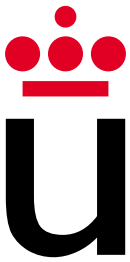


Universidad
Rey Juan Carlos

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

TODO

Agradecimientos

Índice general

Índice de figuras

Índice de cuadros

Listado de abreviaciones

ACO Ant Colony Optimization. 6, 11

AMS Adaptive Multi-Start. 6

AT Asynchronous Teams. 6

CA Cultural Algorithms. 6

EDA Estimation Distribution Algorithms). 6

FANS Fuzzy Adaptive Neighborhood Search. 6

GA Genetic Algorithms. 6

GLS Guided Local Search. 6

GRASP Greedy Randomized Adaptive Search Procedure. 6, 15, 21, 25, 31, 32

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 introduce el tema a tratar partiendo de conceptos previos que ayudarán al lector a entender mejor el desarrollo, siguiendo con la definición del problema y la motivación del mismo, 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 grafos se han convertido en una pieza fundamental en la vida cotidiana de las personas, desde conocidas redes sociales como Twitter o Facebook [9], figuras 1.1 y 1.2 , pasando por las telecomunicaciones, figura 1.5, redes de metro, figura 1.3, se ha simplificado para no contener todas las estaciones, hasta llegar a las simples moléculas, figura 1.4, pues si hay algo que todas estas cosas tienen en común, son los grafos, las conexiones entre nodos por una u otra razón, desde aficiones, hasta enlaces químicos [11], y encontrar bajo cierto criterio un subconjunto en los miles de millones de nodos y conexiones posibles que estos albergan es todo un reto, sin duda, y es más, realizarlo de una manera eficiente y en el menor tiempo posible, es un reto aún mayor.

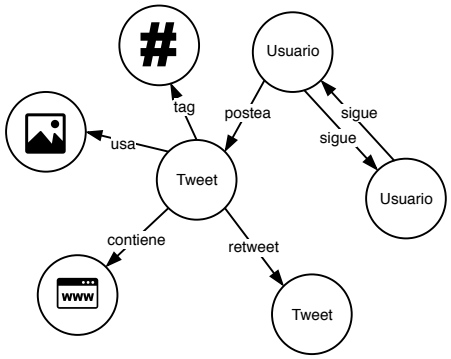


FIGURA 1.1: Grafo de un tweet en Twitter.

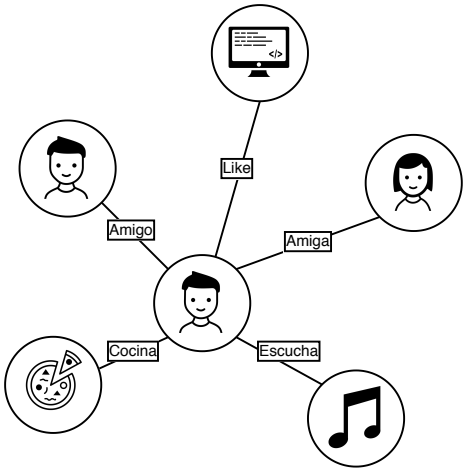


FIGURA 1.2: Grafo relaciones en Facebook.

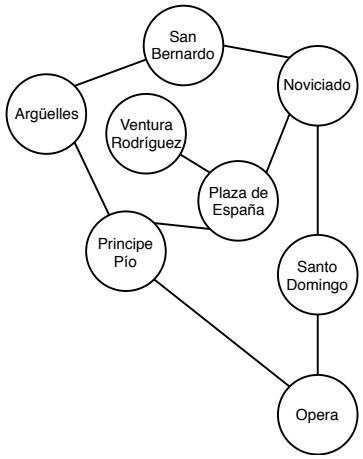


FIGURA 1.3: Grafo sección del metro de Madrid.

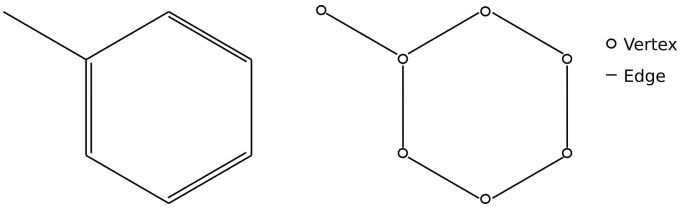


FIGURA 1.4: Representación estándar y en grafo de la molécula del Tolueno.
Fuente: https://www.blopig.com/blog/2019/01/graph-based-methods-for-cheminformatics

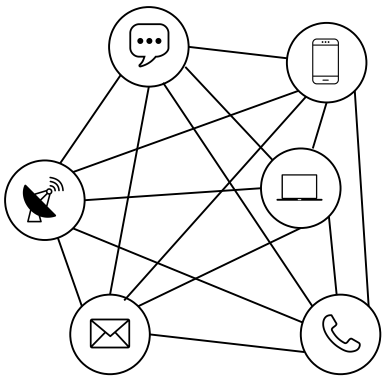


FIGURA 1.5: Grafo de telecomunicaciones.

Muchas son las aplicaciones que se les puede dar a los grafos y a la información que se obtiene de ellos, como nuevas terapias [2], visión computacional y reconocimiento de patrones [6], análisis de datos referentes al mercado bursátil [5] para conocer nuevas

formas de predicción de éxito en inversiones [18], y hacer un uso adecuado y responsable de los mismos es una gran tarea.

Por todo esto, es muy importante tratar los datos y sus relaciones de la manera mas eficiente posible, por lo que se han creado incluso base de datos orientadas a grafos, como es el caso de Neo4j [1], 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 de toda esta memoria, se van a describir algunos conceptos a continuación.

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. 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]. Los problemas que forman parte de la optimización combinatoria tiene en común que es difícil encontrar soluciones factibles puesto que hay muchas soluciones posibles y alguna de estas es óptima.

Algunos problemas conocidos que forman parte de este área, son el problema de la mochila [2] o el problema del vendedor viajero [3].

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 computo reducidos.

[1] https://neo4j.com/
[2] https://www.sciencedirect.com/topics/computer-science/knapsack-problem
[3] https://www.sciencedirect.com/topics/computer-science/travelling-salesman-problem

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], y en relación a esto, el término, fue acuñado por George Polya en su libro "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, aunque no exacta si eficiente, a un problema que no tiene un algoritmo heurístico específico o su aplicación es ineficiente [16]. Este término lo acuño Fred Glover en sus trabajos sobre búsqueda tabú en 1986 [12].

Existen un gran variedad de algoritmo metaheurísticos que se pueden clasificar de diferentes maneras según el enfoque, en la figura 1.6 se muestra una de ellas basada en el número de soluciones.

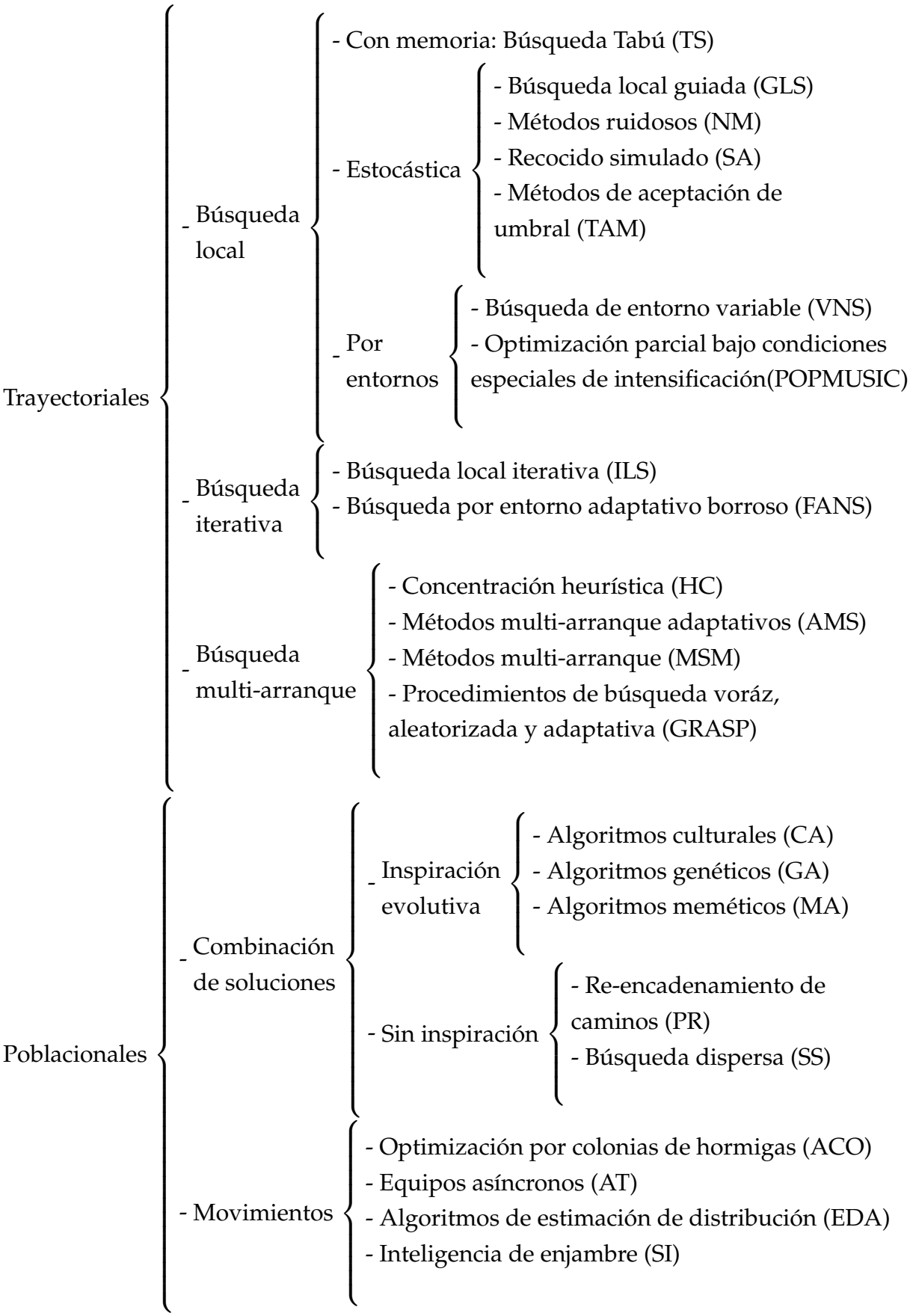


FIGURA 1.6: Clasificación de metaheurísticas

Dos conceptos a tener en cuenta en un algoritmo metaheurístico son la intensificación o explotación en una región, de modo que una alta intensificación hará que el algoritmo realice una búsqueda más exhaustiva en esa región, y la diversificación o exploración de 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 está 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:

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

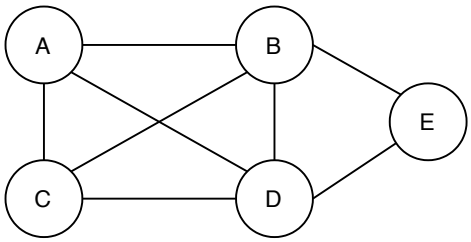
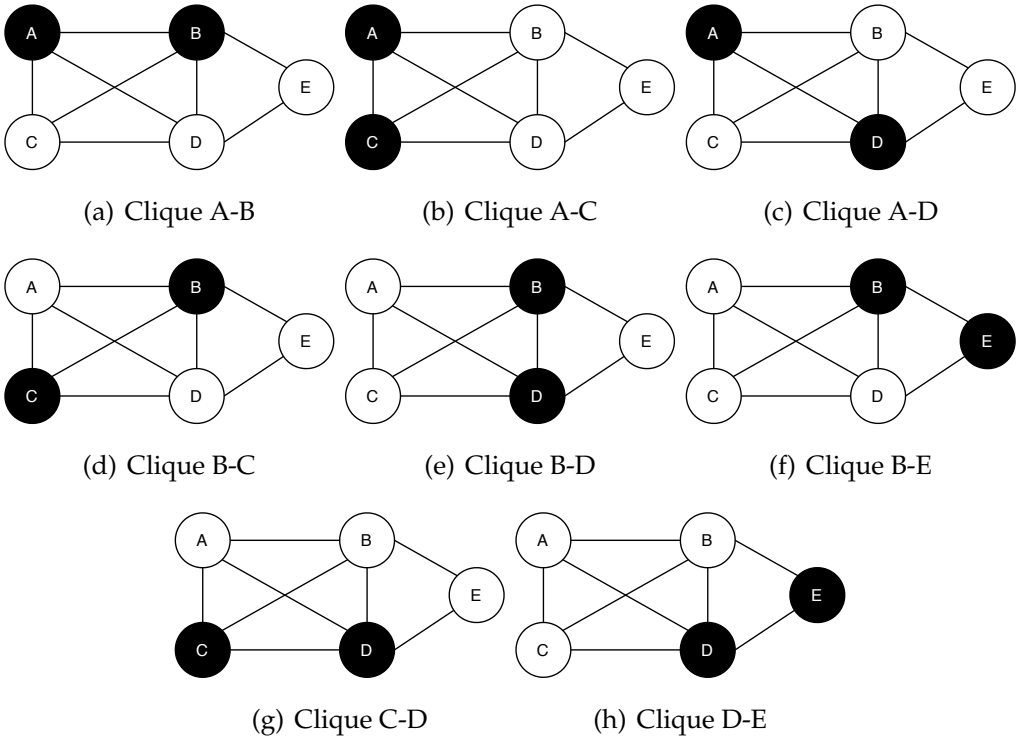


FIGURA 1.7: Diagrama de un grafo.

En la figura 1.7 se muestra un grafo compuesto por el conjunto de nodos $V = \{A, B, C, D, E\}$, en este grafo se encuentran como indican las figuras 1.8 y 1.9 los posibles cliques de este grafo, donde k indica el número de nodos del clique. Cabe resaltar, que los nodos por si solos también forman clique, pero no se ha incluido para no hacer más compleja la figura.

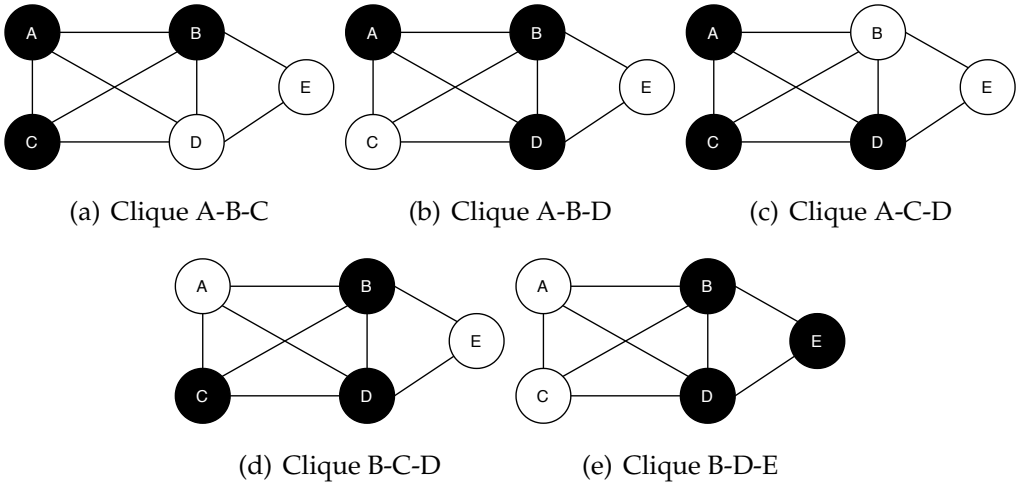


(a) Clique A-B (b) Clique A-C (c) Clique A-D

(d) Clique B-C (e) Clique B-D (f) Clique B-E

(g) Clique C-D (h) Clique D-E

FIGURA 1.8: Cliques $k = 2$ del grafo.



(a) Clique A-B-C (b) Clique A-B-D (c) Clique A-C-D

(d) Clique B-C-D (e) Clique B-D-E

FIGURA 1.9: Cliques $k = 3$ del grafo.

Es importante resaltar que un clique perteneciente a un grafo, no implica que sea máximo, puesto que un clique máximo es el clique el cuál no es posible ampliar, es decir, no se pueden añadir a este más adyacentes que cumplan con las restricciones necesarias

para formar un clique y a su vez es el de mayor tamaño del grafo, como se muestra en la figura 1.10, a diferencia de un clique que se podría denominar simple [32][33]. Y en este caso, se obtiene que:

$$\omega(G) = 4$$

donde ω denota el número de vértices del clique, su cardinalidad.

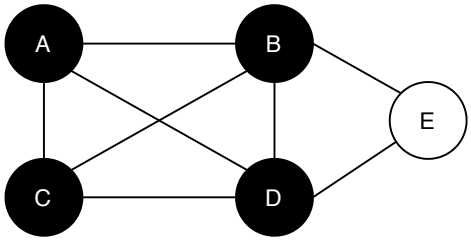


FIGURA 1.10: Clique máximo del grafo.

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 en cuanto a teoría de grafos es fundamental y muy estudiado, y más concretamente la búsqueda del clique máximo dentro de un grafo, a este problema se le conoce como "El problema del clique máximo" o MCP por sus siglas en ingles "Maximum clique problem", y es catalogado según los problemas NP como un problema NP-completo.

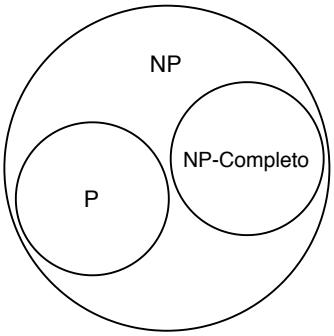


FIGURA 1.11: Diagrama de problemas NP.

Los problemas denominados como NP, figura 1.11, acrónimo de non-deterministic polynomial time o tiempo polinominal no determinista, en teoría de complejidad computacional, 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, además, contiene todos los problemas de tipo P y de tipo NP-completos como es el caso del problema del clique máximo.

Una variante de este problema es el llamado "Problema de clique de peso máximo" o MWCP por sus siglas en inglés a 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 esta estrechamente ligado al problema tratado en este trabajo final de grado, el cual busca el clique de ratio máximo ya que si se asocian dos pesos no negativos a cada vértice se obtiene el problema del clique de ratio máximo MRCP o "Maximum Ratio Clique Problem". En este problema se busca el clique máximo en un grafo con la mayor proporción de ratio. Esta proporción se define como las sumas de los pesos de los vértices:

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

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

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

Por lo tanto, el objetivo del problema es maximizar esta proporción 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, $\{v_1, \ldots, v_n\}$, 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 $\widehat{S}$, siempre y cuando se cumplan las restricciones 1.2 - 1.4.

$$maximizar \qquad f = \frac{\sum_{i=1}^{n} p_i x_i}{\sum_{i=1}^{n} q_i x_i} \tag{1.1}$$

$$sujeto\ a:$$

$$x_i + x_j \leqslant 1 : \forall (v_i, v_j) \notin E,\ i \neq j, \tag{1.2}$$

$$\sum_{i=1}^{n} (a_i j) x_i \ \geqslant 1 : \forall v_j \in V, \tag{1.3}$$

$$x_i \in 0,\ 1 : \forall v_i \in V. \tag{1.4}$$

El problema del clique de ratio máximo ha sido catalogado como un problema NP-difícil o NP-complejo, figura 1.12, por lo que no es posible obtener una solución factible por métodos heurísticos o exactos.

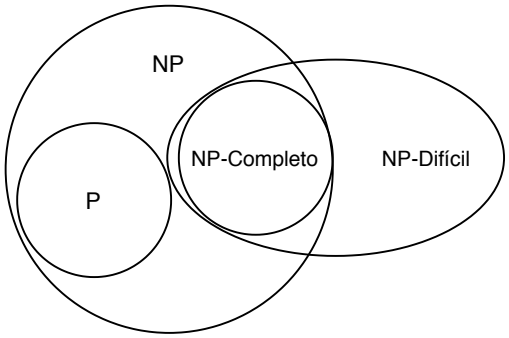


FIGURA 1.12: Diagrama de problemas NP y NP-difícil.

1.5. Estado del arte

El análisis sobre el estado del arte que se ha realizado sobre el problema del clique de ratio máximo y todo lo que rodea al mismo ha sido documentado mediante la siguiente literatura, si bien es cierto que no existe demasiada 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] donde se muestra un algoritmo ACO con un optimizador local K-opt.

También se ha estudiado en mayor medida el problema de la búsqueda del clique de peso máximo, más cercano al tratado en este trabajo final de grado, como se puede

observar en [30] se implementan sendos algoritmos de búsqueda local llamados SCCWalk y SCCWalk4L, [28] en el que se hace uso del muestreo estocástico 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) el cuál provee resultados competitivos, 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 realizados sobre el problema del clique de ratio máximo el de Dominik Goeke, Mahdi Moeini y David Poganiuch [13] el cuál se ha tomado como referencia para realizar este trabajo final de grado, 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 y un tiempo de computo menor.

Capítulo 2

Objetivos

En este capítulo se indican los objetivos, tanto principales como secundarios, para la realización de este trabajo fin de grado:

2.1. Objetivos principales

- 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.

2.2. Objetivos secundarios

- 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.

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 Adaptative 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 α, que oscilará entre 0 y 1 y marca la aleatoriedad de la función de manera que si α 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 μ, 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 μ. 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) $v \leftarrow rnd(V)$
(2) $S \leftarrow \{v\}$
(3) $CL \leftarrow \{u \in V : (u, v) \in E\}$
(4) **mientras** $|CL| \neq 0$ **hacer**
(5) $\quad g_{\min} \leftarrow \min_{c \in CL} g(c)$
(6) $\quad g_{\max} \leftarrow \max_{c \in CL} g(c)$
(7) $\quad \mu \leftarrow g_{\max} - \alpha(g_{\max} - g_{\min})$
(8) $\quad RCL \leftarrow \{c \in CL : g(c) \geq \mu\}$
(9) $\quad u \leftarrow rnd(RCL)$
(10) $\quad S \leftarrow \cup \{u\}$
(11) $\quad CL \leftarrow CL \backslash \{u\} \backslash \{w : (u, w) \notin E\}$
(12) **fin**
(13) **devolver** S

Algoritmo 2: Contructivo voráz.

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

Algoritmo 3: Pseudocódigo método buscarMejor de tipo ratio.

```
(1) ratio ← −1
(2) nodoElegido ← NULO
(3) para nodo hacer
(4)     ratioNodo ← calcularRatio(nodo)
(5)     si ratioNodo > ratio entonces
(6)         ratio ← ratioNodo
(7)         nodoElegido ← 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 ← −1
(2) nodoElegido ← NULO
(3) para nodo hacer
(4)     vecinosNodo ← SeleccionAdyacentes(nodo)
(5)     si |vecinosNodo| > vecinos entonces
(6)         vecinos ← |vecinosNodo|
(7)         nodoElegido ← 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 ← ∅
(2) vecinos ← obtenerVecinos(solucion)
(3) vecinosOrdenados ← ordenarVecinos(vecinos)
(4) para nodo hacer
(5)     solucion ← incluirNodo(solucion, nodo)
(6)     si no esClique(solucion) entonces
(7)         solucion ← excluirNodos(solucion, nodo)
(8)     solucion ← 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.

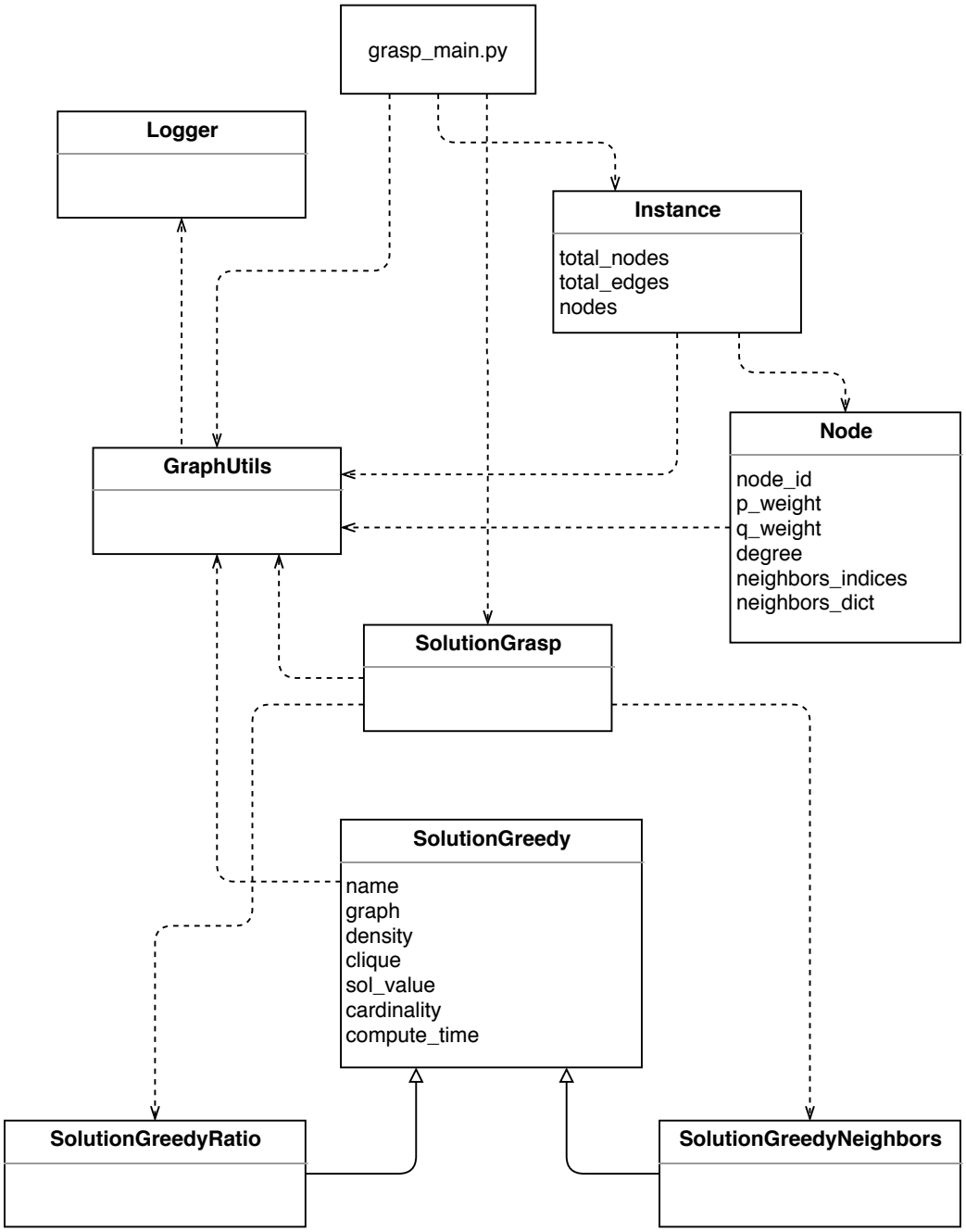


FIGURA 4.1: Diagrama de clases del proyecto.

4.2. Implementación

La implementación de este proyecto se basa en la ejecución de un script[1] 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 α 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.

[1] 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 numero 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[2] 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 μ, calculado como se mostró en el algoritmo 1 con el valor configurado α, 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.

[2]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[3], 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[4], 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.

[3]https://docs.python.org/3/library/unittest.html
[4]https://trello.com/es

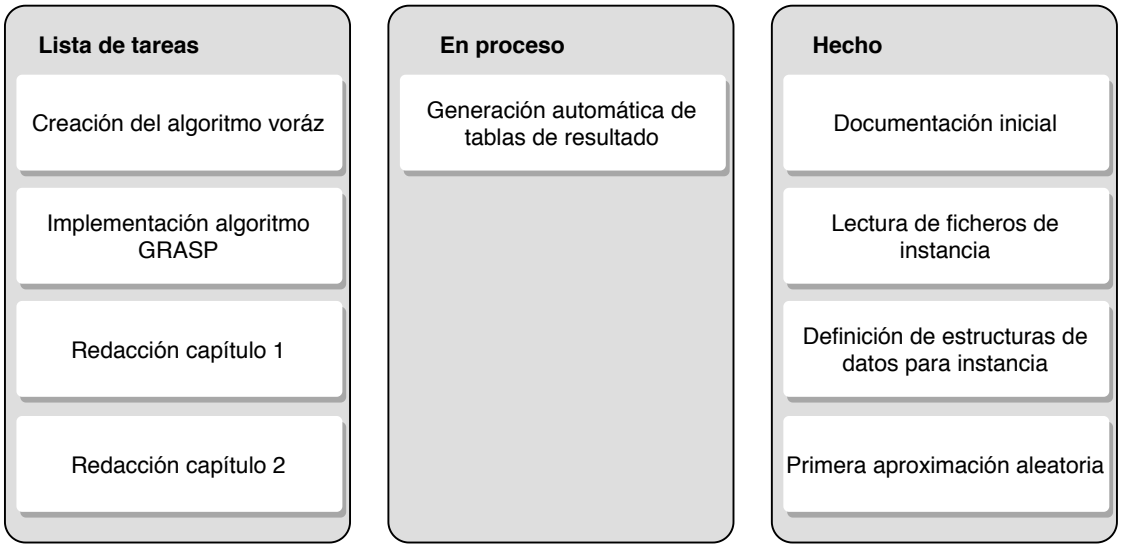


FIGURA 4.2: Ejemplo del sistema de tarjetas de Trello.

En cuanto al mantenimiento de versiones del proyecto se ha usado el sistema de control de versiones Git[5]. Este control de versiones se ha gestionado a su vez a través del portal de alojamiento de repositorios GitHub[6]. 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[7], 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.

[5]https://git-scm.com/
[6]https://github.com/
[7]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[1], en su versión 3.7.4, a través del IDE [2], PyCharm[3] de JetBrains en su versión 2019.3.1.

[1]https://www.python.org/
[2]Entorno de desarrollo integrado.
[3]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 contrucción de una turbina en ese punto.
- **Conjunto de tipo E:** Estas instancias están extraídas del segundo y décimo DIMACS Implementation Challenge[4], 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

Mostrar y comentar los resultados obtenidos.

[4] http://dimacs.rutgers.edu/programs/challenge/

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 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:

- aaa

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 ar tifi cial 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.