of group structure». En: *Psychometrika* 14.2 (jun. de 1949), págs. 95-116. DOI: 10.1007/bf02289146. URL: <https://pubmed.ncbi.nlm.nih.gov/18152948>.
- [16] *Metaheurística*. Oct. de 2019. URL: <https://es.wikipedia.org/wiki/Metaheur%C3%ADstica>.
- [17] Mahdi Moeini. «The Maximum Ratio Clique Problem: A Continuous Optimization Approach and Some New Results». En: *Modelling, Computation and Optimization in Information Systems and Management Sciences*. Ed. por Hoai An Le Thi, Tao Pham Dinh y Ngoc Thanh Nguyen. Cham: Springer International Publishing, 2015, págs. 215-227. ISBN: 978-3-319-18161-5.
- [18] Neo4j Neo4j Neo4j Neo4j. *Nuevas Formas de Predecir el Éxito de Inversiones Tableros Dinámicos 360 de Clientes. Detección de Fraude en Tiempo Real y Agilizando Procesos de AML*. Dic. de 2019. URL: <https://www.youtube.com/watch?v=KUtEee0et-w>.
- [19] A. Nikolaev, M. Batsyn y P. San Segundo. «Reusing the same coloring in the child nodes of the search tree for the maximum clique problem». En: vol. 8994. Springer Verlag, 2015, págs. 275-280. ISBN: 9783319190839.
- [20] Kuk-Kwon Park y col. «GRASP Algorithm for Dynamic Weapon-Target Assignment Problem». En: *Journal of the Korean Society for Aeronautical and Space Sciences* 47 (dic. de 2019), págs. 856-864. DOI: 10.5139/JKSAS.2019.47.12.856.
- [21] George Pólya. *Cómo Plantear y Resolver Problemas*. Ed. por Editorial Trillas. 1965.
- [22] George Pólya. *How to Solve It*. Ed. por Princeton. 1945.

- [23] Julio Ponce y col. «Algoritmo de Colonia de Hormigas para el Problema del Clique Máximo con un Optimizador Local K-opt». En: (ene. de 2008).
- [24] Jorge Ramírez. «Metaheurística GRASP para el problema Vertex Bisection Minimization.» En: 12 (ene. de 2018), págs. 28-41.
- [25] Joao Santos y col. «A parallel hybrid implementation using genetic algorithm, GRASP and reinforcement learning». En: jul. de 2009, págs. 2798 -2803. DOI: 10 . 1109/IJCNN. 2009 .5178938.
- [26] Samyukta Sethuraman y Sergiy Butenko. «The maximum ratio clique problem». En: *Computational Management Science* 12.1 (2015), págs. 197-218. DOI: 10 . 1007 / s10287-013-0197-z. URL: <https://doi.org/10.1007/s10287-013-0197-z>.
- [27] Wang Shaochang y col. «Application of Greedy Random Adaptive Search Algorithm (GRASP) in Flight Recovery Problem». En: ene. de 2018.
- [28] Y. Sun, X. Li y A. Ernst. «Using Statistical Measures and Machine Learning for Graph Reduction to Solve Maximum Weight Clique Problems». En: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2019), págs. 1-1. ISSN: 1939-3539. DOI: 10 . 1109/TPAMI . 2019 . 2954827.
- [29] Víctor Villavicencio. «GRASP para el problema de ruta de vehículos». En: (mar. de 2020).
- [30] Yiyuan Wang y col. «SCCWalk: An efficient local search algorithm and its improvements for maximum weight clique problem». En: *Artificial Intelligence* 280 (2020), pág. 103230. ISSN: 0004-3702. DOI: <https://doi.org/10.1016/j.artint.2019.103230>. URL: <http://www.sciencedirect.com/science/article/pii/S0004370219302164>.
- [31] Eric W. Weisstein. *Clique*. From MathWorld—A Wolfram Web Resource. URL: <https://mathworld.wolfram.com/Clique.html>.
- [32] Eric W. Weisstein. *Maximal Clique*. From MathWorld—A Wolfram Web Resource.
- [33] Eric W. Weisstein. *Maximum Clique*. From MathWorld—A Wolfram Web Resource.
- [34] Gang Yang y col. «An improved competitive Hopfield network with inhibitive competitive activation mechanism for maximum clique problem». eng. En: *Neurocomputing* 130 (2014), págs. 28-35. ISSN: 0925-2312.