Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Diseño, implementación y estudio de un algoritmo heurístico para la optimización de la gestión del instrumento astronómico EMIR

Design, implementation and study of a heuristic algorithm for the optimization of the management for the astronomical instrument EMIR

Lorenzo Gabriel Pérez González

La Laguna, 11 de julio de 2024

D. **Jorge Riera Ledesma**, profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y Sistemas de la Universidad de La Laguna, como tutor.

C E R T I F I C A

Que la presente memoria titulada:

”*Diseño, implementación y estudio de un algoritmo heurístico para la optimización de la gestión del instrumento astronómico EMIR*”

ha sido realizada bajo su dirección por D. **Lorenzo Gabriel Pérez González**.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 11 de julio de 2024

Agradecimientos

Agradezco a mi tutor, Jorge, por su guía, apoyo y consejos a la hora de elaborar el proyecto. Su colaboración ha sido clave para conseguir un resultado del cual sentirnos satisfechos.

A mi familia, por su apoyo constante durante el largo proceso universitario, nada de esto hubiese sido posible sin su ayuda, paciencia y amor.

A mis compañeros y amigos, por suponer una importante fuente de ayuda, ideas y diversión durante todos los años de colegio, instituto y carrera.

A los profesores universitarios, por ayudarme a obtener y perfeccionar mis conocimientos con el objetivo de convertirme en un buen profesional.

Licencia

Resumen

Este Trabajo de Fin de Grado presenta un enfoque heurístico empleado para resolver un problema de optimización complejo que surge durante la gestión del uso de un instrumento de observación de un telescopio de 10.4m. Este instrumento, denominado *Espectrógrafo Multi-objeto Infra-Rojo*, dispone de un campo de visión dividido en 55 bandas adyacentes contiguas, en cada cual existe una asociación con dos barras deslizantes opuestas que pueden posicionarse para observar un objeto astronómico. Es necesario establecer una configuración sincronizada de las barras paralelas para poder crear rendijas en la posición del objeto y poder llevar a cabo su observación.

Los astrónomos que utilizan el dispositivo otorgan una lista de objetos a observar. Dada la alta demanda del dispositivo, los tiempos disponibles para realizar observaciones son limitados, así que surge el problema de seleccionar un conjunto de objetos prioritarios para ser observados. Este problema de optimización se conoce en la literatura como *Selective Routing Problem with Synchronization*.

Este trabajo propone una solución heurística a dicho problema, permitiendo llevar a cabo una selección de los objetos a observar y la correspondiente configuración de las barras paralelas, sujeta a sus respectivas limitaciones temporales y de sincronización. Dicha solución heurística consiste en un algoritmo constructivo junto a procedimientos de intensificación y diversificación. Estos elementos se han integrado en diferentes frameworks, cuyos comportamientos han sido estudiados experimentalmente.

Esta solución heurística ha sido probada utilizando una batería de instancias artificiales, dando lugar a un estudio computacional en el que se evalúa la calidad de las soluciones factibles obtenidas.

Palabras clave: Algoritmo heurístico, espectrógrafo, sincronización

Abstract

This Final Degree Project introduces a heuristic approach used to solve a complex optimization problem that arises during the management of the use of a 10.4m telescope's observation instrument. This instrument, called the Multi-Object Infrared Spectrograph*, has a field of view divided into 55 contiguous adjacent bands, each associated with two opposing sliding bars that can be positioned to observe an astronomical object. It is necessary to establish a synchronized configuration of the parallel bars to create slits at the object's position to conduct the observation.*

Astronomers using the device provide a list of objects to observe. Given the high demand for the device, the available times for observations are limited, leading to the problem of selecting a set of priority objects to be observed. This optimization problem is known in the literature as the Selective Routing Problem with Synchronization*.*

This work proposes a heuristic solution to this problem, allowing the selection of objects to be observed and the corresponding configuration of the parallel bars, subject to their respective temporal and synchronization constraints. The heuristic solution consists of a constructive algorithm along with intensification and diversification procedures. These elements have been integrated into different frameworks, whose behaviors have been studied experimentally.

This heuristic solution has been tested using a set of artificial instances, resulting in a computational study that evaluates the quality of the feasible solutions obtained.

Keywords: Heuristic algorithm, spectrograph, synchronization

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

En esta sección del trabajo, se describirá la naturaleza del problema y el contexto en el que surge. También, se abordará una descripción general de la problemática a resolver y la justificación del proyecto. De la misma forma, se plantearán algunos objetivos e hitos que se han planeado desde el inicio del proyecto.

1.1. Instituto Astrofísico de Canarias

El Instituto de Astrofísica de Canarias (IAC) [1] es un centro de investigación localizado en las Islas Canarias. Es un consorcio público que cuenta con la colaboración de la Administración General del Estado Español, la Administración Pública de la Comunidad Autónoma de Canarias, la Universidad de La Laguna (ULL) y el Consejo Superior de Investigaciones Científicas (CSIC), además de diversas instituciones internacionales. Su misión radica en realizar y promover cualquier tipo de investigación astrofísica, así como en desarrollar su tecnología y difundir sus conocimientos científicos a nivel mundial. El IAC está formado por las siguientes componentes:

- Instituto de Astrofísica: Sede central del IAC, ubicada en el municipio de San Cristóbal de La Laguna. A su cargo están dos de los observatorios más importantes del mundo, el Observatorio del Roque de los Muchachos y el Observatorio del Teide.
- Observatorio del Roque de los Muchachos: Ubicado en la Caldera de Taburiente, en la isla de La Palma. Alberga el mayor telescopio óptico del mundo, el Gran Telescopio Canarias (GRANTECAN), y la mayor concentración de telescopios del hemisferio norte.
- Observatorio del Teide: Se encuentra en la zona de Izaña, en la isla de Tenerife. Su principal función es el estudio del Sol, albergando uno de los telescopios más sofisticados de Europa, el GREGOR, de 1.5 metros de diámetro.

1.1.1. GRANTECAN

El Gran Telescopio de Canarias [2] (GTC o GRANTECAN) es el mayor telescopio óptico del mundo, ubicado en el Observatorio del Roque de los Muchachos, en la isla de La Palma. Su principal función es la de estudiar en profundidad la naturaleza de los agujeros

[1] IAC: Instituto de Astrofísica de Canarias
[2] GRANTECAN: Gran Telescopio de Canarias

negros, la formación de las estrellas y las galaxias en el universo temprano, la física de los planetas distantes y la naturaleza de la materia y energía oscuras en el universo. Puede verse una imagen del mismo en la figura 1.1.

Figura 1.1: Gran Telescopio de Canarias en el Roque de los Muchachos.[3]

El GRANTECAN está formado por un conjunto de instrumentos astronómicos, desde espectrógrafos infrarrojos a generadores de imágenes de alta resolución, que permiten llevar a cabo las observaciones de la forma más precisa. Entre dichos instrumentos, se encuentra EMIR, el dispositivo para el cual va a estar destinado este trabajo.

1.1.2. EMIR

El dispositivo EMIR [4] (Espectrógrafo Multiobjeto Infra-Rojo) es capaz de observar objetos desde una visión espectroscópica en un campo de 4 x 6.64 minutos de arco al cuadrado. Véase el instrumento EMIR en la figura 1.2.

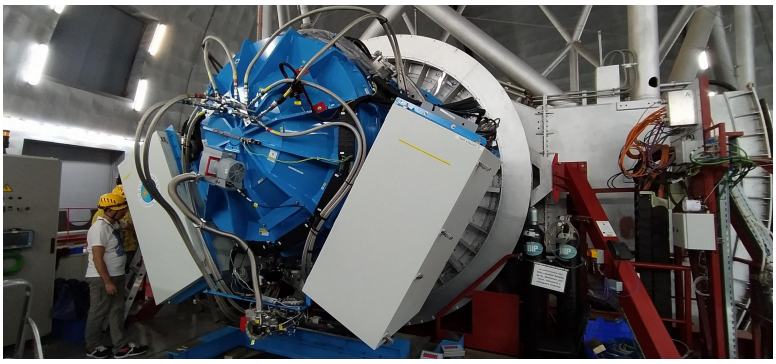

Figura 1.2: Espectrógrafo Multiobjeto Infra-Rojo.[5]

Uno de los aspectos más novedosos del dispositivo es el uso de una unidad robótica criogénica que permite la configuración remota de patrones de rejilla múltiple en su campo de visión. Esta unidad, llamada Cold Slit Unit (CSU) tiene la estructura de 55 pares de barras opuestas que tienen la capacidad de ser configuradas de forma individual y sincronizada para generar rejillas, con las que se podrá observar el cuerpo celeste que quede encerrado en ellas. La unidad puede verse en la figura 1.3.

[3]Imagen: GRANTECAN
[4]EMIR: Espectrógrafo Multiobjeto Infra-Rojo
[5]Imagen: EMIR

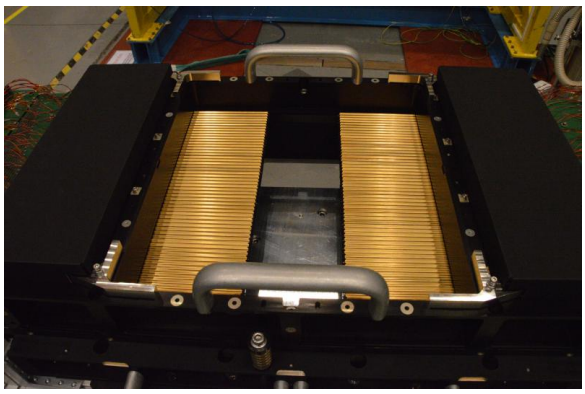

Figura 1.3: Unidad Robótica Criogénica.

1.2. Descripción del problema

Cada semestre, astrónomos de todo el mundo realizan peticiones al GRANTECAN para llevar a cabo observaciones con sus diferentes instrumentos. En el caso de las observaciones mediante el uso de EMIR, se deben de resolver dos problemas principales:

- Seleccionar alrededor de tres campos de visión que cubrirá la observación: Cada campo de observación vendrá definido por la orientación del telescopio y la CSU.
- Selección de un subconjunto de objetos a observar: Se realizará una lista de prioridad con los cuerpos celestes a observar y se seleccionará un conjunto de los mismos para adaptarse a la limitación temporal.

Algunos objetos en el campo de visión podrían requerir la rendija creada por las barras de varias bandas contiguas. Las bandas asignadas a cada objeto y el ancho de la rendija dependen de las características del objeto y de su posición en el campo de visión.

Cada rendija, tiene una altura de *h* segundos de arco, fijados por la altura de las barras. Asimismo, la anchura *w* de las rendijas, puede configurarse por el propio astrónomo, dependiendo de las condiciones de observación o la resolución espectral determinada.

Cada barra paralela tiene una zona muerta superior y otra inferior que no permite realizar una observación de manera efectiva en dichas áreas. A la zona muerta superior de una barra, la llamaremos d_1 y a la zona muerta inferior, la denominaremos d_2. Se muestra en la figura 1.4 un diagrama de una barra con sus áreas muertas.

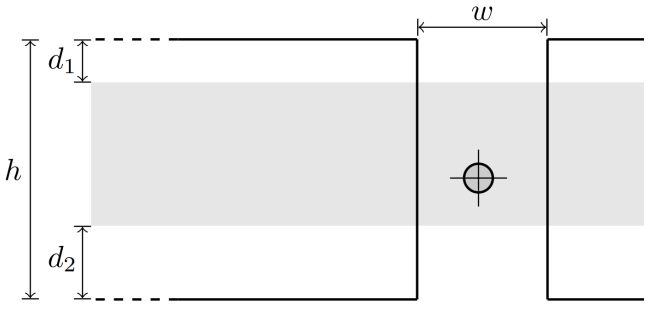

Figura 1.4: Barra deslizante de la CSU.

Un objeto situado en el área central de una banda puede ser observado mediante la única rendija formada por las dos barras de dicha banda. Cuando un objeto se encuentra en la zona muerta superior o inferior de una banda, por razones técnicas, la observación de este objeto requiere la rendija creada por las barras de (al menos) dos bandas contiguas. La figura 1.5 muestra la configuración de dos barras para observar un objetivo.

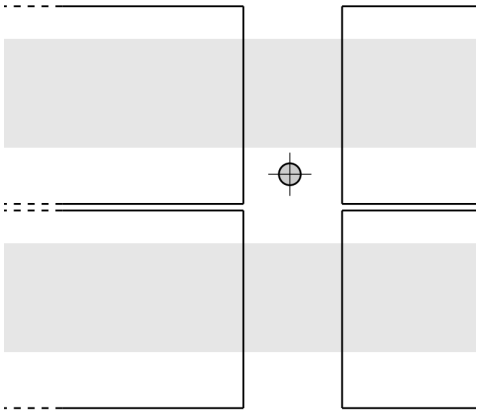

Figura 1.5: Objeto en zona muerta.

Dado que las barras pueden configurarse de manera independiente, EMIR puede realizar muchas observaciones simultáneas (hasta 55), pero la sincronización de las barras, necesaria para cada observación, es fundamental para una gestión eficiente del telescopio. Además, mover una barra consume bastante tiempo, por lo que los tiempos de reconfiguración en el CSU son largos. El uso eficiente del tiempo del telescopio impone una planificación cuidadosa para maximizar el rendimiento científico en el tiempo de observación disponible y limitado.

Dado un área en el cielo y una restricción temporal, este Trabajo Fin de Grado (TFG) aborda el problema de seleccionar objetos de una lista y configurar las barras del CSU. La lista de potenciales objetos a ser observados, junto con sus prioridades, también son entradas al problema. El objetivo es procesar tantas observaciones priorizadas como sea posible.

Nos referimos a este problema de optimización como Orienteering Problem with Synchronization (OPS) [19], dado que tiene características del Orienteering Problem (véase, [21]) con sincronización entre las barras paralelas de la unidad.

1.3. Justificación del proyecto

Las observaciones en el GRANTECAN disponen de un tiempo reducido, gran coste y mucha demanda. Esto supone la necesidad de elaborar un método para aprovechar al máximo las valiosas observaciones.

Una de las mayores dificultades del problema radica en la sincronización entre los diferentes procesadores, que supone la reconfiguración de la posición de los mismos. Esto consume una gran cantidad de recursos y por ello será importante llevar a cabo la planificación de la secuencia de pasos para realizar la observación de la forma más organizada y rápida posible.

Como se ha podido señalar, el problema que se está tratando es en absoluto sencillo de resolver, puesto que se trata de un problema de complejidad exponencial, que hace imposible su resolución exacta en tiempo polinomial. De esta forma, se propone llevar a cabo un algoritmo heurístico, que haga posible la obtención de una solución de calidad empleando menos recursos temporales que el algoritmo exacto.

1.4. Objetivos

El objetivo general de este proyecto es llevar a cabo el diseño e implementación de un algoritmo heurístico para la optimización de la gestión de uso del instrumento astronómico EMIR en el lenguaje de programación C++. De la misma forma, interesa realizar un estudio que permita comparar los resultados con los que pueda obtener un algoritmo exacto, tanto en tiempo como en la calidad de su solución.

1.4.1. Objetivos específicos

- Análisis del estado del arte en la materia de los algoritmos heurísticos y los problemas de optimización y sincronización.
- Realizar un generador de instancias aleatorias que permita llevar a cabo pruebas que complementen el estudio comparativo con ambos algoritmos.
- Llevar a cabo mejoras en el algoritmo heurístico una vez éste haya sido resuelto e, idealmente, estudiar si las mejoras son sustanciales.
- Desarrollar algoritmos aproximados (voraz y estocástico) y diferentes configuraciones del algoritmo heurístico que permitan comparar los recursos necesarios para su ejecución y los resultados.

1.5. Estructura del documento

En el segundo capítulo, se expone un marco teórico para comprender el resto del trabajo. Además, se realiza una revisión de la literatura relacionada con el problema a resolver y se dan a conocer las diferentes técnicas algorítmicas para su resolución.

Posteriormente, en la tercera sección, se mostrará el desarrollo de todos los componentes del trabajo. Se empezará mostrando el generador de instancias y, posteriormente, se describirán todos los algoritmos realizados.

El cuarto capítulo permitirá exponer los resultados obtenidos mediante un pequeño estudio comparativo entre los algoritmos desarrollados, donde se podrán observar las diferencias temporales y sus resultados.

La quinta sección estará compuesta por una pequeña conclusión acerca del trabajo realizado y una exposición de las líneas futuras o los posibles pasos a realizar para ampliar el proyecto.

El sexto capítulo dispondrá de los temas tratados en el anterior en inglés.

La última sección, el séptimo capítulo, mostrará un presupuesto, donde se desglosará el trabajo y su coste.

Finalmente, se muestran los apéndices del documento y la bibliografía, en la que figuran las referencias empleadas en la redacción de la presente memoria.

Capítulo 2

Estado del arte

Este apartado está destinado a la presentación de diferentes conceptos matemáticos de interés en el desarrollo de este proyecto, por las características del problema y el procedimiento llevado a cabo para darle solución. También se adentrará en una revisión de la literatura acerca de los problemas de orientación y sincronización, claves en la definición del problema a resolver.

2.1. Teoría de Grafos

Formalmente, un grafo viene definido por la siguiente estructura:

- $V(G)$: Conjunto no vacío de elementos llamados nodos o vértices del grafo.
- $E(G)$: Conjunto de pares ordenados de arcos o aristas del grafo.

$E(G) \subseteq V(G) \times V(G)$ define que las aristas que forman el grafo supondrán un subconjunto de todas las posibles conexiones entre los vértices del mismo, pero sin necesidad de igualarlo [17, p. 151]. Las aristas de un grafo se describen como: $E(G) = (A, B)$, donde el nodo A contiene una arista con dirección hacia el nodo B.

Entre las aristas de un grafo, existe el concepto de direccionalidad. De esta forma, un grafo dirigido es aquel cuyas aristas presentan un nodo inicial y un nodo final en sus extremos, permitiendo la comunicación unidireccional entre ambos. En la figura 2.1 puede verse un ejemplo de grafo dirigido.

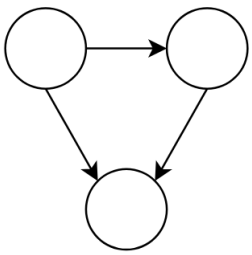

Figura 2.1: Grafo dirigido.

Por otra parte, un grafo no dirigido es aquel cuyas aristas tienen la condición de bidireccionalidad, permitiendo la comunicación en ambos sentidos para una arista. Como ejemplo, podemos ver que, si la colección de aristas de un grafo es: $E(G) = \{(A, B), (B, A)\}$, se

cumple la condición de que los nodos A y B están conectados en ambos sentidos. Por último, se muestra un ejemplo de grafo no dirigido en la figura 2.2.

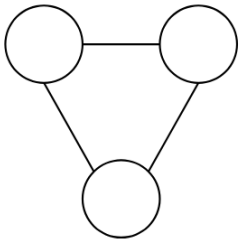

Figura 2.2: Grafo no dirigido.

Un grafo puede contener la condición de bucle, implicando que, desde un vértice v_1, puede formarse un camino por el resto de los nodos del grafo (o consigo mismo) que termine en el mismo nodo inicial v_1. Los bucles y su detección serán cruciales en secciones posteriores del proyecto. Puede observarse en la figura 2.3 un grafo dirigido con bucle.

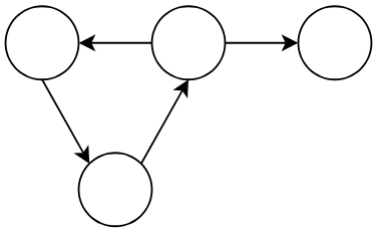

Figura 2.3: Grafo dirigido con bucle.

2.2. Complejidad Computacional

La complejidad computacional se define como la disciplina de las ciencias de la computación que se encarga del análisis y clasificación de los problemas informáticos en base a la necesidad de los recursos espaciales y temporales que precisan para su resolución. Entre los diferentes problemas, se podrá clasificar entre tratables o intratables, dependiendo de si los recursos temporales necesarios se encuadran en la categoría de resoluble en tiempo polinomial o exponencial, respectivamente [1]. Se entiende que un problema no está completamente resuelto a menos que se encuentre un algoritmo que sea capaz de resolverlo de forma óptima en tiempo polinomial.

Además, existen diferentes planteamientos a la hora de analizar la complejidad temporal de un algoritmo. La base de dichos enfoques reside en el análisis asintótico, que pretende describir el comportamiento de un algoritmo en sus límites. De esta forma, se aportan cotas superiores o inferiores que sean capaces de contener el comportamiento del algoritmo [20, p. 276–281].

- Notación O: La notación O busca definir una función matemática $g(N)$ que sea capaz de acotar superiormente a otra función matemática $f(N)$ si existen dos constantes positivas c y N_0 tales que: $f(N) \leq c \cdot g(N) \ \forall \ N \geq N_0$. De esta forma, se habrá encontrado una cota superior a $f(N)$ representada por la función matemática $g(N)$, definida como $f(N) = O(g(N))$. Resumidamente, se representa la función $g(N)$ como

aquella que crece más rápido que la función $f(N)$. Véase en la figura 2.4 un ejemplo de la notación O.

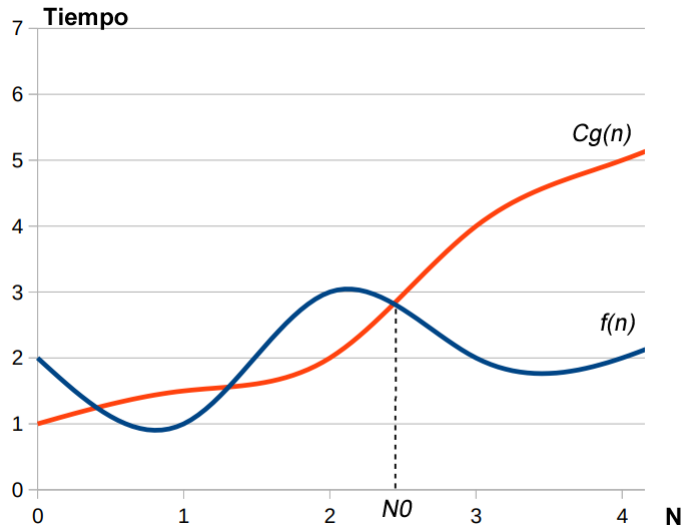

Figura 2.4: Notación O.

- Notación Ω: Por otra parte, la notación Ω busca encontrar una función matemática $g(N)$ que sea capaz de acotar en su límite inferior a otra función matemática $f(N)$. De esta forma, si existen dos constantes positivas c y N_0 tales que: $f(N) \geq c \cdot g(N) \; \forall \; N \geq N_0$, se habrá encontrado una cota inferior representada por la función $g(N)$ y se definirá dicha cota como: $f(N) = \Omega(g(N))$. En la figura 2.5, se representa una cota inferior a la función $f(N)$.

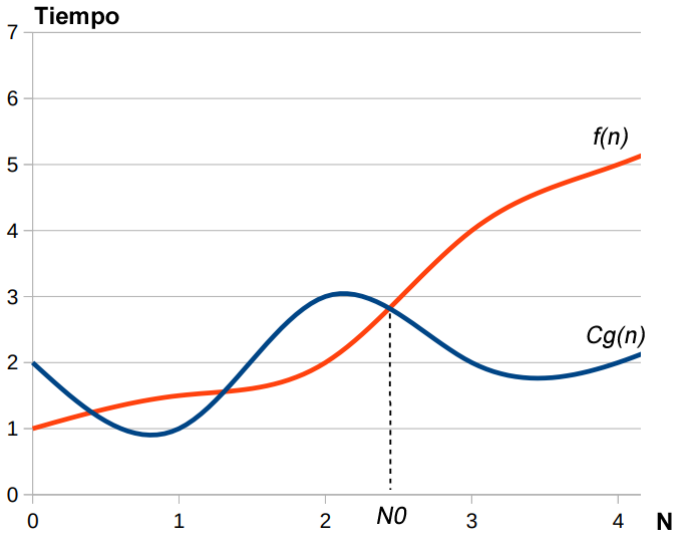

Figura 2.5: Notación Ω.

- Notación Θ: Finalmente, existe una notación que permite acotar inferior y superiormente una función matemática, la notación Θ. De esta forma, se buscan tres constantes positivas c_1, c_2 y N_0 que permitan que una función $g(N)$ acote inferior y superiormente a la función $f(N)$. Así, definimos la notación Θ como: $c_2 \cdot g(N) \leq f(N) \leq c_1 \cdot g(N) \; \forall \; N \geq N_0$. Véase un ejemplo de la notación Θ en la figura 2.6.

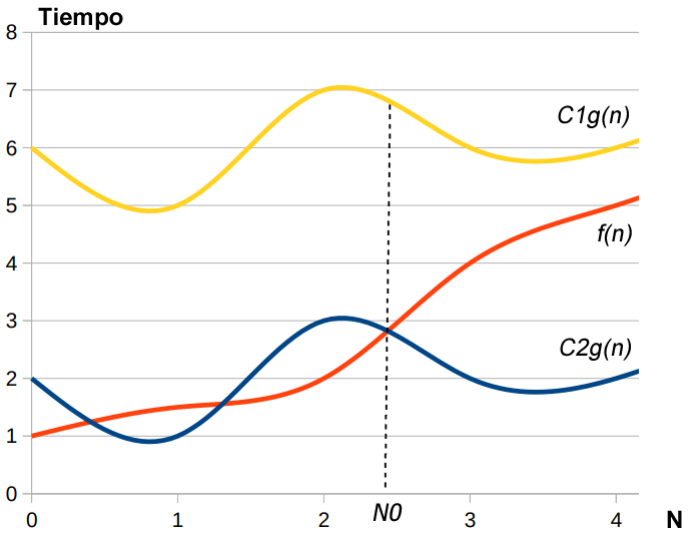

Figura 2.6: Notación Θ.

2.2.1. Clasificación de problemas

Cuando se realiza un análisis de los problemas según su complejidad, el modelo computacional que se empleará será la Máquina de Turing Determinista (DTM) y la Máquina de Turing No Determinista (NDTM). Entre las clases a definir, se tiene que señalar que, los problemas que las conforman tienen la naturaleza de problemas de decisión. Sin embargo, la última clase a definir, la de los problemas NP-duros, donde se engloba el problema a resolver en este trabajo, está compuesta tanto por problemas de decisión como de distinta índole.

Clase P

La clase de problemas P representa aquellos problemas informáticos que pueden ser resueltos en tiempo polinomial por una máquina de Turing determinista. Estos problemas, por sus características, se identifican como problemas simples, puesto que los recursos espaciales y temporales necesarios para su resolución permiten que sean practicables con los algoritmos y los computadores conocidos en la actualidad.

Además, los problemas englobados en esta definición están resueltos por la comunidad científica [20, p. 275], puesto que la misma define que se ha conseguido dar solución a un problema cuando los recursos destinados para ello se encuentran englobados en los límites temporales polinomiales. En esta sección, se representarán visualmente las clases de complejidad, empezando por la clase P, representada en la figura 2.7.

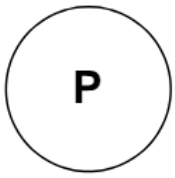

Figura 2.7: Clase P.

Clase NP

Seguidamente, la clase de problemas NP representa a aquellos problemas que pueden ser verificados por una Máquina de Turing Determinista en tiempo polinomial, pero no tienen por qué ser resolubles por dicha máquina en el tiempo señalado. Sin embargo, su resolución debe darse en tiempo polinomial por una Máquina de Turing No Determinista. Estos problemas no se han podido resolver por la comunidad científica mediante técnicas exactas, puesto que su crecimiento temporal entra en la categoría del crecimiento exponencial. Como puede verse en la figura 2.8, la clase P está contenida por la clase NP, puesto que cumple la característica de ser verificable en tiempo polinomial por una DTM.

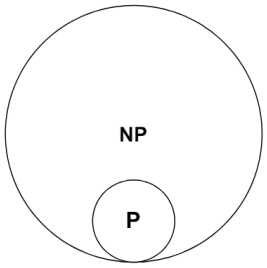

Figura 2.8: Clase NP.

Clase NP-completo

Dentro de la clase de problemas NP figura un conjunto de problemas llamados NP-completos. Esta distinción fue propuesta por Stephen Cook [5] y Leonid Levin [14] en los años 70s y propone la existencia de un conjunto de problemas relacionados entre sí con capacidad de ser reducidos en tiempo polinomial por una Máquina de Turing Determinista a cualquier otro problema de la misma clase [10, p. 34]. Esta característica hace posible que, si un problema de la clase NP-completo se consigue resolver en tiempo polinomial por una Máquina de Turing Determinista, se podrá realizar una reducción hacia cualquiera de los otros problemas de su clase y resolverlos igualmente.

Por otra parte, una propiedad importante de la clase NP-completo es que los problemas que forman parte de ella, deben de ser los más complejos entre los problemas en la clase NP. Esto se puede verificar mediante la reducción de la que se ha hablado previamente. Se establece que si un problema L_1 puede reducirse en tiempo polinomial a un problema L_2, se entiende que el problema L_1 es más simple que el problema L_2. Por ello, cualquier problema perteneciente a la clase NP-completo debe de tener la capacidad de que todos los problemas de la clase NP se puedan reducir a él en tiempo polinomial por una DTM. En la figura 2.9 se puede ver la categoría NP-completo dentro de la categoría NP.

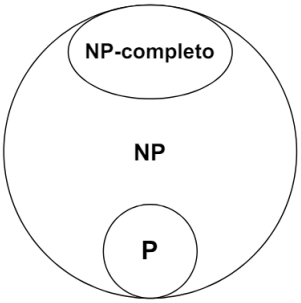

Figura 2.9: Clase NP-completo.

Clase NP-duro

La clase de problemas NP-duro es aquella que contiene los problemas de la categoría NP-completo y un conjunto de problemas de distinta índole que tampoco son resolubles en tiempo polinomial por una Máquina de Turing Determinista [15]. En la figura 2.10 puede verse un diagrama con la clase NP-duro y la intersección que presenta con la clase NP, mediante la clase NP-completo.

Dentro de esta clase, se pueden ver problemas de optimización, como el problema del Viajante de Comercio, que se encuentra estrechamente relacionado con el problema de Selección de Rutas con Sincronización, que se pretende resolver en este trabajo. Puesto que los problemas de la categoría NP-duro son de alta complejidad, se justifica la creación de un algoritmo heurístico o aproximado para su resolución en tiempo polinomial.

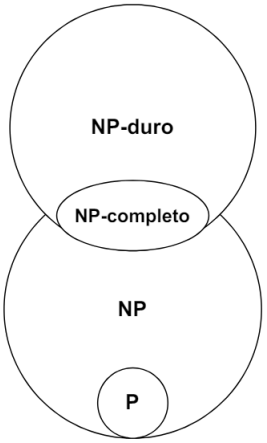

Figura 2.10: Clase NP-duro.

2.3. Algoritmia

2.3.1. Algoritmos Exactos

Un algoritmo exacto es aquel que permite resolver un problema con la garantía de que se obtendrá la solución óptima. Por las características de estos algoritmos, su aplicación en problemas de alta complejidad no es practicable, puesto que, aunque se asegura la obtención de la mejor solución, los tiempos de ejecución pueden ascender de forma exponencial [8].

Este tipo de algoritmos son inevitables si se busca la obtención de una solución óptima, pero si dicha condición no es obligatoria, se abre la opción de emplear algoritmos aproximados, que permitan la obtención de una solución de calidad, relativamente cercana al óptimo global [9].

2.3.2. Algoritmos Aproximados

Los problemas complejos, aquellos que se encuentran fuera de la clase P en cuanto a su complejidad computacional, necesitan técnicas aproximadas o heurísticas para su resolución en tiempo polinomial. Dichas técnicas permiten obtener soluciones a problemas sin

necesidad de cumplir con que sean óptimas, pero acercándose a dicho valor dependiendo de los métodos que se puedan emplear y del tiempo que se permita destinar a la ejecución del programa. Además, los algoritmos aproximados disponen de la capacidad de emplear un valor ajustable que defina la precisión que debe de tener la solución obtenida respecto a un valor resultado y garantizan la obtención de las soluciones en tiempo polinomial.

De esta forma, existen diversas técnicas para la construcción de los algoritmos aproximados, como puede verse en [24], que muestra las aplicaciones de los algoritmos voraces o algoritmos estocásticos para aportar las soluciones aproximadas a un problema, entre otras categorías.

Algoritmos Voraces

Los algoritmos voraces son una técnica algorítmica que permite la construcción de soluciones para un problema en tiempos muy reducidos. Se muestra un pseudocódigo para un algoritmo voraz en el código 1.

De este modo, los algoritmos voraces se basan en la búsqueda de sucesivos óptimos locales sin tener en cuenta que los mismos pueden no acercarse a la solución óptima global. Además, por su estructura, no se permite emplear técnicas de "backtracking", puesto que, una vez se incluye un elemento en la solución, no se revisa ni se retira [23].

Algoritmos Estocásticos

Los algoritmos estocásticos son técnicas que emplean aleatoriedad para construir soluciones aproximadas a un problema. Esto puede ser útil cuando interesa llevar a cabo pruebas con soluciones muy diferentes, al contrario de lo que se consigue usando algoritmos voraces, cuya solución inicial es invariable.

Estas técnicas pueden ser realmente interesantes cuando se ejecutan múltiples veces, permitiendo diversas evaluaciones y seleccionando aquella solución que permita obtener el mejor resultado en el valor objetivo. Como puede verse en el artículo [4], las técnicas de búsqueda aleatoria permiten la obtención de soluciones que pueden acercarse al óptimo global, siempre y cuando se dote del suficiente tiempo de ejecución al algoritmo o se le aplique un factor de precisión alto.

Búsquedas Locales

Las búsquedas locales son técnicas que permiten la mejora de una solución (posiblemente obtenida mediante un algoritmo voraz o estocástico), para acercarse a un óptimo local respecto a las condiciones iniciales de dicha solución [22].

Para que las búsquedas locales sean posibles, es necesario dotar a los programas que las utilizan con mecanismos para modificar las soluciones iniciales. De esta forma, se introduce la figura del movimiento simple, que se basa en la opción de añadir, eliminar o sustituir un elemento en la solución para encontrar mejores soluciones en la estructura de entorno asociada al movimiento. Para que la búsqueda local sea efectiva, las modificaciones se deben de hacer con un único paso, permitiendo explorar lo que se conoce como el área de soluciones vecinas.

En el área de soluciones vecinas, se puede mejorar o empeorar la solución inicial, dependiendo del valor de la función objetivo que se consiga con el movimiento. Si una solución mejora, se establece como solución preferida. En cambio, si la solución empeora, se prueba un movimiento diferente respecto a la solución previa. Será necesario aplicar las mejoras locales hasta que se llegue a lo que se conoce como valle (en caso de que el problema sea de minimización) o colina (en caso de que el problema sea de maximización). Dichos valles y colinas pueden representarse en el espacio de soluciones como mínimos o máximos respectivamente.

2.3.3. Algoritmos Heurísticos

Los algoritmos heurísticos son técnicas para la resolución de problemas de gran complejidad que suponen un campo de estudio al que se destinan grandes esfuerzos a día de hoy. Las bases de los algoritmos heurísticos residen en tener la capacidad de resolver problemas complejos mediante la mejora de soluciones iniciales y el escape de los óptimos locales. De esta forma, el espacio de soluciones se podrá explorar, teniendo la esperanza de obtener soluciones de mejor calidad e incluso llegar al descubrimiento de un óptimo global.

Entre los algoritmos heurísticos existen múltiples planteamientos, puesto que, como se propone en el documento [8], no existe una receta única para desarrollar una buena heurística. En cualquier caso, entre los planteamientos más estudiados, se puede identificar un conjunto que será realmente importante en el desarrollo de este proyecto.

GRASP

GRASP (Greedy Randomized Adaptive Search Procedures) es un método heurístico para la resolución de problemas de alta complejidad. El procedimiento se basa en la generación de una solución inicial mediante un algoritmo voraz con aleatoriedad. Esto supone la selección de un subconjunto de mejores elementos a incluir en la solución y seleccionar uno de ellos de manera aleatoria.

Una vez se lleva a cabo la solución inicial, se busca aplicar un procedimiento de mejoría local que permita conseguir un óptimo en el espacio de soluciones vecino. De esta manera, cada vez que se ejecute un procedimiento mediante GRASP, se generarán soluciones variadas en el espacio de soluciones y se conseguirán óptimos locales respecto a las mismas.

De la misma forma que ocurre con los algoritmos aproximados estocásticos, el uso de técnicas de multi-arranque serán cruciales para obtener soluciones cercanas al óptimo global. Esto supone ejecutar múltiples veces el procedimiento y seleccionar la mejor solución obtenida. En la referencia [7], se puede ver en detalle la construcción de procedimientos GRASP. Además, se muestra un algoritmo con la estructura del procedimiento en el procedimiento 2.

VND

Variable Neighbourhood Descent es un procedimiento que permite explorar el espacio de soluciones local a una solución inicial. La principal diferencia respecto a las técnicas básicas de búsqueda local radican en que se dispone de un conjunto de vecinos mayor, es decir, los movimientos que alteran la solución inicial, no son de un único tipo, sino que se puede explorar usando varios. Por ejemplo, en vez de usar el espacio de soluciones vecinas únicamente a partir de movimientos de adición, se podrán fusionar movimientos de adición, sustitución y eliminación de elementos en el mismo procedimiento.

Este método aporta mayor flexibilidad a la hora de explorar el espacio de soluciones, permitiendo encontrar óptimos locales con mayor facilidad [13]. Cuando se lleva a cabo un procedimiento VND, se necesita tener una solución inicial y el conjunto de movimientos que se podrán aplicar sobre dicha solución. Se presenta un pseudocódigo que ilustre el funcionamiento en la figura 3. En ella, puede verse que, en la función de exploración, se recibe como parámetro un valor numérico $Movement$ que representa el identificador del movimiento a realizar y que, con cada iteración, irá aumentando. Se señala que, de esta forma, todos los movimientos definidos tomarán parte del procedimiento VND.

GVNS

El procedimiento GVNS o General Variable Neighbourhood Search implica un método para escapar de óptimos locales y conseguir un acercamiento de la solución a un posible óptimo global (véase [2]). Este método juega con el concepto de orden de vecindad, que define el número de movimientos que se aplican a la solución inicial para conseguir escapar de un óptimo local. Dicho orden aumentará a medida que se ejecuta el procedimiento, permitiendo explorar espacios de soluciones cada vez mayores. En el artículo [16] se puede observar cómo desarrollar correctamente un método GVNS y sus variantes.

Para desarrollar un procedimiento GVNS se debe de disponer de una solución inicial y un conjunto de movimientos que permitirán modificar la solución para explorar el espacio de soluciones. Se muestra un pseudocódigo en la figura 4 que ilustra su funcionamiento. En ella, se puede observar una función que realiza el proceso de "Shaking", que consiste en una variación de la solución en un orden de vecindad establecido. Dicho orden representa cuántos movimientos se realizarán para aplicar una variación significativa a la solución, e irá en aumento hasta un valor definido.

2.4. Problemática

El problema de selección de rutas con sincronización (SRPS) contiene características de un conjunto de problemas que se mostrarán en este apartado con el objetivo de lograr un mayor entendimiento del mismo. Se hará referencia al artículo [19], donde se define el problema a resolver en extensión.

2.4.1. Problemas de Orientación

Los problemas de orientación pueden verse detallados en los documentos [21] y [12], donde se definen sus características básicas y posibles técnicas de resolución. De manera

general, los problemas de orientación comienzan con una colección de tareas a llevar a cabo por uno o más clientes con una limitación temporal. El objetivo reside en seleccionar un subconjunto de las tareas a resolver por los clientes, intentando optimizar el valor aportado por las mismas sin superar el tiempo disponible para llevarlas a cabo. El tiempo final no tiene que ser minimizado, puesto que sólo representa una limitación que se ha de cumplir.

Las similitudes respecto al problema que se trata de resolver en este trabajo son claras, pero existen dos diferencias que deben señalarse:

- En los problemas de orientación, no existe la sincronización de forma obligatoria.
- La lista de clientes no tiene por qué estar asociada a la lista de tareas a resolver. Esto es, un cliente cualquiera puede encargarse de una tarea determinada, pero puede darse el caso de que otro cliente se encargue de esa tarea también. Sin embargo, en el problema a resolver en este trabajo, los clientes (barras deslizantes de la unidad robótica) pueden encargarse únicamente de los objetivos que se encuentren en su área de acción.

2.4.2. Problemas de Sincronización

Los problemas de sincronización se generan, mayormente, en el ámbito de la logística. Entre ellos puede verse al VRP (Vehicle Routing Problem) con sincronización, en el cual se da la situación en la que dos o más vehículos necesiten llegar a un almacén determinado al mismo tiempo o tengan la obligación de terminar una operación de descarga a la vez (véase [6] para una descripción intensiva acerca del VRP con sincronización).

La sincronización puede estar definida por una ventana temporal, en la que se lleva a cabo la acción sincronizada con algo más de libertad o puede representarse como un instante de tiempo, siendo una sincronización mucho menos permisiva. En el segundo caso, la figura de la sincronización introduce el concepto del tiempo de espera, aspecto clave a la hora de gestionar los clientes.

De esta forma, la figura de la sincronización representa un aspecto limitador del comportamiento de los clientes de los que se compone el problema y su gestión será fundamental a la hora de construir un algoritmo de calidad que lo resuelva. En el documento [3] se propone un estudio acerca de los problemas de optimización con sincronización y el uso de algoritmos heurísticos con aleatoriedad sesgada para su resolución.

Capítulo 3

Desarrollo

En este apartado se llevará a cabo una descripción general de la metodología de trabajo. Además, se realizará un recorrido por las diferentes secciones de desarrollo, presentando los pasos realizados en cada una de ellas, desde el comienzo del proyecto hasta su completitud.

3.1. Esquema de trabajo

Inicialmente, se llevó a cabo un análisis del estado del arte, de forma que se pudiese disponer de una perspectiva lo suficientemente informada como para desarrollar el proyecto. Posteriormente, se empezó a programar el código del mismo, mediante la elaboración de un generador de instancias aleatorias para el problema. A continuación, se llevó a cabo una revisión del algoritmo exacto, que se emplearía para comparar los resultados de los algoritmos aproximados y heurísticos. Seguidamente, se desarrollarían los algoritmos heurísticos, la sección central del proyecto y por ende, la más extensa y en la que se tuvo que tomar la mayoría de las decisiones críticas. Finalmente, se desarrollarían los últimos dos algoritmos, el aproximado estocástico y el voraz, los cuales fueron una extensión del proyecto inicial, con el objetivo de representar un complemento para el estudio comparativo final, que representaría el cierre del proyecto. En la figura 3.1, se puede ver el recorrido del desarrollo del proyecto.

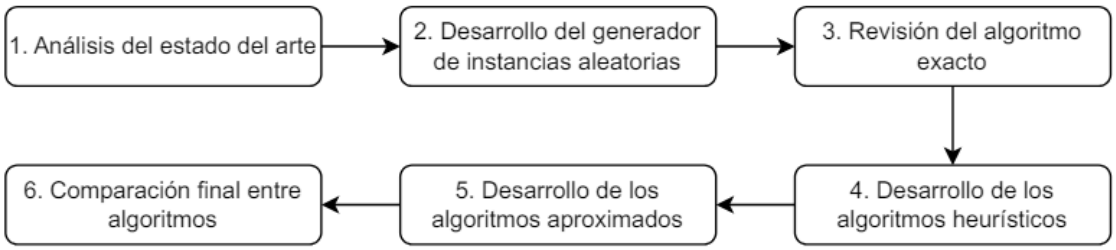

Figura 3.1: Esquema de trabajo.

3.2. Herramientas de desarrollo

Esta sección presentará las herramientas más importantes durante el desarrollo del proyecto.

3.2.1. Visual Studio Code

Visual Studio Code es un editor de código fuente desarrollado por Microsoft con soporte de depuración y control de versiones con Git. Además, dispone de la opción de ser personalizado, lo que permite un desarrollo de código cómodo y limpio. Por ello, fue el editor elegido para llevar a cabo todas las secciones de código del trabajo [1].

3.2.2. Git y Github

Git es un sistema de control de versiones que permite llevar a cabo el seguimiento de los cambios realizados a un proyecto. Por otra parte, Github es una plataforma de desarrollo colaborativo en la cual pueden alojarse proyectos en la nube y usando Git como sistema de control de versiones. Este sistema fue empleado en el proyecto por su facilidad para compartir el código fuente y realizar el seguimiento de los cambios al mismo [2].

3.2.3. C++

El lenguaje de programación C++ supuso una extensión del lenguaje de programación C en la que se permitió dotar al lenguaje de la manipulación de objetos manteniendo el nivel de rendimiento que presentaba C. De esta forma, siguiendo la programación orientada a objetos como el paradigma a emplear en el desarrollo de código y aprovechando los niveles de rendimiento que aporta el lenguaje C++, se eligió como el idóneo para llevar a cabo el proyecto.

3.3. Generador de instancias aleatorias

Inicialmente, se llevó a cabo un programa que permitiese generar una batería de instancias de dimensión variable para poder ejecutar los algoritmos desarrollados. En la figura 3.2 puede verse el diagrama de clases del generador de instancias.

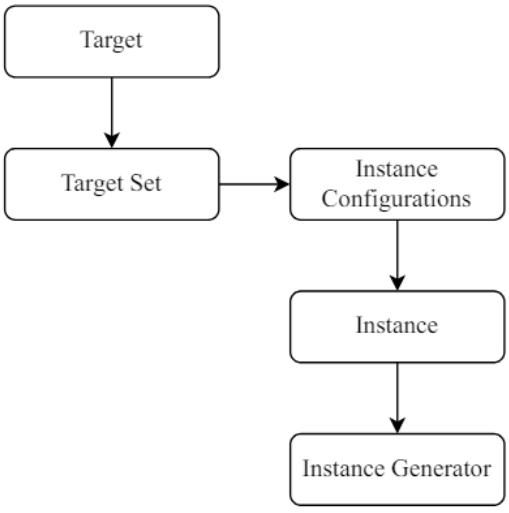

Figura 3.2: Diagrama de clases simplificado para el generador de instancias.

[1] Visual Studio Code
[2] Github

3.3.1. Clase Target

Para el generador de instancias, fue necesario definir estructuras de datos que permitiesen representar los objetivos en el plano bidimensional. De esta forma, se definió una clase *Target* que contendría los siguientes datos:

- Coordenadas del objetivo.
- Tiempo de exposición: Tiempo que tarda el objetivo en ser observado.
- Prioridad: Valor objetivo a maximizar que permitirá definir el beneficio del objetivo observable.

3.3.2. Clase Target Set

Después de desarrollar los objetivos, se creó una clase que permitiese definir un conjunto de los mismos. De esta forma, se podrían incluir objetivos a una colección de datos y calcular una matriz de costes que almacenase los tiempos de configuración de la CSU entre todos los pares de objetivos que formen el problema.

La matriz de costes estaría compuesta por tantas filas y columnas como objetivos observables. A dichas dimensiones se les sumaría un objetivo inicial y un objetivo final, que definirían las configuraciones de la unidad robótica en ambas situaciones, esto es, en configuración por defecto. Estos objetivos se incluyen puesto que existen dos condiciones dentro del periodo de observación:

- Al iniciar la observación, se parte de una CSU en configuración básica.
- Antes de terminar las observaciones, se debe de configurar la CSU tal y como se encontraba al principio.

3.3.3. Clase Instance Configurations

La clase Instance Configurations se emplea para recoger los valores asociados a las características de la instancia a generar aleatoriamente. Por consiguiente, se tomarán los datos relativos a la CSU, tales como el número de barras paralelas y su velocidad de movimiento, las zonas muertas de las barras y las dimensiones totales del campo de visión de la CSU.

Por otra parte, se tomarían los datos de la propia instancia, que definirían rangos numéricos entre los que se podrían generar datos aleatorios. De esta manera, las coordenadas de los objetivos, el tiempo de exposición y el valor de prioridad de los mismos, además del tiempo total de observación, se podrían generar aleatoriamente en base a los límites establecidos. Por otra parte, se dispuso de un atributo que permitiese almacenar el tiempo permitido para la fase de mejora de los algoritmos aproximados y heurísticos, de modo que se pudiese ajustar dicho valor en diferentes instancias.

3.3.4. Clase Instance

Una vez se dispone de los datos asociados a la instancia, puede definirse una clase para la misma, en la que se dispondrá de los siguientes datos:

- Conjunto de objetivos.
- Matriz de costes.
- Estructura de datos J_k: Define una colección de objetivos observables para cada uno de los procesadores de la CSU. Es necesario disponer de esta estructura puesto que, para cada procesador, existe un conjunto de objetivos a observar invariable desde que se configura el campo de visión del telescopio.
- Estructura de datos K_j: Dispone de los procesadores asignados a cada objetivo. Es una estructura inversa a J_k.

Dentro de la clase Instance se podrá generar la matriz de costes y las estructuras de datos J_k y K_j en base a los objetivos definidos.

3.3.5. Clase Instance Generator

Finalmente, la clase Instance Generator está dispuesta de las clases definidas anteriormente, puesto que será capaz de tomar un conjunto de configuraciones de generación de instancias para que, a partir de ellas, se pueda construir una colección de objetivos observables aleatoria y una instancia asociada a dicha colección. Cabe destacar que el formato en el que se codificarán la instancia y los objetivos generados es JSON, por lo que se dispondrá de métodos de lectura y escritura para dicho formato en las clases descritas.

Los objetivos generados pueden verse representados en formato JSON en la figura B.1. Por otro lado, los datos de la instancia se muestran en la figura B.2, que contiene la matriz de costes, la estructura J_k y la lista de prioridad para cada objetivo.

Se muestra una sección de la matriz de costes para el objetivo inicial (configuración básica de la CSU) en B.3 y una fracción de la estructura J_k en la figura B.4, donde puede verse el conjunto de objetivos asociados a los dos procesadores superiores de la CSU.

3.4. Entrada de datos

La entrada de datos define el procedimiento que se realizará para leer las instancias generadas y proporcionar dicha información a los algoritmos desarrollados en este trabajo. Una de las componentes de la entrada de datos es la clase Instance que se mostró en el apartado anterior. En ella, se dispone de un método para leer las instancias en formato JSON y variables para almacenar la información obtenida. De esta forma, sólo será necesaria una clase adicional que permita acceder a la información necesaria de la forma más rápida posible. A esta clase la llamaremos Input.

3.4.1. Clase Input

La clase Input dispone de una instancia y una matriz de costes, que estará relacionada a la matriz ubicada en la propia instancia, pero con algunas modificaciones que facilitarán el desarrollo de los algoritmos de resolución del problema.

De esta forma, la modificación realizada a la matriz de costes original reside en el ajuste de ciertos valores nulos en la misma. Dichos cambios pueden mostrarse tomando el ejemplo del objetivo final del problema, que nos señala que la CSU debe de finalizar en configuración básica. Este aspecto define que el objetivo final no puede estar sucedido por ningún otro, así que debe de tomarse el tiempo de configuración entre él mismo y cualquier otro par de objetivos y ajustarlo a un valor que llamaremos infinito. Con ello, conseguimos etiquetar ciertas configuraciones como no válidas, puesto que la CSU no debe de tener la capacidad ni la necesidad de realizarlas.

3.5. Algoritmo exacto

El algoritmo exacto se desarrolla mediante el uso de CPLEX, como un modelo matemático compacto. Su diseño se basa en el estudio [19], donde se muestra el modelo para la definición del OPS de manera inequívoca.

Caracterizamos una solución factible para el OPS mediante un camino elemental $P_k \subset A_k$ desde 0 hasta $n+1$ en G_k para cada procesador $k \in K$. El camino P_k representa la secuencia de trabajos que el procesador k debe procesar, comenzando y terminando en las posiciones de estacionamiento. No todos los trabajos necesitan ser visitados por un camino, pero cuando un camino visita un trabajo j, entonces todos los caminos P_k con $k \in K_j$ deben visitar j. Además, debe existir un valor s_j para cada $j \in V$ que cumpla:

$$s_i - s_j \leq -t_{ij} \qquad k \in K, (i,j) \in P^k \tag{3.1}$$

$$s_{n+1} - s_0 \leq L \tag{3.2}$$

El valor s_j representa el instante de comienzo del trabajo j y, junto con las restricciones 3.1, garantiza la sincronización de los procesadores al realizar ese trabajo. Es decir, todos los procesadores involucrados en el procesamiento de un trabajo específico j pueden llegar a la configuración para comenzar a procesar j en diferentes momentos, pero todos necesitan comenzar el proceso de j al mismo tiempo s_j. La desigualdad 3.2 limita el tiempo de procesamiento total.

Para cada procesador $k \in K$ y cada arco $(i,j) \in A^k$, la variable binaria x_{ij}^k representa si el procesador k está configurado para procesar el trabajo j inmediatamente después de haber procesado el trabajo i. Para cada trabajo $j \in J$, la variable binaria y_j determina si el trabajo j es seleccionado para ser procesado.

Definimos la siguiente notación estándar. Dado un procesador $k \in K$ y un subconjunto de vértices $S \subset V^k$, utilizamos $\delta^k\_(S) = \{(i,j) \in A^k \mid i \in V^k \setminus S, j \in S\}$ y $\delta_+^k(S) = \{(i,j) \in A^k \mid i \in S, j \in V^k \setminus S\}$. Si $S = \{i\}$, simplemente escribimos $\delta^k\_(i)$ y $\delta_+^k(i)$. También utilizamos $\delta\_(S) = \{(i,j,k) \in A \mid (i,j) \in \delta^k\_(S), k \in K\}$ y $\delta_+(S) = \{(i,j,k) \in A \mid (i,j) \in \delta_+^k(S), k \in K\}$.

Escribimos $x^k(F)$ en lugar de $\sum_{a \in F} x_\alpha^k$ si $F \subseteq A^k$, y $x(F)$ en lugar de $\sum_{a \in F} x_\alpha$ si $F \subseteq A$. El modelo requiere variables continuas s_j para cada tarea j en V. La variable s_j representa el instante de inicio de la tarea j.

$$\text{máx} \sum_{j \in J} b_j y_j \tag{3.3}$$

sujeto a:

$$x^k(\delta^k_+(0)) = 1 \qquad k \in K \tag{3.4}$$
$$x^k(\delta^k\_(n+1)) = 1 \qquad k \in K \tag{3.5}$$
$$x^k(\delta^k_+(j)) - x^k(\delta^k\_(j)) = 0 \qquad k \in K, j \in J^k \tag{3.6}$$
$$x^k(\delta^k\_(j)) = y_j \qquad k \in K, j \in J^k \tag{3.7}$$
$$s_i - s_j + L x^k_{ij} \leq L - t_{ij} \qquad k \in K, (i,j) \in A^k \tag{3.8}$$
$$s_{n+1} - s_0 \leq L \tag{3.9}$$
$$x^k_{ij} \in 0, 1 \qquad k \in K, (i,j) \in A^k \tag{3.10}$$
$$y_j \in 0, 1 \qquad j \in J \tag{3.11}$$

La función objetivo 3.3 maximiza la ganancia asociada con las tareas seleccionadas. Las ecuaciones 3.4 y 3.5 establecen, respectivamente, la configuración de inicio y fin para cada procesador. Las ecuaciones 3.6 y 3.7 definen las variables y_j, y exigen que cada procesador $k \in K_j$ debe procesar la tarea j si $y_j = 1$ para cada $j \in J$. Tienen un triple papel: primero, garantizan la conectividad de las configuraciones para cada procesador en la tarea j con sus respectivas configuraciones de estacionamiento inicial y final, 0 y $n+1$; segundo, fuerzan que cada procesador $k \in K_j$ esté adecuadamente configurado antes de comenzar a procesar una tarea seleccionada j; y tercero, establecen una limitación en la duración del tiempo en cada procesador. Finalmente, las cotas 3.10 y 3.11 establecen los límites de las variables x^k_{ij} y y_j, respectivamente.

3.6. Algoritmos heurísticos

Los algoritmos heurísticos se han desarrollado siguiendo el esquema de clases presente en la figura 3.3.

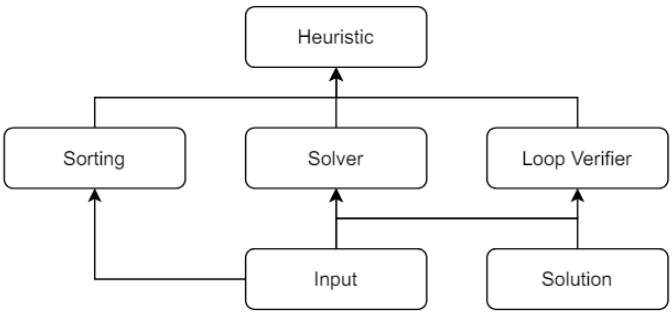

Figura 3.3: Diagrama de clases simplificado para el algoritmo heurístico.

3.6.1. Clase Solution

La CSU del instrumento EMIR dispone de un conjunto de barras laterales o procesadores independientes que permiten la configuración de rejilla múltiple. Esto da la posibilidad

de crear soluciones individuales para cada procesador por medio de grafos dirigidos, que definirán caminos, representando la secuencia de pasos para completar las observaciones de cuerpos celestes.

Diseño de las soluciones

Las soluciones se han diseñado mediante listas de adyacencia, que podrán ser representadas como grafos. De esta manera, se dispondrá de una lista de adyacencia que describa la secuencia de pasos para un procesador cualquiera k y otra lista de adyacencia que contenga la solución inversa, esto es, el camino desde el último trabajo hasta el primero. Por razones de diseño, se ha decidido disponer de ambas estructuras.

Supongamos un ejemplo en el cual existen N objetivos observables. Se muestra en la figura 3.4 una solución para un procesador k y los objetivos que observará. Por otra parte, puede observarse la lista de adyacencia de predecesores, que definirá la solución inversa en la figura 3.5.

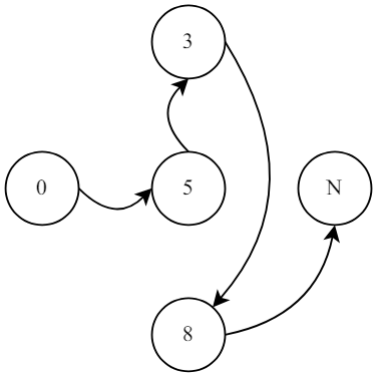

Figura 3.4: Grafo asociado a una solución de un procesador único.

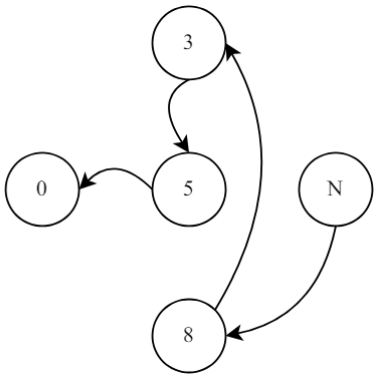

Figura 3.5: Grafo asociado a una solución inversa de un procesador único.

Asimismo, se ha decidido incorporar a la clase Solution otra estructura de datos que permita almacenar los costes asociados a cada sección del grafo, así como los respectivos tiempos de espera.

Diseño de áreas de entorno

Dentro de la clase Solution se dispone de una colección de métodos que serán cruciales en el desarrollo del algoritmo heurístico. Esto presenta la figura de las áreas de entorno, que permitirán la exploración del espacio de soluciones vecino a una solución inicial, haciendo posible la búsqueda local.

Estructura de entorno de adición

- Procedimiento $Push$: Permite añadir un trabajo a observar justo antes del trabajo final del grafo. En la figura 3.6 puede verse la inclusión del objetivo 2 justo antes del trabajo final (las líneas discontinuas suponen la configuración previa al movimiento).
- Procedimiento $Include$: Añade flexibilidad a la inclusión de trabajos, permitiendo incluir un objetivo observable a la solución después de un determinado nodo del grafo. En la figura 3.7 se incluye el objetivo 2 después del trabajo inicial.

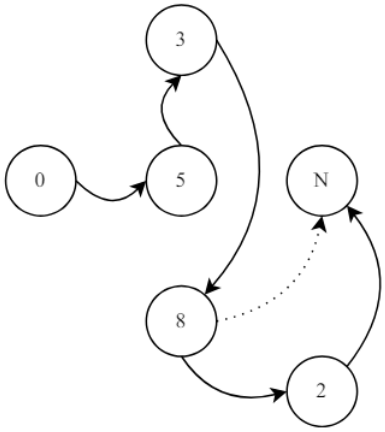

Figura 3.6: Grafo asociado a un movimiento de adición Push.

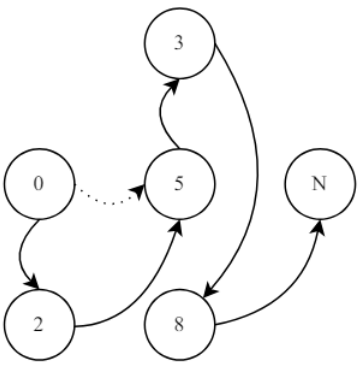

Figura 3.7: Grafo asociado a un movimiento de adición Include.

Estructura de entorno de eliminación

El movimiento de eliminación tiene la capacidad de borrar objetivos observables de la solución. En la figura 3.8 se observa la eliminación del objetivo 8.

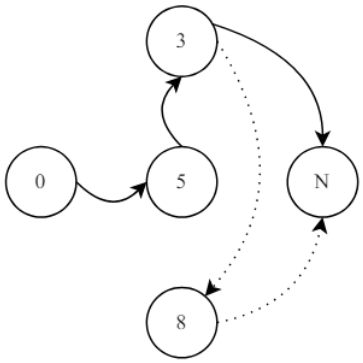

Figura 3.8: Grafo asociado a un movimiento de eliminación.

Estructura de entorno de intercambio

El movimiento de intercambio permite intercambiar las posiciones de dos objetivos incluidos en la solución. La figura 3.9 define un intercambio entre los objetivos 3 y 5.

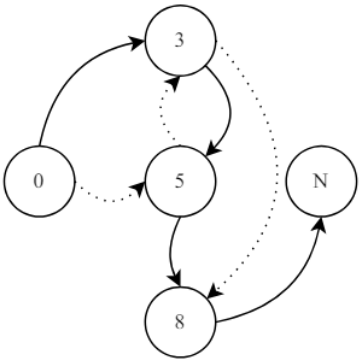

Figura 3.9: Grafo asociado a un movimiento de intercambio.

Estructura de entorno de sustitución

Finalmente, se ha desarrollado un movimiento de sustitución que tome un objetivo incluido en la solución y lo sustituya por otro que no se encuentre en la misma. En la figura 3.10 se muestra un movimiento de sustitución.

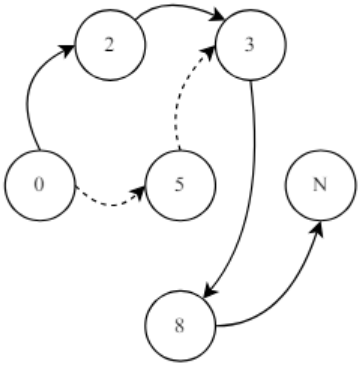

Figura 3.10: Grafo asociado a un movimiento de sustitución.

3.6.2. Clase Sorting

La clase Sorting se emplea para ordenar los objetivos observables de mayor a menor prioridad. Este procedimiento será útil en el algoritmo constructivo, que tomará los objetivos observables que tengan un mayor valor de prioridad y seleccionará, entre ellos, uno al azar en cada iteración. Disponer de una lista ordenada facilita el proceso de selección de dichos objetivos más prioritarios, puesto que estarán situados en la parte delantera de la propia lista.

- Procedimiento $ProfitListA$: Este procedimiento tomará los objetivos observables de la instancia del problema y los ordenará según su prioridad de forma descendente. Esto es, los objetivos más prioritarios estarán situados al inicio de la lista. Se muestra en la figura 3.11 la lista ordenada para una instancia con 5 objetivos.

Job	Profit
5	10000
1	10000
3	1000
2	100
4	100

Figura 3.11: Lista de prioridad A.

- Procedimiento $ProfitListB$: Por otra parte, este procedimiento permite ordenar la lista de objetivos observables de forma que los más prioritarios se encuentren en el comienzo de la lista, pero además se ordenarán aquellos objetivos con valores de prioridad iguales en base a la cantidad de procesadores que sean necesarios para su observación. Esto es, los objetivos que puedan ser observados empleando una menor cantidad de recursos, serán "mejores"que aquellos que necesiten de un mayor gasto. Se muestra en la figura 3.12 la lista ordenada para una instancia con 5 objetivos.

Job	Profit	Processors
4	10000	1
1	10000	2
2	10000	3
5	1000	2
3	1000	2

Figura 3.12: Lista de prioridad B.

3.6.3. Clase Loop Verifier

Durante los procesos de generación y mejora de soluciones, puede darse la situación de bucle, puesto que las mismas están formadas por grafos dirigidos. Esta condición debe de gestionarse, declarando como no factible toda aquella solución que contenga una estructura cíclica.

Los ciclos que pueden llegar a producirse en el algoritmo heurístico vienen definidos por la figura de la sincronización entre dos procesadores que observen un mismo objetivo. De esta forma, podemos construir un grafo como la fusión entre los grafos asociados a ambos procesadores, permitiendo que el análisis de la detección de bucles sea posible. En la figura 3.13 se observan tres grafos. Los dos grafos superiores representan soluciones individuales para dos procesadores k_1 y k_2, mientras que el grafo inferior representa la fusión de ambos grafos individuales, fruto de la sincronización entre los procesadores. Cuando se analizan los grafos pertenecientes a los procesadores k_1 y k_2 por separado, no es posible identificar una situación cíclica, pero cuando se analiza teniendo en cuenta la sincronización de ambos procesadores, se observa un ciclo entre los nodos 3 y 5, que produciría una solución no factible en el problema a resolver.

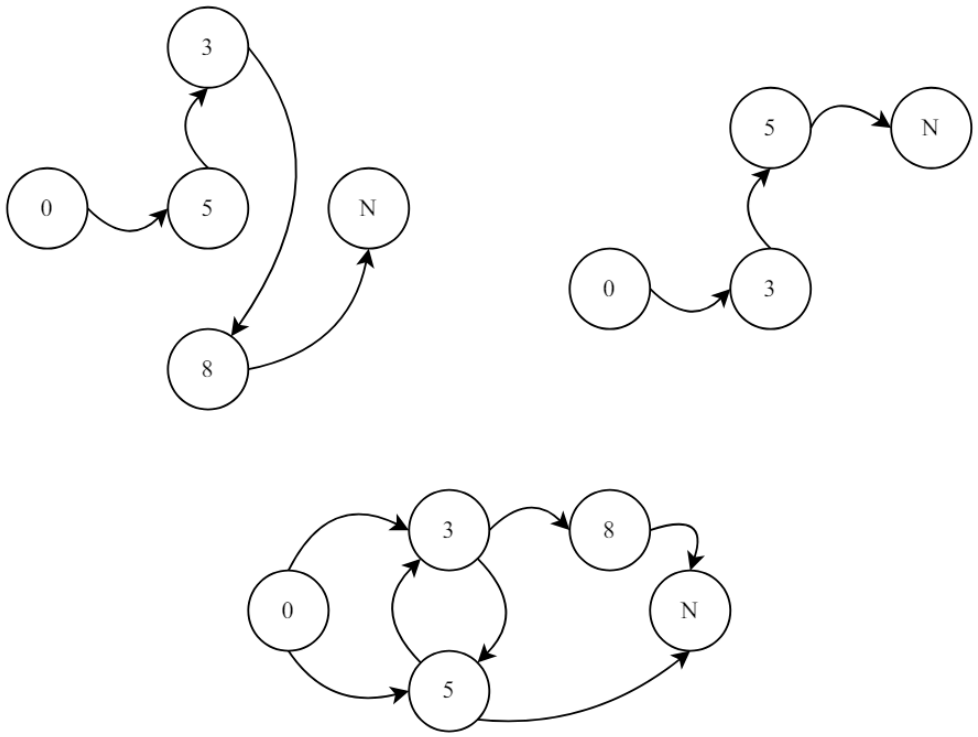

Figura 3.13: Presencia de bucles en una solución.

Para realizar la detección de bucles, se ha decidido programar, mediante patrón estrategia, una clase Loop Verifier que funcione como interfaz para dos clases que describirán algoritmos de comprobación de bucles. Dichos algoritmos serán el DFS (Depth First Search) y el algoritmo de Tarjan.

- Procedimiento DFS: La clase DFS define una implementación del algoritmo Depth First Search para la detección de bucles en el grafo de soluciones mediante el uso de procesos recursivos. Se toma un nodo inicial y se explora en profundidad por sus ramas hasta llegar al último nodo de las mismas y, si en el camino recorrido por la

exploración de una rama se visita más de una vez un mismo nodo, se detectará un bucle.

- Procedimiento $Tarjan$: Por otra parte, se ha desarrollado un método de detección de bucles mediante el algoritmo de Tarjan. Se propone un algoritmo modificado para detectar la existencia de componentes fuertemente conexas en el grafo, más que para realizar un recuento de todas las componentes que existan en la estructura.

Ambos algoritmos disponen de complejidad temporal lineal, pero se ha decidido programar el algoritmo de Tarjan por su extrema velocidad a la hora de encontrar componentes fuertemente conexas en grafos con gran número de aristas, como pueden ser los grafos generados por las soluciones del problema a resolver.

3.6.4. Clase Solver

Los algoritmos heurísticos se desarrollaron, de la misma forma que los programas previos, usando una clase abstracta genérica de la que se derivarían los dos algoritmos heurísticos: un algoritmo GRASP con multi-arranque y un algoritmo GVNS. Dentro de la clase Solver, se pueden ver diversos métodos:

- Verificación de bucles: Realiza una selección del algoritmo de detección de ciclos a usar, en base a un parámetro recibido. De esta forma se podrá usar el algoritmo DFS o Tarjan mediante la modificación de dicho valor.

- Cálculo de la factibilidad: Lleva a cabo un algoritmo para asignar los tiempos de procesamiento y de espera para cada sección de los grafos de solución. Esto es, para cada nodo, se dispondrá de un valor que represente el tiempo que el procesador ha invertido en llegar hasta él y el tiempo de espera necesario para procesar dicho objetivo, en el caso de que la sincronización exista.

  Para conseguir dicho objetivo, se trabajará con la estructura de soluciones para todos los grafos, de forma que se recorrerán cada uno de ellos, asignando los tiempos de ejecución para cada sección de los mismos. Con ello, se consigue determinar el tiempo de procesamiento hasta cada objetivo incluido en la solución mediante una estructura que guardará el máximo tiempo invertido hasta procesar el mismo. Así, si en dicha estructura, para procesar un objetivo x_1 se requiere de un tiempo t_1 y en un grafo de solución para un procesador de la CSU se encuentra dicho objetivo con un tiempo menor a t_1, se podrá ajustar el tiempo de espera, puesto que existirá sincronización entre procesadores.

- Métodos de área de entorno: Las áreas de entorno señaladas anteriormente, se han programado para aplicarse a una solución individual. Por ello, ha sido necesario incluir métodos para aplicar dichos movimientos a la estructura completa de grafos para todos los procesadores. De esta forma, si se intenta, por ejemplo, añadir o eliminar un objetivo de la solución, el procedimiento se encargará de llevar a cabo la búsqueda de los procesadores que intervienen en su observación y realizar los ajustes necesarios en los mismos.

Procedimiento VND

Por otra parte, se ha decidido incluir el procedimiento VND para las búsquedas locales en la clase Solver, puesto que los algoritmos heurísticos a desarrollar emplearán el mismo procedimiento para llevar a cabo la exploración del espacio de soluciones local.

Para desarrollar este algoritmo, se han seleccionado dos áreas de entorno: la adición y la sustitución de objetivos. Las razones por la que no se usan los movimientos de intercambio y eliminación son las siguientes:

- El movimiento de eliminación desciende el valor objetivo de la solución, haciendo imposible llegar a un óptimo local.
- El movimiento de intercambio no modifica el valor objetivo, sino que cambia el orden en el que los objetivos se observan. De la misma forma, no puede acercarse a un óptimo local.

Puesto que los movimientos de adición y sustitución son capaces de aumentar el valor objetivo de la solución, se ha decidido emplear dichos movimientos para construir el procedimiento VND. Además, se tendrá en cuenta la aplicación de un único movimiento en cada iteración del bucle de mejora, puesto que así se explorará el espacio de soluciones local, únicamente. Cabe señalar que el movimiento de adición será el primero en aplicarse, mientras que el de sustitución será el segundo. Siempre y cuando se consiga mejorar la solución en alguna iteración del bucle, se volverá a repetir el proceso hasta que no queden áreas de entorno que aplicar.

Cabe destacar que el proceso de búsqueda en cada iteración del procedimiento VND se llevará a cabo mediante el criterio de la mejor solución respecto al área de entorno aplicada. Es decir, la solución obtenida en cada paso garantizará la obtención de un óptimo local en relación a dicho área.

3.6.5. Clase Heuristic

El algoritmo heurístico principal del proyecto se basa en la generación de una solución inicial mediante un procedimiento GRASP, con su respectivo algoritmo VND para realizar las búsquedas del óptimo local. Además, dispone de un procedimiento GVNS que permitirá explorar el espacio de soluciones global mediante un parámetro que ajuste el tiempo de ejecución del mismo. De esta forma, se espera que el algoritmo pueda obtener soluciones cercanas al óptimo global mediante la exploración del espacio de soluciones si se destina el tiempo suficiente a su ejecución.

Procedimiento GRASP

El algoritmo GRASP para la generación de soluciones iniciales al problema se lleva a cabo partiendo de una lista ordenada de objetivos observables, mediante los métodos de ordenación mostrados anteriormente.

Así, seleccionando los primeros N objetivos de la lista (siendo N un valor ajustable, conocido como lista de candidatos), se podrá elegir, en cada iteración de la construcción

un elemento de forma aleatoria, incluyéndolo en la solución, hasta que no se puedan añadir más objetivos a la misma. Por último, se realizará una búsqueda local mediante el procedimiento VND para mejorar la solución generada.

Procedimiento GVNS

Seguidamente, el algoritmo GVNS se desarrolló empleando las áreas de entorno de eliminación e intercambio de objetivos. De esta forma, se llevaría a cabo una perturbación aleatoria en el espacio de soluciones asociado a un área de entorno determinado y se aplicaría una búsqueda local mediante VND para su correspondiente mejoría.

Al tratarse de un procedimiento GVNS, se trabajaría con una variable que representa el orden de vecindad (valor que determinará cuantos movimientos de solución se aplican para llevar a cabo la perturbación aleatoria), permitiendo aumentar dicho orden cada vez que no se encontrase una solución con potencial para mejorar el valor objetivo de la solución inicial.

3.6.6. Clase MultiGRASP

De manera adicional, se ha desarrollado un segundo algoritmo heurístico empleando el procedimiento GRASP incluido en el algoritmo presentado anteriormente, aunque aplicando la figura del multi-arranque para generar una batería de soluciones variadas y seleccionar la mejor que se haya encontrado.

De esta forma, empleando el tiempo de mejora de soluciones ubicado en las instancias generadas previamente, se conseguirá aplicar GRASP en múltiples ocasiones, durante dicha ventana de tiempo, para conseguir la mejor solución posible.

3.7. Algoritmos aproximados

Los algoritmos aproximados que se han decidido incorporar en el proyecto son dos procedimientos: un algoritmo voraz y un algoritmo estocástico. Esta familia de algoritmos se ha llevado a cabo tomando como base la entrada al algoritmo heurístico, así como la estructura del programa, con los algoritmos de ordenación y los procedimientos de detección de bucles. De la misma forma, se emplea el patrón estrategia para construir el algoritmo genérico de resolución aproximada, permitiendo derivar clases heredadas del mismo.

3.7.1. Algoritmo voraz

El algoritmo voraz tendrá la capacidad de encontrar una solución factible mucho más rápido que cualquier otro algoritmo desarrollado, aunque no dispondrá de procedimientos de mejora de soluciones.

Para la generación de la solución, tomará una lista con los objetivos observables ordenados por su prioridad, tal y como ocurre en la fase constructiva del algoritmo heurístico. La principal diferencia reside en que el componente aleatorio se elimina, así

que el objetivo observable a incluir será, en cualquier caso, aquel que sume un mayor valor objetivo, siempre y cuando mantenga la factibilidad de la solución.

3.7.2. Algoritmo estocástico

El algoritmo estocástico dispondrá de un método de generación de soluciones similar a la fase constructiva del algoritmo heurístico, pero sin haber ordenado previamente la lista de objetivos observables. Además, la obtención del trabajo a incluir en la solución se llevará a cabo mediante una selección aleatoria sobre todos los objetivos de la lista.

Mediante técnicas de multi-arranque, se podrá generar una batería de soluciones variadas que podrán compararse entre ellas para tomar aquella que tenga un beneficio total más alto. Así, en cada iteración del algoritmo, se generará una solución y se comparará con la mejor solución encontrada hasta el momento, decidiendo si se actualiza dicho valor o se conserva. De esta forma, si se asigna suficiente tiempo a la ejecución del algoritmo aproximado estocástico, la batería de soluciones será lo suficientemente extensa como para encontrar una solución de calidad para el problema.

3.8. Salida de datos

Una vez los algoritmos se han ejecutado, los datos relativos a la solución deben de mostrarse por pantalla, con el objetivo de evaluar la calidad de la misma y poder conocer las secuencias necesarias para configurar la CSU.

3.8.1. Clase Output

En la clase Output se dispone de los datos asociados a la entrada de datos al problema, además de la solución final al mismo, el valor objetivo conseguido y el tiempo de observación empleado en su totalidad. Por ende, se podrá verificar la calidad de la solución y si está enmarcada en la limitación temporal otorgada por el tiempo de observación.

Respecto a la funcionalidad de la clase Output, se ha de señalar que dispone de una función capaz de calcular el valor objetivo final por medio de la suma de todos los valores de beneficio de los objetivos observables que se han incluido en la solución. Además, contendrá un amplio conjunto de métodos de visualización de datos:

- Datos relativos a la solución: Se muestra la secuencia de objetivos observables por cada procesador. Puede verse en la figura B.5 el conjunto de los datos señalados.
- Datos relativos a los tiempos: Se muestra una secuencia para cada procesador con los tiempos del recorrido hasta dicho instante, además de los tiempos de espera para cada procesador en cada sección. En la figura B.6 pueden verse los tiempos para cada sección de los caminos y en la figura B.7 se muestran los tiempos de espera para cada sección.

Capítulo 4

Experimento comparativo entre algoritmos

Para comprobar el funcionamiento y los resultados de los algoritmos programados, se procederá a realizar una comparación entre los mismos. De esta forma, se encontrarán las fortalezas y debilidades de los programas en relación a las diferentes instancias, según su tamaño.

4.1. Diseño del experimento

Para llevar a cabo el experimento comparativo, se ha necesitado generar una batería de instancias variadas, mediante el uso del generador aleatorio, de forma que se pueda verificar el comportamiento de los diferentes algoritmos respecto al tamaño de las mismas.

De esta manera, se han separado las instancias en 4 categorías:

- Instancias pequeñas: 50 objetivos.
- Instancias medianas: 100 objetivos.
- Instancias grandes: 200 objetivos.
- Instancias muy grandes: 400 objetivos.

Estas distinciones serán realmente importantes cuando se ejecute el algoritmo exacto, puesto que, para las instancias más grandes, sus tiempos de ejecución deben de ser muy elevados. Sin embargo, para los problemas más pequeños, la obtención de la solución óptima en bajos tiempos de ejecución estará prácticamente asegurada. Además, cada categoría estará formada por 3 instancias generadas aleatoriamente.

Por otra parte, en las tablas de cada algoritmo se mostrarán datos asociados al tamaño de la instancia, su tiempo de observación máximo, el valor final de la solución y el tiempo de ejecución en segundos. A su vez, para los algoritmos heurísticos y el algoritmo estocástico, se ejecutarán los programas en múltiples ocasiones, cambiando el valor del parámetro que define el tiempo de ejecución de los procedimientos de mejora o multi-arranque.

4.2. Algoritmo exacto

Se muestran los resultados del algoritmo exacto en la tabla C.1. En ella, se pueden ver diversos campos que no toman un valor determinado, tales como lo son los datos asociados a las instancias de más de 200 objetivos: en la columna asociada al valor de la solución, donde en algunos casos se muestra un valor acotado a un máximo ajustable a medida que se ejecuta el programa (cota dual) y en la columna correspondiente al tiempo de ejecución, que al ser tan grande, se ha seleccionado un límite inferior de una hora para acotarlo, aunque el tiempo de ejecución real será notablemente superior a dicha cota, en especial, cuando se ejecute la instancia de 400 objetivos observables.

Al observar la tabla con los resultados del algoritmo exacto, se puede determinar su buen funcionamiento para instancias pequeñas y medianas, esto es, instancias de 50 y 100 objetivos. Para dichos problemas, es capaz de encontrar la solución óptima en tiempos muy reducidos, de menos de un segundo en cualquier caso.

Por otra parte, los problemas grandes, de 200 objetivos, muestran un crecimiento que produce que el algoritmo exacto tarde en ejecutar más de una hora, por lo que tomaremos dichos problemas como los casos en los que el programa encuentra sus mayores debilidades. Los problemas de 400 objetivos observables deben de tomarse como irresolubles, puesto que el tiempo necesario para su ejecución es notablemente superior al empleado para resolver los problemas de la categoría anterior, creciendo de forma exponencial.

4.3. Algoritmo voraz

Se muestran los resultados del algoritmo voraz en la tabla C.2.

Tal y como se esperaba, los tiempos de ejecución del programa son cortos para cualquier instancia, puesto que no se toma ninguna búsqueda para obtener los objetivos a observar. Además, usando la lista ordenada de objetivos observables, el algoritmo tiene que seleccionar, en cualquier caso, la primera opción de la lista, permitiendo que los resultados sean más rápidos aún.

De esta forma, podemos ver como se obtienen resultados muy cercanos a los valores resultantes del algoritmo exacto para instancias de 100 o menos objetivos, mientras que en instancias de más de 200, obtiene resultados relativamente alejados de la cota dual, mostrada en la tabla previa. Se espera que, a medida que la dimensión del problema aumente, dichos valores estén menos ajustados, puesto que existen más alternativas para obtener la solución óptima.

Otra de las debilidades del algoritmo voraz se basa en la imposibilidad de generar soluciones variadas, que puedan proporcionar mejores alternativas a alguna otra solución generada anteriormente por el programa. Por ello, las técnicas de multi-arranque o heurísticas pueden obtener resultados más atractivos, aunque con tiempos notablemente superiores.

4.4. Algoritmo estocástico

Se muestran los resultados del algoritmo estocástico en las tablas C.3 y C.4. Se ha procedido a ejecutar el programa usando diferentes ventanas de tiempo ascendentes para cada instancia, de forma que se pueda observar la mejoría de las soluciones obtenidas a medida que se dote de más tiempo de ejecución a las mismas.

Inicialmente, podemos identificar la obtención de las soluciones óptimas para instancias de 50 objetivos en la ventana de tiempo de un segundo, por lo que no sería necesario ampliar la ventana en dichos casos.

Seguidamente, se identifica la mejoría de las soluciones resultantes a medida que los tiempos de multi-arranque aumentan, independientemente del tamaño de la instancia. Esto supone que, si se dota del suficiente tiempo de ejecución al algoritmo estocástico, se podrán obtener soluciones cada vez mejores. Este aspecto, para instancias medianas, de 100 objetivos, no trae demasiadas ventajas, puesto que el algoritmo exacto, e incluso el voraz, pueden obtener mejores soluciones, pero para los problemas de gran tamaño, se pueden obtener soluciones realmente atractivas si se dispone del tiempo suficiente.

De cualquier forma, con los tiempos empleados para realizar el experimento, parece que el algoritmo estocástico, no puede ajustarse a los valores óptimos, por lo que se necesitaría de mayores tiempos de ejecución o la aplicación de técnicas más sofisticadas, como las heurísticas GRASP con multi-arranque, que pueden suponer una versión mejorada del presente programa, empleando tiempos más reducidos si se trata de conseguir un acercamiento al óptimo global.

4.5. Algoritmo heurístico GRASP con multi-arranque

Se muestran los resultados del algoritmo heurístico GRASP con multi-arranque en las tablas C.5 y C.6. Si tomamos la comparación del presente algoritmo con el estocástico, ambos empleando multi-arranque, puede observarse la utilidad del procedimiento VND, puesto que los resultados logran valores notablemente más altos.

Incluso con la figura del procedimiento VND, que produce un aumento de los tiempos de ejecución por cada iteración del algoritmo, los tiempos de ejecución son mucho menores en relación a la calidad de la solución obtenida. Esto puede verse en las instancias más grandes, puesto que sería muy complicado para el algoritmo estocástico acercarse a los valores que toma la alternativa heurística empleando los tiempos de ejecución que tarda la misma.

Por otra parte, el algoritmo heurístico supone un procedimiento más seguro, puesto que encuentra las cumbres en el espacio de soluciones local respecto a la solución generada en cada iteración, mientras que el algoritmo estocástico depende, en gran medida, de la suerte que se pueda tener en su ejecución, dado que es un procedimiento que depende totalmente de la pseudo-aleatoridad, sin reparar en búsquedas locales.

4.6. Algoritmo heurístico GVNS

Se muestran los resultados del algoritmo heurístico GVNS en las tablas C.7 y C.8. En ellas, se pueden ver algunas diferencias respecto al algoritmo heurístico previo, pero son, en su mayor parte, resultados relativamente similares.

Los resultados entre los cuales se observan más diferencias se engloban en los valores finales y los tiempos de ejecución de las instancias de gran tamaño, puesto que el programa toma mayor cantidad de recursos temporales, por sus repetidas iteraciones del procedimiento VND en el bucle de ejecución del algoritmo GVNS, que busca agitar la solución inicial y aplicar la búsqueda a dicha agitación, siendo un proceso costoso.

En cualquier caso, el algoritmo GVNS muestra resultados mejores que el anterior algoritmo para instancias de gran tamaño, quizás por la capacidad del procedimiento GVNS de agitar la solución para explorar, de forma más efectiva, el espacio de soluciones, en relación al procedimiento GRASP con multi-arranque.

4.7. Comparación final

En conclusión, se puede observar que los algoritmos programados son capaces de obtener resultados en tiempos de ejecución notablemente inferiores a los del algoritmo exacto, siempre y cuando se trate de instancias grandes.

Por otra parte, el algoritmo voraz ha sido capaz de proporcionar resultados muy rápidos y de alto valor final, gracias a la lista ordenada de candidatos. Por ello, si tenemos en cuenta el tiempo de ejecución en relación a la calidad de los resultados, el algoritmo voraz ha cumplido totalmente con sus expectativas. Mientras, el algoritmo estocástico necesita de tiempos de ejecución realmente elevados para empezar a dar buenos resultados, puesto que la evolución que presenta desde la ventana de ejecución de un segundo hasta la de dos minutos, es realmente lenta. Además, el valor resultante que ha conseguido obtener en dichas ventanas es notablemente inferior al de cualquier algoritmo desarrollado en el proyecto.

Seguidamente, los algoritmos heurísticos presentan la capacidad de obtener soluciones mejores que el algoritmo voraz, aunque aplicando mayores tiempos de ejecución, sin entrar en los límites del algoritmo exacto. De esta forma, se convierten en una gran alternativa si se pretende obtener resultados variados y potencialmente mejores que los proporcionados por el algoritmo voraz. Asimismo, el algoritmo GVNS parece ser ligeramente mejor que el algoritmo GRASP con multi-arranque, puesto que el método de agitación ha sido capaz de explorar el espacio de soluciones global de manera más efectiva que el procedimiento de multi-arranque.

Capítulo 5

Conclusiones y líneas futuras

5.1. Conclusiones

En el presente proyecto, se ha llevado a cabo el diseño e implementación de un algoritmo heurístico aplicando GVNS, con el objetivo de resolver el problema de orientación con sincronización [19], que surge durante el uso de un espectrógrafo multi-objeto infrarrojo (EMIR) en el Gran Telescopio de Canarias. De la misma forma, se ha realizado un experimento comparativo del programa desarrollado con un algoritmo exacto que resuelve el mismo problema, así como un conjunto de algoritmos aproximados y heurísticos, diseñados para complementar el experimento, donde se han podido determinar las fortalezas y debilidades de cada alternativa.

Mediante el estudio comparativo, se muestra la debilidad del algoritmo exacto para instancias grandes, puesto que sus tiempos de ejecución aumentan de forma exponencial, mientras que, el resto de algoritmos, permiten obtener resultados usando menos recursos temporales.

El algoritmo voraz ha sido capaz de obtener soluciones a cualquier instancia usando recursos temporales reducidos, pero sus resultados disponen de un límite superior que no se puede sobrepasar, dada su característica de invariabilidad. Por otro lado, el algoritmo estocástico necesita de tiempos de ejecución realmente altos para alcanzar los resultados que muestra el algoritmo voraz, aunque puede obtener soluciones cada vez mejores respecto al tiempo invertido en su ejecución.

Por otra parte, los algoritmos heurísticos presentan un equilibrio entre los recursos temporales necesarios para su ejecución y el resultado final. Inicialmente, el algoritmo GRASP con multi-arranque permite aprovechar el mismo procedimiento usado para el algoritmo estocástico, añadiendo la figura del procedimiento VND para alcanzar las cumbres en el espacio local a la solución obtenida inicialmente, mediante el procedimiento GRASP, teniendo la capacidad de obtener resultados notablemente superiores de forma consistente.

Por último, el algoritmo GVNS, aunque consume más recursos temporales que la alternativa previa, permite alcanzar, normalmente, resultados de mayor calidad. Gracias a la figura del procedimiento GVNS, se puede explorar mejor el espacio de soluciones global y encontrar mejores alternativas, a costa de aplicar más búsquedas locales.

5.2. Líneas futuras

Respecto a las líneas futuras asociadas al algoritmo heurístico GVNS, se plantea que las estructuras de entorno empleadas para el procedimiento VND y el método de agitación pueden sustituirse por alternativas más sofisticadas, que permitan encontrar los máximos con mayor velocidad. Además, también puede modificarse el criterio de mejora para la búsqueda local del procedimiento VND, mediante la obtención de la primera mejor solución encontrada, respecto a un área de entorno, en contraposición a la búsqueda de la mejor solución, alternativa de la que se dispone en este proyecto. Estos cambios, podrían acercar al algoritmo heurístico GVNS a los valores resultantes del algoritmo exacto, aplicando menos recursos temporales en su ejecución.

Igualmente, se puede modificar el planteamiento del algoritmo heurístico, mediante el uso de técnicas como la Búsqueda Tabú [11], Búsqueda Dispersa o "Path-Relinking"[18], que permiten diseñar métodos estratégicos y sistemáticos, que dependan menos de la aleatoriedad del procedimiento GVNS.

Capítulo 6

Summary and Conclusions

6.1. Summary

In this project, the design and implementation of a heuristic algorithm applying GVNS has been carried out, with the objective of solving the orientation problem with synchronization [19], which arises during the use of a multi-object infrared spectrograph (EMIR) at the Gran Telescopio de Canarias. In the same way, a comparative study of the developed algorithm with an exact algorithm that solves the same problem has been conducted, as well as a set of approximate and heuristic algorithms, programmed to complement the experiment, where the strengths and weaknesses of each alternative have been determined.

Through the comparative study, the weakness of the exact algorithm for large instances is shown, as its execution times increase exponentially, while the rest of the algorithms allow obtaining results using fewer temporal resources.

The greedy algorithm has been able to obtain solutions to any instance using reduced temporal resources, but its results have an upper bound that cannot be surpassed, given its invariability characteristic. On the other hand, the stochastic approximate algorithm requires really high execution times to achieve the results shown by the greedy algorithm, although it can obtain increasingly better solutions concerning the time invested in its execution.

Moreover, the heuristic algorithms present a balance between the temporal resources needed for their execution and the final result. Initially, the GRASP algorithm with multi-start allows taking advantage of the same procedure used for the stochastic algorithm, adding the VND procedure to reach the peaks in the local space of the initially obtained solution, through the GRASP procedure, having the ability to consistently obtain significantly superior results.

Finally, the GVNS algorithm, although it consumes more temporal resources than its previous alternative, usually allows reaching higher quality results. Thanks to the GVNS procedure, the global solution space can be better explored and better alternatives can be found, at the cost of applying more local searches.

6.2. Future work

Regarding future directions associated with the GVNS heuristic algorithm, it is proposed that the neighborhood structures used for the VND procedure and the shaking method can be replaced by more sophisticated alternatives, allowing for finding maxima more quickly. Additionally, the improvement criterion for the local search of the VND procedure can be modified by obtaining the first best solution found, concerning a neighborhood area, as opposed to searching for the best solution, which is the alternative available in this project. These changes could bring the GVNS heuristic algorithm closer to the results of the exact algorithm while applying fewer temporal resources in its execution.

Similarly, the heuristic algorithm approach can be modified using techniques such as Tabu Search [11], Scatter Search, or Path-Relinking [18], which allow for designing strategic and systematic methods that depend less on the randomness of the GVNS procedure.

Capítulo 7

Presupuesto

Esta sección muestra la estimación de los costes de desarrollo del proyecto. Se estima un coste por hora de trabajo de 22€.

7.1. Costes materiales

Concepto	**Presupuesto**
Equipo informático	1.600 €
Total:	**1.600 €**

Cuadro 7.1: Costes materiales para el desarrollo del proyecto.

7.2. Coste de horas de trabajo

Concepto	**Duración**	**Presupuesto**
Revisión bibliográfica	30 h	660 €
Preparación del entorno de desarrollo	5 h	110 €
Construcción de estructuras de datos iniciales	25 h	550 €
Desarrollo del generador de instancias	20 h	440 €
Desarrollo de entrada de datos	30 h	660 €
Desarrollo del algoritmo voraz	10 h	220 €
Desarrollo del algoritmo estocástico	10 h	220 €
Desarrollo del algoritmo GRASP con multi-arranque	50 h	1.100 €
Desarrollo del algoritmo GVNS	50 h	1.100 €
Desarrollo de salida de datos	10 h	220 €
Experimento comparativo de algoritmos	30 h	660 €
Redacción de la memoria	80 h	1.760 €
Total:	**350 h**	**7.700 €**

Cuadro 7.2: Coste de las horas de trabajo para el desarrollo del proyecto.

7.3. Costes totales

Concepto	**Presupuesto**
Costes materiales	1.600 €
Coste de horas de trabajo	7.700 €
Total:	**9.300 €**

Cuadro 7.3: Costes totales del proyecto.

Apéndice A

Pseudocódigos

A.1. Algoritmo Greedy

Algorithm 1 Greedy

```
Input: Entrada del problema
procedure Greedy
    Solution = EmptySolution;
    while !IsComplete(Solution) do
        BestItem = FindBest(Solution);
        NewSolution = Solution;
        if NewSolution.Include(BestItem) is feasible then
            Solution.Include(BestItem);
        end if
        Input.RemoveItem(BestItem)
    end while
    returnSolution;
end procedure
```

A.2. Algoritmo GRASP

Algorithm 2 GRASP

```
Input: Entrada del problema
Output: Solución final
procedure GRASP
    while Criterion not satisfied do
        Solution = ConstructGreedyRandomizedSolution(Input);
        Solution = LocalSearch(Solution);
    end while
    returnSolution;
end procedure
```

A.3. Algoritmo VND

Algorithm 3 VND

Input: Solución inicial
Output: Solución final
procedure VND
  $Solution = Input$;
  while *Exists improvement* **do**
    $Movement = 1$;
    while $Movement < MaxMovement$ **do**
      $NewSolution = Exploration(Solution, Movement)$;
      if $NewSolutionbetterthanSolution$ **then**
        $Solution = NewSolution$;
        $Movement = 1$;
      else
        $Movement+ = 1$;
      end if
    end while
  end while
end procedure

A.4. Algoritmo GVNS

Algorithm 4 GVNS

Input: Solución inicial
Output: Solución final
procedure GVNS
  $Solution = Input$;
  while *Exists improvement* **do**
    $NeighbourLevel = 1$;
    while $NeighbourLevel \leq MaxNeighbourLevel$ **do**
      $NewSolution = Shaking(Solution, NeighbourLevel)$;
      $VND(NewSolution)$;
      if *NewSolution better than Solution* **then**
        $Solution = NewSolution$;
        $NeighbourLevel = 1$;
      else
        $NeighbourLevel+ = 1$;
      end if
    end while
  end while
end procedure

Apéndice B

Resultados

B.1. Generador de instancias

B.1.1. Objetivos

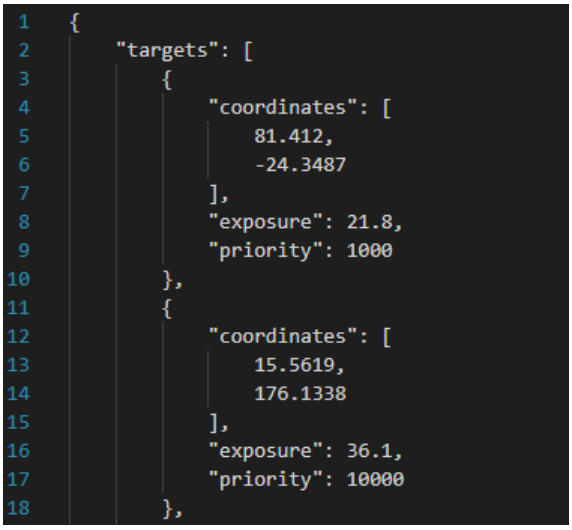

Figura B.1: Objetivos generados aleatoriamente.

B.1.2. Instancias

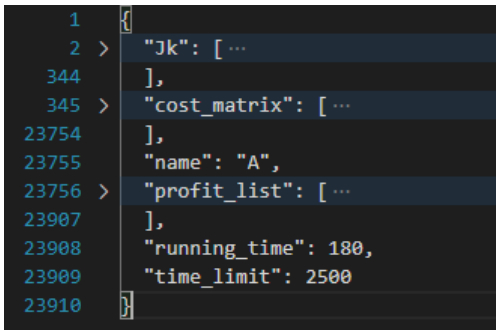

Figura B.2: Instancia en formato JSON.

```
"cost_matrix": [
  [
    0,
    407,
    77,
    351,
    2,
    178,
    116,
    561,
    90,
    491,
    203,
    546,
```


Figura B.3: Matriz de costes.


```
"Jk": [
  [],
  [],
  [
    16,
    19,
    21
  ],
  [
    7,
    87,
    92,
    104,
    114,
    139,
    142,
    146
  ],
```


Figura B.4: J_k.

B.2. Salida de datos

```
Se ha ejecutado el algoritmo heurístico para la instancia G.
La instancia dispone de 200 objetivos.
Tiempo máximo de observación: 3000 unidades
Tiempo alcanzado: 2981 unidades
Porcentaje del tiempo total alcanzado: 99.366669
Tiempo de ejecución: 728159 microsegundos
Valor objetivo final: 52560
Mostrando caminos y tiempos:
Procesador 1: 0 -> 201
Procesador 2: 0 -> 44 -> 201
Procesador 3: 0 -> 120 -> 44 -> 107 -> 201
Procesador 4: 0 -> 120 -> 107 -> 201
Procesador 5: 0 -> 100 -> 201
Procesador 6: 0 -> 100 -> 201
Procesador 7: 0 -> 30 -> 157 -> 162 -> 201
Procesador 8: 0 -> 117 -> 15 -> 96 -> 201
Procesador 9: 0 -> 117 -> 1 -> 96 -> 201
Procesador 10: 0 -> 1 -> 201
Procesador 11: 0 -> 176 -> 163 -> 164 -> 201
```

Figura B.5: Salida de datos de la solución y caminos por cada procesador.

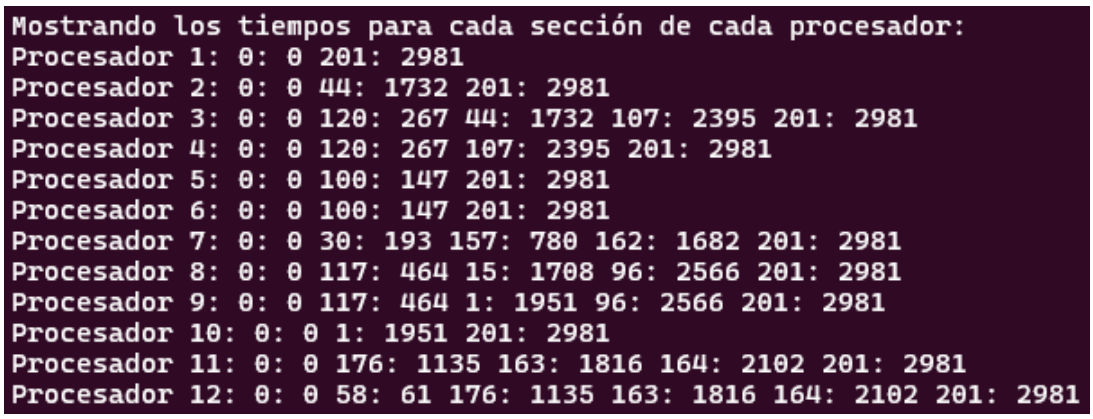

Figura B.6: Tiempos por cada sección de los caminos de cada procesador.

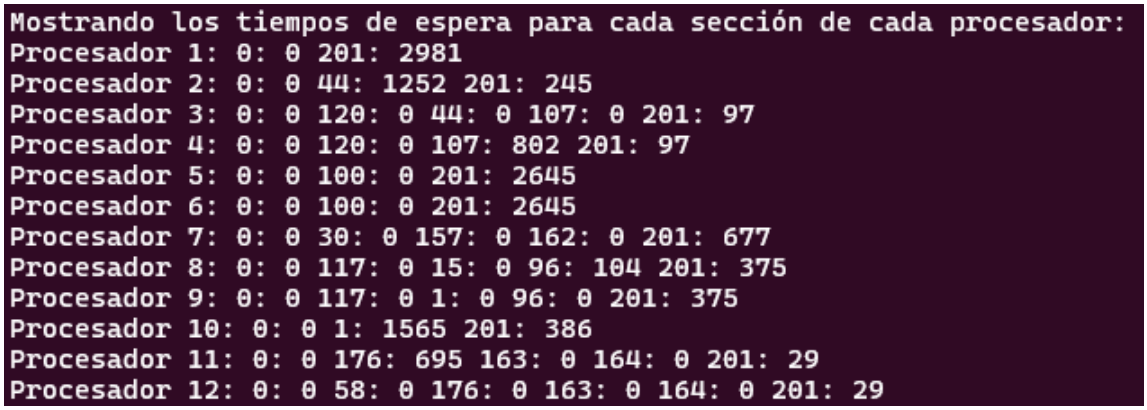

Figura B.7: Tiempos de espera por cada sección de los caminos de cada procesador.

Apéndice C

Tablas

C.1. Algoritmo exacto

	Parámetros		*Resultados*	
Instancia	n	L	*Objetivo*	*Tiempo (sec.)*
A	50	1000	4 480	0.03
B			4 030	0.12
C			11 240	0.04
D	100	2000	30 720	0.47
E			27 560	0.66
F			27 030	0.64
G	200	3000	$<$ 64 470	$>$ 1 hora
H			$<$ 69 721	$>$ 1 hora
I			$<$ 66 095	$>$ 1 hora
J	400	4000	$<$ 149 800	$>$ 1 hora
K			$<$ 139 090	$>$ 1 hora
L			$<$ 153 400	$>$ 1 hora

Cuadro C.1: Resultados de ejecución para el algoritmo exacto.

C.2. Algoritmo voraz

	Parámetros		*Resultados*	
Instancia	n	L	*Objetivo*	*Tiempo (sec.)*
A	50	1000	4 480	0.01
B			4 030	0.01
C			11 240	0.01
D	100	2000	30 430	0.07
E			26 330	0.07
F			25 690	0.07
G	200	3000	52 560	0.71
H			59 570	0.70
I			50 810	0.72
J	400	4000	103 640	4.18
K			95 360	4.22
L			99 480	4.26

Cuadro C.2: Resultados de ejecución para el algoritmo voraz.

C.3. Algoritmo estocástico

	Parámetros			*Resultados*	
Instancia	n	L	*Mejora (sec.)*	*Objetivo*	*Tiempo (sec.)*
A	50	1000	1	4 480	1.00
B			1	4 030	1.00
C			1	11 240	1.01
D	100	2000	1	24 660	1.00
			60	27 270	60.06
			120	27 650	120.08
			180	28 640	180.02
			240	29 250	240.01
E	100	2000	1	22 770	1.00
			60	23 360	60.02
			120	23 740	120.01
			180	24 670	180.01
			240	25 260	240.04
F	100	2000	1	22 020	1.00
			60	23 120	60.05
			120	23 130	120.02
			180	23 560	180.04
			240	23 510	240.05
G	200	3000	1	32 420	1.41
			60	36 650	60.41
			120	38 490	120.65
			180	39 480	180.11
			240	42 640	240.50
H	200	3000	1	37 780	1.46
			60	42 670	60.73
			120	42 450	120.47
			180	43 160	180.19
			240	43 380	240.67
I	200	3000	1	34 980	1.43
			60	37 690	60.03
			120	37 980	120.29
			180	40 200	180.27
			240	40 260	240.20

Cuadro C.3: Resultados de ejecución para el algoritmo aproximado estocástico.

	Parámetros			*Resultados*	
Instancia	n	L	*Mejora (sec.)*	*Objetivo*	*Tiempo (sec.)*
J	400	4000	1	58 090	4.21
			60	58 900	60.49
			120	62 340	123.18
			180	64 060	183.23
			240	61 300	243.61
K	400	4000	1	42 720	4.12
			60	58 770	60.36
			120	57 400	122.72
			180	54 740	182.38
			240	54 660	241.23
L	400	4000	1	53 850	4.17
			60	58 900	64.16
			120	58 960	123.05
			180	61 300	180.28
			240	62 710	241.14

Cuadro C.4: Continuación de resultados de ejecución para el algoritmo aproximado estocástico.

C.4. Algoritmo GRASP con multi-arranque

	Parámetros			*Resultados*	
Instancia	n	L	*Mejora (sec.)*	*Objetivo*	*Tiempo (sec.)*
A	50	1000	1	4 480	1.06
B	50	1000	1	4 030	1.09
C	50	1000	1	11 240	1.04
D	100	2000	1	28 500	1.12
			60	30 510	60.34
			120	30 610	121.69
			180	30 610	180.52
			240	30 610	240.33
E	100	2000	1	26 550	2.21
			60	26 650	63.96
			120	27 550	121.28
			180	27 550	180.93
			240	27 530	240.82
F	100	2000	1	25 670	1.21
			60	26 810	60.57
			120	26 810	122.89
			180	26 910	180.56
			240	26 890	240.20
G	200	3000	1	53 910	8.60
			60	55 000	74.42
			120	54 140	134.83
			180	55 710	183.99
			240	55 930	244.46
H	200	3000	1	57 680	26.93
			60	59 340	76.91
			120	59 760	129.40
			180	60 680	223.64
			240	61 760	268.76

Cuadro C.5: Resultados de ejecución para el algoritmo GRASP con multi-arranque.

	Parámetros			*Resultados*	
Instancia	n	L	*Mejora (sec.)*	*Objetivo*	*Tiempo (sec.)*
I	200	3000	1	51 540	5.50
			60	49 530	67.26
			120	52 710	121.02
			180	52 180	198.10
			240	53 830	246.15
J	400	4000	1	99 020	347.29
			360	105 590	679.57
			720	107 050	761.38
			1080	107 320	1 476.11
K	400	4000	1	91 350	380.54
			420	90 720	658.23
			840	89 650	1 104.24
			1260	92 130	1 469.49
L	400	4000	1	92 390	153.47
			180	97 350	227.00
			360	97 290	383.49
			540	99 440	584.64

Cuadro C.6: Continuación de resultados de ejecución para el algoritmo GRASP con multi-arranque.

C.5. Algoritmo GVNS

	Parámetros			*Resultados*	
Instancia	n	L	*Mejora (sec.)*	*Objetivo*	*Tiempo (sec.)*
A	50	1000	1	4 480	1.36
B	50	1000	1	4 030	1.27
C	50	1000	1	11 240	1.31
D	100	2000	1	29 520	1.59
			60	30 530	61.76
			120	30 520	120.76
			180	30 530	181.83
			240	30 520	242.35
E	100	2000	1	25 330	2.63
			60	25 550	62.64
			120	26 660	121.06
			180	26 550	182.34
			240	27 550	242.55
F	100	2000	1	23 910	3.22
			60	26 820	61.26
			120	27 010	121.56
			180	27 030	182.70
			240	27 010	242.80
G	200	3000	1	51 930	25.55
			60	54 400	87.44
			120	53 670	148.22
			180	53 950	208.01
			240	55 920	249.41
H	200	3000	1	56 990	57.06
			60	61 060	91.19
			120	60 730	131.48
			180	60 610	215.69
			240	61 500	305.48

Cuadro C.7: Resultados de ejecución para el algoritmo GVNS.

	Parámetros			*Resultados*	
Instancia	n	L	*Mejora (sec.)*	*Objetivo*	*Tiempo (sec.)*
I	200	3000	1	50 650	27.01
			60	51 550	76.48
			120	53 730	159.38
			180	52 150	221.67
			240	52 640	273.25
J	400	4000	1	99 530	460.51
			480	94 640	810.07
			960	102 250	1 293.95
			1440	108 830	1 762.66
K	400	4000	1	89 990	666.32
			700	87 910	922.23
			1400	91 870	1 624.76
			2100	94 600	2 874.96
L	400	4000	1	97 280	266.02
			300	97 130	773.98
			900	94 560	1 308.57
			1500	101 310	1 840.98

Cuadro C.8: Continuación de resultados de ejecución para el algoritmo GVNS.

Apéndice D

Códigos

D.1. Repositorio Github

Se muestran los códigos del proyecto en este repositorio.

Bibliografía

[1] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Princeton University, 2007.

[2] Sinaide Nunes Bezerra, Sérgio Ricardo de Souza, and Marcone Jamilson Freitas Souza. A gvns algorithm for solving the multi-depot vehicle routing problem. *Electronic Notes in Discrete Mathematics*, 66:167–174, 2018. 5th International Conference on Variable Neighborhood Search.

[3] J. Castañeda, M. Neroni, M. Ammouriova, J. Panadero, and Á. Juan. Biased-randomized discrete-event heuristics for dynamic optimization with time dependencies and synchronization. *Algorithms*, 15(289), 08 2022.

[4] Pierre Collet and Jean-Philippe Rennard. Stochastic optimization algorithms. In Vijayan Sugumaran, editor, *Intelligent Information Technologies: Concepts, Methodologies, Tools, and Applications*, pages 1121–1137. IGI Global, 2008.

[5] Stephen A. Cook. The complexity of theorem-proving procedures. *Proceedings of the Third Annual ACM Symposium on Theory of Computing (STOC)*, pages 151–158, 1971.

[6] Michael Drexl. Synchronization in vehicle routing—a survey of vrps with multiple synchronization constraints. *Transportation Science*, 46:297–316, 08 2012.

[7] Thomas Feo and Mauricio Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 03 1995.

[8] P. Festa. A brief introduction to exact, approximation, and heuristic algorithms for solving hard combinatorial optimization problems. In *2014 16th International Conference on Transparent Optical Networks (ICTON)*, pages 1–20, 2014.

[9] Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. Springer-Verlag, Berlin, Heidelberg, 1st edition, 2010.

[10] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA, 1990.

[11] Fred Glover. Tabu search—part i. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[12] Aldy Gunawan, Hoong Chuin Lau, and Pieter Vansteenwegen. Orienteering problem: A survey of recent variants, solution approaches and applications. *European Journal of Operational Research*, 255(2):315–332, 2016.

[13] Pierre Hansen, Nenad Mladenovic, and José Moreno-Pérez. Variable neighbourhood search: Methods and applications. *4OR*, 175:367–407, 02 2010.

[14] Leonid A. Levin. Universal sequential search problems. *Problems of Information Transmission*, 9(3):115–116, 1973.

[15] Zoltan Mann. The top eight misconceptions about np-hardness. *Computer*, 50:72–79, 05 2017.

[16] Nenad Mladenovic. A tutorial on variable neighborhood search. Technical report, GERAD, University of Montreal, 2003.

[17] R. Pérez Aguila. *Una introducción a las matemáticas discretas y teoría de grafos*. El Cid Editor, 2013.

[18] Mauricio Resende, Celso Ribeiro, Fred Glover, and Rafael Marti. *Scatter Search and Path-Relinking: Fundamentals, Advances, and Applications*, pages 87–107. 09 2010.

[19] J. Riera-Ledesma and J.-J. Salazar-González. Selective routing problem with synchronization. *Computers & Operations Research*, 135:105465, 2021.

[20] M. Sipser. *Introduction to the Theory of Computation*. Course Technology, 2013.

[21] P. Vansteenwegen, W. Souffriau, and D. Oudheusden. The orienteering problem: A survey. *European Journal of Operational Research*, 209(1):1–10, 2011.

[22] Christos Voudouris and Edward P. K. Tsang. *Guided Local Search*, pages 185–218. Springer US, Boston, MA, 2003.

[23] Yizhun Wang. Review on greedy algorithm. *Theoretical and Natural Science*, 14:233–239, 11 2023.

[24] D. P. Williamson and D. B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, Cambridge, 2011.