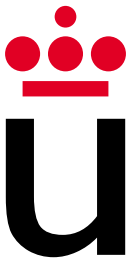


Universidad
Rey Juan Carlos

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

Resumen

TODO

Agradecimientos

Todo este trabajo no habría sido posible sin mi familia, sin su apoyo y cariño, en especial a Rebeca Muñoz, 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 cuadros

Lista de Abreviaciones

GRASP	**G**reedy **R**andomized **A**daptive **S**earch **P**rocedure
IDE	**I**ntegrated **D**evelopment **E**nvironment

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. 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.

1.2.1. Optimización combinatoria

aaa

1.2.2. Heurística y Metaheurística

Se define heurística, según el Diccionario de la Real Academia de la Lengua Española [3], 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" [12], más tarde traducido a "Cómo plantear y resolver problemas" [11].

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 [9]. Este término lo acuño Fred Glover en sus trabajos sobre búsqueda tabú en 1986 [5].

Existen un gran variedad de algoritmo metaheurísticos que se pueden clasificar de diferentes maneras según el enfoque, en la figura 1.1 se muestra una de ellas basada en el número de soluciones.

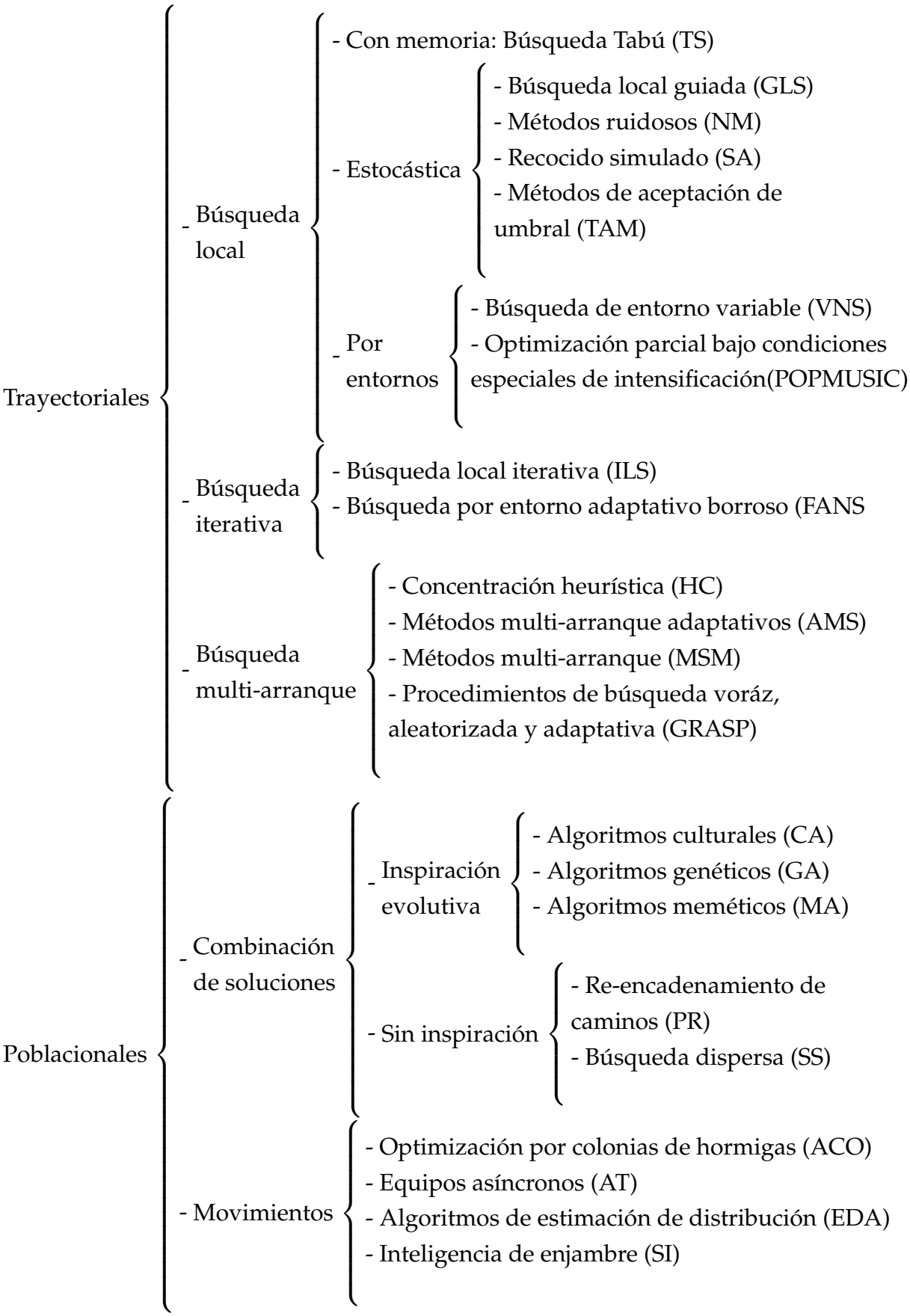


FIGURA 1.1: Clasificación de metaheurísticas

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].

1.2.3. 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 [8].

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 [19], un clique se define como:

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

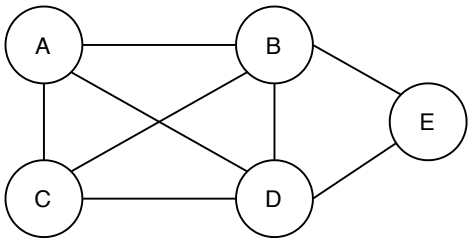
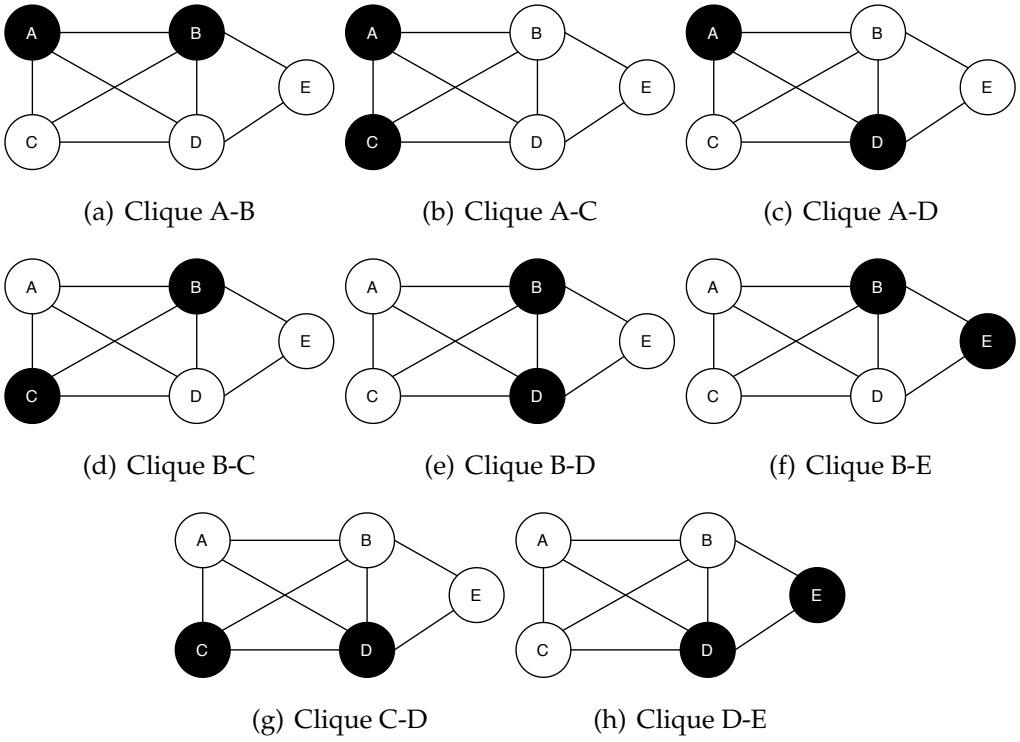


FIGURA 1.2: Diagrama de un grafo.

En la figura 1.2 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 1.3 y 1.4 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.

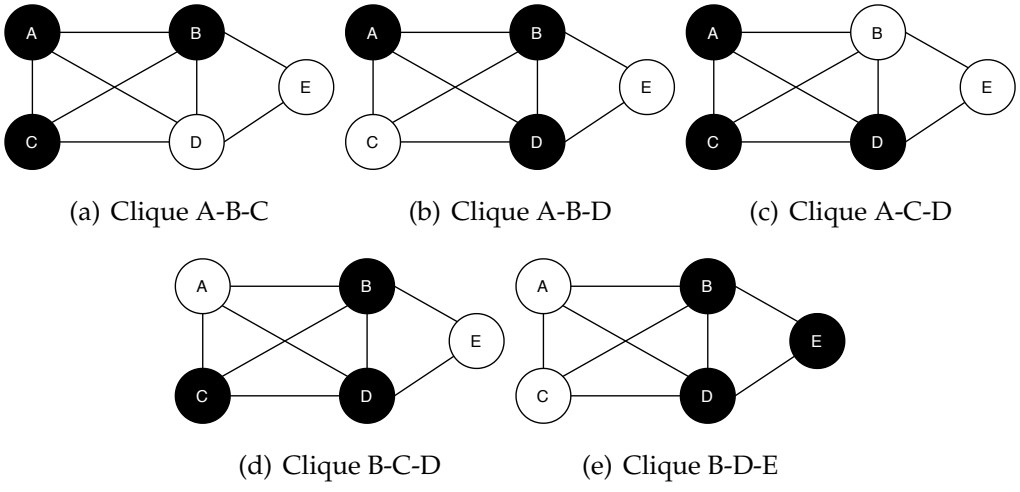


(a) Clique A-B (b) Clique A-C (c) Clique A-D

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

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

FIGURA 1.3: 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 1.4: 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 1.7, a diferencia de un clique que se podría denominar simple [20][21]. Y en este caso, se obtiene que:

$$\omega(G) = 4]$$

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

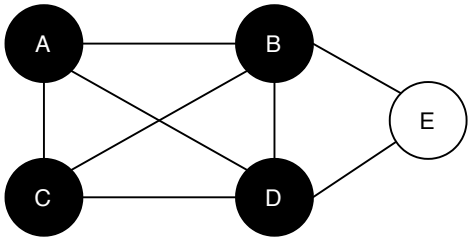


FIGURA 1.5: Clique máximo del grafo.

1.3. Definición y motivación del problema

1.3.1. Problema del clique de ratio máximo

Como se introdujo en la sección 1.2.3 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 como un problema NP-completo.

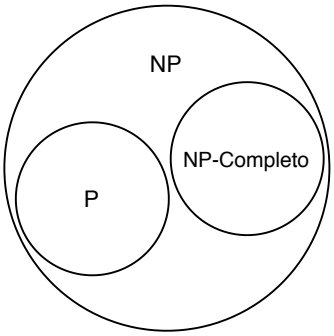


FIGURA 1.6: Clique máximo del grafo.

Los problemas denominados como NP, 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 del 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. 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.

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

$$sujeto\ a:$$

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

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

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

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

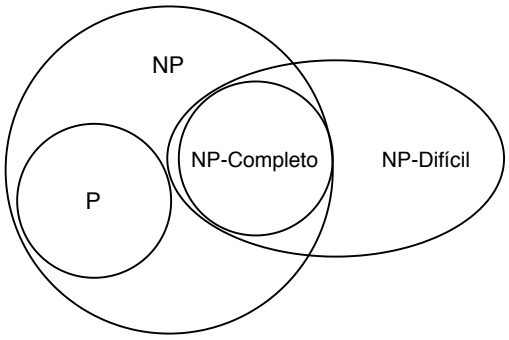


FIGURA 1.7: Clique máximo del grafo.

1.3.2. Motivación del problema

aaaaa

1.4. Estado del arte

El análisis sobre el estado del arte que se ha realizado sobre el problema del clique de ratio máximo

Moeini dca

[13]

[16]

Cabe destacar entre todos los trabajos realizados sobre el problema del clique de ratio máximo el de Dominik Goeke, Mahdi Moeini y David Poganiuch [6] el cuál se ha tomado como referencia para realizar este trabajo final de grado.

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.
- Implementación del algoritmo metaheurístico GRASP para la resolución del problema.

2.2. Objetivos secundarios

- 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 [4].

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 [17] [15] [10] [2] [18] [14] 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 para este problema. Dicho algoritmo 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 1.3.1, 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 una solución.

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

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.

Algoritmo 2: Contructivo voráz.

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

En primer lugar 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 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, 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

Definición de la fase de búsqueda.

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

aaa

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 como aplicacion standalone[2] mediante el comando:

```
python grasp_main.py
```

Este script, en adelante Grasp Main, tiene la configuración necesaria para ajustar el programa:

- Número de iteraciones a realizar por cada fichero.

[1] script
[2] standalone

- Ruta donde se encuentran los ficheros de definición de los grafos a procesar.
- Ruta de los ficheros de resultados generados por el programa.

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.
- 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 en Grasp Main, y por cada tipo de constructivo de los que dispone la aplicación, en este caso, constructivo ratio y constructivo adyacente, los cuales serán explicados más adelante. La implementación del algoritmo GRASP se encuentra en la clase SolutionGrasp y contiene los métodos necesarios para la obtención mediante el algoritmo 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.

//TODO ¿se debe explicar cada método que forma parte de esta funcion?

Esta función se apoya en otra auxiliar, nombrada como $find\_solution\_aux$, la cuál implementa el algoritmo utilizado, de esta manera se desacopla el tipo de algoritmo de la

búsqueda de una solución, dando la posibilidad a un cambio posterior si fuera necesario. Dicha función, procesará los nodos del grafo tal como se describió en el capítulo anterior. Para la fase constructiva del algoritmo, según el tipo elegido en la configuración inicial, se creará un objeto del contructivo específico, SolutionGreedyRatio o SolutionGreedyAdjacent. Estos heredan de la clase abstracta[3] 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 voráz buscan una solución factible en un tiempo muy reducido.

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.

Para probar las diferentes funciones del proyecto se han escrito casos de prueba mediante la librería de Python unittest[4], para conseguir un código tolerable a posibles fallos y mantenible.

//TODO Explicación del proceso, partiendo de la lectura de fichero, luego crear solucion voraz

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.

[3] 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.

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

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 la utilidad Trello[5], la cuál mediante tarjetas sobre el tablero del proyecto permite conocer las tareas del proyecto, así como conocer su estado, y añadir nuevas si así fuera necesario.

En cuanto al mantenimiento de versiones del proyecto se ha usado el sistema de control de versiones Git[6] a través del portal de alojamiento 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 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://trello.com/es
[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 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] Integrated Development Environment o 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:

- aaaa

Estos objetivos se han cumplido de manera satisfactoria:

6.2. Conocimientos adquiridos

Durante el proceso de realización de este trabajo fin de grado me he encontrado con diferentes retos los cuales me han permitido adquirir muchos conocimientos sobre el desarrollo de software y la gestión de un proyecto. También he 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 destaco:

- El aumento y mejorar mis 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.
- El aprendizaje sobre LaTeX, al utilizarlo para documentar el trabajo fin de grado.

6.3. Líneas de desarrollo futuras

ccc

Bibliografía

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

[2] 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.

[3] Real Academia Española. *Heurística*.

[4] 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.

[5] Fred Glover. «Future paths for integer programming and links to ar tifi cial intelligence». En: *Computers operations research* 13 (1986), págs. 533-549.

[6] Dominik Goeke, Mahdi Moeini y David Poganiuch. «A Variable Neighborhood Search heuristic for the maximum ratio clique problem». En: *Computers and 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.

[7] *Heurística*. Jul. de 2019. URL: https://conceptodefinicion.de/heuristica/.

[8] 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.

[9] *Metaheurística*. Oct. de 2019. URL: https://es.wikipedia.org/wiki/Metaheur%C3%ADstica.

[10] 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.

[11] George Pólya. *Cómo Plantear y Resolver Problemas*. Ed. por Editorial Trillas. 1965.

[12] George Pólya. *How to Solve It*. Ed. por Princeton. 1945.

[13] 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).

[14] Jorge Ramírez. «Metaheurística GRASP para el problema Vertex Bisection Minimization.» En: 12 (ene. de 2018), págs. 28-41.

[15] 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.

[16] 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.

[17] Wang Shaochang y col. «Application of Greedy Random Adaptive Search Algorithm (GRASP) in Flight Recovery Problem». En: ene. de 2018.

[18] Víctor Villavicencio. «GRASP para el problema de ruta de vehículos». En: (mar. de 2020).

[19] Eric W. Weisstein. *Clique*. From MathWorld–A Wolfram Web Resource. URL: https://mathworld.wolfram.com/Clique.html.

[20] Eric W. Weisstein. *Maximal Clique*. From MathWorld–A Wolfram Web Resource.

[21] Eric W. Weisstein. *Maximum Clique*. From MathWorld–A Wolfram Web Resource.