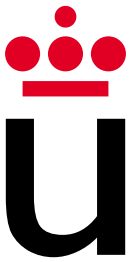


Universidad
Rey Juan Carlos

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 se explica como se ha abordado el problema del clique de ratio máximo o Maximal Clique Problem (MRCP). Durante el proceso se expone la metaheurística empleada para la obtención de una solución, en este caso Greedy Randomized Adaptive Search Procedure (GRASP) o Procedimiento de Búsqueda Voraz Aleatoria y Adaptativa, el cual ofrece buenas soluciones en un tiempo de computo razonable. Se explica, además, las distintas variantes del problema y las diversas aplicaciones de este, como son las redes sociales. Finalmente se analizan los datos obtenidos y comparados con el estudio previo del problema tratado por Dominik Goeke, Mahdi Moeini y David Poganiuch.

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

Índice de figuras

Índice de tablas

Listado de abreviaciones

API Application Programming Interface. 2

GIL Global Interpreter Lock. 35

GRASP Greedy Randomized Adaptive Search Procedure. 13, 20, 24, 31, 33, 34, 36, 37

IDE Integrated Development Environment. 27

MCP Maximum Clique Problem. 8, 34

MRCP Maximum Ratio Clique Problem. 9

MSM Multi-Start Methods. 11

MWCP Maximum Weight Clique Problem. 9, 33

VNS Variable Neighborhood Search. 11

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 de este, así como conceptos previos relacionados.
- 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 estos.
- Capítulo 6, en este capítulo se muestran las conclusiones 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[1], 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 [2] con la que poder interactuar con los datos que son recopilados en ella [9].

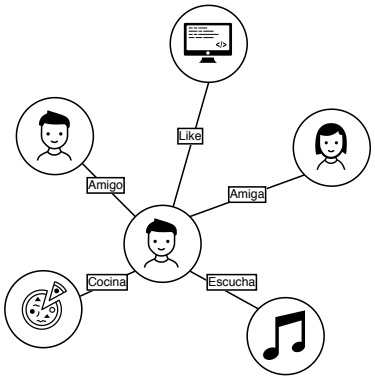


FIGURA 1.1: Grafo de relaciones en Facebook.

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 comunicación 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.

[1] https://es-es.facebook.com/

[2] 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.

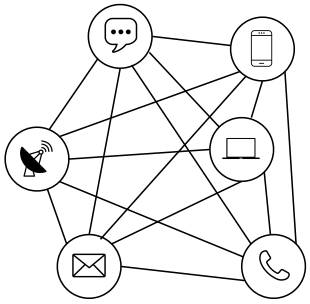


FIGURA 1.2: Grafo de telecomunicaciones.

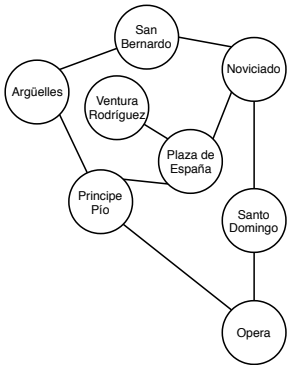


FIGURA 1.3: Grafo sección del metro de Madrid.

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

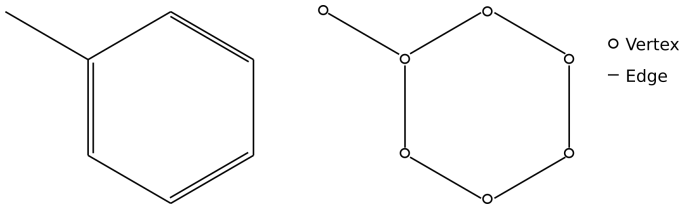


FIGURA 1.4: Representación estándar y en grafo de la molécula del Tolueno.
Fuente: https://www.blopig.com/blog/2019/01/graph-based-methods-for-cheminformatics

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 [3], 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 [3], 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.

[3]https://neo4j.com/

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 esta 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[4] o el problema del vendedor viajero[5]. 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, 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."[8]. 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 "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].

[4]https://www.sciencedirect.com/topics/computer-science/knapsack-problem
[5]https://www.sciencedirect.com/topics/computer-science/travelling-salesman-problem

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.

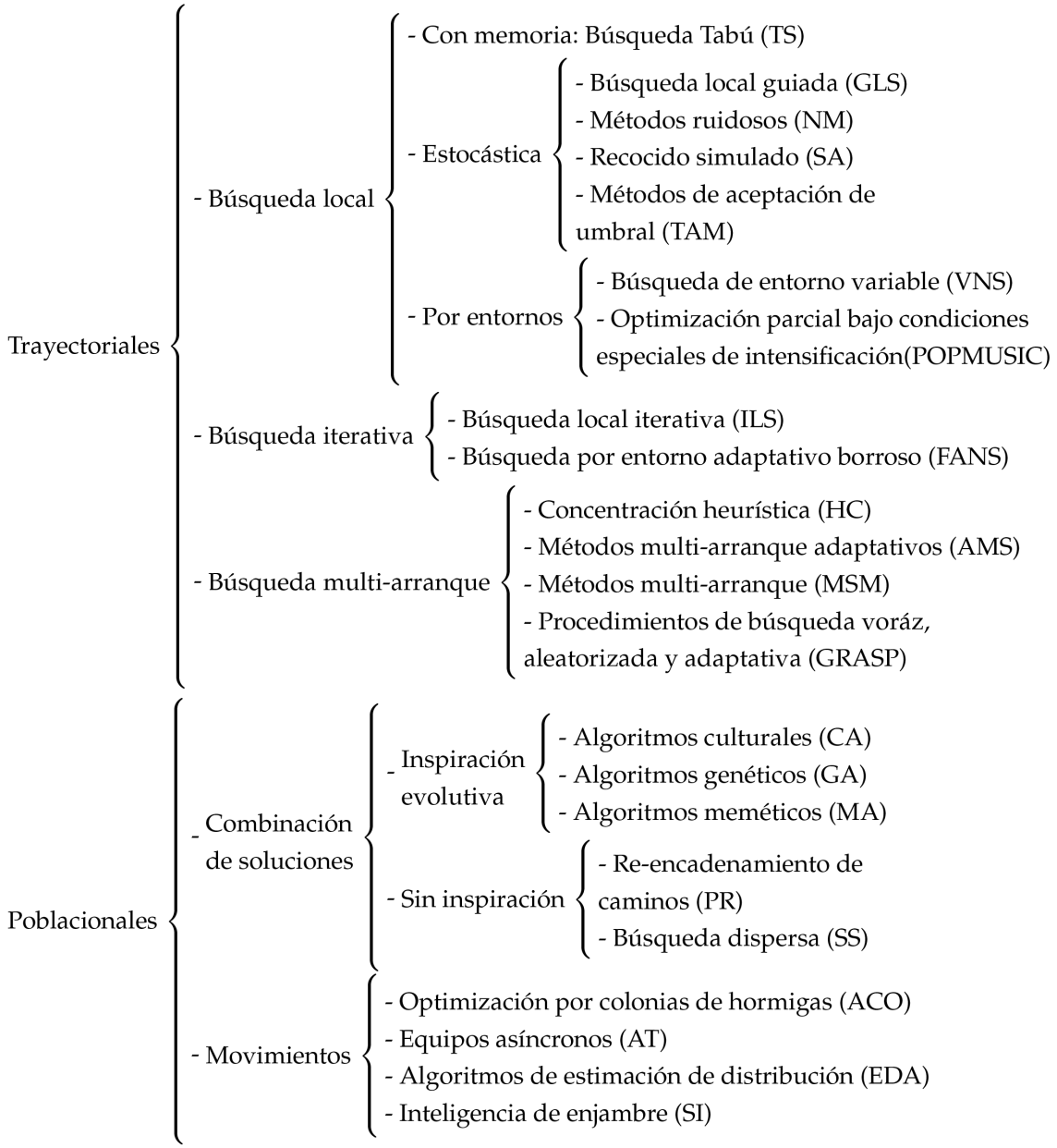


FIGURA 1.5: Clasificación de metaheurísticas

Dos conceptos importantes 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 [2].

1.3.3. Clique

El término clique define a un grupo de personas que tienen intereses comunes. Según el Cambridge Dictionary un clique es "un grupo pequeño de personas que invierten su tiempo juntos y excluyen al resto de personas que no forman parte de este" [1]. Asemejando esta definición con un grafo, se tienen los nodos que serían las personas que forman el grupo y las aristas del grafo, que corresponderían con los intereses comunes entre esas personas. [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:

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

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

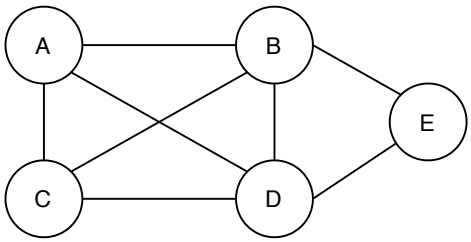
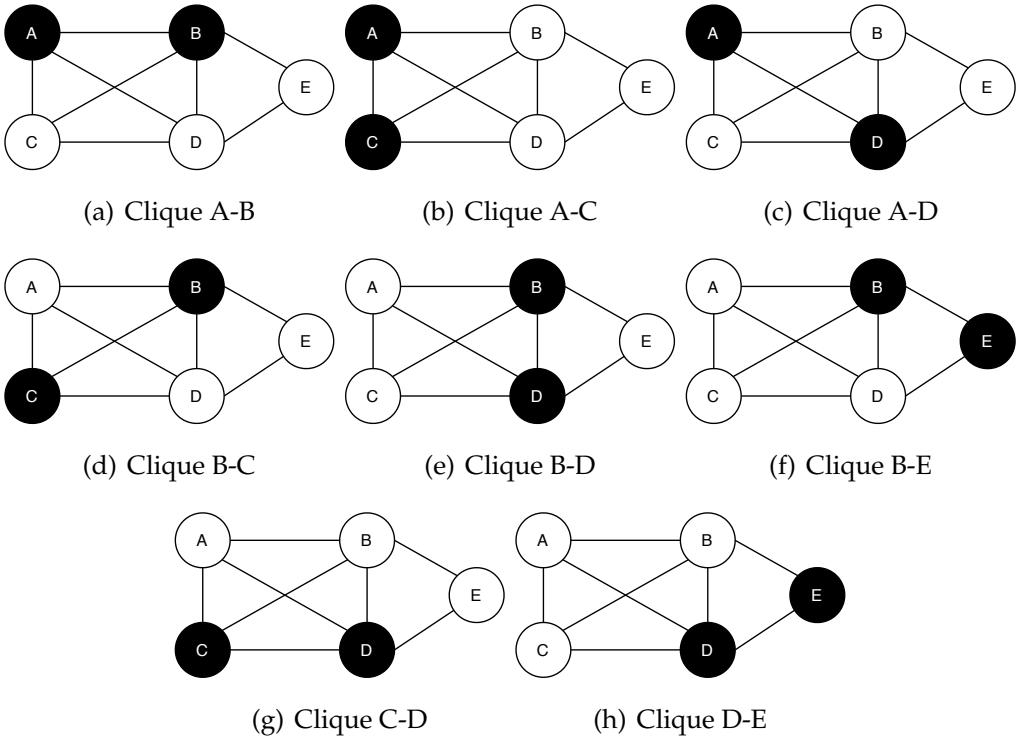


FIGURA 1.6: Diagrama de un grafo.

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.



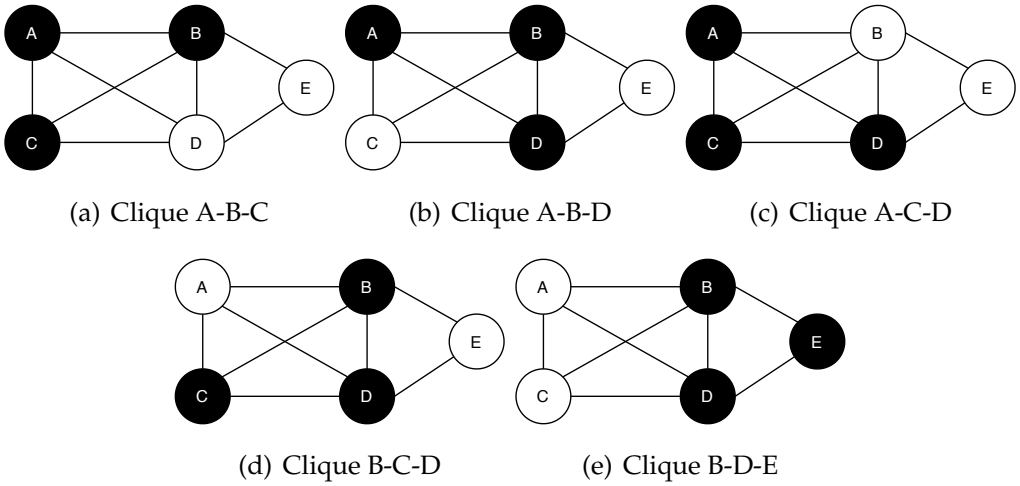
(a) Clique A-B (b) Clique A-C (c) Clique A-D

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

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

FIGURA 1.7: Cliques de cardinalidad 2 del grafo.

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



(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 1.8: 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].

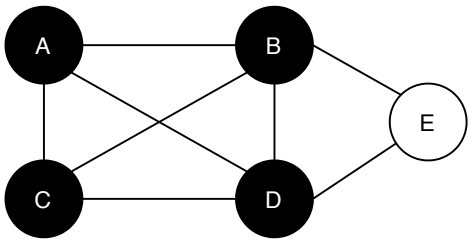


FIGURA 1.9: Clique máximo del grafo.

En este caso, se obtiene $\omega(G) = 4$, donde ω 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.

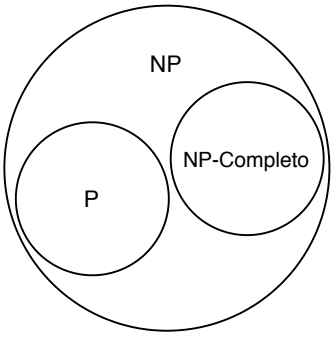


FIGURA 1.10: Diagrama de problemas NP.

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 esta 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, $\{v_1, \ldots, v_n\}$, 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.

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

$$sujeto\ a:$$

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

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

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

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

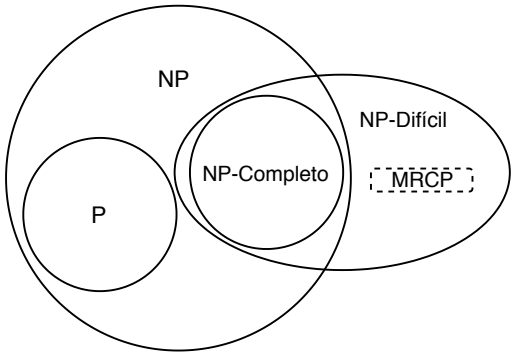


FIGURA 1.11: MRCP clasificado en los problemas NP-difícil.

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, [4], [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 (Greedy Randomized Adaptative Search Procedure) o en castellano procedimiento de búsqueda voraz aleatorizado y adaptativo, 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 uno de ellos es una iteración de un procedimiento que está constituido por dos partes bien diferenciadas. Por un lado, la fase constructiva, en la que se obtiene una solución de buena calidad, y por otro una fase de mejora, en la que, partiendo de la solución obtenida en la fase anterior, se intenta mejorar localmente [2]. En [27] [25] [20] [5] [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.

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) $\quad c \leftarrow CL$

(6) $\quad g_{min} \leftarrow \min_{c \in CL} g(c)$

(7) $\quad g_{max} \leftarrow \max_{c \in CL} g(c)$

(8) $\quad \mu \leftarrow g_{max} - \alpha(g_{max} - g_{min})$

(9) $\quad RCL \leftarrow \{c \in CL : g(c) \geq \mu\}$

(10) $\quad u \leftarrow rnd(RCL)$

(11) $\quad S \leftarrow S \cup \{u\}$

(12) $\quad CL \leftarrow CL \backslash \{u\} \backslash \{w : (u, w) \notin E\}$

(13) **fin**

(14) **devolver** S

Dichos algoritmos se explicarán aplicados al problema tratado de la siguiente manera:

Teniendo un grafo $G = (V, E)$ donde V son los vértices o nodos del grafo, y E las aristas que unen estos nodos.
Se parte del paso 1 del algoritmo 1, donde se toma un vértice v aleatorio de entre los vértices del grafo. En el paso 2 el vértice 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, como se indica en el paso 3, definida como todos los nodos adyacentes a v que forman parte de la lista de nodos del grafo de partida. A continuación, se toma un elemento de la lista de candidatos, paso 5, con el que se obtiene mediante una función voraz un listado de nodos. Esta función será determinada antes de iniciar el proceso e indicará si se deben obtener los nodos según su ratio o según el número de adyacentes al nodo tratado, será explicada con mayor detalle en la sección 3.1.1. De este listado se escogen los valores de ratio máximo y mínimo de los nodos, descrito en los pasos 6 y 7, y junto a α se obtiene el valor de μ, mostrado en el paso 8.

El valor de μ indica el umbral de la lista restringida de candidatos RCL. Esta lista se genera mediante la lista de candidatos CL ordenada de mayor a menor valor y, como se muestra en la figura 3.1, se toman los primeros valores hasta alcanzar el valor umbral, esa función está descrita en el paso 9.

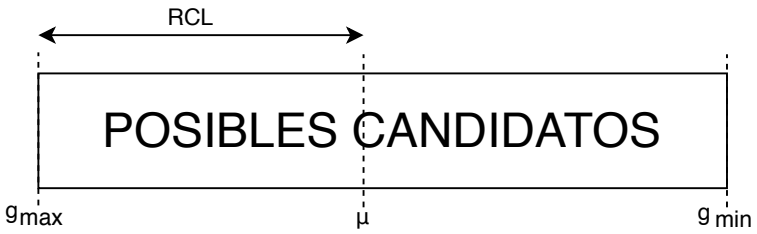


FIGURA 3.1: Detalle de la lista restringida de candidatos (RCL).

Respecto al valor de α, será configurado antes de este proceso y oscilará entre 0 y 1. Este valor va a indicar la cantidad de nodos que se incluirían en la RCL. Por un lado, un valor igual a 1 supondría añadir el mejor candidato, convirtiendo el algoritmo en más voraz. Por el contrario, si se configura un valor de 0, se estarían incluyendo más candidatos en el listado.

De la RCL se elegirá, de manera aleatoria, un nodo u como se muestra en el paso 10, se añadirá a la lista solución S, paso 11 y será eliminado de la lista de candidatos junto con los nodos que no sean adyacentes a este, paso 12.

Este procedimiento será repetido hasta que la lista de candidatos este vacía, obteniéndose en ese momento la lista S final, que conformará la solución preliminar se devolverá en el paso 14 del algoritmo 1. Esta solución será usada en la fase de mejora, explicada con detalle en la sección 3.1.2, con el fin de optimizarla.

3.1.1. Fase constructiva

En esta fase se ha usado el algoritmo 2 para definir la función constructiva. Esta es común a ambos constructivos y cada uno de ellos se especializa en como obtener el mejor nodo a incluir en el listado que será devuelto.

Algoritmo 2: Pseudocódigo del constructivo voraz.

```
(1)  S ← ∅
(2)  Adyacentes ← SeleccinAdyacentes(nodo)
(3)  mientras |Adyacentes| ≠ 0 hacer
(4)  |  candidato ← buscarMejor(Adyacentes)
(5)  |  si formaClique(candidato) entonces
(6)  |  |  Adyacentes ← Adyacentes ∩ SeleccinAdyacentes(candidato)
(7)  |  |  S ← S ∪ {candidato}
(8)  |  fin
(9)  |  en otro caso
(10) |  |  Adyacentes ← Adyacentes\{candidato}
(11) |  fin
(12) fin
(13) devolver S
```

Partiendo del nodo candidato que se obtuvo en el paso 5 del algoritmo 1 se crea un listado solución vacío en el paso 1. A continuación se obtienen, como se indica en el paso 2, sus nodos adyacentes. De esta lista se selecciona el mejor candidato, paso 4. La manera de seleccionarlo dependerá de que constructivo se configuró y será descrita con mayor detalle en las secciones 3.1.1.1 y 3.1.1.2.

Tras la selección, en el paso 5 se comprueba si este candidato es factible. Esta comprobación verifica que el candidato es adyacente a todos los nodos que formen la solución S actual. Si es factible, se eliminan del listado de adyacentes los nodos que no son adyacentes al nodo, paso 6., y se añade al listado solución que se creó al inicio, paso 7. En caso de no ser factible, se eliminará este candidato del listado de adyacentes como se aprecia en el paso 10. Finalmente, una vez es vacía la lista de adyacentes, se devuelve el listado solución, paso 13.

3.1.1.1. Constructivo en función del ratio

Esta especialización del constructivo voraz se basa en el algoritmo 3 que obtiene una solución buscando el mayor ratio de cada nodo adyacente al de partida.

Algoritmo 3: Pseudocódigo método buscarMejor de tipo ratio.

```
(1) ratio ← −1
(2) nodoElegido ← NULO
(3) para nodo ϵ adyacentes hacer
(4)     ratioNodo ← calcularRatio(nodo)
(5)     si ratioNodo > ratio entonces
(6)         ratio ← ratioNodo
(7)         nodoElegido ← nodo
(8) devolver nodoElegido
```

Partiendo del listado de nodos adyacentes, en el paso 1 se marca un ratio de referencia. Mediante un bucle, por cada nodo del listado, en el paso 4, se calcula el ratio del mismo, a continuación se comprueba si es mayor este ratio que el de referencia. En caso afirmativo, se actualiza, paso 6, y se guarda el nodo, paso 7. Este proceso finalizará al haber tratado todos los nodos del listado. Una vez finalizado se devuelve el nodo que se ha elegido y el cual tiene mayor ratio.

3.1.1.2. Constructivo en función de los adyacentes

En este caso, se define la especialización del constructivo basándose en el algoritmo 4 que obtiene el nodo con mayor número de vecinos respecto al de partida.

Al igual que en la sección 3.1.1.1 se parte del listado de nodos adyacentes. En el paso 1 se almacena un valor de referencia con el número de vecinos. A continuación, se compara cada nodo del listado de adyacentes, pasos 4 y 5, con el de referencia, buscando el que mayor número de adyacentes tenga. En caso de encontrar un nodo con mayor número de vecinos, se actualiza el valor de referencia y se almacena el nodo elegido, pasos 6 y 7. Cuando se han tratado todos los nodos del listado, el nodo elegido es devuelto por la función, paso 8.

Algoritmo 4: Pseudocódigo método buscarMejor de tipo adyacentes.

```
(1) vecinos ← −1
(2) nodoElegido ← NULO
(3) para nodo ϵ adyacentes hacer
(4)     vecinosNodo ← SeleccionAdyacentes(nodo)
(5)     si |vecinosNodo| > vecinos entonces
(6)         vecinos ← |vecinosNodo|
(7)         nodoElegido ← nodo
(8) devolver nodoElegido
```

A continuación, se muestra un ejemplo de la búsqueda del mejor nodo, partiendo del grafo formado por los nodos $\{A, B, C\}$, donde los siguientes nodos a añadir son D y E.

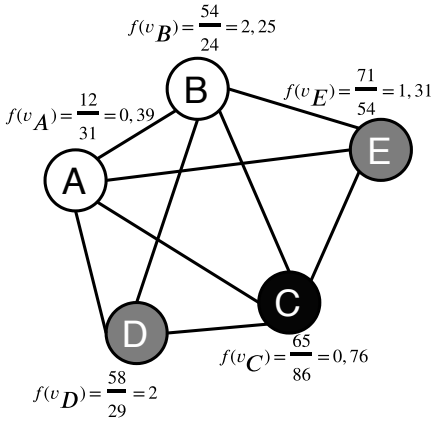


FIGURA 3.2: Elección de nodo por mayor ratio.

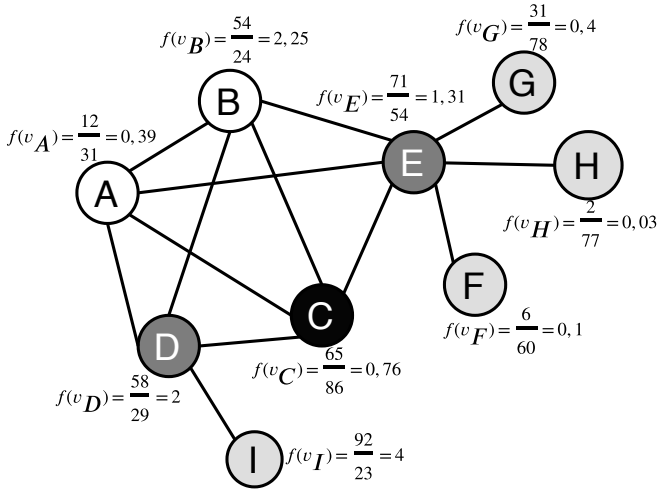


FIGURA 3.3: Elección de nodo por mayor número de adyacentes.

En el caso de la figura 3.2 se escoge el nodo D que tiene una relación mayor entre sus pesos p y q, $f(v_D) = \frac{58}{29} = 2$, respecto del nodo E. Como siempre se selecciona un solo nodo en el paso 4 del algoritmo 2, en lugar de un subconjunto de nodos, este caso puede suponer un mayor ratio final.

Por otro lado, en la figura 3.3, se observa la opción de escoger el nodo con mayor número de vecinos. A priori, puede suponer una mejor solución añadir mayor cantidad de adyacentes y por lo tanto, mayor ratio. Al tratarse de un algoritmo voraz,

esta estrategia puede dejar atrás nodos más prometedores para alcanzar mayor ratio en la solución final. En este ejemplo, se escoge como mejor nodo el E, ya que tiene más adyacentes. Posteriormente se añade, por ejemplo, el nodo G, obteniendo un ratio de $f(\{v_A, v_B, v_C, v_E, v_G\}) = 0,96$. Esta selección está dejando atrás los nodos D e I, los cuáles aportan mayor ratio a la solución, con un valor en este caso de $f(\{v_A, v_B, v_C, v_D, v_I\}) = 1,46$, superior al obtenido mediante el otro constructivo.

3.1.2. Fase de mejora

Para esta segunda fase, se ha definido el algoritmo 5, el cuál parte de la solución obtenida previamente en la fase constructiva, formada por los nodos que forman un clique.

Algoritmo 5: Pseudocódigo algoritmo búsqueda local.

```
(1) vecinos ← ∅
(2) vecinos ← obtenerVecinos(solucin)
(3) vecinosOrdenados ← ordenarVecinos(vecinos)
(4) para nodo ϵ vecinosOrdenados hacer
(5)     solucion ← incluirNodo(solucin, nodo)
(6)     si no esClique(solucin) entonces
(7)         solucin ← excluirNodo(solucin, nodo)
(8)     solucin ← incluirAdyacentes(solucin, nodo)
(9) devolver solución
```

Con esta solución se obtienen en el paso 2 todos los vecinos de cada nodo de esta. Estos son ordenados de mayor a menor ratio en el paso 3. Esta ordenación se realiza con el fin de aumentar las posibilidades de obtener un mejor valor de ratio. Se obtiene el primer nodo del listado y se añade a la solución, paso 6, comprobando posteriormente si esta solución forma o no un nuevo clique en el paso 6. Esta comprobación verifica que todos los nodos son adyacentes entre sí, formando un clique.

Si añadir el nodo no formara una solución, en el paso 7 se excluyen todos los nodos que impiden que se forme una solución factible. En caso contrario, ese nodo añadido en el paso 5 se mantendría y a su vez, en el paso 8, se añadirían todos sus nodos adyacentes, con el fin de obtener un clique de tamaño máximo, como marca la restricción del problema. Finalmente, una vez comprobados todos los nodos se devuelve el listado solución en el paso 9.

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 de este.

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 planteado la arquitectura que se muestra en el diagrama de clases de la figura 4.1. La implementación 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 que 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 voraz. La definición completa se refiere a los atributos density, el cuál indica la densidad del grafo del que parte el proceso; clique, que 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, en este caso, con mayor número de nodos vecinos.

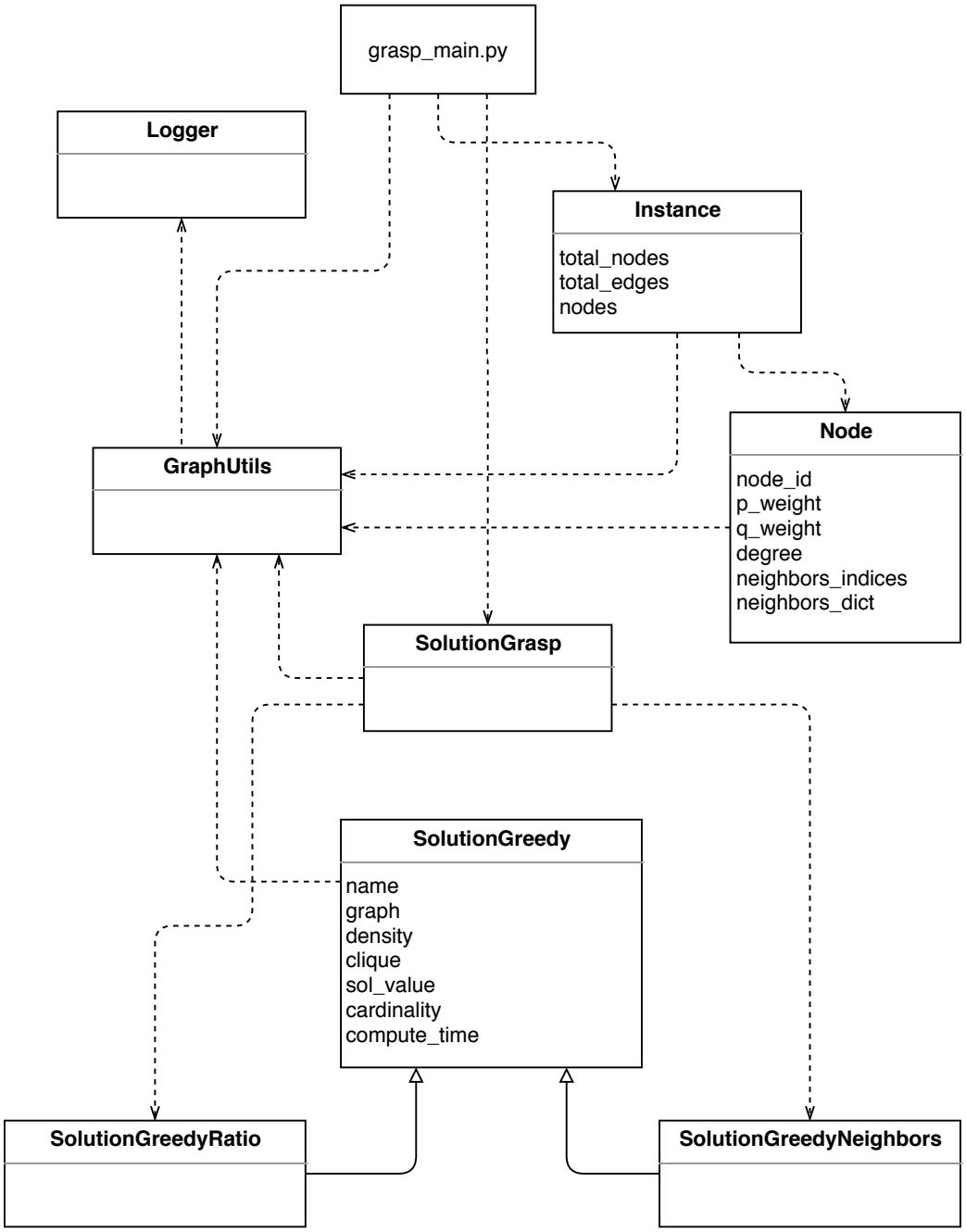


FIGURA 4.1: Diagrama de clases del proyecto.

4.2. Implementación

La implementación de este proyecto se basa en la ejecución de un script[1], escrito en lenguaje Python, que parte de una clase principal caracterizada por contener la siguiente sentencia de código:

```
if__name__ == "__main__"
```

Dicha sentencia 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 encuentren en la ruta de recursos definida y, por cada uno de los ficheros encontrados, crea un objeto de tipo Instance, añadiendo la siguiente 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 tenga 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 interpretadas para su ejecución.

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

Tras terminar esta operación se 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, los cuales serán explicados más adelante.

A partir de este punto se invoca la función $find\_grasp\_solution$ disponible en la clase SolutionGrasp y que implementa al algoritmo GRASP anteriormente descrito. Esta función 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* que obtiene un listado de posibles candidatos. 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 cual 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.

Una vez se obtiene el listado de candidatos, es procesado mediante la función *get_rcl*, la cual haciendo uso del valor de μ, calculado como se mostró en el algoritmo 1 con el valor de α, permite obtener la lista de candidatos restringida denominada en el algoritmo como RCL. A partir de este momento el siguiente paso es escoger de manera aleatoria un nodo de esta lista e incluirlo en el conjunto solución, eliminando de la lista CL los nodos 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 el código ordenado se ha implementado la clase GraphUtils, la cuál contiene información necesaria y métodos útiles para el procesado de las instancias, así como la exportación a ficheros de tipo CSV[3] de 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.

[3]Es un tipo de ficheros de texto simple en el que se almacenan datos separados en columnas por comas o por punto y coma, y las filas por salto de línea.

Con el fin de probar las diferentes funciones del proyecto se han escrito casos de prueba mediante la librería de Python unittest[4].

4.3. Metodología empleada

Para el desarrollo de este proyecto se ha optado por seguir una metodología de tipo iterativa e incremental, la cual permite evolucionar el proyecto progresivamente e ir adaptándose a 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 o 4 semanas entre ellas, corrigiendo errores de la iteración anterior si los hubiera y aumentando la funcionalidad del producto a entregar, de esta manera se consigue una evolución progresiva y segura de los requerimientos que el proyecto exige.

Para mantener el control y administrar las tareas a realizar, las que se están desarrollando y las que han finalizado, se ha usado el administrador de proyectos Trello[5] el cuál mediante tarjetas permite conocer las tareas del proyecto, así como su estado, detalles de esta 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.

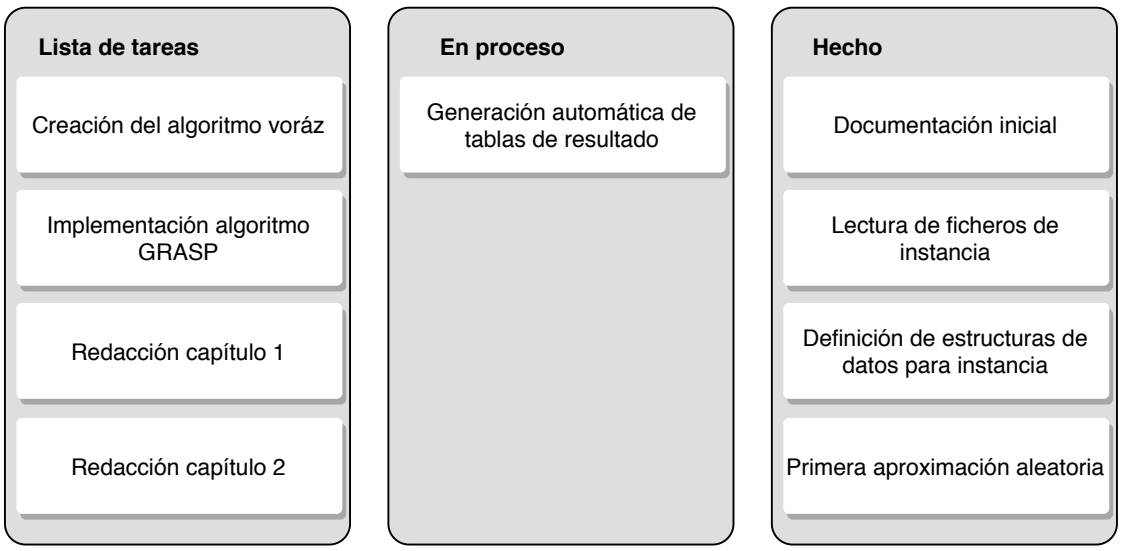


FIGURA 4.2: Ejemplo del sistema de tarjetas de Trello.

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

En cuanto al mantenimiento de versiones del proyecto, se ha usado el sistema de control de versiones Git[6] y gestionado a su vez a través del portal de alojamiento remoto de repositorios GitHub[7]. 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[8] el cual permite, mediante su sencilla e intuitiva interfaz, 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.

[6]https://git-scm.com/
[7]https://github.com/
[8]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 cómo, 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 empleados para el desarrollo del código, así como las instancias utilizadas para comprobar la calidad del algoritmo.

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(R) Core(TM) i5-5257U CPU 2.70 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 entorno de desarrollo integrado o IDE (Integrated Development Environment) PyCharm[2] de JetBrains en su versión 2019.3.1.

[1]https://www.python.org/
[2]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 conjuntos de grafos, tanto generados de manera aleatoria como obtenidos de diferentes fuentes de datos, como precios del mercado de valores o turbinas de viento.

A continuación, se detallan los diferentes conjuntos en los que se dividen las instancias:

- **Conjuntos de tipo A y B:** Estos conjuntos son instancias de grafos generadas mediante una distribución de probabilidad uniforme, variando su número de nodos entre 100 y 500, la densidad de estos grafos oscila entre el 45,78 % y el 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[3], 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 esta, 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. Estos resultados se han obtenido en dos fases, una primera fase preliminar, con

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

un conjunto reducido de las instancias y una segunda fase, con todas las instancias del problema para su posterior comparación con los resultados de estudios previos.

5.2.1. Experimentos preliminares

En esta fase ha sido 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 α en varias iteraciones, estas se han elegido de todos los conjuntos para ampliar el rango de las pruebas. El número de nodos en estas instancias varia 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, descritos en la sección 4.2, y a su vez se han aplicado los valores de $\alpha \in \{0.25, 0.5, 0.75\}$ y adicionalmente un valor de α 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.

En las tablas 5.1 y 5.2 se muestran los datos resumidos que se han recopilado durante todo el procesado preliminar. Estos han sido comparados con el análisis previo mediante el algoritmo MS-VNS, donde:

- f: Indica el valor medio de la función objetivo.
- t(sec): Indica el tiempo medio empleado en el procesado de las instancias.
- Desv.(%): Indica el porcentaje de desviación de la función objetivo respecto al análisis previo.
- # Mejor: Indica el número de veces que ese constructivo ha sido mejor que el análisis previo.

Por un lado, la tabla 5.1 muestra los datos para el constructivo por adyacentes, se percibe que el valor medio de f se acerca mucho al que se obtuvo en el análisis previo del problema, esto indica que el algoritmo cumple cierta calidad en la obtención de una solución factible.

	f	t (sec)	Desv.(%)	# Mejor
MS-VNS	8789,20	31,42	0,21	21
GRASP	8074,21	4,31	48,16	1
GRASP + BL	8202,39	5,42	45,19	1

TABLA 5.1: Tabla resumen constructivo de adyacentes.

Por otro lado, en la tabla 5.2, se exponen los resultados para el constructivo por ratio, que en este caso consigue alcanzar mejor promedio para f. Cabe añadir que incluir la mejora aumenta en ambos casos el tiempo de cómputo. Sin embargo, también aumenta significativamente el valor de la función objetivo, por lo que se añadirá a la experimentación final.

	f	t (sec)	Desv (%)	# Mejor
MS-VNS	8789,20	31,42	0,00	22
GRASP	8139,99	5,00	47,79	0
GRASP + BL	8202,43	5,89	45,35	0

TABLA 5.2: Tabla resumen constructivo de ratio.

Aunque que el constructivo mediante el ratio no alcance un valor que supere la función objetivo, se han obtenido de media mejores resultados en relación ratio-cardinalidad, con una desviación menor respecto al algoritmo MS-VNS. Por lo tanto, el constructivo basado en el ratio permite obtener valores para la función objetivo mayores y cliques con mayor número de nodo. A esto hay que añadir que el valor de α con el que se han obtenido mejores resultados ha sido 0.75, llegando a 16 soluciones mejores de las 22 instancias procesadas en este caso respecto del constructivo por adyacentes.

Todos los datos que se han recopilado durante estos experimentos preliminares se pueden encontrar en los anexos 7.1 y 7.2. En estas tablas se encuentran los mejores resultados tras todas las iteraciones de ambos constructivos y la posterior mejora del algoritmo con la búsqueda local.

5.2.2. Experimento final

En esta sección se han recopilado todos los datos obtenidos en las pruebas preliminares para el procesado final y se ha configurado de la siguiente manera:

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

Los datos recopilados durante el experimento final han sido comparados con los obtenidos previamente en el trabajo realizado por Dominik Goeke, Mahdi Moeini y David Poganiuch [13], los cuales se pueden observar en la tabla 2 del mismo.

En la tabla 5.3 se muestran los resultados resumidos tras el procesado de todas las instancias, se puede apreciar que el valor promedio de f es muy cercano al valor promedio que se obtuvo en el análisis de referencia.

	f	**t(sec)**	**Desv.(%)**	**# Mejor**
MS-VNS	1700,88	23,46	5,28	102
GRASP + BL	1671,81	6,09	37,73	7

TABLA 5.3: Tabla comparativa corregida de la experimentación final.

Es importante indicar que en esta tabla no se han añadido los resultados que en el estudio previo de Dominik Goeke, Mahdi Moeini y David Poganiuch no incluyeron o no procesaron. Si se añaden estos valores, se obtiene la tabla 5.4, donde se puede observar que el valor medio de f es mayor que en el estudio previo. También cabe añadir que este algoritmo ha obtenido los resultados en un tiempo medio muy bajo respecto del estudio previo. Además, aplicar este algoritmo y posterior mejora a supuesto la menor desviación, respecto del algoritmo MS-VNS, de todas las experimentaciones.

	f	**t(sec)**	**Desv.(%)**	**# Mejor**
MS-VNS	1700,88	23,46	5,28	102
GRASP + BL	2384,61	6,84	35,75	13

TABLA 5.4: Tabla comparativa de la experimentación final.

En la tabla 7.3 que se anexa se pueden encontrar los datos tras el procesado de todas las instancias durante el experimento final.

Capítulo 6

Conclusiones

En este capítulo se describen las conclusiones 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 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.

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 (MWCP)

y el clásico problema del clique máximo (MCP). 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 ampliar conocimientos sobre el desarrollo de software y la gestión de un proyecto. También se ha profundizado en conceptos sobre algoritmia y estructuras de datos para obtener soluciones mejores y más eficientes al planteamiento de problemas de optimización. Destacando:

- 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, enfocado 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 describen algunas ideas como:

En primer lugar, una opción que ofrece un considerable aumento en el rendimiento del código es la utilización de Cython[1] mediante pequeñas modificaciones en el código para ajustarlo a su lenguaje propio. Este compilador optimizado permite aumentar el rendimiento de las funciones escritas en Python. Algunas de sus características son la

[1] https://cython.org/

unificación de la legibilidad del código escrito en Python con el rendimiento de C, y la interacción eficiente con grandes conjuntos de datos mediante NumPy[2].

Otra opción podría ser la implementación del procesado paralelo aprovechando el módulo *concurrent.futures* que ofrece Python, este hace uso del módulo multiprocessing, por lo tanto no se ve afectado por el GIL[3] a diferencia del módulo multithreading, y permite crear un conjunto de procesos, mediante la función ProcessPoolExecutor, para ejecutar llamadas asíncronas.

Esta opción se puede añadir para el cálculo de los valores a partir de la lista de todos los nodos candidatos, como se describió en la sección 3.1 en el algoritmo 1, con el fin de obtener los resultados de manera paralela, reduciendo considerablemente el tiempo de cómputo, ya que para instancias con una gran cantidad de nodos y una alta densidad este tiempo puede ser excesivo.

[2]https://numpy.org/
[3]https://docs.python.org/3/glossary.html?highlight=gil#term-global-interpreter-lock

Capítulo 7

Anexos

7.1. Resultados del experimento preliminar con el constructivo por adyacentes

En la tabla 7.1 se muestran los resultados tras analizar las instancias seleccionadas para la experimentación preliminar mediante el constructivo por adyacentes, separada por el uso del algoritmo GRASP y la posterior adición de la mejora, 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.
- desv (%): Indica la variación respecto al valor de la función objetivo mediante el algoritmo MS-VNS.
- # Mejor: Indica si la solución obtenida mediante el algoritmo es mejor o no que la del trabajo anterior.

TABLA 7.1: Resultados de los experimentos preliminares con constructivo adyacentes.

Instancias	GRASP					GRASP y búsqueda local				
	f	c	t (sec)	desv(%)	#Mejor	f	c	t (sec)	desv(%)	#Mejor
random-1	1.242	15	0.062	0.621	0	1.242	15	0.143	0.621	0
random-4	1.017	26	1.673	0.789	0	1.465	27	5.761	0.697	0
random-5	0.913	31	3.651	0.749	0	1.459	32	3.407	0.599	0
random-6	0.981	19	0.099	0.147	0	0.967	18	0.259	0.159	0
random-8	0.855	22	0.714	0.281	0	0.923	23	1.969	0.224	0

Continúa en la siguiente página

Tabla 7.1 Continuación de la página anterior

Instancias	GRASP					GRASP y búsqueda local				
	f	c	t (sec)	desv(%)	#Mejor	f	c	t (sec)	desv(%)	#Mejor
random-10	0.899	31	3.323	0.344	0	1.069	30	2.584	0.220	0
market-1	1.543	8	0.030	0.927	0	2.143	8	0.255	0.899	0
market-2	1.231	8	0.051	0.948	0	1.326	8	0.306	0.944	0
market-5	1.472	8	0.023	0.896	0	2.812	8	0.388	0.802	0
wind-2005	90 391.750	4	0.003	0.045	0	90 391.750	4	0.016	0.045	0
wind-2006	87 219.512	5	0.005	0.114	0	90 034.200	5	0.052	0.086	0
brock200-2	0.545	12	0.062	0.008	0	0.545	12	0.824	0.008	0
football	0.526	10	0.004	0.123	0	0.526	10	0.165	0.123	0
johnson16-2-4	0.556	10	0.064	0	1	0.556	10	0.411	0	1
karate	0.545	6	0.001	0.091	0	0.545	6	0.010	0.091	0
keller4	0.542	13	0.081	0.015	0	0.542	13	0.506	0.015	0
c-fat500-10	1.008	126	84.088	0.002	0	1.008	126	98.450	0.002	0
email	1.453	12	0.004	0.975	0	1.453	12	0.004	0.975	0
jazz	0.889	30	0.216	0.878	0	0.889	30	0.063	0.878	0
p-hat1000-1	0.806	9	0.147	0.977	0	2.075	10	0.654	0.941	0
polbooks	2.975	6	0.001	0.757	0	2.975	6	0.004	0.757	0
sanr400-0-7	1.335	18	0.586	0.906	0	2.047	18	3.103	0.856	0

Concluido

7.2. Resultados del experimento preliminar con el constructivo por ratio

En la tabla 7.2 se muestran los resultados tras analizar las instancias seleccionadas para la experimentación preliminar mediante el constructivo de ratio, separadas por el uso del algoritmo GRASP y su posterior mejora, 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.
- desv (%): Indica la variación respecto al valor de la función objetivo mediante el algoritmo MS-VNS.
- # Mejor: Indica si la solución obtenida mediante el algoritmo es mejor o no que la del trabajo anterior.

TABLA 7.2: Resultados de los experimentos preliminares con constructivo ratio.

Instancias	GRASP					GRASP y búsqueda local				
	f	c	t (sec)	desv(%)	#Mejor	f	c	t (sec)	desv(%)	#Mejor
random-1	1.242	15	0.046	0.621	0	1.242	15	0.117	0.621	0
random-4	0.982	28	1.356	0.797	0	1.186	28	2.568	0.754	0
random-5	0.809	31	1.353	0.778	0	1.459	32	2.924	0.599	0
random-6	0.951	18	0.079	0.173	0	0.967	18	0.257	0.159	0
random-8	0.907	22	0.211	0.238	0	0.923	23	0.831	0.224	0
random-10	0.901	30	1.390	0.342	0	1.069	30	2.539	0.220	0
market-1	1.898	8	0.026	0.911	0	2.143	8	0.273	0.899	0
market-2	0.879	8	0.047	0.963	0	1.326	8	0.323	0.944	0
market-5	1.250	8	0.058	0.912	0	3.862	7	0.363	0.728	0
wind-2005	90 391.750	4	0.003	0.045	0	90 391.750	4	0.015	0.045	0
wind-2006	88 663.400	5	0.004	0.100	0	90 034.200	5	0.022	0.086	0
brock200-2	0.524	11	0.054	0.048	0	0.524	11	0.462	0.048	0
football	0.526	10	0.004	0.123	0	0.526	10	0.172	0.123	0
johnson16-2-4	0.529	9	0.073	0.001	0	0.529	9	0.282	0.001	0
karate	0.545	6	0.001	0.091	0	0.545	6	0.010	0.091	0
keller4	0.522	12	0.076	0.051	0	0.522	12	0.313	0.051	0
c-fat500-10	1.008	126	99.983	0.002	0	1.008	126	113.830	0.002	0
email	1.453	12	0.008	0.975	0	1.453	12	0.002	0.975	0
jazz	0.889	30	0.170	0.878	0	0.889	30	0.024	0.878	0
p-hat1000-1	2.981	9	0.158	0.915	0	2.075	10	0.783	0.941	0
polbooks	2.975	6	0.001	0.757	0	2.975	6	0.003	0.757	0
sanr400-0-7	2.919	18	0.796	0.794	0	2.395	19	3.559	0.831	0

Concluido

7.3. Resultados del experimento final

En la tabla 7.3 se muestran los resultados tras analizar todas las instancias durante la experimentación final, 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.
- desv(#): 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 7.3: Resultados del experimento final.

Instancias	f	c	t (sec)	desv(%)	#Mejor
random 1	1.242	15	0.132	0.621	0
random 2	1.082	20	0.372	0.769	0
random 3	0.947	21	0.440	0.775	0
random-4	1.523	27	2.453	0.685	0
random-5	1.459	32	3.445	0.599	0
random-6	0.967	18	0.274	0.159	0
random-7	1.047	19	0.331	0.127	0
random-8	0.950	23	0.703	0.201	0
random-9	0.967	28	1.432	0.267	0
random-10	0.991	31	3.661	0.277	0
market-1	2.143	8	0.281	0.899	0
market-2	1.326	8	0.324	0.944	0
market-3	2.017	8	0.350	0.855	0
market-4	2.368	8	0.352	0.874	0
market-5	2.812	8	0.314	0.802	0
wind-2004	91 286.000	4	0.026	0	1
wind-2005	90 391.750	4	0.017	0.045	0
wind-2006	90 034.200	5	0.024	0.086	0
Set E					
brock200-1	0.514	19	1.731	0.031	0
brock200-2	0.526	10	0.460	0.043	0
brock200-3	0.519	14	0.735	0.040	0
brock200-4	0.517	15	1.030	0.024	0
c-fat200-1	0.520	13	0.516	0	1
c-fat200-2	0.510	25	0.873	0	1
c-fat200-5	0.504	59	8.409	0	1
C-fat500-1	0.517	15	5.527	0.005	0
c-fat500-2	0.509	27	4.514	0.001	0
c-fat500-5	0.504	65	19.157	0	1
c-fat500-10	0.502	127	146.890	0	1
hamming6-2	0.508	33	1.188	0.024	0
hamming6-4	0.556	5	0.028	0.074	0
hamming8-4	0.515	17	0.980	0.080	0
johnson8-2-4	0.556	5	0.008	0.008	0
johnson8-4-4	0.517	15	0.138	0.024	0
johnson16-2-4	0.529	9	0.281	0.001	0
johnson32-2-4	0.515	17	15.608	0.009	0

Continúa en la siguiente página

Tabla 7.3 Continuación de la página anterior

Instancias	f	c	t (sec)	desv(%)	#Mejor
keller4	0.522	12	0.287	0.051	0
p-hat300-1	0.533	8	0.690	0.064	0
p-hat300-2	0.511	23	1.312	0.053	0
p-hat300-3	0.508	31	5.527	0.023	0
p-hat500-1	0.529	9	2.228	0.071	0
p-hat700-1	0.526	10	3.370	0.077	0
p-hat1000-1	0.526	10	7.568	0.077	0
san200-0-7-1	0.511	23	1.534	0.036	0
san200-0-7-2	0.516	16	1.445	0.044	0
san200-0-9-1	0.504	64	16.631	0.012	0
san200-0-9-2	0.506	44	11.385	0.008	0
san200-0-9-3	0.507	34	9.206	0.005	0
san400-0-5-1	0.526	10	2.616	0.060	0
san400-0-7-1	0.511	23	7.395	0	1
san400-0-7-2	0.514	19	6.310	0.031	0
san400-0-7-3	0.515	17	5.141	0	1
san400-0-9-1	0.504	59	49.706	0	1
san1000	0.524	11	23.804	0.048	0
sanr200-0-7	0.514	18	1.377	0.030	0
sanr200-0-9	0.507	36	11.158	0.006	0
sanr400-0-5	0.520	13	1.764	0.055	0
sanr400-0-7	0.514	19	4.844	0.031	0
karate	0.545	6	0.010	0.091	0
dolphins	0.545	6	0.029	0.091	0
polbooks	0.538	7	0.173	0.103	0
adjnoun	0.545	6	0.137	0.091	0
football	0.526	10	0.181	0.123	0
jazz	0.508	31	0.583	0.153	0
celegans-metabolic	0.526	10	9.544	0.123	0
email	0.520	13	94.598	0.133	0
Set F					
brock200-1	1.502	18	1.117	0.782	0
brock200-2	1.683	11	0.233	0.890	0
brock200-3	3.323	12	0.379	0.761	0
brock200-4	2.109	14	0.625	0.756	0
c-fat200-1	1.138	12	0.021	0.067	0
c-fat200-2	1.000	24	0.215	0.130	0
c-fat200-5	1.020	58	6.140	0	1

Continúa en la siguiente página

Tabla 7.3 Continuación de la página anterior

Instancias	f	c	t (sec)	desv(%)	#Mejor
c-fat500-1	1.075	14	0.037	0.147	0
c-fat500-2	1.075	26	0.410	0	1
c-fat500-5	1.008	64	9.261	0.031	0
c-fat500-10	1.008	126	116.288	0.002	0
hamming6-2	1.000	32	0.666	0.394	0
hamming6-4	1.000	4	0.004	0.660	0
hamming8-4	1.000	16	0.456	0.497	0
johnson8-2-4	1.071	4	0.003	0.068	0
johnson8-4-4	1.000	14	0.099	0.291	0
johnson16-2-4	1.021	8	0.172	0.190	0
johnson32-2-4	1.127	16	9.243	0	1
keller4	1.513	11	0.157	0.636	0
p-hat300-1	6.291	7	0.059	0.756	0
p-hat300-2	1.005	21	1.094	0.866	0
p-hat300-3	1.602	32	4.747	0.778	0
p-hat500-1	4.387	8	0.231	0.777	0
p-hat700-1	4.536	8	0.442	0.785	0
p-hat1000-1	2.075	10	0.827	0.941	0
san200-0-7-1	1.140	25	0.996	0.896	0
san200-0-7-2	2.752	14	1.235	0.848	0
san200-0-9-1	1.193	55	11.854	0.758	0
san200-0-9-2	1.328	43	10.215	0.734	0
san200-0-9-3	1.115	35	9.704	0.750	0
san400-0-5-1	1.703	9	1.258	0.970	0
san400-0-7-1	3.866	22	5.207	0.691	0
san400-0-7-2	3.327	18	4.608	0.794	0
san400-0-7-3	3.413	16	3.157	0.810	0
san400-0-9-1	1.757	55	50.877	0	1
san1000	2.912	10	9.435	0.965	0
sanr200-0-7	1.358	17	0.765	0.865	0
sanr200-0-9	1.445	36	10.060	0.703	0
sanr400-0-5	1.006	12	0.704	0.934	0
sanr400-0-7	4.066	19	3.494	0.713	0
karate	0.159	5	0.001	0.990	0
dolphins	1.203	5	0.001	0.896	0
polbooks	2.975	6	0.004	0.757	0
adjnoun	0.464	5	0.020	0.976	0
football	1.966	9	0.002	0.794	0

Continúa en la siguiente página

Tabla 7.3 Continuación de la página anterior

Instancias	f	**c**	**t (sec)**	**desv(%)**	**#Mejor**
jazz	0.889	30	0.051	0.878	0
celegans-metabolic	1.210	9	0.026	0.809	0
email	1.453	12	0.002	0.975	0

Concluido

Bibliografía

[1] URL: https://dictionary.cambridge.org/es/diccionario/ingles-espanol/clique.

[2] Micael Gallego Carrillo Abraham Duarte Muñoz Juan José Pantrigo Fernández. *Metaheurísticas*. Ed. por Servicio de Publicaciones Universidad Rey Juan Carlos. 2010.

[3] 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.

[4] 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.

[5] 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.

[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.