Hiperheurísticas: Aplicación a problemas de asignación de horario y metaoptimización

Trabajo de Fin de Grado

Curso 2022-2023

UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE CIENCIAS MATEMÁTICAS

Grado en Matemáticas

José Ignacio Alba Rodríguez

Tutor: Fernando Rubio Diez

Madrid, a 28 de junio de 2023

Resumen

El campo de las hiperheurísticas en una rama de la Inteligencia Artificial que está en auge debido a su capacidad de obtener una gran generalidad y a la forma en la que trabaja sobre otros algoritmos. En este trabajo se realiza una investigación sobre las hiperheurísticas y sus aplicaciones. Algunas de estas son implementadas para resolver problemas de asignación de horarios de la competición ITC2007 y para investigar los principios de la metaoptimización, para lo cual será necesario aplicar técnicas de la optimización multiobjetivo.

Palabras clave

Hiperheurísticas, optimización, asignación de horarios, ITC2007, metaoptimización, optimización multiobjetivo, Inteligencia Artificial.

Abstract

Hiperheuristics research is a field within Artificial Intelligence which is rapidly growing due to its ability to achieve great generality and the way it works among other algorithms. In this paper, research is conducted on hyper-heuristics and their applications. Some of these are implemented to solve time-tabling problems from the ITC2007 competition and to investigate the principles of metaoptimization, where multi-objective optimization techniques are to be applied in order to achieve off-line learning.

Keywords

Hiper-heuristics, optimization, Time-Tabling, ITC2007, metaoptimization, multi-objective optimization, Artificial Intelligence.

Índice general

Capítulo 1

Introducción y estructura del trabajo

Debido a la imposibilidad de encontrar algoritmos eficientes para la mayoría de problemas de optimización o de carácter combinatorio que surgen del mundo real, nace la necesidad de desarrollar técnicas que permitan obtener soluciones aceptables en un tiempo razonable. A raíz de esta necesidad, surge el campo de las metaheurísticas. Posteriormente, se trató de estudiar el desempeño de distintos algoritmos sobre un mismo problema. A menudo, cuando el problema cumple unas determinadas características, un algoritmo funciona mejor que otro. De este estudio, surge la idea de utilizar las propias técnicas de la heurística aplicadas a la selección del mejor algoritmo para cada situación. Esta idea da lugar al campo de las hiperheurísticas, que consisten en algoritmos que son capaces de trabajar sobre otros algoritmos heurísticos. Estas hiperheurísticas seleccionan o generan heurísticas apropiadas para el problema que se esté tratando. En el Capítulo 2, se realiza una introducción más precisa sobre este campo y se establece una clasificación de las diferentes hiperheurísticas que aparecen en la literatura. Posteriormente, se muestran algunas de las aplicaciones que tienen las hiperheurísticas.

En el Capítulo 3 se introduce una hiperheurística para resolver problemas de asignación de horarios. En los problemas de asignación de horarios o Time-Tabling se tiene una serie de eventos a los que se les busca asignar a un horario de forma que se cumplan un número de restricciones obligatorias y se minimice el coste de las restricciones blandas. Estos problemas tienen un gran interés debido a que existen muchas variaciones que son directamente aplicables a situaciones del mundo real, como por ejemplo: la asignación de eventos deportivos, la asignación de horarios para exámenes universitarios, la organización de los turnos de las enfermeras en un hospital, etc. Ante el gran número de variaciones que pueden ser consideradas, las hiperheurísticas son técnicas muy apropiadas para esta clase de problemas debido a la gran generalidad que pueden aportar. En este trabajo se va a explicar y desarrollar una hiperheurística capaz de resolver problemas de este tipo. Los problemas utilizados han sido obtenidos de la competición ITC2007, cuya página oficial puede ser accesible a través de la herramienta WayBackMachine. Utilizando estos datos aseguramos que los problemas que están siendo evaluados tengan un cierto nivel de dificultad, para posteriormente resolverlos mediante el uso de las hiperheurísticas.

En el Capítulo 4 se estudian los conceptos de metaoptimización y optimización multiobjetivo. El objetivo de este capítulo es implementar una hiperheurística que seleccione

los mejores coeficientes para Particle Swarm Optimization (PSO) en función de su tiempo máximo de ejecución. Esto se realizará por medio del aprendizaje Offline sobre diferentes funciones de entrenamiento. Gracias a este tipo de aprendizaje, los coeficientes obtenidos permitirán garantizar la convergencia de la metaheurística al aplicarla en problemas sobre los que no ha sido entrenada. Debido a la necesidad de aplicar múltiples funciones de entrenamiento, ha sido necesario explorar las técnicas de la optimización multiobjetivo.

Finalmente, se introducen una serie de futuras líneas de investigación para explorar nuevas aplicaciones de las hiperheurísticas, así como su combinación con redes neuronales para obtener algoritmos de inteligencia artificial mucho más potentes. Todas las implementaciones de las metaheurísticas e hiperheurísticas explicadas, así como los resultados de estas, pueden ser encontradas en el siguiente repositorio de GitHub.

Capítulo 2

Definiciones sobre las Hiperheurísticas

El objetivo de este capítulo es introducir los distintos conceptos de heurística, metaheurística e hiperheurística tomando como base los artículos [3–5,15,28]. Finalmente, se introduce la clasificación que aparece en [5] para ampliar sobre las distintas clases de hiperheurísticas que aparecen en la literatura y cómo pueden ser aplicadas.

2.1. Introducción a las heurísticas

Una de las principales aplicaciones de las Matemáticas consiste en encontrar respuesta a problemas que, a menudo, tienden a estar relacionados con encontrar el mínimo de una función o encontrar la mejor forma de escoger unas variables. Estos problemas pueden aparecer en ámbitos muy distintos, desde el mundo financiero, el diseño industrial o el transporte de mercancías hasta la planificación de rutinas o eventos. Debido a las múltiples aplicaciones reales que tienen estos problemas, resulta interesante desarrollar algoritmos que sean capaces de resolver cualquier instancia de estos.

Sin embargo, a pesar de que pueda existir una solución óptima (por ejemplo si el conjunto de soluciones es finito), existen problemas para los cuales parece ser imposible encontrar un algoritmo que encuentre dicha solución en un tiempo razonable. Esto ocurre para la mayoría de ejemplos que surgen del mundo real. En general, encontrar la mejor solución posible requerirá de realizar una exploración exhaustiva sobre todo el espacio de soluciones factibles. El carácter combinatorio de dicho espacio puede provocar que la cantidad de tiempo requerido para comprobar todas las posibilidades sea demasiado grande, y por tanto, no será posible realizar esta búsqueda en problemas de gran escala.

Para estos casos se suele recurrir a algoritmos heurísticos. Una heurística es un método o estrategia que permite guiar la búsqueda de soluciones mediante reglas basadas en la intuición y la experiencia, con el objetivo de encontrar una solución aproximada sin necesidad de realizar una búsqueda exhaustiva completa. Mediante estos algoritmos es posible encontrar soluciones aceptables para ser tomadas como una aproximación, de forma que, a cambio de sacrificar la garantía de encontrar la solución óptima, estos se ejecutan en una cantidad de tiempo mucho menor. Y aunque existirán mejores soluciones que la encontrada por la heurística, para muchos de los problemas del mundo real, basta con encontrar una solución

"suficientemente buena, suficientemente barata y lo mas rápido posible"[4]. Si la diferencia entre la solución encontrada por la heurística y la óptima no es demasiado grande, la pérdida por no estar usando la solución óptima es mucho menor que el coste en recursos que llevaría encontrarla (si es que fuera posible), por lo que estas metodologías son ampliamente utilizadas en una multitud de ámbitos distintos.

2.2. Metaheurísticas

Los primeros algoritmos heurísticos nacen de reglas intuitivas basadas en la experiencia sobre el problema que pretenden resolver. Sin embargo, esta intuición no suele ser extrapolable a distintos problemas. Por esta razón, los algoritmos de este tipo solo sirven para el caso específico para el cual han sido diseñados. Esto hace que encontrar heurísticas buenas tenga un menor interés desde el punto de vista de la investigación, debido a su falta de generalidad.

Para intentar resolver esta cuestión, nacieron nuevas propuestas que trataban de crear algoritmos heurísticos capaces de abstraer el problema que buscaban resolver. Esta abstracción normalmente se realiza representando las soluciones como objetos matemáticos y reduciendo el problema mediante una función objetivo con una serie de restricciones. Esta modelización "esconde" la naturaleza del problema detrás de una "caja negra", de forma que un algoritmo que sea capaz de actuar sobre esta "caja negra" será también aplicable a cualquier problema que admita la misma abstracción. Este tipo de algoritmos fueron denominados como metaheurísticas por primera vez en 1986. Este nombre deriva de la composición de los términos griegos *heurisken*, que significa "encontrar" y *meta*, que significa "más allá". Antes de la adopción de este término, solían ser denominados "heurísticas modernas"[3]. Sin embargo, el término metaheurística ha sido ampliamente aceptado y su campo de estudio ha evolucionado mucho en los últimos años.

En [3], se presentan una serie de definiciones sobre el término *metaheurística* empleadas por distintos autores, destacando la siguiente definición: *"A metaheuristic is a set of concepts that can be used to define heuristic methods that can be applied to a wide set of different problems. In other words, a metaheuristic can be seen as a general algorithmic framework which can be applied to different optimization problems with relatively few modifications to make them adapted to a specific problem."* Aunque no existe consenso sobre una definición precisa de las metaheurísticas, la mayor parte de los autores coincide en destacar las siguientes características:

- Una metaheurística es un esquema algorítmico genérico que abstrae el problema, por lo que es independiente de este y es, por tanto, aplicable a distintos problemas.
- Las metaheurísticas constan de estrategias cuyo objetivo es guiar el proceso de búsqueda hacia soluciones cercanas a la óptima.
- Son métodos aproximativos y, en general, no deterministas.
- Cuentan con mecanismos (generalmente estocásticos) que les permiten evitar quedar atrapados en mínimos locales.

- Las estrategias que emplean pueden constar de búsquedas locales, técnicas de aprendizaje, uso de memoria y hasta de un conjunto de heurísticas más específicas controladas por una estrategia de alto nivel.

Este campo ha evolucionado mucho en los últimos años y ha sido aplicado a múltiples disciplinas científicas, a la optimización de recursos de empresas e incluso a la propia informática. Debido a la evolución y a las aplicaciones de las metaheurísticas en los últimos años, el número de metaheurísticas distintas y variaciones de las mismas ha crecido de forma exponencial. Por esta razón, se buscan formas de clasificar las distintas metaheurísticas, algunas de las categorías más aceptadas son las siguientes:

- **Según la fuente de inspiración.** La mayor parte de las metaheurísticas surgen de la inspiración en la simulación de eventos de la naturaleza estudiados por otras disciplinas de la Ciencia, como por ejemplo de la Psicología (algoritmos de aprendizaje y redes de neuronas), la Termología (algoritmos de recocido simulado), la Geología (algoritmos basados en la formación de ríos) y principalmente de la Biología (algoritmos genéticos, Particle Swarm Optimization, Cuckoo Search, Ant Colony Optimization...). Por otro lado, también existen algoritmos que no están basados en la naturaleza y se aprovechan de conceptos estadísticos u otras técnicas algorítmicas como "divide y vencerás" o algoritmos voraces, por ejemplo la búsqueda tabú o Iterated Local Search.

- **Según el número de soluciones empleadas.** Se distingue entre: las metaheurísticas de trayectoria, que toman una única solución, la cual se desplaza guiada por una estrategia describiendo una trayectoria de donde se obtiene la mejor solución encontrada, como Tabu Search o Simmulated Annaealing; y las metaheurísticas poblacionales, que parten de un conjunto amplio de soluciones, el cual es evolucionado, como Particle Swarm Optimization (PSO) o los algoritmos genéticos.

- **Según el determinismo.** Las metaheurísticas se pueden clasificar como deterministas, es decir, dados unos mismos parámetros de entrada, se devuelve siempre la misma solución; o estocásticas (no deterministas), si tienen mecanismos basados en la aleatoriedad que provocan que se obtengan resultados distintos para ejecuciones con los mismos parámetros. Como ya se mencionó anteriormente, la mayor parte las metaheurísticas pertenecen a este segundo grupo.

- **Según el uso de memoria y técnicas de aprendizaje.** El uso de la memoria y de técnicas de aprendizaje permiten que la metaheurística sea capaz de tomar decisiones basándose en la información de estados anteriores, mientras que las que no usan memoria se ven forzadas a tomar la siguiente decisión únicamente con la información de su estado actual. Las primeras metaheurísticas no solían usar memoria ni estrategias de aprendizaje, pero con la aparición de la Tabu Search, esto comenzó a ser más utilizado hasta el punto de que es una de las características esenciales de las metaheurísticas modernas.

- **Según el funcionamiento** Finalmente, existen dos grandes categorías de metaheurísticas: Constructivas o Iterativas. Las primeras constan de estrategias que parten de una solución "vacía" a la que se le van añadiendo nuevas componentes en cada paso hasta obtener una solución completa. A menudo tienen relación con algoritmos voraces y

tienen una terminación natural, que ocurre una vez la solución es completada. Por otro lado, las iterativas, también llamadas de búsqueda local o perturbativas, parten de una solución del problema y tratan de ir mejorándola aplicando "movimientos" de forma iterativa. Los movimientos consisten en cambios pequeños en la solución, de manera que se obtienen soluciones "cercanas", llamadas vecinas. Además, existen acercamientos híbridos, que combinan los dos tipos de funcionamiento, por ejemplo, generando una nueva solución mediante técnicas constructivas una vez que la estrategia iterativa es incapaz de mejorar la solución anterior.

Por definición, todas las metaheurísticas buscan una forma de reducir el tiempo empleado en encontrar una solución, y esto pasa por descartar gran parte de las posibles soluciones del espacio de búsqueda. Sin embargo, si descartamos demasiadas soluciones, a menudo descartaríamos también las soluciones más próximas a la óptima. Por ello, toda metaheurística debe mantener un equilibrio entre *diversificación* e *intensificación*. La intensificación consiste en tratar de encontrar la mejor solución dentro de un vecindario concreto, mientras que la diversificación se entiende como la capacidad de explorar nuevas áreas del espacio de búsqueda para tener la posibilidad de encontrar zonas con una "mayor calidad de soluciones" (generalmente por medios estadísticos). Un algoritmo muy centrado en la intensificación será muy rápido, pero a menudo encontrará peores soluciones al no haber aprovechado todo el espacio de búsqueda, mientras que un algoritmo centrado en la diversificación terminará explorando la práctica totalidad del espacio de búsqueda, encontrando mejores soluciones pero requiriendo casi tanto tiempo como la solución exacta. Por ello, un equilibrio entre estas dos cualidades será necesario para poder considerar una metaheurística como aceptable.

A modo de ejemplo, se explicará más en detalle el funcionamiento de algunos algoritmos metaheurísticos que serán empleados más adelante. Una implementación de cada uno de ellos puede ser encontrada en el repositorio de GitHub del trabajo.

2.2.1. Particle Swarm Optimization

La optimización por enjambres de partículas (PSO por sus siglas en inglés) es una metaheurística introducida en [11] en el año 1995. Es un algoritmo que buscaba simular el comportamiento de cada uno de los individuos en grupos sociales grandes como bandadas de pájaros o bancos de peces. Para ello, se considera una población de partículas, donde cada partícula representa una solución, que van desplazándose sobre el espacio de búsqueda intercambiando información entre ellas sobre las mejores posiciones encontradas. De esta forma, cada partícula recorre una trayectoria distinta, explorando diferentes zonas del espacio de búsqueda, a la vez que son atraídas hacia los mínimos encontrados por otras partículas.

El algoritmo tiene un conjunto P de partículas, donde cada partícula $p \in P$ tiene una posición $x_p = (x_p^1, x_p^2, \ldots, x_p^n) \in \mathbb{K}^n$, una velocidad $v_p = (v_p^1, v_p^2, \ldots, v_p^n) \in \mathbb{K}^n$ y un vecindario $N_p \subset P$, que es el conjunto de partículas con las que intercambia y recibe información. En la inicialización, la posición y la velocidad de cada partícula es determinada de forma aleatoria. Posteriormente, en cada iteración t, cada partícula se desplaza y actualiza su velocidad según las siguientes ecuaciones:

$$v^i_{p,t+1} = wv^i_{p,t} + \varphi_1 r_1 \left(b^i_{p,t} - x^i_{p,t}\right) + \varphi_2 r_2 \left(l^i_{p,t} - x^i_{p,t}\right) \quad \forall i = 1, 2, \ldots, n$$

$$x^{i+1}_{p,t} = x^i_{p,t} + v^i_{p,t} \quad \forall i = 1, 2, \ldots, n$$

donde r_1 y r_2 son valores aleatorios que siguen una distribución $U(0, 1)$ y se generan cada vez que se utilizan; w es el coeficiente de **inercia**, que indica la cantidad de velocidad del paso anterior que se conserva en la próxima velocidad; $b_{p,t} = (b^1_{p,t}, b^2_{p,t}, \ldots, b^n_{p,t})$ es la mejor solución encontrada por la partícula p en las t primeras iteraciones; φ_1 es el coeficiente **cognitivo**, que regula la influencia que tiene la mejor posición encontrada por la partícula en la nueva velocidad; $l_{p,t} = (l^1_{p,t}, l^2_{p,t}, \ldots, l^n_{p,t})$ es la mejor solución entre las encontradas por las partículas del vecindario N_p en las t primeras iteraciones, llamada "mejor posición local"; y φ_2 es el coeficiente **social**, que regula el peso que tiene la mejor posición local en la nueva velocidad.

Siguiendo estas ecuaciones, las partículas se desplazan por el espacio de búsqueda de forma que se ven atraídas hacia el mínimo que han encontrado durante su propia trayectoria y a su vez son también atraídas hacia el mínimo que ha encontrado alguna partícula de su vecindario. Estas desviaciones se multiplican por valores aleatorios en cada componente para permitir que la partícula se mueva sin seguir líneas rectas y reduciendo la distancia trasladada, que, junto a la gran cantidad de partículas que se consideran, contribuye a la diversificación. Además, si $w < 1$, cuanto más cerca estén las partículas de sus vecinas, más lento se desplazarán en cada iteración, de forma que se intensifica la búsqueda cuando se aproxima mucho a valores buenos para el problema, mientras que si $w > 1$, en general, las soluciones cada vez se mueven más rápido y no será posible llevar a cabo el proceso de intensificación, por lo tanto es recomendable utilizar valores pequeños de w. Más adelante, se estudiará la influencia de estos parámetros a través de las técnicas de metaoptimización, en la Capítulo 4.

Debido a la naturaleza de la metaheurística, esta es aplicable a problemas donde el espacio de soluciones es continuo. Sin embargo, también existen variaciones de este algoritmo que le permiten trabajar sobre espacios de soluciones discretos. A su vez, también se distinguen las variantes síncronas y asíncronas. En la primera, todas las partículas reciben la información de sus vecinas al mismo tiempo, después de que todas las partículas se hayan desplazado. En la segunda versión, cada partícula se mueve de forma independiente y actualiza su velocidad justo después de moverse, usando la información recibida de sus vecinas hasta ese momento. Esta segunda versión puede ser paralelizada en diferentes procesos, donde cada proceso corresponderá con una partícula, pero a cambio puede requerir de más iteraciones para alcanzar la misma calidad, ya que la información que se recibe de cada partícula es información sesgada, pues puden no haberse desplazado el mismo número de veces.

En el Algoritmo 1 se presenta el pseudocódigo de este algoritmo.

Algoritmo 1 Particle Swarm Optimization

for $i = 1, 2, \ldots, N$ **do**
  $p_i \leftarrow solucionAleatoria(limites)$
  $v_i \leftarrow velocidadAleatoria(p_i, limites)$
  $x_i \leftarrow f_{obj}(p_i)$
  $b_i \leftarrow p_i$
end for
$x_{global} \leftarrow \min_{i=1\ldots N}(x_i)$
while not$(condicionParada)$ **do**
  for $i = 1, 2, \ldots, N$ **do**
    $p_i \leftarrow p_i + v_i$
    if $f_{obj}(p_i) < x_i$ **then**
      $x_i \leftarrow f_{obj}(p_i)$
      $b_i \leftarrow p_i$
      $x_{global} \leftarrow min(x_{global}, x_i)$
    end if
  end for
  for $i = 1, 2, \ldots, N$ **do**
    $j \leftarrow arg\min_{i \in N_i}(x_i)$
    $r1,\ r2 \leftarrow U(0, 1)$
    $v_i \leftarrow wv_i + \varphi_1 r_1(b_i - p_i) + \varphi_2 r_2(b_j - p_i)$
  end for
end while

2.2.2. Algoritmos Genéticos

Los algoritmos genéticos fueron una nueva categoría de metaheurísticas introducidas en 1970, dando lugar a la programación evolutiva, una de las ramas más exitosas de la inteligencia artificial. Estos algoritmos toman como fuente de inspiración los procesos biológicos descritos por la teoría Darwinista de la evolución. En la naturaleza, cada especie es el resultado de innumerables mutaciones, donde aquellas que permiten una mayor supervivencia se transmiten a las nuevas generaciones, mientras que las mutaciones que tienen un efecto negativo son descartadas por medio de la selección natural. El resultado de este proceso evolutivo es una amplia diversidad de individuos especializados a las condiciones de su entorno.

Los algoritmos genéticos intentan simular este proceso de forma más simplificada que en la naturaleza. Para ello, se toma una población P de "cromosomas", la cual se busca evolucionar. Cada uno de estos cromosomas corresponde con una codificación de una solución del problema a tratar. Estas codificaciones dependerán del tipo de problema, por ejemplo, para el problema de la mochila, cada cromosoma vendrá dado por una cadena de ceros y unos donde un 1 en la posición i indica que el elemento e_i ha sido introducido en la mochila.

En cada paso del algoritmo, se producen los siguientes procesos:

- **Selección.** Se realiza una criba para elegir cuales serán los cromosomas que producirán una descendencia en los pasos posteriores. Esta criba se realiza de forma que

los cromosomas con un mejor valor de la función objetivo tengan más probabilidades de ser seleccionados. No siempre será lo mejor escoger las mejores soluciones, ya que si esto ocurre, a menudo sus descendientes estarán muy próximos y de esta forma el algoritmo puede acabar atrapado en un mínimo local. Por eso, añadir una componente de aleatoriedad a la selección puede ser positivo de cara a la diversificación. Existen diversas estrategias de selección, algunas de las más utilizadas son:

  - **Selección mediante un orden lineal.** La selección se realiza ordenando las soluciones según el valor de la función objetivo y tomando las k mejores soluciones.
  - **Selección por ruleta.** A cada cromosoma se le asocia una probabilidad según la calidad de su solución, de forma que las mejores soluciones tengan una mayor probabilidad de ser elegidas y posteriormente se escoge un número de cromosomas de forma aleatoria siguiendo estas probabilidades.
  - **Selección por torneos.** Los cromosomas son agrupados aleatoriamente en k subconjuntos llamados "torneos" y de cada torneo se escoge un número m de ganadores, que corresponderán con las soluciones que tengan un mayor valor de la función objetivo.

- **Cruce.** Representa la reproducción de dos cromosomas seleccionados. Los cromosomas son recombinados para generar nuevos cromosomas que comparten algunas de las características de sus progenitores. También existen diferentes estrategias de cruce, de las cuales algunas serán mejores para un tipo de problemas que otras. Algunos ejemplos son:

  - **Cruce por un punto.** Se determina un índice j de corte y se intercambian las secciones de los cromosomas a partir del dicho índice. De esta forma se obtienen dos nuevos cromosomas donde cada cual comparte las j primeras componentes con uno de los progenitores y el resto con el otro progenitor.
  - **Cruce Aritmético.** El nuevo cromosoma se obtiene mediante una suma ponderada de los progenitores, por ejemplo la media aritmética. Suele ser utilizado cuando el cromosoma toma valores numéricos reales.
  - **Cruce Uniforme.** Cada una de las componentes del nuevo cromosoma es seleccionada de forma aleatoria entre dicha componente de sus progenitores. Esta estrategia permite una mayor diversidad y es aplicable a todo tipo de cromosomas, pero puede llegar a producir un gran numero de soluciones idénticas si la probabilidad de mutación es baja.

- **Mutación.** Simulando las mutaciones que ocurren en la naturaleza, de forma aleatoria se introduce alguna modificación en un subconjunto pequeño de las partículas generadas. Esto permite ampliar la diversificación y alcanzar zonas que puedan no haber sido exploradas por las generaciones anteriores.

- **Reemplazo.** Se seleccionan los cromosomas que van a formar parte de la nueva generación. Estos pueden ser seleccionados de entre los generados en la fase de cruce y mutación, aunque también pueden tomarse alguno de los seleccionados de la generación anterior. En este segundo caso, se habla de elitismo, el cual consiste en tomar uno o

varios de las mejores cromosomas de la población anterior para garantizar que no haya un pérdida de calidad.

La población inicial puede tomarse de forma aleatoria, siguiendo una distribución uniforme sobre el espacio de búsqueda. En cada iteración del algoritmo se llevan a cabo estos pasos con el objetivo de ir mejorando las soluciones obtenidas. Cuando se cumpla un criterio de parada, por ejemplo un máximo de iteraciones, se devuelve la mejor solución encontrada hasta la fecha y ese será el resultado del algoritmo.

Debido a su gran adaptabilidad, estos algoritmos han sido empleados con éxito en un gran campo de problemas de distinta índole. Además, a menudo son utilizados como estrategias de alto nivel para diferentes hiperheurísticas o siendo combinados con otras técnicas de mayor complejidad, convirtiéndose así en la metaheurística más popular y estudiada. Un ejemplo de su utilización como estrategia de alto nivel se encuentra descrito en la Capítulo 4, aplicado a la metaoptimización.

2.2.3. Tabu Search

La búsqueda tabú es una metaheurística creada por Fred W. Glover en el año 1986 y posteriormente ampliada en el libro [16]. Esta supuso un gran avance para el campo de las metaheurísticas, ya que introdujo el uso de memoria adaptativa en estos algoritmos.

La búsqueda tabú es un método de optimización basado en la búsqueda local, donde se añaden estructuras de memoria que actúan como "prohibiciones" temporales para evitar que se vuelva a explorar zonas visitadas recientemente. Además de estas prohibiciones, se permite la posibilidad de aceptar soluciones que empeoren el resultado, con el objetivo de escapar de mínimos locales y aumentar la diversificación.

Se parte de una solución inicial $x \in X$, que puede ser obtenida aleatoriamente o mediante alguna heurística constructiva. Sobre esta solución se aplica un conjunto de movimientos, obteniendo así su vecindario $N(x) \subset X$. Los movimientos son alteraciones de la solución que tratan de encontrar soluciones cercanas o similares, pueden ser por ejemplo intercambiar el orden de dos elementos de una lista o cambiar el valor de uno, por ejemplo. Posteriormente, dependiendo del tipo de memoria que se esté utilizando, la búsqueda tabú modificará el vecindario obteniendo uno nuevo $N^*(x)$. Finalmente, se selecciona un $x' \in N^*(x)$ y se repite el proceso de forma iterativa sobre x' hasta que se cumpla un criterio de parada. La selección del x' puede realizarse mediante distintas estrategias, como *deepest descent*, que escoge como x' aquel que verifique $f(x') \leq f(y) \quad \forall y \in N^*(x)$.

El uso de la memoria más utilizado es la memoria a corto plazo. En este, se toma una lista tabú l_{tabu} de una longitud determinada. Llamamos tenencia tabú a esta longitud, que indica el número de iteraciones en las que descartaremos una solución que acaba de entrar en lista tabú. Los elementos seleccionados en cada iteración son introducidos en esta lista durante dicho número de iteraciones, permitiendo que la búsqueda encuentre nuevos vecindarios en vez de volver siempre a uno ya visitado. Una vez un elemento ya ha pasado ese número de itera-

ciones en la lista tabú, es eliminado de esta para dejar hueco a la próxima solución explorada. Así, en cada iteración, $N^*(x) = N(x) \setminus l_{tabu}$, y de este conjunto seleccionaremos el próximo x'.

Se distingue además entre memoria explicita y memoria atributiva. Si la memoria es explícita, la lista tabú guarda las soluciones explícitas que va a descartar En cambio, si la memoria es atributiva, la lista tabú guarda los cambios que ha habido entre la solución previa y la seleccionada. En este ultimo caso, se descartan de $N(x)$ los elementos que han sido obtenidos aplicando a la solución x alguno de los cambios que aparecen en la lista tabú.

Otra característica de esta metaheurística a considerar es el *criterio de aspiración*. Este es un criterio que nos permite considerar una solución a pesar de que esta aparezca en la lista tabú. Por tanto, actúa como una relajación de la lista. El criterio de aspiración más utilizado consiste en tomar un elemento de la lista tabú si el valor de la función objetivo es menor que el mínimo valor encontrado por la búsqueda tabú hasta ese momento. En ocasiones es interesante añadir este criterio para asegurarse de que no dejamos buenas soluciones sin evaluar. Suele ser más empleado cuando se está utilizando memoria atributiva o cuando el resultado de la función objetivo no es determinista (por ejemplo, si es una función con ruido o como veremos más adelante, si se utiliza sobre otras heurísticas).

El pseudocódigo para la búsqueda tabú con memoria explícita puede verse en el Algoritmo 2. Esta metaheurística será empleada como estrategia de alto nivel en el Capítulo 3.

Algoritmo 2 Tabu Search con memoria explícita

```
x ← solucionInicial()
mejorSolucion ← x
solucionActual ← x
listaTabu ← [x]
while not(condicionParada) do
    V ← vecindario(solucionActual, movimientos)
    V' ← [v ∈ V | v ∉ listaTabu ∨ criterioAspiracion(v)]
    solucionActual ← seleccion(V')
    if funObj(solucionActual) < funObj(mejorSolucion) then
        mejorSolucion ← solucionActual
    end if
    if length(listaTabu) == tenenciaTabu then
        listaTabu.remove(0)
    end if
    listaTabu.add(solucionActual)
end while
```

2.3. Hiperheurísticas

Para cualquier clase de problemas, existe una gran variedad de heurísticas que se pueden desarrollar, teniendo cada una de ellas una serie de puntos fuertes y puntos débiles. Depen-

diendo de los datos del problema que se está tratando de resolver, puede ocurrir que una heurística, en general, aporte mejores resultados que otra. Esto se ejemplifica en [4], donde se analizan dos heurísticas diferentes para problemas de Bin-Packing. Se observa que una de ellas es capaz de encontrar soluciones muy buenas siempre y cuando la relación entre el tamaño de los objetos que queremos guardar y la capacidad de las papeleras sea grande. Sin embargo, para los problemas que tienen muchos objetos muy pequeños a guardar, las soluciones que encuentra dejan bastante que desear. Mientras tanto, la otra heurística se comporta al revés, es capaz de encontrar mejores soluciones cuando esta relación es pequeña. A raíz de este análisis, se plantea la posibilidad de crear un algoritmo que fuese capaz de identificar el tipo de problema a tratar de forma automática. De esta forma, el propio algoritmo ejecutaría la primera heurística en los casos en los que el problema tuviese objetos de un tamaño relativamente grande respecto a la capacidad de cada basura, mientras que si esto no ocurriese, ejecutaría la segunda heurística.

A partir de esta idea surge el concepto de *hiperheurísticas*. En un primer momento, estas fueron definidas como "heurísticas (o metaheurísticas) capaces de elegir la mejor heurística (o metaheurística) para el problema que se está tratando de resolver"[4]. En vez de trabajar sobre el espacio de búsqueda de las soluciones del problema, una hiperheurística consistiría de una heurística de alto nivel que trabaja sobre un espacio de búsqueda de heurísticas de más bajo nivel, con el objetivo de encontrar la que mejor se adapte al problema a tratar.

Según esta definición, las hiperheurísticas son a su vez un tipo de metaheurística, pero que opera sobre un espacio de búsqueda distinto. Un ejemplo de esta diferencia puede encontrarse en las aplicaciones de los algoritmos genéticos (descritos en la Sección 2.2.2). Una aplicación directa de un algoritmo genético para el problema de la mochila puede consistir en tomar una población de soluciones, expresadas como una lista de valores booleanos donde el i-ésimo elemento representa si introducimos o no el i-ésimo objeto en la mochila. El algoritmo evolucionará una población de vectores solución hasta obtener una solución aceptable. Por otro lado, una aplicación de los algoritmos genéticos como una hiperheurística podría consistir en tomar una población formada por heurísticas constructivas. En este segundo caso, estaríamos evolucionando esta población de heurísticas en vez de trabajar directamente sobre el espacio de soluciones, y acabaríamos obteniendo una población de nuevas heurísticas mejoradas. Posteriormente podemos seleccionar la mejor de esta población para hallar una solución.

Más adelante, debido a la evolución que ha experimentado este campo, los mismos autores propusieron una nueva definición más general en [5]: "*Una hiperheurística es un método automatizado a través del cual se pueden generar o seleccionar heurísticas para resolver un problema de búsqueda computacional.*" Con esta nueva definición se acepta que la estrategia de alto nivel pueda no ser necesariamente una heurística, ya que existen propuestas que basan esta toma de decisiones en redes neuronales u otro tipo de inteligencias artificiales. Además, el espacio de búsqueda puede estar formado no solo de heurísticas, si no también de sus componentes más esenciales, donde en estos casos la estrategia de alto nivel funcionará combinando estas componentes para obtener nuevas heurísticas que no seríamos capaces de encontrar con los mecanismos de diseño basados en intuición. Algunos ejemplos de artículos donde se proponen este tipo de combinaciones son: [1], donde se utiliza una red neuronal como estrategia de alto nivel para generar heurísticas constructivas; [18], donde se emplean redes

neuronales para acompañar a la estrategia de alto nivel y reducir el coste computacional de la hiperheurística; y [13], donde se aplica una hiperheurística basada en algoritmos genéticos sobre una *Stacked Neural Network*, es decir, una combinación de redes neuronales que se complementan entre sí, para tratar de resolver el problema multiobjetivo de optimizar los pesos asociados a las pilas.

Con el objetivo de explicar cuáles son las capacidades y el funcionamiento de las hiperheurísticas, se introduce la siguiente clasificación. Cabe destacar que es posible que sea necesario reformular esta clasificación para adaptarla a las nuevas técnicas que sean desarrolladas en el futuro. La clasificación presentada en [5] consta de dos categorías, que son las siguientes:

Según el tipo de aprendizaje

1. **Sin aprendizaje:** Se considera que una hiperheurística tiene aprendizaje si la estrategia de alto nivel es capaz de recolectar información del desempeño de las heurísticas de bajo nivel y guardar algún tipo de registro que permita modificarlas o seleccionarlas en base a resultados anteriores. Esta primera clase está formada por los algoritmos hiperheurísticos que no tienen ningún tipo de aprendizaje.

2. **Aprendizaje Online:** Una hiperheurística tiene un aprendizaje Online si es capaz de recopilar la información durante su ejecución sobre un problema. El objetivo del aprendizaje consiste en ir adaptando la estrategia para que se adapte al problema que se está tratando de la mejor forma posible, por lo que este aprendizaje solo servirá para el problema en cuestión. La heurística obtenida finalmente estará muy especializada para una única instancia del problema, por lo que se puede utilizar para encontrar una solución a este. También se les suele llamar "de usar y tirar", ya que una vez se ha resuelto la instancia, no tiene sentido intentar aplicarla para un problema en el que no ha sido especializada.

3. **Aprendizaje Offline:** Una hiperheurística tiene un aprendizaje Offline si el aprendizaje tiene lugar después de su ejecución sobre un problema. El objetivo de este tipo de aprendizaje es realizarlo a través de varios problemas de entrenamiento, de forma que se obtenga una heurística que pueda ser utilizada sobre instancias nunca vistas antes. El conjunto de problemas de entrenamiento debe estar cuidadosamente seleccionado para no sesgar el algoritmo sobre los problemas con los que ha sido entrenado. Debido a que la heurística obtenida como resultado es aplicable a distintas instancias del problema, normalmente son llamadas "reutilizables".

Según el funcionamiento de la estrategia de alto nivel

1. **Selección:** Una hiperheurística de selección consiste de una estrategia de alto nivel que en cada iteración debe elegir una heurística de bajo nivel y aplicarla sobre una solución. La estrategia puede realizar la elección en función de la información que disponga, tanto del rendimiento anterior, como por ejemplo del tiempo que lleva una heurística sin ser aplicada. El objetivo de esta puede ser aplicar una nueva heurística para determinar cuál es mejor, intentar mejorar la solución o intentar escapar de mínimos locales. A su vez, pueden distinguirse dos subcategorías dependiendo del funcionamiento de las heurísticas de bajo nivel:

a) **Sobre constructivas:** Si las heurísticas de bajo nivel son constructivas, entonces la hiperheurística de selección sobre constructivas partirá de una solución vacía (o varias). En cada paso, selecciona una heurística constructiva y ejecuta un paso de esta (no necesariamente debe completar la solución). Posteriormente, se seleccionará otra heurística constructiva y se repetirá el proceso hasta alcanzar una solución completa, generando una cadena. Una vez esta es alcanzada, se evalúa el rendimiento de la cadena de heurísticas seleccionada y se comienza de nuevo para tratar de encontrar la mejor cadena. Finalmente, esa cadena servirá como una heurística que podemos aplicar para encontrar una solución sin necesidad de partir de una solución previa. Es esperable que la solución obtenida tenga mayor calidad que las que se obtendrían aplicando directamente una de las heurísticas del conjunto base. Posteriormente podemos tomar la solución generada por esta cadena como el resultado final o, en cambio, se pueden aplicar heurísticas perturbativas sobre ella que traten de mejorarla, de manera que se obtenga una heurística híbrida.

b) **Sobre iterativas:** En este caso, se comenzará desde una solución completa. El objetivo es aplicar en cada iteración alguna de las heurísticas de bajo nivel con el objetivo de ir mejorando la solución iterativamente. La estrategia de selección consta de dos pasos: en el primer paso se selecciona una heurística de bajo nivel en base a la información que contiene la estrategia y posteriormente se aplica a la solución, después, en el segundo paso la estrategia decide si se conserva o no la nueva solución e incorpora la información sobre el desempeño de la última heurística utilizada para ir mejorando la selección en los pasos futuros (en caso de contar con algún tipo de aprendizaje). En ocasiones nos interesará conservar la nueva solución a pesar de que el resultado de la función objetivo haya empeorado. Esto se debe a que mediante la aceptación de resultados peores será posible escapar de mínimos locales y además nos permitirá explorar una parte mayor del espacio de búsqueda de heurísticas.

2. **Generación:** Estas hiperheurísticas tienen el objetivo de generar nuevas heurísticas de forma automática que sean apropiadas para el problema en cuestión. La mayor parte de las hiperheurísticas de este tipo utilizan algoritmos genéticos como estrategia de alto nivel. A partir de una serie de componentes de heurísticas conocidas, se codifican unas nuevas heurísticas como genes. Cada gen corresponderá, por tanto, con un nuevo algoritmo que combina las componentes del conjunto base. Esta modelización permite evolucionar los genes, obteniendo finalmente un conjunto de heurísticas adecuadas para el problema. Debido a este proceso de generación, es posible considerar nuevas heurísticas que no sería posible generar a través de procesos intuitivos por ningún ser humano. En general, estas hiperheurísticas sirven para generar algoritmos que puedan ser usados en distintos problemas, por lo que suelen ser de aprendizaje Off-Line.

   a) **Sobre constructivas:** Si el conjunto de heurísticas (o componentes de estas) son constructivas, la estrategia de alto nivel generará un conjunto de nuevas heurísticas también constructivas, que ejecutará para evaluarlas a partir de la solución final que generen. Finalmente, se quedará con la que mejores resultados ofrezca. Esta heurística nueva es esperable que genere mejores resultados que las que se

han dado como conjunto base, por lo que puede ser aplicada para obtener una solución factible a partir de una solución vacía o incluso pueden generarse varias heurísticas distintas para ser utilizadas dentro de otra hiperheurística de selección.

b) **Sobre iterativas:** Si el conjunto de heurísticas o componentes es de perturbativas, el objetivo de la estrategia de alto nivel será ir evolucionando la población de estas, evaluándolas según la mejora o el cambio que produzcan sobre varias soluciones. En cada iteración tratará de ir evolucionándolas, conservando las que producen mayores cambios positivos. Finalmente se obtendrá una o varias heurísticas perturbativas que pueden ser aplicadas directamente sobre una solución para mejorarla, o como conjunto base de una hiperheurística de selección u otras estructuras de inteligencia artificial.

Combinando estas estrategias, es posible crear también hiperheurísticas híbridas. Por ejemplo, estas pueden utilizar hiperheurísticas de generación para alimentar el conjunto base y después aplicar una de selección con un conjunto de heurísticas mejoradas. A su vez, el mayor nivel de abstracción permite que el mismo esquema sea reutilizable para distintos problemas tan solo cambiando el conjunto base de heurísticas y la representación del problema. Para esto último es necesario que la estrategia de alto nivel no dependa de la estructura del problema y sea capaz de funcionar en base únicamente al desempeño de las heurísticas de bajo nivel sobre las que trabaja. Esto permite alcanzar el objetivo de la generalidad con el que nacieron las metaheurísticas y desarrollar metodologías más amplias que puedan ser estudiadas desde distintos ángulos.

Debido a su naturaleza, la mayor parte de hiperheurísticas de selección tienden a tener un aprendizaje Online, ya que la selección de las heurísticas más apropiadas variará mucho según el problema. A su vez, las de generación tenderán a ser Offline, pues su objetivo es crear heurísticas que puedan ser aplicables a cualquier instancia. Sin embargo, existen ejemplos en la literatura de las distintas combinaciones. Como norma general, las heurísticas Offline serán más rápidas (una vez ya hayan sido entrenadas), mientras que es de esperar que las que tengan un aprendizaje Online obtengan un mejor resultado, ya que no necesitan obtener un método general, si no que estará especializado para un único problema.

En los capítulos posteriores se presentan algoritmos hiperheurísticos aplicados a la resolución de problemas de TimeTabling y a la metaoptimización.

Capítulo 3

Hiperheurística basada en Coloración de Grafos para problemas de TimeTabling

En este capítulo se va a presentar una hiperheurística para resolver problemas de Timetabling o asignación de horarios mediante una modelización como problemas de coloración de grafos. La hiperheurística utilizada consistirá en una Búsqueda Tabú sobre un conjunto ampliamente estudiado de heurísticas constructivas de coloración de grafos introducidas en [8]. Esta hiperheurística ha sido sujeto de investigación en múltiples artículos, como [6,7,23,24]. Finalmente, se utilizará para resolver problemas de la competición ITC2007, cuyo formato detallado puede encontrarse en [19].

3.1. Problema de Timetabling General

El problema de Timetabling es un problema con múltiples variaciones que puede ser aplicado a prácticamente cualquier área del mundo real, como en la organización de eventos deportivos, la colocación de exámenes o cursos universitarios, la distribución de tareas de los enfermeros de un hospital, etc.

Este problema consta de un conjunto de eventos que deben ser asignados a un grupo de horarios específico. Las asignaciones deben estar sujetas a una serie de restricciones, que se clasifican en dos tipos:

- **Restricciones Obligatorias:** Son una serie de restricciones que deben cumplirse ante cualquier circunstancia y delimitan el espacio de búsqueda de soluciones factibles. Solo consideraremos como válidas las soluciones que respeten todas estas restricciones. Permiten modelizar situaciones como las siguientes: Un profesor no puede estar en dos exámenes al mismo tiempo. Dos partidos no pueden realizarse en el mismo estadio al mismo tiempo.

- **Restricciones Blandas:** Son una serie de restricciones que preferiblemente deben cumplirse, siempre y cuando sea posible. En caso de no ser respetadas, añadirán una penalización que será evaluada por la función objetivo para intentar determinar cuál es la mejor solución. Permiten modelizar preferencias no esenciales, por ejemplo: Intentar

evitar que un alumno tenga dos exámenes en días consecutivos. Evitar que un par de partidos de alto riesgo tengan lugar en la misma ciudad.

Este tipo de problemas pertenece a la clase NP-duro y, debido a su carácter combinatorio, los algoritmos exactos requerirán de demasiado tiempo para explorar todas las posibles asignaciones hasta encontrar la mejor. Además, como fue mencionado anteriormente, existe un gran número de variaciones, que dependen en gran medida del carácter de las posibles restricciones que añadamos al problema, así como del tipo de asignaciones, por ejemplo asignar a cada examen un horario, una sala y un inspector, en vez de solo los horarios. Por esta razón, este problema es de los más indicados para ser resuelto mediante hiperheurísticas, ya que la generalidad que estas aportan será clave para extender el marco del algoritmo a diferentes variaciones del problema.

En concreto, se estudiará una versión de asignación de horarios para exámenes universitarios. Esta variante fue inicialmente estudiada en [8] con conjuntos de datos tomados a partir de universidades reales, principalmente de Canadá. Los conjuntos de datos tomados en este artículo se convirtieron en un estándar para este tipo de problemas debido a su directa aplicación al mundo real. Sin embargo, son problemas con un pequeño número de restricciones. Por ello, para este trabajo se ha tratado de tomar un conjunto de datos con una mayor dificultad. Finalmente, el conjunto seleccionado fue el proporcionado en la competición ITC2007, que se detalla a continuación.

3.2. Caso de estudio: ITC2007

En esta sección se va a introducir el tipo concreto de problemas contra el cual se va a enfrentar la hiperheurística desarrollada en la Sección 3.4. El modelo de problema elegido es el que fue utilizado en la competición ITC2007[19] (International Timetabling Competition). Esta fue la segunda edición de la competición organizada por PATAT Conference (Practice and Theory of Automated Timetabling). Este es un grupo cuyo objetivo es promover la investigación y automatización de esta clase de problemas, para lo que realizan conferencias y competiciones. La información sobre esta competición puede encontrarse en su página oficial, aunque debido a la antigüedad y falta de mantenimiento, la página puede no ser accesible a día de hoy. Para poder acceder a versiones antiguas es posible utilizar The Way Back Machine mediante el siguiente enlace. En cualquier caso, dejaré subido a mi repositorio de GitHub los datos utilizados para facilitar su acceso. Aquí también se puede encontrar la implementación de esta hiperheurística y los resultados obtenidos.

3.2.1. Datos del problema

En este caso, los datos del problema vienen dados por:

- un conjunto E de exámenes, donde cada examen $e \in E$ tiene una duración determinada y un conjunto de alumnos S_e que deben presentarse al examen e.
- un conjunto S de estudiantes, donde cada estudiante debe participar en una serie de exámenes.

- un conjunto limitado H de horarios, donde cada horario $h \in H$ está fijado a un día y una hora concreta, con una duración fija.

- un conjunto R de salas donde puede ser alojado más de un examen al mismo tiempo, y cada sala cuenta con una capacidad máxima.

El objetivo del problema es encontrar una asignación $a : E \longrightarrow H \times R$ de manera que cada examen quede asignado en un horario a una sala. La asignación a no tiene por que ser necesariamente inyectiva, esto es, se permite la realización de dos exámenes distintos simultáneamente en el mismo aula. La asignación debe respetar todas las restricciones obligatorias y se busca una asignación que minimice el coste asociado a las restricciones blandas que no sean respetadas.

3.2.2. Restricciones Obligatorias

Toda solución válida del problema debe verificar las siguientes restricciones obligatorias:

- Cada estudiante debe de poder acceder a todos sus exámenes, por tanto, ningún estudiante puede tener dos exámenes asignados al mismo tiempo.

- La longitud de los periodos debe ser respetada, es decir, los exámenes no podrán ser asignados a horarios que tengan una menor duración que el examen.

- La capacidad de cada sala no debe ser superada en ningún momento durante un horario específico.

  Estas restricciones son comunes a todas las instancias de los problemas. Además de estas, se pueden añadir las siguientes restricciones que vienen determinadas por cada instancia del problema y están relacionadas con el uso concreto de algunas salas u horarios:

- Restricciones extra asociadas a los horarios:

  - Puede requerirse que un examen e_1 sea asignado estrictamente después que otro examen e_2.
  - Puede requerirse que dos exámenes e_1 y e_2 sean siempre asignados al mismo horario, aunque no necesariamente a las mismas salas.
  - Puede requerirse que dos exámenes e_1 y e_2 sean asignados a periodos distintos.

- Restricciones extra asociadas a las salas:

  - Puede requerirse que un examen e_1 sea realizado en una sala exclusiva, es decir, ningún otro examen puede ser asignado a la misma sala en el mismo horario.

3.2.3. Restricciones Blandas

El objetivo es que la solución minimice el coste asociado a incumplir las siguientes restricciones. Puede que no exista ninguna solución que respete todas las restricciones asociadas al mismo tiempo, por lo que estas restricciones pueden ser violadas si no queda otra opción. Cada una de estas restricciones tiene un coste asociado que depende de cada instancia del problema. El coste total será evaluado por la función objetivo y servirá para determinar la solución con mayor calidad.

- Se debe intentar evitar que algún alumno tenga dos exámenes en dos horarios seguidos en el mismo día. Esta restricción solo se aplica para exámenes que compartan día, es decir, si un estudiante tiene exámenes en los periodos 14 y 15 pero estos están asignados a días distintos, la restricción se mantiene como respetada.

- Se debe intentar evitar que algún alumno tenga dos exámenes en el mismo día. Esta restricción es excluyente con la anterior, es decir, si tiene dos exámenes en el mismo día y están seguidos, se considerará violada la restricción anterior y esta no. Por tanto, esta restricción solo se aplica a días que tengan al menos 3 periodos distintos.

- Con el objetivo de dejar un tiempo de descanso similar a todos los estudiantes, cada instancia del problema añade un espacio g, de manera que esta restricción se considera violada si algún alumno tiene dos exámenes asignados en un espacio de tiempo menor o igual que g. Esta restricción no es excluyente con las dos anteriores y, por tanto, si un alumno tiene dos exámenes seguidos seguramente también incumplirá esta restricción.

- Para evitar el cansancio, dependiendo de los valores de n y m definidos en cada instancia, se debe intentar minimizar el número de exámenes de entre los n con más asistencia que hayan sido asignados a alguno de los m últimos periodos.

- Debido a que la finalización de un examen puede distraer a los alumnos que aún siguen examinándose, se debe evitar que dos exámenes con duraciones distintas sean asignados al mismo aula en el mismo horario.

- Cada horario tiene una penalización asociada, que será sumada cada vez que un examen sea asignado a ese horario.

- Cada sala tiene una penalización asociada, que será sumada cada vez que un examen sea asignado a esa sala.

La función objetivo sirve para determinar la calidad de las soluciones. Cada instancia cuenta con una serie de parámetros w^{2R}, w^{2D}, w^{PS}, w^{FL}, w^{NMD} que determinan la importancia de las restricciones anteriores, y los superíndices quieren decir "2 in a Row", "2 in a Day", "Period Spread", "Front Load" y "No Mixed Duration". Además, cada sala $r \in R$ y cada horario $h \in H$ cuentan con su penalización asociada w_r^R y w_h^H, respectivamente, determinadas en cada instancia del problema.

Así, la función objetivo se puede calcular como:

$$w^{2R}C^{2R} + w^{2D}C^{2D} + w^{PS}C^{PS} + w^{FL}C^{FL} + w^{NMD}C^{NMD} + C^H + C^R$$

donde:

- $X^H_{e,h} \in \{0,1\}$ determina si el examen e está asignado al horario h.
- $X^R_{e,r} \in \{0,1\}$ determina si el examen e está asignado a la sala r.
- $C^{2R} = \sum_{s \in S} C^{2R}_s$ y C^{2R}_s corresponde con el número de veces que el estudiante s participa en dos exámenes e_1 y e_2 tales que están asignados a los horarios h_1 y h_2 donde $d(h_1, h_2) = 1$ y están asignados al mismo día.
- $C^{2D} = \sum_{s \in S} C^{2D}_s$ y C^{2D}_s corresponde con el número de veces que el estudiante s participa en dos exámenes e_1 y e_2 tales que están asignados a los horarios h_1 y h_2 donde $d(h_1, h_2) > 1$ y están asignados al mismo día.
- $C^{PS} = \sum_{s \in S} C^{PS}_s$ y C^{PS}_s corresponde con el número de veces que el estudiante s participa en dos exámenes e_1 y e_2 tales que están asignados a los horarios h_1 y h_2 donde $d(h_1, h_2) \leq g$.
- $C^{FL} = \sum_{\substack{h \in H_{FL} \\ e \in E_{FL}}} X^H_{e,h}$ y donde $H_{FL} \subset H$ corresponde con los m últimos periodos y $E_{FL} \subset E$ corresponde con los n exámenes con mayor asistencia.
- $C^{NMD} = \sum_{h \in H} \sum_{h \in H} \sum_{d \in D} U^d_{h,r}$ donde D es el conjunto de distintas duraciones de los exámenes y $U^d_{h,r} \in \{0,1\}$ indica si hay algún examen de duración d asignado al horario h y en la sala r.
- $C^H = \sum_{h \in H} \sum_{e \in E} w^H_h X^H_{e,h}$, es decir, la suma de las penalizaciones de cada horario por el número de exámenes asignados a dicho horario.
- $C^R = \sum_{r \in R} \sum_{e \in E} w^R_r X^R_{e,r}$, es decir, la suma de las penalizaciones de cada sala por el número de exámenes asignados a dicha sala.

3.3. Modelización mediante Coloración de Grafos

La cuestión de encontrar una solución factible para un problema de Timetabling general puede modelizarse como un problema de coloración de grafos. Si nos atenemos únicamente a un conjunto sencillo de restricciones obligatorias, podemos representar cada uno de los eventos como un vértice en el grafo. Como nuestro objetivo consiste en encontrar una asignación factible de los horarios a cada uno de los eventos, podemos considerar un color por cada horario, y así el objetivo será equivalente a encontrar una coloración factible de todos los nodos. De esta forma, las aristas juegan el papel de impedir que dos eventos estén asignados en el mismo horario, por lo que podemos modelizar cada una de las restricciones obligatorias como una serie de aristas en el grafo.

Para modelizar el tipo de problema que se presenta en ITC2007, se proponen dos acercamientos distintos, que fueron surgiendo durante el proceso de intentar resolver el problema.

3.3.1. Primer Acercamiento

El primer intento de modelización parte de la idea de separar el problema en dos fases. De esta forma, se pretende aprovechar el hecho de que las restricciones sobre los periodos no dependen de la asignación de las salas. La primera fase consistirá en asignar a cada examen un horario, intentando minimizar el coste asociado de la función objetivo, ignorando las restricciones asociadas a las salas. Una vez se haya conseguido una solución para la primera fase con una calidad aceptable, se puede pasar a la segunda fase. Esta segunda fase parte de las asignaciones obtenidas en la primera fase y consiste en asignar las salas a cada examen.

En la primera fase consideramos únicamente las restricciones asociadas a los horarios. Para cada examen $e \in E$ generamos un nodo n_e^E. Por cada estudiante $s \in S$ tomamos el conjunto $E_s = \{e \in E \mid s \in S_e\}$ como el conjunto de exámenes en los que e debe participar. Para cada par $\{e_1, e_2\} \subset E_s$, $e_1 \neq e_2$, como s no puede participar en los dos exámenes al mismo tiempo, añadimos la arista $\{n_{e_1}^E, n_{e_2}^E\}$ al grafo. Con esto, tenemos asegurada la primera restricción obligatoria.

Para asegurar que los exámenes no son asignados a horarios de menor duración, para cada horario $h \in H$ se añade un nuevo nodo n_h^H, el cual se colorea con el color asociado a h. Este nodo representará al horario h y una arista que una un examen con este nodo tendrá el efecto de impedir que dicho examen sea asignado al horario h. Por tanto, por cada horario $h \in H$ de duración l, para cada examen $e \in E$, si e tiene una duración mayor que l, se añade la arista $\{n_h^H, n_e^E\}$ al grafo, asegurando así la restricción de la longitud de los horarios.

De esta forma, solo queda por modelizar las restricciones que dependen de la instancia del problema. Si se tiene que e_1 y e_2 deben ser asignados a horarios distintos, basta con añadir la arista $\{n_{e_1}^E, n_{e_2}^E\}$. Para las restricciones que exijan que e_1 y e_2 sean asignados al mismo horario, debemos asegurar en la implementación que siempre que se coloree uno de un color, el otro nodo sea coloreado con el mismo color inmediatamente. Finalmente, si e_1 debe ser asignado a un horario estrictamente posterior al de e_2, podemos realizar los siguientes cambios: se añade la arista $\{n_{e_1}^E, n_{e_2}^E\}$ ya que el orden debe ser estricto, y si h_0 y h_f son respectivamente el primer y último horario de H, entonces añadimos las aristas $\{n_{h_0}^H, n_{e_1}^E\}$ y $\{n_{h_f}^H, n_{e_2}^E\}$, ya que asignar e_1 al primer horario o e_2 al último dejaría sin posibilidades al otro. Además, cuando e_1 sea coloreado con el color h_i, añadiremos las aristas $\{n_{h_j}^H, n_{e_2}^E\}$ $\forall j > i$, para asegurar que e_2 no es asignado a ningún horario posterior. Análogamente, cuando e_2 sea coloreado con el color h_i, añadiremos las aristas $\{n_{h_j}^H, n_{e_2}^E\}$ $\forall j < i$.

El algoritmo será ejecutado sobre este grafo para obtener primero una asignación $a_1 : E \to H$ válida para esta primera fase, intentando reducir el coste asociado a la función objetivo restringida a las restricciones asociadas a los horarios.

Una vez se haya encontrado una asignación $a_1 : E \to H$ factible, el problema se reduce a encontrar una asignación $a_2 : E \to R$ para las salas y de esta forma podemos tomar la asignación $a(e) = (a_1(e), a_2(e))$ como solución final. Siguiendo las mismas ideas, podemos modelizar esta segunda fase utilizando un nuevo grafo en el cual cada arista une dos exámenes asignados al mismo horario que no pueden ser asignados a la misma sala. Para ello, el

nuevo grafo debe contener un nodo sin color n_e^E para cada examen $e \in E$ y un nodo n_r^R para cada sala $r \in R$, con el color asociado a esta.

Para cada periodo y cada sala, se guarda un registro donde se indica la capacidad disponible de la sala en ese momento, esto es $C_r^h = C_r - \sum_{e \in E} X_{e,h}^H X_{e,r}^R \|S_e\|$ donde C_r es la capacidad total de la sala r. Para modelizar las restricciones obligatorias, para cada examen e exclusivo, añadimos las aristas $\{n_e^E, n_{e'}^E\}$ $\forall e' \in E \mid a_1(e) = a_1(e')$, asegurando así el uso exclusivo de la sala de e durante su transcurso. Finalmente, para asegurar que ninguna sala supera su capacidad máxima en ningún momento, cada vez que un examen e sea asignado a la sala r, actualizamos el valor de $C_r^{a_1(e)}$ como $C_r^{a_1(e)} = C_r^{a_1(e)} - \|S_e\|$ y comprobamos cuáles son los exámenes que no van a poder ser asignados a la misma sala. Esto es, añadir las aristas: $\{n_{e'}^E, n_r^R\} \quad \forall e' \in E \mid a_1(e) = a_1(e') \wedge \|S'_e\| > C_r^{a_1(e)}$.

Este primer intento de modelización posee tres grandes ventajas:

- Es bastante rápido, ya que al dividirlo en dos fases, el número de aristas de cada grafo es menor y el número de colores posibles es más reducido.
- La segunda fase solo contiene aristas que unen nodos asociados a exámenes con otros nodos asociados a exámenes en el mismo horario o con nodos asociados a salas (que no necesitan ser coloreados). Esto permite paralelizar esta segunda fase generando un grafo distinto por cada horario, incluyendo únicamente los nodos de exámenes en dicho horario y una réplica de cada nodo asociado a una sala. De esta forma, cada nuevo grafo es independiente de los demás y puede ser coloreado mediante un proceso paralelo, y además cada grafo tiene un número de nodos mucho menor, haciendo incluso viable aplicar algoritmos exactos de forma rápida.
- Gracias a la generalidad que aporta la hiperheurística, ambas fases pueden ser resueltas por el mismo algoritmo. Lo único que sería necesario adaptar son las restricciones que suceden al colorear un nodo en cada grafo y la función objetivo sobre la que operará la hiperheurística.

A la hora de realizar los experimentos, la primera fase obtenía unos resultados muy superiores a las soluciones ganadoras de la competición (siempre que se restringiese a las penalizaciones asociadas a los horarios). Sin embargo, a la hora de asignar las salas, el algoritmo era incapaz de encontrar soluciones factibles. Después de un análisis en profundidad, se desveló que este problema no era culpa del algoritmo, si no de la modelización escogida. La raíz de esta cuestión radicaba en que la primera fase del problema asignaba en el mismo horario una cantidad de exámenes cuyo alumnado superaba a la suma total de todas las capacidades, haciendo absolutamente imposible encontrar una asignación a_2 compatible con a_1. Aunque existiese la posibilidad de que una asignación a_1 estuviera lo suficientemente repartida como para poder encontrar una solución factible en la segunda fase, esta situación no iba a darse en el caso general. Se estudió la posibilidad de añadir restricciones de manera que la suma de los alumnos presentes en cada examen no pudiese superar la capacidad total máxima de todas las salas, sin embargo se seguiría sin poder asegurar la existencia de factibilidad al asignar las salas debido a la posibilidad de que un examen exclusivo dejase huecos libres. Por tanto, esta modelización debió ser descartada, a pesar de ser considerablemente más rápida.

3.3.2. Segundo Acercamiento

Las conclusiones obtenidas al intentar aplicar la modelización anterior indicaban que la única forma de poder modelizar el problema con coloración de grafos pasaban por asignar al mismo tiempo la sala y el horario. Por ello, en esta segunda aproximación, cada color vendrá dado por un par (h, s) que corresponde con la asignación en el horario h y la sala s. Con esta elección de colores, el significado de las aristas no puede ser el mismo, puesto que necesitamos dos tipos de prohibiciones: prohibición de horario (por ejemplo dos exámenes con alumnos en común) y prohibición de horario y sala (por ejemplo un examen exclusivo con cualquier otro examen). Por esta razón, se distinguirán dos tipos de aristas, según los nodos que unen. Por tanto, se consideran cuatro tipos de nodo: asociado a un examen, asociado a un horario, asociado a un color y asociado a la exclusividad de un examen. Las aristas que unan un examen con otro examen o un horario representarán la prohibición de que ambos nodos compartan horario, mientras que si une un examen con un nodo asociado a un color o a una exclusividad, representará la prohibición de que ambos compartan color (es decir, horario y sala simultáneamente). Estas restricciones deben ser contenidas dentro de la "caja negra" en la que consiste la estructura de datos utilizada. Así, será posible emplear el mismo algoritmo, pues será capaz de trabajar con esta caja negra, abstrayendo el cambio de modelo.

El grafo de esta modelización debe contener un nodo n_e^E para cada examen $e \in E$, un nodo n_h^H para cada horario $h \in H$ con un color asociado al horario, n_c^C para cada color $c \in H \times S$ con el color c y un nodo n_e^{Ex} para cada examen e con la restricción de exclusividad. Cada arista que una un nodo $n_{e_1}^E$ con otro nodo $n_{e_2}^E$ de color (p, s) o con otro nodo n_h^H, descarta todas las posibles coloraciones (p, s') $\forall s' \in S$ para el nodo $n_{e_1}^E$. Mientras que si está unido a un nodo n_c^C o un nodo n_e^{Ex} con color c, únicamente descarte el color c.

Para mantener las restricciones obligatorias en cada momento, se irán añadiendo aristas de forma dinámica. En la inicialización, se añade una arista $\{n_{e_1}^E, n_{e_2}^E\}$ entre cada par de exámenes que compartan algún alumno o estén dentro de una restricción de exclusión mutua. Se añaden también las aristas $\{n_e^E, n_h^H\}$ para cada examen y horario tal que la duración del examen es mayor que la del horario y $\{n_e^E, n_c^c\}$ para cualquier $c = (h, r)$ tal que la capacidad de la sala r es menor que el número de asistentes al examen e. Cada vez que un nodo sea asignado a $c = (h, r)$, se resta el número de participantes del examen a la capacidad restante de dicho color y se añade una arista $\{n_{e'}^E, n_c^C\}$ para cada examen e' que supere la nueva capacidad restante. Si n_e^E es asignado a (h, s) y debe estar en el mismo horario que $n_{e'}^E$, se añaden las aristas $\{n_{e'}^E, n_{h'}^H\}$ $\forall h' \in H \setminus \{h\}$. Si n_e^E debe ser asignado a un horario posterior que $n_{e'}^E$, replicando el modelo anterior, en la inicialización se añaden las aristas $\{n_e^E, n_{e'}^E\}$, $\{n_e^E, n_{h_0}^H\}$ y $\{n_{e'}^E, n_{h_f}^H\}$, y cuando n_e^E sea asignado a (h, s), se añade una arista que una $n_{e'}^E$ con todos los nodos n_h^H para h' posterior a h y viceversa si $n_{e'}^E$ es asignado. Finalmente, se añade una arista $\{n_e^E, n_{e'}^{Ex}\}$ $\forall e \in E \mid e \neq e'$ por cada examen e' que deba ser asignado en exclusiva a una sala. El nodo $n_{e'}^{Ex}$ será siempre coloreado del mismo color a la vez que su nodo asociado $n_{e'}^E$, y así las aristas con el nodo exclusivo son menos restrictivas que las aristas con el nodo del propio examen. Cada vez que un nodo cualquiera sea asignado a un color $c = (h, r)$, debemos añadir la arista $\{n_{e'}^E, n_c^C\}$ y de esta forma se garantiza que cualquier coloración factible encontrada es a su vez solución del problema.

3.4. Hiperheurística para el problema

Para tratar de resolver estos problemas, se va a utilizar la hiperheurística que va a ser detallada en este capítulo. Esta usará como estrategia de alto nivel la búsqueda tabú sobre un conjunto de secuencias de heurísticas constructivas de coloración de grafos introducidas en [8]. Este acercamiento a la resolución de problemas de Time-Tabling ha sido estudiado en multitud de artículos [6,7,23,24]. A raíz de su publicación, se ha despertado el interés de la comunidad sobre las hiperheurísticas, y a menudo son utilizadas variaciones de este algoritmo para resolver problemas de Time-Tabling.

3.4.1. Heurísticas de coloración de grafos

El espacio de búsqueda sobre el que operará la estrategia de alto nivel está formado por secuencias de heurísticas de coloración de grafos. La aplicación de una secuencia de heurísticas consiste en aplicar k pasos de cada una de las heurísticas de la secuencia en orden hasta alcanzar una solución completa. Es decir, si se pretende aplicar la secuencia $[h_1, h_2, h_3]$, se parte de una solución vacía y primero se toma h_1, se ejecuta hasta que haya asignado un color a k elementos, posteriormente se selecciona h_2 y se vuelve a ejecutar k pasos y se procede de forma iterativa hasta completar la solución.

La idea detrás de las heurísticas que van a formar parte de la secuencia consiste en determinar en cada paso el nodo que sea más difícil de asignar. Este será el nodo elegido y posteriormente se buscará el color que aumente en menor medida el valor de la función objetivo. Paso a paso se completará la solución de forma voraz. El objetivo es dejar los nodos más fáciles para el final, ya que los últimos nodos en ser coloreados serán los que experimentarán más restricciones debido al color asignado de vecinos. El funcionamiento de estas heurísticas puede dividirse en dos partes: La primera parte consiste en seleccionar el nodo que va a ser coloreado y la segunda parte consiste en seleccionar el color al que va a ser asignado.

Para seleccionar el nodo, se van a utilizar los siguientes criterios, que darán nombre a las heurísticas que formarán. Estos tratan de ordenar los nodos acorde con una medida de la dificultad de asignarlos, para tomar el primero y darle un color:

- **Largest Degree (LD).** Se ordenan todos los nodos según su grado y se toma el de mayor grado.
- **Largest Enrollment (LE).** Se ordenan los nodos según el número de estudiantes que participan en el examen y se toma el mayor.
- **Largest Weighted Degree (LWD).** Se ordenan todos los nodos según su grado multiplicado por el número de estudiantes que participan en el examen y se toma el mayor.
- **Largest Colored Degree (LCD).** Se ordenan todos los nodos según su "grado de color", es decir, el número de aristas que comparte con nodos con un color asignado, y se toma el de mayor grado de color.

- **Least Saturation Degree (LS).** Se ordenan los nodos según el número de colores disponibles con los que puedan ser asignados y se escoge el que tenga el menor número de posibilidades.

- **Random (R).** Se escoge un nodo de forma aleatoria. Esto servirá para dar variedad al espacio de búsqueda y diversificar las soluciones encontradas.

  Estas son las heurísticas aplicadas en los artículos mencionados anteriormente. Sin embargo, debido a que las restricciones que se completan de forma dinámica no son necesariamente consideradas por estas heurísticas, decidí añadir la siguiente heurística. El objetivo es poder dar prioridad a los nodos que tengan restricciones más fuertes no modelizadas, y así solucionar los casos en los que estos no fuesen considerados suficientemente pronto por no haber tenido las aristas añadidas y se alcanzasen soluciones infactibles.

- **GradoRestricciones (GR).** Se calcula el grado del nodo y a este se le suma una cantidad en función de las restricciones extra que haya sobre este nodo. Si es exclusivo, se le suma el número de exámenes, pues el nodo exclusivo asociado esta unido a cada examen distinto. Si debe ser asignado antes o después que otro examen, se le suma la mitad de la cantidad de periodos disponibles y si debe ser asignado al mismo horario que otro examen, se suma el valor asociado a dicho examen. Finalmente, se ordenan los nodos según estos valores y se toma el mayor. Si ningún nodo tiene restricciones especiales, esta heurística funcionará igual que *LargestDegree*. En caso contrario, el nodo que tenga restricciones especiales tendrá una mayor prioridad para ser seleccionado.

A su vez, existen diferentes estrategias para determinar el color al que va a ser asignado. Una opción es seleccionar dicho color de forma completamente aleatoria, lo que permite unos cómputos muy ágiles. La agilidad tomará una importancia mayor dentro del marco de la hiperheurística, pues esta aplicará la selección muchas veces por cada iteración. Otra posible selección consistiría en tratar de predecir el incremento en la función objetivo que va a suponer asignar al nodo cada color y escoger el color que minimice este incremento. Esta estrategia brindará mejores resultados, pero tendrá un coste mayor y funciona de forma determinista, por lo que se explorará una menor parte del espacio de soluciones. En[24], se emplea la estrategia de la ruleta, que consiste en predecir los incrementos y en función de estos asignar una probabilidad a cada color de forma que los que menor coste añadan tengan mayores probabilidades. Posteriormente, se selecciona un color al azar siguiendo estas probabilidades. Esta estrategia permitirá obtener mejores resultados en algunos casos, ya que al incluir no determinismo, se amplía el espacio de soluciones que puede encontrar. Otras estrategias posibles serían aplicar búsqueda local o alguna otra heurística para evitar tener que calcular el incremento de la función objetivo de todos los colores y ganar eficiencia sin perder demasiada calidad de la solución.

En el Algoritmo 3 se presenta el pseudocódigo general de las heurísticas de coloración de grafos.

Algoritmo 3 Heurísticas de coloración de grafos

$a \leftarrow \emptyset$
$i \leftarrow 0$
while $i < k$ **and not** $completa(a)$ **do**
  $nodo \leftarrow seleccionNodo(grafo)$
  for $color \in coloresDisponibles(nodo)$ **do**
    $a \leftarrow a \cup \{(nodo, color)\}$
    $predicciones.add(f_{obj}(a))$
    $a \leftarrow a \setminus \{(nodo, color)\}$
  end for
  $mejorColor \leftarrow seleccionColor(predicciones)$
  $x \leftarrow x \cup \{(nodo, color\}$
end while

3.4.2. Estrategia de alto nivel

La heurística que va a ser empleada como estrategia de alto nivel es la búsqueda tabú, explicada en la Sección 2.2.3. El espacio de búsqueda sobre el que opera es el conjunto de listas formadas por las posibles combinaciones de las heurísticas descritas en el apartado anterior.

Se parte de una lista inicial. Esta lista inicial puede tomarse de forma aleatoria o usando una lista donde todas las componentes sean de la heurística que mejores resultados obtiene cuando es aplicada en solitario. El peso de la heurística inicial en la solución final será estudiado en la Sección 3.5. En cada iteración de la búsqueda tabú, se genera el nuevo vecindario que se obtiene cambiando de forma aleatoria un número de heurísticas de la lista por otras heurísticas distintas. Posteriormente, se evalúa cada una de las secuencias del vecindario aplicando $k = 2$ pasos constructivos de cada una de las heurísticas de la lista sobre una solución vacía. El valor asociado a la secuencia será tomado como el valor de la función objetivo sobre la solución completa generada. La tenencia seleccionada en base a los experimentos realizados y a la literatura es $t = 9$.

Siguiendo el esquema de la búsqueda tabú, se repite este proceso sobre la nueva heurística seleccionada durante un número de iteraciones suficientemente alto. El resultado final obtenido será una secuencia de heurísticas, que es a su vez una nueva heurística. Se puede tomar como solución al problema de TimeTabling la solución que genere esta heurística. Si esta tiene un comportamiento no determinista (selección de color por ruleta), es posible ejecutarla un número de veces y quedarse con la mejor solución obtenida. Lo esperable es que esta heurística obtendrá mejores resultados que las heurísticas que se han utilizado de base para seleccionarla, ya que seleccionará la mejor combinación en cada paso.

En los casos en los que ninguna heurística produzca una solución factible, la selección de la próxima heurística que va a ser iterada será aquella que haya sido capaz de colorear un número mayor de nodos antes de encontrar una infactibilidad. De esta forma, se pretende que la heurística evolucione hacia una nueva que sea capaz de colorear todos los nodos y así habrá encontrado una solución factible.

Por último, siguiendo las recomendaciones de [6], se añade una lista de secuencias fallidas. Esto permite evitar la evaluación repetida de heurísticas que ya se conoce que van a producir una solución infactible. Para ello, se añade a la lista de secuencias fallidas el prefijo de longitud igual al número de pasos que ha tardado en fallar. Cualquier nueva heurística evaluada que tenga este prefijo volverá a fallar, por lo que omitimos su ejecución para ahorrar recursos en la computación.

3.5. Resultados

En esta sección se van a comparar los resultados obtenidos por la hiperheurística con los ganadores de la competición ITC2007 y con los resultados conseguidos mediante las heurísticas de bajo nivel que toma como base. Esto último permitirá analizar la mejora que es capaz de realizar la estrategia de alto nivel al iterar sobre su espacio de búsqueda. Para obtener estas soluciones, se han considerado los casos en los que la selección del color en cada paso constructivo se realiza mediante la elección del mejor color y mediante el método de la ruleta. Las soluciones obtenidas han sido validadas mediante la herramienta Online disponible en este enlace y se encuentran publicadas en el repositorio de GitHub. Las tablas correspondientes a los resultados de esta sección se encuentran en el Apéndice B.

3.5.1. Resultados de las heurísticas de bajo nivel

En primer lugar, se analizan los resultados obtenidos mediante la aplicación directa de cada una de las heurísticas definidas en la Sección 3.4.1. Se consideran los casos en los que la selección del color se realiza escogiendo el color que menos aumenta el valor de la función objetivo en cada paso constructivo, cuyos datos corresponden a la tabla B.1; y mediante la estrategia de selección por ruleta, visibles en la tabla B.2. Cada columna de la tabla corresponde con el valor de la función objetivo obtenida sobre la solución después de aplicar cada una de las heurísticas, denotadas según sus iniciales. En el caso de la selección por ruleta, se han realizado varias ejecuciones debido a que la solución obtenida puede variar por el no determinismo. Las posiciones vacías de las tablas indican que no se ha obtenido una solución factible.

En general, las heurísticas que son capaces de obtener soluciones factibles en un mayor número de problemas son *LeastSaturationDegree*, *LargestColoredDegree* y sobre todo *GradoRestricciones*, por lo que se puede considerar que introducir esta heurística ha sido un éxito. Estas son las heurísticas que se espera que aparezcan con mayor frecuencia en la lista de heurísticas de las soluciones finales. La heurística *GradoRestricciones* ha sido capaz de obtener mejores resultados en los problemas con un gran número de restricciones adicionales sobre exámenes concretos. Esto se debe a que es capaz de priorizar estos exámenes cuyas restricciones se van añadiendo de forma dinámica. El resto de heurísticas no son capaces de considerar la dificultad de estos nodos desde el principio, pues las restricciones que estos añaden solo se verán representadas una vez se haya coloreado el nodo asociado. Por tanto, si un examen de estas características es asignado en la parte final, de forma súbita añadirá una

gran cantidad de restricciones que no habían sido consideradas en los pasos anteriores, con la posibilidad de dejar algún nodo sin colores disponibles. Añadir esta heurística al conjunto de estrategias de bajo nivel de la hiperheurística puede permitir que, en pasos tempranos, esta sea ejecutada, coloreando los nodos con este tipo de restricciones y añadiendo la información correspondiente para que el resto de heurísticas pueda trabajar con normalidad a partir de este punto. En algunos casos esto permitirá alcanzar la factibilidad.

La selección del mejor color aporta buenas soluciones de forma determinista, mientras que la selección por ruleta obtiene resultados dispares. Es cierto que en algunos casos es capaz de obtener mejores resultados que la variante determinista, destacando principalmente en los problemas asociados al *set5* y al *set8*. Además, es capaz de encontrar factibilidad en ejecuciones donde antes no había sido posible. Sin embargo, en otras ocasiones, los resultados obtenidos empeoran en gran medida con respecto a la solución determinista. Una forma de paliar este efecto sería mediante la ejecución repetida de la heurística con ruleta, quedándose con en mínimo valor obtenido por esta. Esto, sin embargo, añadirá un coste adicional que dentro del marco de la hiperheurística puede resultar muy importante, ya que en cada iteración de la heurística de alto nivel se ejecutan tantas heurísticas como el número de vecinos de la secuencia sobre la que se está iterando. Si además se repiten estas heurísticas varias veces, el tiempo de cómputo puede incrementar bastante.

3.5.2. Resultados de la hiperheurística

En base a los resultados obtenidos por experimentos preliminares y los resultados obtenidos en [6], la ejecución de la hiperheurística se ha realizado con los parámetros $k = 2$, tenencia $t = 9$ y movimientos que seleccionan 2, 5 y 10 elementos de la lista los cambian de forma aleatoria. También se ha estudiado la diferencia al modificar estos parámetros. Se han realizado los experimentos aplicando las heurísticas con y sin selección por ruleta, tomando como secuencias iniciales la dada por *LeastSaturationDegree* y otra generada de forma aleatoria. La lista de *LeastSaturationDegree* se utiliza debido a su capacidad de encontrar factibilidad y a que es la más empleada como inicialización a lo largo de la literatura. Por otro lado, se emplea la lista generada de forma aleatoria para añadir más variedad a la solución, pues puede que si el número de iteraciones no es suficientemente alto, no se cambien buena parte de las heurísticas de la secuencia. Los resultados obtenidos están representados en la tabla B.4.

En primer lugar, se realizaron experimentos con distintos valores de k y distintos movimientos. Añadir más movimientos tiene, en general, un coste en tiempo mucho mayor, pues por cada movimiento añadido debe repetirse la evaluación de la secuencia una vez más. Sin embargo, la diferencia en el valor de la solución no fue significativo. Esto se puede deber a que los elementos que varían están seleccionados de formas aleatorias. Se deberían estudiar formas de poder aumentar el número de movimientos sin suponer un coste tan alto. Una posibilidad podría consistir en que si varias heurísticas del vecindario comparten los m pasos iniciales, se ejecuten estos m pasos una única vez y posteriormente cada secuencia recibe una copia del grafo tras los m pasos para terminar de completarla. En cuanto al valor de k, cuanto mayor es este, se reduce en gran medida el espacio de búsqueda de las soluciones, ya que cuantos más pasos realice cada heurística de la lista, menor longitud tendrá esta y

menor será el numero de posibles combinaciones. Esto puede ayudar a la búsqueda local, evitando que la solución realice muchos caminos similares pasando por puntos muy cercanos pero que no estén en la lista tabú por ser diferentes en una única posición. Sin embargo, un valor muy alto de k puede empeorar los resultados, ya que cuanto mayor sea este, mayor será la similaridad de las secuencias con las heurísticas que forman el conjunto base de la hiperheurística. Realizando los experimentos con $k = 1$, la longitud de las secuencias era tan grande que era difícil que la secuencia se repitiese, por lo que no se llegaba a aplicar el criterio de la búsqueda tabú. Finalmente se seleccionó $k = 2$ porque con este dato se obtenían buenos resultados.

Por otro lado, se ha observado que, a menudo, la solución deja de mejorar a partir de las 300 primeras iteraciones. Esto puede deberse a que los movimientos seleccionados proporcionen una gran diversificación en las iteraciones iniciales, mientras que la aleatoriedad que estos involucran dificulte la intensificación a partir de cierto punto. Para solventar esta situación, sería posible implementar algún método que fuese capaz de seleccionar los movimientos en cada iteración de forma más inteligente. En [23] se estudia esto mediante la aplicación de *Variable Neighbourhood Search* como estrategia de alto nivel. Esta estrategia tiene la capacidad de ir modificando la estructura de los vecindarios a lo largo de las diferentes iteraciones, de forma que permite premiar la intensificación o la diversificación cuando estas son necesarias.

Otra característica de la hiperheurística es que esta trabaja sobre el espacio de búsqueda de las secuencias, en lugar de sobre el espacio de búsqueda de las soluciones directamente. Esto tiene el efecto de que la solución obtenida en cada iteración no se desplaza de forma lenta sobre el espacio de soluciones, si no que tiene la capacidad de "saltar" sobre este. Dos secuencias vecinas pueden producir soluciones muy lejanas. Esto, por un lado, permite alcanzar zonas de factibilidad de forma más eficaz, sin embargo, dificulta la intensificación. Además, tiene un efecto negativo muy importante. A diferencia de las heurísticas convencionales, este efecto de salto hace imposible alcanzar todas las soluciones. Es decir, hay múltiples soluciones para las cuales no existe una secuencia de heurísticas que las alcance. Por esta razón, en [6,23] se considera la aplicación de una pequeña búsqueda local sobre la solución obtenida en cada iteración. En estos casos, el espacio de búsqueda de la hiperheurística estará formado por heurísticas híbridas, que aplican la búsqueda local sobre la solución obtenida de forma constructiva. Esta búsqueda permite mejorar las soluciones obtenidas por la secuencia, explorando una mayor parte del espacio de búsqueda de soluciones y minimizando el efecto de "salto".

El tiempo que ha empleado el algoritmo en cada problema ha variado mucho. Problemas como *set4*, *set9* o *set12* son problemas con pocos exámenes o salas para asignar pero con unas restricciones muy duras, son los que el algoritmo ha resuelto más rápido. Esto ha permitido realizar los cálculos con un número mayor de iteraciones en estos problemas, lo cual ha sido de gran utilidad ya que estos problemas están entre los más difíciles de encontrar soluciones factibles para los ganadores de la competición. Por otro lado, los problemas con más salas y exámenes como *set2* o los que contienen un gran número de estos y tienen muchas restricciones específicas de las salas, como *set3*, han resultado mucho más costosos para el algoritmo.

Los resultados obtenidos superan con cierto margen a los obtenidos por las heurísticas

base, sobre todo en la capacidad de encontrar soluciones factibles. Esto se debe a que, a menudo, cambiar el criterio de selección en pasos concretos puede conseguir crear decisiones que sean más fáciles de completar en pasos futuros. Se concluye que la hiperheurística ha resultado exitosa a la hora de mejorar las heurísticas base, adaptándolas al problema en cada situación.

Aunque existen excepciones, como el *set12*, en general, la selección por ruleta obtiene peores resultados. A pesar de que esta pueda llegar a mejorar la solución con respecto a la selección del mejor color, en este esquema iterativo, este no determinismo puede hacer que la solución no se desplace a la mejor vecina y por tanto no se alcancen zonas de igual calidad que en el caso determinista. Para poder aplicar la selección por ruleta con éxito sería necesario que en cada iteración se ejecutasen varias repeticiones de la secuencia y tomar como resultado la que tuviese un valor asociado más pequeño. Esto añadiría un coste mucho mayor, que en los problemas más pequeños podría ser aplicable pero en los problemas grandes puede llegar a hacerlos impracticables.

En cuanto a la inicialización, la lista inicial que empieza con *LargestSaturationDegree* suele obtener mejores resultados en general, aunque la diferencia de la calidad no es demasiado grande. En algunos problemas, la inicialización aleatoria ha conseguido resultados superiores, como en el *set5* y *set8*. Analizando la composición de la secuencia resultante se observa que la inicialización por *LS* a menudo mantiene una gran proporción de estos pasos en la secuencia, mientras que la inicialización aleatoria alcanza secuencias mucho más distribuidas. Es de esperar que a medida que aumente el número de iteraciones, estas diferencias disminuyan. Las heurísticas que aparecen con mayor frecuencia en las secuencias finales son *LWD*, *LS*, *LCD* y *GR*.

3.5.3. Comparación frente a los ganadores de la competición

Los resultados obtenidos por los ganadores de la competición ITC2007 se encuentran en la tabla B.3. En esta se muestra el resultado medio obtenido después de 10 ejecuciones y el número de ocasiones en los que no se ha encontrado una solución factible. La mayoría de estos resultados se ha obtenido mediante metaheurísticas perturbativas que involucran Simmulated Annaealing.

Sorprende ver que, en algunos conjuntos de datos, las propias heurísticas del conjunto base son capaces de superar a la mayoría de finalistas, como en los casos de los conjuntos *set2* y *set7*. Esto indica que es poco probable que hubiese muchas participaciones basadas en la coloración de grafos. Por otro lado, es normal que, con el paso del tiempo, la capacidad de computación y el desarrollo de las técnicas algorítmicas permitan cada vez mejorar las soluciones encontradas. Las soluciones ganadoras de la competición en ocasiones son incapaces de encontrar soluciones factibles. Por ejemplo, el algoritmo del primer puesto en el *set11* solo es capaz de encontrar una solución factible en las 10 ejecuciones. También ocurre que la técnica del 4º queda rezagada por el gran número de problemas para los que no es capaz de hallar factibilidad, a pesar de obtener muy buenos resultados, superando incluso al segundo puesto, cuando alcanza factibilidad.

Los resultados obtenidos mediante la hiperheurística quedan entre los 3 primeros puestos en todos los problemas, por lo que se puede confirmar que sirve para obtener buenos resultados capaces de competir con las mejores técnicas. En los problemas *set6* y *set11* incluso llega a quedar en primera posición, alcanza 6 veces el segundo puesto y 4 veces el tercero. Cabe destacar que en los problemas *set3* y *set4* esta queda por detrás de problemas que a pesar de obtener buenos resultados, presentan dificultades para obtener soluciones factibles. La hiperheurística cuenta con dos ventajas fundamentales respecto a las soluciones ganadoras. La primera es que no necesita partir de ninguna solución, ya que las heurísticas base son todas constructivas. Por tanto, es capaz de construir una solución a partir de una vacía, mientras que los de la competición necesitan apoyarse en otra técnica para poder generar una solución inicial. La segunda ventaja es que el algoritmo propuesto en este trabajo siempre ha sido capaz de encontrar soluciones factibles, mientras que las soluciones ganadoras presentan algunas dificultades en este aspecto sobre algunos problemas.

Además, la mayoría de las técnicas empleadas por las soluciones ganadoras están basadas en metaheurísticas perturbativas. Si se utiliza la hiperheurística para obtener una solución inicial, se conseguirá una bastante buena solución de partida que puede ser mejorada mediante estas técnicas perturbativas. Aprovechando esta hibridación, sería posible mejorar en gran medida las soluciones ganadoras de la competición, ya que en algunos casos la solución de partida es directamente mejor que las soluciones encontradas después de la ejecución sin hibridación. De esta forma también se garantizaría encontrar factibilidad. En concreto, el algoritmo empleado en el 4º puesto se vería muy beneficiado por esta hibridación, ya que su mayor dificultad es encontrar soluciones factibles de partida, pero cuando las encuentra es capaz de obtener muy buenos resultados.

Finalmente, se concluye que la hiperheurística construida en este capítulo ha cumplido todos los objetivos deseados. Es capaz de encontrar soluciones factibles con un buen valor de la función objetivo asociado, consiguiendo competir con las mejores soluciones aportadas a la competición en 2007. Además, también es capaz de mejorar con un cierto margen el resultado producido por las heurísticas que toma como base, produciendo una nueva heurística mejorada y adaptada al problema que se está tratando empleando aprendizaje On-line. Por último, se ha estudiado el efecto de la selección por ruleta y se han estudiado posibles mejoras que pueden introducirse para agilizar el cómputo e incluso mejorar los resultados obtenidos por medio de implementar una búsqueda local auxiliar o mediante la hibridación las técnicas empleadas por las soluciones ganadoras.

Capítulo 4

Metaoptimización de PSO mediante algoritmos genéticos

En este capítulo se busca explorar la metaoptimización y la optimización multiobjetivo desde la perspectiva de las hiperheurísticas. Estos conceptos son introducidos en [14,20,21,25]. El objetivo final consiste en aplicar una hiperheurística basada en algoritmos genéticos que operarán sobre un espacio de búsqueda formado por los diferentes PSO en función de sus parámetros, tratando de encontrar los mejores parámetros para este.

4.1. Definición de Metaoptimización

Otra aplicación de las hiperheurísticas consiste en tratar de adaptar o mejorar metaheurísticas previamente concebidas para favorecer su rendimiento sobre un grupo de problemas. Esta técnica es conocida como *metaoptimización* y consiste en aplicar un método de optimización para poner a punto a otro método de optimización distinto.

Frecuentemente, las metaheurísticas contienen una serie de parámetros que pueden afectar en gran medida al desempeño del problema. A menudo, queda en la mano de la persona que vaya a aplicar la heurística el seleccionar estos parámetros. Por ejemplo, para el Particle Swarm Optimization descrito en la Sección 2.2.1, se tienen los coeficientes de inercia, cognitivo y social. En la Tabla 4.1 se ejecutan varias instancias de PSO con 100 partículas y 100 iteraciones para distintas elecciones arbitrarias de coeficientes sobre la función de *Rastrigin* (definida en el Apéndice A). Se puede comprobar como la convergencia de la solución es muy sensible a la elección de estos parámetros. Por esta razón, tiene sentido tratar de encontrar los mejores coeficientes posibles para garantizar la convergencia de la heurística sobre un determinado conjunto de problemas. Además, aplicando técnicas de metaoptimización se puede evitar que la selección de estos parámetros parta de presunciones o fallos de concepción sobre el efecto que tienen en la solución.

En general, la metaoptimización se realiza por medio de un algoritmo de optimización que actúa sobre un conjunto de heurísticas, donde cada heurística viene codificada a través de una serie de parámetros que regulan su funcionamiento. Este algoritmo toma como función objetivo el resultado de aplicar las heurísticas sobre las que trabaja, y de esta forma es

Coeficientes	$w = 1, \varphi_1 = 1, \varphi_2 = 1$	$w = 0.3, \varphi_1 = 0.8, \varphi_2 = 0.2$	$w = 0.2, \varphi_1 = 0.7, \varphi_2 = 0.7$
Valores	12.0172	3.0934	0.9952
	10.2453	3.4128	3.98
	12.6782	3.8531	1.9899
	14.5388	3.5809	0.995
	10.8671	6.6356	0.995
Media	12.0693	4.1152	1.791

Tabla 4.1: Valores encontrados por PSO para Rastrigin con 100 partículas y 100 iteraciones.

capaz de encontrar los parámetros para los cuales se consiguen mejores resultados. Por lo cual este algoritmo de metaoptimización es a su vez una hiperheurística, cuyo resultado es una heurística adaptada al problema o problemas sobre los que ha sido entrenada.

Además, si el conjunto de entrenamiento utilizado es suficientemente general, sería posible tomar estos coeficientes para utilizarlos en la resolución de nuevos problemas sobre los que no se ha aplicado este estudio. De esta forma, obtendríamos un aprendizaje Offline con el que garantizar la convergencia sobre múltiples funciones. Este será el objetivo final de este capítulo.

4.2. Criterios para la Optimización Multiobjetivo

La utilización de diferentes problemas durante la metaoptimización es importante para añadir generalidad a la solución. Puesto que el entrenamiento va a ser realizado resolviendo varias instancias del problema múltiples veces, el objetivo de esta metaoptimización debería estar enfocado a resolver problemas nunca vistos durante el entrenamiento. De lo contrario, será más eficaz directamente ejecutar el algoritmo base con un número mayor de iteraciones sobre un único problema. Sin embargo, la utilización de varios problemas durante la optimización trae consigo una dificultad adicional a la hora de decidir la mejor solución. A menudo ocurrirá la situación en la que la elección de parámetros que mejor solución encuentra para un determinado problema p, no es capaz de encontrar soluciones de buena calidad para un problema distinto p'. Mientras que otra elección de parámetros distinta puede encontrar soluciones más consistentes para ambos problemas, aunque no sea capaz de alcanzar una calidad semejante sobre el problema p. Esto lleva a la necesidad de un criterio que permita decidir cual de estas dos soluciones es mejor con respecto a todos los diferentes objetivos sobre los que se está optimizando. ¿Es mejor tomar la elección consistente o la que mejora más el primer problema? ¿Tienen todos los problemas la misma importancia?

El campo que se encarga de estudiar estos posibles criterios es el de la optimización multiobjetivo. En un problema de optimización multiobjetivo, se pretende resolver:

$$\begin{aligned} &\text{mín} \quad f(x) = (f_1(x), f_2(x), \ldots, f_n(x)) \\ &sujeto\ a: \quad x \in F \end{aligned}$$

Donde, en el caso de estudio que nos ocupa, x consistiría en una asignación de los parámetros,

F es el conjunto de parámetros que son considerados y $f_i(x)$ sería el resultado obtenido después de aplicar el método de optimización con dicha asignación de parámetros sobre el i-ésimo problema. A continuación, se describen algunos de los criterios más utilizados para decidir cuál es este mínimo.

4.2.1. Suma ponderada

La forma más directa de tomar esta decisión consiste en determinar unos pesos w_i asociados a cada problema, que determinan la importancia de optimizarlo respecto a los demás. Posteriormente, puede tomarse la suma ponderada de los $f_i(x)$ y escoger la menor. Sin embargo, esto puede tener complicaciones debido a que el rango de valores que toma cada problema puede no ser el mismo. Si esto ocurriese, sería necesario normalizar los valores de $f_i(x)$ con respecto al rango del i-ésimo problema para permitir una comparación más justa.

La función auxiliar que se utilizará para determinar el mínimo vendrá dada por

$$g(x) = \sum_{i=1}^{n} w_i y_i(x)$$

donde $y_i(x)$ es el valor normalizado de $f_i(x)$. Este acercamiento permite decidir sobre la multioptimización de forma muy directa, sin embargo, requiere conocer el rango de valores de cada problema para poder normalizarlo y establecer un peso para cada problema.

4.2.2. Ranking medio

Otra forma de determinar la mejor solución para los múltiples objetivos cuando se tiene una población de soluciones es utilizar el ranking medio. En este método, para cada solución $x_m \in P$ de la población, se determina su ranking para cada problema, es decir, se toma $R(x_m) = (r_1(x), r_2(x_m), \ldots, r_n(x_m))$ donde $r_i(x_m)$ es el lugar que ocupa el valor $f_i(x_m)$ con respecto a los $f_i(x') \quad \forall x' \in P$ ordenados. Posteriormente puede calcularse la media de cada vector $R(x_m)$ y de esta forma se obtiene un valor numérico respecto al cual determinar la mejor solución.

Por tanto, la función auxiliar que permite ordenar las soluciones viene dada por

$$g(x_m) = \sum_{i=1}^{n} r_i(x_m)$$

Este método no requiere conocer el rango de valores del problema, aunque a cambio es necesario que en cada iteración se tenga una población de soluciones y establecer de nuevo la comparación cada vez que esta cambie. Por lo tanto, el valor de g asociado a cada solución es contextual con el resto de la población y debe ser modificado a pesar de que la solución permanezca (por ejemplo cuando ocurra elitismo en los algoritmos genéticos), por lo que este método no permite establecer comparaciones absolutas.

En una fase avanzada de los experimentos, es probable que la distribución de los rankings sea muy desigual, es decir, que no haya una solución que quede consistentemente la primera

para todos los problemas, si no que esta alcance buenas posiciones para un problema pero malas para otros. Para determinar la distancia entre estos rangos se puede aplicar el Test de Friedman[26] dado por

$$Q = \frac{12n}{k(k+1)} \sum_{m=1}^{k} \left(g(x_m) - \frac{k+1}{2} \right)^2$$

donde $k = |P|$. Esta fórmula permite tener una medida de la importancia del ranking en la población de soluciones. Cuanto menor sea este coeficiente Q, más distribuidas estarán las soluciones, pues mayor será la distancia media con respecto a $\frac{k+1}{2}$, que es la media de los rangos. Mientras que cuando este sea mayor significa que hay soluciones muy superiores en la mayoría de los problemas.

En el caso de estudio de este trabajo, lo normal es que en la inicialización este coeficiente sea alto y vaya reduciéndose a medida que el problema realiza iteraciones. Por lo tanto, se podría aplicar el Test de Friedman como un criterio de parada, por ejemplo, el algoritmo puede parar cuando el p-valor $P\left(\chi^2_{k-1} \geq Q\right)$ sea menor que una cierta cantidad. De esta forma, se detendría el algoritmo cuando comience a ser muy difícil determinar qué solución es mejor para el problema multiobjetivo, ya que todas las soluciones en ese momento se espera que tengan buena calidad a lo largo de todos los problemas.

4.2.3. Dominancia de Pareto

Este método es el más comúnmente aceptado y se basa en encontrar una ordenación de las soluciones según su nivel de dominancia de pareto. Dadas dos soluciones $x, x' \in F$, se dice que x tiene dominancia de Pareto sobre x', $(x \prec_P x')$ si se verifica:

$$\forall i \in \{1, 2, \ldots, n\} : f_i(x) \leq f_i(x') \wedge$$
$$\exists i \in \{1, 2, \ldots, n\} : f_i(x) < f_i(x')$$

Es decir, x es mejor o igual en todos los objetivos y hay al menos un objetivo en el que x es mejor que x'. Además, se dice que x es un punto de Pareto si no está dominado por ningún otro punto de P. Los puntos de Pareto representan las soluciones para las cuales no se puede encontrar una mejora sobre un problema sin perjudicar a otro problema distinto. Considerando estas como las mejores soluciones para el problema multiobjetivo, se puede establecer un ranking según la dominancia de Pareto de la siguiente forma: Para cada solución x de la población X, se comprueba si es punto de Pareto, es decir, se considera x como candidato de Pareto y para cada punto $x' \in X$, $x' \neq x$ se comprueba si $x' \prec_P x$. En este caso, x no es un punto de Pareto. Si ningún otro punto domina a x, entonces x es punto de Pareto y se le asigna el rango $r_x = 1$. Se eliminan los puntos de Pareto y se repite el proceso para asignar el rango 2 a los puntos de Pareto del nuevo conjunto. Se repite la operación hasta que el conjunto quede vacío. Así, $\{x \in X \mid r_x = n\}$ es el conjunto de los puntos de Pareto de $X^n = X \setminus \{x \in X \mid r_x < n\}$ y de esta forma podemos determinar las mejores soluciones de la población con la función $g(x) = r_x$. En el Algoritmo 4 se presenta un esquema algorítmico para determinar este ranking.

Este tipo de clasificación permite dar una idea de cuáles son las soluciones que debemos tratar de seguir mejorando. Las soluciones que no están en el frente de Pareto son las que

Algoritmo 4 Ranking de Pareto

$X_{restantes} \leftarrow X$
$i \leftarrow 0$
while $X_{restantes} \neq \emptyset$ **do**
  $C_{pareto} \leftarrow \emptyset$
  for $x \in X_{restantes}$ **do**
    $C_{pareto}.add(x)$
    for $x' \in X_{restantes} \setminus \{x\}$ **do**
      if $x' \prec_P x$ **then**
        $C_{pareto}.remove(x)$
        break
      end if
    end for
  end for
  for $x \in C_{pareto}$ **do**
    $r_x \leftarrow i$
    $X_{restantes}.remove(x)$
  end for
  $i \leftarrow i + 1$
end while

ya han sido objetivamente superadas con respecto a la optimización multiobjetivo. Esta clasificación es muy útil cuando la dimensión del problema multiobjetivo, es decir, el número de problemas sobre el que estamos optimizando es bajo. Sin embargo, cuando esta dimensión aumenta, el frente de Pareto pasa de ser una curva a ser una superficie o una variedad de mayor dimensión, de forma que existen cada vez más puntos de Pareto y determinar la clasificación tiene un coste mucho mayor.

4.3. Hiperheurística para la Metaoptimización

El objetivo final de este capítulo es implementar una hiperheurística basada en las técnicas estudiadas para determinar los mejores parámetros de la metaheurística Particle Swarm Optimization, presentada en la Sección 2.2.1, para que pueda ser aplicada sobre problemas generales. La hiperheurística utilizada se basa en algoritmos genéticos (véase Sección 2.2.2), donde cada cromosoma representa una selección de parámetros del PSO y, por tanto, codifica al PSO asociado a estos parámetros. El problema recibe una serie de funciones de entrenamiento definidas en el Apéndice A. Estos problemas de entrenamiento son también empleados en [20,21,27]. Un análisis más en profundidad sobre algunas de estas funciones de entrenamiento puede encontrarse en [12].

La hiperheurística utilizada consta de una población de cromosomas que son evolucionados en cada iteración mediante selección por torneo, recombinación uniforme y reemplazo con elitismo. En cada iteración, para cada cromosoma y cada función de entrenamiento, se ejecuta el PSO codificado por el cromosoma para obtener una solución para dicho problema. Esta ejecución se hace para todos los cromosomas en cada iteración, a pesar de que el

cromosoma provenga de una generación anterior. La repetición de la ejecución del PSO asociado permite recalcular el valor de este para evitar que un cromosoma, de forma aleatoria, obtenga una solución muy superior a lo que dicho algoritmo suele obtener y se reproduzca de forma indefinida "contaminando" el resto de cromosomas. Esta repetición de cálculos además premia a las elecciones de parámetros más consistentes, pues para perdurar durante varias generaciones debe obtener siempre soluciones de buena calidad.

El hecho de aplicar varias funciones de entrenamiento distintas obliga al algoritmo a enfrentarse a un problema de optimización multiobjetivo (véase Sección 4.2). El criterio utilizado para seleccionar las mejores soluciones de cada generación es el ranking medio. Una vez calculada la ejecución de cada PSO asociado a un cromosoma x sobre cada función, se ordenan los parámetros según sus resultados sobre cada función de entrenamiento f, obteniendo el rango r_f^x. Posteriormente, se calcula el rango medio para cada x, y este es el escalar que permite realizar la selección. Este rango medio debe recalcularse en cada iteración, pues su valor es contextual a la población, es decir, depende de los valores del resto de cromosomas de la misma generación.

Finalmente, la nueva generación es remplazada con un número de elitismo pequeño. Esto permite garantizar que no se pierde calidad. Además, como fue explicado anteriormente, permite premiar la consistencia de una serie de parámetros, lo cual es un característica muy deseable de las metaheurísticas no deterministas.

Una primera aproximación consistía en realizar un estudio cualitativo del valor de dichos parámetros. Es decir, encontrar los parámetros que globalmente encontrasen mejores soluciones. En este tipo de estudio, no tiene sentido tratar de buscar el mejor número de partículas o el mejor número de iteraciones puesto que incrementar estas cantidades nunca puede tener ningún efecto negativo sobre la calidad de la solución del problema. Por esta razón, incrementarlo siempre encontraría mejores soluciones y estas cantidades tenderían a infinito. Más aún, el hecho de que estas cantidades crezcan independientemente del resto de parámetros puede contaminar las soluciones ya que peores parámetros podrían encontrar mejores soluciones empleando más tiempo (es decir, más iteraciones y partículas).

Sin embargo, este estudio no tiene por que realizarse únicamente desde el punto de vista puramente cualitativo. En ocasiones, puede que estos algoritmos vayan a ser utilizados muchas veces dentro de otra estrategia de alto nivel, como por ejemplo en el ajuste de pesos de una red neuronal. Para estas situaciones, el tiempo de ejecución empieza a tener un peso mayor, por lo que es interesante encontrar los coeficientes que aseguren una convergencia más rápida, a pesar de que esta sea menos fina. Debido a las restricciones de tiempo, en esta situación sí tiene mayor sentido tratar de encontrar los mejores valores para el número de partículas y el número de iteraciones. ¿Qué será mejor, aumentar el número de iteraciones para aumentar la intensificación o aumentar el número de partículas para mejorar la diversificación?

Para ello, cada cromosoma consta de cuatro parámetros: el coeficiente de inercia w, el coeficiente cognitivo φ_1, el coeficiente social φ_2 y el número de partículas. La ejecución de cada PSO se realiza durante tantas iteraciones como tenga tiempo a ejecutar en una cantidad

de tiempo determinada por el usuario. Esto tiene dos principales ventajas: Permite obtener algoritmos con la convergencia inicial más rápida. Las partículas de los algoritmos con coeficientes más bajos se desplazarán de forma más lenta pero más intensiva sobre al espacio de búsqueda. Requerirán de más iteraciones para alcanzar soluciones buenas, pero una vez las encuentren son capaces de mejorarlas, mientras que unos coeficientes más grandes permiten un desplazamiento muy veloz en las primeras etapas, aunque menor capacidad de intensificación. En función del tiempo determinado, se verá beneficiada una estrategia u otra, de manera que la propia hiperheurística será capaz de encontrar el equilibrio para alcanzar las mejores soluciones en esa cantidad de tiempo. La segunda ventaja es que el tiempo que va a emplear la metaoptimización es conocido a priori, este tiempo vendrá dado aproximadamente por la multiplicación del número de iteraciones de la estrategia de alto nivel, el número de partículas de la estrategia de alto nivel, el número de funciones de entrenamiento empleadas y esta cantidad de tiempo determinada.

4.4. Resultados

Para la ejecución del algoritmo se han utilizado 50 cromosomas con coeficientes inicializados aleatoriamente: $w, \varphi_1, \varphi_2 \sim U(-1, 1)$; $n \sim U(5, 150)$ y 50 iteraciones. Se han calculado los mejores coeficientes para $t = 0.1$ y $t = 0.5$, con 3, 5 y 10 dimensiones de la función asociada al PSO, para comprobar si los coeficientes son constantes o difieren mucho con respecto del tiempo o el número de dimensiones. En las tablas C.1 y C.2 se muestran los resultados obtenidos para $t = 0.1$ y $t = 0.5$ respectivamente.

La intuición nos guía a suponer que los coeficientes del PSO deberían ser números en el intervalo $[0, 1]$. Sin embargo, para estos experimentos no se ha puesto ninguna limitación sobre los coeficientes, a excepción de que el número de partículas sea un natural mayor que 1. De esta forma, los coeficientes pueden llegar a alcanzar valores mayores que 1 o incluso negativos mediante la mutación. Cada vez que un elemento es mutado, se le añade a cada coeficiente continuo un valor $r \sim N(0, 0.5)$ y al número de partículas se le suma un entero aleatorio entre -10 y 10. El objetivo de esta decisión es comprobar si la intuición es correcta y corresponde con los mejores intervalos obtenidos en la práctica. Además, esta libertad permite obtener unos resultados que reduzcan sesgo humano.

Todas las funciones que han sido utilizadas para el entrenamiento y muestreo aparecen en el Apéndice A. En este experimento, las seleccionadas como entrenamiento han sido las funciones de *Rosenbrock*, *Griewank*, *Ackley*, *Schwefell1-2*, *Schwefell2-22* y la función *Step* (esta última aporta un entrenamiento sobre funciones discontinuas, lo cual puede resultar muy interesante para el algoritmo). El resto de funciones han sido utilizadas para evaluar la calidad de estos parámetros. Para ello, se han realizado 50 ejecuciones de cada una con los parámetros obtenidos en el tiempo determinado. La elección de estas funciones se debe a que la función de *Rastrigin* es la más complicada de optimizar por medio de PSO debido a sus numerosos mínimos locales y por tanto servirá de medida de la calidad de los coeficientes. Mientras que las funciones *Esfera* y *Schwefell2-21* son las más sencillas de minimizar, por lo que utilizarlas dentro del entrenamiento puede contaminar la muestra o aumentar el

tiempo de muestreo sin aportar demasiado al entrenamiento, aunque a pesar de ello pueden ser utilizadas para comprobar la diferencia con otros parámetros.

La hipótesis de partida es que, en una cantidad de tiempo pequeña, se favorecerá principalmente la diversificación, es decir, coeficientes mayores en valor absoluto, que permitan un desplazamiento más rápido. Mientras que a medida que el algoritmo disponga de más tiempo, se favorecerá más la intensificación, con movimientos ligeros pero mayor número de iteraciones. Por otro lado, el número de partículas se verá limitado por el tiempo, por lo que a tiempos menores se encontrarán menor número de partículas. En cualquier caso tiene interés evaluar la relación entre el número de iteraciones y el número de partículas. Cuanto mayor sea el número de iteraciones con respecto al de partículas, se estará premiando en mayor medida la búsqueda realizada por el algoritmo, mientras que cuando este sea menor, se premiará más el azar que genera la diversificación en las soluciones iniciales.

En cuanto a la dimensión, un mayor número de dimensiones amplifica en gran medida el espacio de búsqueda, a la vez que también aumenta el tiempo empleado en la evaluación de la función objetivo. Por lo tanto el número de iteraciones será menor. Ante esta situación, es posible que el algoritmo dependa más de la diversificación, ya que será menos probable poder escapar de mínimos locales.

Por último, es muy probable que no exista una configuración de parámetros absolutamente superior, si no que exista un conjunto de buenas configuraciones en las que ninguna sobresale en gran medida de las demás. Esto se debe a que con tiempos tan reducidos, el algoritmo tiene un gran componente de aleatoriedad. Un estudio que puede ser realizado consistiría en ejecutar múltiples veces el algoritmo, obteniendo una muestra experimental de este conjunto y aplicando técnicas de estadística o relanzando el algoritmo a partir de los cromosomas dados por estas configuraciones.

A la vista de los resultados, se observa que aparecen coeficientes bastante dispares, aunque se observa una relativa similaridad entre algunos de ellos. Es posible que estas congifuraciones similares se agrupen en distintos vecindarios con una diferencia de calidad muy pequeña. Ejecutando el algoritmo un gran número de veces, se podría obtener un conjunto mayor de datos sobre el que aplicar técnicas para analizar esta similaridad. Del mismo modo, estos datos pueden reintroducirse como la población inicial del algoritmo para que este vuelva a evolucionar los coeficientes para acabar seleccionando aquellos que mejor comportamiento presenten.

Por otro lado, se puede comprobar como a medida que aumenta la dimensión del problema, se incrementa también el número de partículas empleado. Esto se realiza a costa de poder ejecutar un menor número de iteraciones y por tanto disminuir esta relación. A medida que aumenta la dimensión, aumenta el tamaño del espacio de búsqueda y es más importante priorizar la diversificación. Además, se observa que cuando aumenta la dimensión, el coeficiente de inercia aumenta considerablemente. Esto permite a las partículas avanzar más rápido para alcanzar zonas de buena calidad en un menor número de iteraciones, a costa de reducir la capacidad de intensificación una vez todas las partículas se han acercado suficiente. También se observa que el aumento de la dimensión tiene grandes consecuencias en

los resultados de las funciones de muestra, principalmente en la de *Rastrigin*. Cuanto mayor es el número de dimensiones, resulta mucho más complicado encontrar el mínimo local de esta función, pues tiene un gran número de mínimos locales cerca del óptimo.

En cuanto a la diferencia en función del tiempo, los coeficientes son ligeramente inferiores, pero esta diferencia no es suficiente para confirmar la hipótesis pues los resultados son muy dispares y haría falta un mayor número de experimentos para corroborarlo. En cambio, la relación entre el número de iteraciones y el número de partículas si que ha aumentado significativamente, permitiendo explotar más las capacidades del algoritmo y dependiendo menos de la aleatoriedad en la inicialización. En cuanto a los límites de los parámetros, en la mayoría de casos la inercia se queda por debajo de 1, ya que de superar esta cantidad la solución explotaría. En los casos de los coeficientes social y cognitivo, en la mayor parte de las configuraciones estos parámetros exceden esta cantidad, desafiando a la intuición previa. Por otro lado, en todas las pruebas solo ha ocurrido un único caso en el que algún coeficiente ha tomado valores negativos, lo que sí que acompaña la intuición.

A continuación, se realizan 50 ejecución de PSO en los tiempos determinados con los coeficientes utilizados en la tabla 4.1 para la función de *Rastrigin* en distintas dimensiones, para poder realizar la comparativa con los resultados de las tablas C.1 y C.2 para esta misma función. Los resultados medios se pueden encontrar en la tabla 4.4.

A excepción de la configuración $w = 0.2, \varphi_1 = 0.7, \varphi_2 = 0.7$ para 3 dimensiones y 0.1s, los resultados obtenidos para la función de *Rastrigin* con los coeficientes obtenidos por la metaoptimización obtienen mejores resultados que las configuraciones de parámetros seleccionados arbitrariamente de la tabla 4.4. Esta diferencia se hace mucho mayor a medida que aumenta la dimensión. Estos resultados confirman que los parámetros encontrados son buenos y que además estos pueden ser utilizados con éxito de forma Offline para problemas con los que no ha sido entrenado, ya que la función de *Rastrigin* no estaba dentro del entrenamiento. Aquí se demuestra la utilidad que puede tener un algoritmo de metaoptimización como este para garantizar los mejores resultados posibles cuando se implementa otra hiperheurística.

Para concluir, cabe destacar que este experimento puede ser realizado con mínimos cambios para distintos criterios. Otra opción hubiese sido realizar la metaoptimización para un número fijo de partículas y de iteraciones. Sin embargo, considero la aproximación tratada en este trabajo más interesante ya que también permite evolucionar el número de partículas, y por tanto también se puede realizar la comparación sobre la relación entre estas dos cantidades, que, de otra forma, quedarían impuestas por los criterios del algoritmo.

$t = 0.1$			
Dimensiones	$w = 1, \varphi_1 = 1, \varphi_2 = 1$	$w = 0.3, \varphi_1 = 0.8, \varphi_2 = 0.2$	$w = 0.2, \varphi_1 = 0.7, \varphi_2 = 0.7$
3	2.2060	1.6148	0.8537
5	13.3614	8.5262	3.4605
10	60.2619	40.2102	12.5855
$t = 0.5$			
Dimensiones	$w = 1, \varphi_1 = 1, \varphi_2 = 1$	$w = 0.3, \varphi_1 = 0.8, \varphi_2 = 0.2$	$w = 0.2, \varphi_1 = 0.7, \varphi_2 = 0.7$
3	1.7184	0.8877	0.69649
5	11.3883	5.0898	3.06576
10	56.3373	31.1626	10.7128

Tabla 4.2: Valores encontrados por PSO para Rastrigin con 100 partículas en 0,1s y 0,5s.

Capítulo 5

Conclusiones y futuras líneas de investigación

En este trabajo se ha introducido el campo de las hiperheurísticas, una rama de la inteligencia artificial que tiene la capacidad de dar un gran nivel de generalidad gracias a operar sobre un espacio de búsqueda de propios algoritmos. Además, se han implementado algunas hiperheurísticas para resolver con éxito los problemas planteados por la competición ITC2007 y para obtener buenas configuraciones de parámetros para otras metaheurísticas en un tiempo determinado, que posteriormente pueden ser aplicables sobre otras estrategias de alto nivel.

Para lograr estos objetivos ha sido necesario investigar y aprender sobre técnicas metaheurísticas e hiperheurísticas y modelización de problemas, así como entender los principios de la metaoptimización y aplicar criterios de optimización multiobjetivo para poder implementar hiperheurísticas con aprendizaje Offline.

Sin embargo, este trabajo solo explora una pequeña parte de lo que puede llegar a alcanzarse mediante el uso de las hiperheurísticas y, por tanto, puede ser considerado como una introducción al mundo de la inteligencia artificial. Las técnicas aquí explicadas pueden ser también empleadas sobre otras estrategias de otras ramas de la inteligencia artificial y el diseño de algoritmos. Por lo que una futura línea de investigación puede pasar por explorar nuevas disciplinas y combinar los nuevos conocimentos con las hiperheurísticas descritas aquí. Por ejemplo, en [1] se implementa una hiperheurística generativa que toma como estrategia de alto nivel una red neuronal. Por otro lado, en [18], se toma una hiperheurística cuya eficiencia se pretende mejorar mediante la combinación de la estrategia de alto nivel junto a una red neuronal que permite decidir sobre qué elementos calcular la función objetivo, para poder así ahorrar estos cálculos que son tan costosos dentro de la hiperheurística. Finalmente, en [13] se explora la situación recíproca. En este último artículo se emplea una hiperheurística para tratar de mejorar los pesos asociados a una pila de redes de neuronas. Existen muchos métodos de combinar este tipo de estrategias y estos artículos pueden ser tomados como base para comenzar a explorar tanto el campo de las hiperheurísticas como el de las redes neuronales para futuros trabajos.

Por otro lado, también pueden considerarse líneas de investigación que no necesariamente

tienen que escapar de este ámbito, explorando nuevas aplicaciones de las hiperheurísticas. Por ejemplo, en [2,22] se implementan hiperheurísticas generativas para resolver problemas de Bin-Packing. Estas hiperheurísticas codifican heurísticas constructivas como un árbol, donde los nodos son operaciones y las ramas son los valores con los que operan, de forma que la heurística codificada decide en qué caja introducir cada elemento a partir de las operaciones de estos árboles. Posteriormente, estos árboles son evolucionados mediante algoritmos genéticos, produciendo nuevas heurísticas que escapan a la intuición humana. Una investigación por esta línea conllevaría un estudio en mayor profundidad de los algoritmos genéticos aplicados sobre árboles, algo que resulta muy interesante por la forma en la que es codificado un algoritmo para tratarlo como un ente matemático.

Finalmente, la conferencia PATAT ha organizado diferentes competiciones de Time-Tabling a lo largo del tiempo, tales como ITC2019 o ITC2021. Cada una estas competiciones explora un nuevo modelo de problemas, y todos ellos tienen muchas aplicaciones directamente relacionadas con el mundo real. Por ello, una futura línea de investigación puede ir de la mano de resolver estos problemas, mejorando las técnicas aquí empleadas mediante la hibridación de las heurísticas base o mediante hiperheurísticas selectivas que operen sobre la solución generada con la hiperheurística del Capítulo 3. Una forma de resolver estos problemas muy efectiva ha sido mediante Simulated Annaealing[10], y por ejemplo podría investigarse una forma de realizar la modelización por coloración de grafos para ITC2021 siguiendo [17] para combinar la hiperheurística GHH con las basadas en Simulated Annaealing en [9].

Bibliografía

[1] Mohamad Alissa, Kevin Sim, and Emma Hart. A Neural Approach to Generation of Constructive Heuristics. In *2021 IEEE Congress on Evolutionary Computation (CEC)*, pages 1147–1154, Kraków, Poland, June 2021. IEEE.

[2] Shahriar Asta, Ender Özcan, and Andrew J. Parkes. CHAMP: Creating heuristics via many parameters for online bin packing. *Expert Systems with Applications*, 63:208–221, November 2016.

[3] Christian Blum and Andrea Roli. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.

[4] Edmund Burke, Graham Kendall, Jim Newall, Emma Hart, Peter Ross, and Sonia Schulenburg. Hyper-Heuristics: An Emerging Direction in Modern Search Technology. In Fred Glover and Gary A. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57, pages 457–474. Kluwer Academic Publishers, Boston, 2003. Series Title: International Series in Operations Research & Management Science.

[5] Edmund K Burke, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and John R Woodward. A classification of hyper-heuristic approaches. *Handbook of metaheuristics*, pages 449–468, 2010.

[6] Edmund K. Burke, Barry McCollum, Amnon Meisels, Sanja Petrovic, and Rong Qu. A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research*, 176(1):177–192, January 2007.

[7] E.K. Burke, G. Kendall, and E. Soubeiga. A Tabu-Search Hyperheuristic for Timetabling and Rostering. *Journal of Heuristics*, 9(6):451–470, December 2003.

[8] Michael W Carter, Gilbert Laporte, and Sau Yan Lee. Examination Timetabling: Algorithmic Strategies and Applications. *Journal of the operational research society*, 47:373–383, 1996.

[9] Angelos Dimitsas, Christos Gogos, Christos Valouxis, Alexandros Tzallas, and Panayiotis Alefragis. A Pragmatic Approach for Solving the Sports Scheduling Problem. In *In Proc. 13th Int. Conf. Pract. Theory Autom. Timetabling*, volume 3, pages 195–207, 2022.

[10] Kathryn A. Dowsland, Eric Soubeiga, and Edmund Burke. A simulated annealing based hyperheuristic for determining shipper sizes for storage and transportation. *European Journal of Operational Research*, 179(3):759–774, June 2007.

[11] R. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pages 39–43, 1995.

[12] Steffen Finck, Nikolaus Hansen, Raymond Ros, and Anne Auger. Real-Parameter Black-Box Optimization Benchmarking 2010: Presentation of the Noiseless Functions. Technical report, Technical Report RR-6828, INRIA, France, 2010.

[13] Renata Furtuna, Silvia Curteanu, and Florin Leon. Multi-objective optimization of a stacked neural network using an evolutionary hyper-heuristic. *Applied Soft Computing*, 12(1):133–144, January 2012.

[14] Mario Garza-Fabre, Gregorio Toscano Pulido, and Carlos A. Coello Coello. Ranking Methods for Many-Objective Optimization. In Arturo Hernández Aguirre, Raúl Monroy Borja, and Carlos Alberto Reyes Garciá, editors, *MICAI 2009: Advances in Artificial Intelligence*, pages 633–645, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[15] F. Glover and K. Sörensen. Metaheuristics. http://www.scholarpedia.org/article/Metaheuristics, 2015. [Online; descargado 15-Marzo-2023].

[16] Fred Glover and Manuel Laguna. *Tabu search I*, volume 1. Springer, January 1999. ORSA Journal on Computing.

[17] R. Lewis and J. Thompson. On the application of graph colouring techniques in round-robin sports scheduling. *Computers & Operations Research*, 38(1):190–204, January 2011.

[18] Jingpeng Li, Edmund K. Burke, and Rong Qu. Integrating neural networks and logistic regression to underpin hyper-heuristic search. *Knowledge-Based Systems*, 24(2):322–330, March 2011.

[19] Barry McCollum, Paul McMullan, Edmund K. Burke, Andrew J. Parkes, and Rong Qu. The Second International Timetabling Competition: Examination Timetabling Track. Technical report, Technical Report QUB/IEEE/Tech/ITC2007/-Exam/v4. 0/17, Queen's University, 2007.

[20] Magnus Erik Pedersen. Good Parameters for Particle Swarm Optimization. Technical report, Hvass Lab., Copenhagen, Denmark, Tech. Rep. HL1001, 2010.

[21] M.E.H. Pedersen and A.J. Chipperfield. Simplifying Particle Swarm Optimization. *Applied Soft Computing*, 10(2):618–628, March 2010.

[22] Nelishia Pillay. A Study of Evolutionary Algorithm Selection Hyper-Heuristics for the One-Dimensional Bin-Packing Problem. *South African Computer Journal*, 48, June 2012.

[23] R Qu and E K Burke. Hybridizations within a graph-based hyper-heuristic framework for university timetabling problems. *Journal of the Operational Research Society*, 60(9):1273–1285, September 2009.

[24] Nasser R. Sabar, Masri Ayob, Rong Qu, and Graham Kendall. A graph coloring constructive hyper-heuristic for examination timetabling problems. *Applied Intelligence*, 37(1):1–11, July 2012.

[25] Wikipedia. Metaoptimización — wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/MetaoptimizaciÃşn, 2023. [Internet; descargado 20-mayo-2023].

[26] Wikipedia contributors. Friedman test — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Friedman_test&oldid=1137467078, 2023. [Online; descargado 20-junio-2023].

[27] Xin Yao, Yong Liu, and Guangming Lin. Evolutionary programming made faster. *IEEE Transactions on Evolutionary Computation*, 3(2):82–102, July 1999.

[28] Xin-She Yang. Metaheuristic Optimization - Scholarpedia. http://www.scholarpedia.org/article/Metaheuristic_Optimization, 2023. [Online; descargado 15-Marzo-2023].

Apéndice A

Funciones de Entrenamiento

Las funciones de entrenamiento utilizadas para la hiperheurística de metaoptimización del Capítulo 4 fueron obtenidas de [12,20,27] y son las siguientes:

- **Función Esfera.**

$$f_{esfera} : \mathbb{R}^n \to \mathbb{R}$$
$$(x_1, \ldots, x_n) \mapsto \sum_{i=1}^{n} x_i^2$$

- **Función Ackley.**

$$f_{Ackley} : \mathbb{R}^n \to \mathbb{R}$$
$$(x_1, \ldots, x_n) \mapsto e + 20 - 20 \cdot e^{\left(-0.2 \cdot \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right)} - e^{\left(\frac{1}{n}\sum_{i=1}^{n} \cos(2\pi x_i)\right)}$$

- **Función Griewank.**

$$f_{Griewank} : \mathbb{R}^n \to \mathbb{R}$$
$$(x_1, \ldots, x_n) \mapsto 1 + \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right)$$

- **Función Rastrigin.**

$$f_{Rastrigin} : \mathbb{R}^n \to \mathbb{R}$$
$$(x_1, \ldots, x_n) \mapsto \sum_{i=1}^{n} \left(x_i^2 + 10 - 10 \cdot \cos\left(2\pi x_i\right)\right)$$

- **Función Rosenbrock.**

$$f_{Rosenbrock} : \mathbb{R}^n \to \mathbb{R}$$
$$(x_1, \ldots, x_n) \mapsto \sum_{i=1}^{n-1} \left(100 \cdot \left(x_{i+1} - x_i^2\right)^2 + \left(x_i - 1\right)^2\right)$$

- **Función Schwefel1-2.**

$$f_{Schwefel1-2} : \mathbb{R}^n \to \mathbb{R}$$
$$(x_1, \ldots, x_n) \mapsto \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j \right)^2$$

- **Función Schwefel2-21.**

$$f_{Schwefel2-21} : \mathbb{R}^n \to \mathbb{R}$$
$$(x_1, \ldots, x_n) \mapsto \text{máx}\{|x_i| : i \in \{1, \cdots, n\}\}$$

- **Función Schwefel2-22.**

$$f_{Schwefel2-22} : \mathbb{R}^n \to \mathbb{R}$$
$$(x_1, \ldots, x_n) \mapsto \sum_{i=1}^{n} |x_i| + \prod_{i=1}^{n} |x_i|$$

- **Función Step.**

$$f_{Step} : \mathbb{R}^n \to \mathbb{R}$$
$$(x_1, \ldots, x_n) \mapsto \sum_{i=1}^{n} (|x_i + 0.5|)^2$$

Apéndice B

Resultados de los problemas de Time-Tabling

B.1. Resultados de las heurísticas seleccionando el mejor color

	LD	LS	LCD	GR	LWD	LE	R
set1	-	7066	6382	-	-	7871	-
set2	665	550	575	687	623	1438	1030
set3	12452	13119	13788	13066	14224	15318	-
set4	-	-	-	-	-	-	-
set5	5104	4632	4512	4666	-	-	16628
set6	-	-	-	32420	-	-	-
set7	5231	5372	5477	5705	5646	6090	7250
set8	12395	19686	16830	12733	19605	12406	14127
set9	1329	1353	1348	1322	1332	1278	1768
set10	-	-	-	-	-	-	-
set11	-	-	-	40320.0	-	-	-
set12	-	-	-	-	-	-	-

Tabla B.1: Resultados obtenidos por las heurísticas de coloración de grafos seleccionando el mejor color

B.2. Resultados de las heurísticas seleccionando el color mediante la ruleta

	LD	LS	LCD	GR	LWD	LE	R
	-	6695	6276	-	-	8691	-
set1	-	7502	-	-	8040	7988	-
	-	7561	6110	-	-	7803	-
	626	638	644	1256	869	634	1203
set2	639	696	622	655	694	1478	1267
	641	1116	634	782	660	685	1367
	12702	12524	12767	12765	-	-	-
set3	-	12212	11729	14036	14656	14082	-
	13350	13660	13319	12793	-	-	-
	-	-	-	-	-	-	-
set4	-	-	-	-	-	-	-
	-	-	-	-	-	-	-
	4752	4499	4364	4677	-	5825	-
set5	4846	4666	4763	4806	5122	7002	-
	5327	3925	4708	4895	5304	-	9470
	36945	-	-	41335	-	-	-
set6	-	-	-	-	-	-	-
	-	38965	37015	33160	-	-	-
	5894	5399	5837	5683	5704	5725	7923
set7	6140.0	6007	5913	5656	6039	5674	8058
	5877	5709	5960	5847	6045	5726	8427
	10793	19881	10337	10617	10888	10770	30236
set8	18952	10356	10681	10755	10777	11793	16870
	11309	10652	10706	10381	20091	11173	16412
	1366	1451	1342	1378	-	1486	2777
set9	1349	1273	1269	1379	1370	1620	3579
	1256	1434	1406	1379	1344	1305	3426
	-	21109	-	40940	-	-	-
set10	-	-	-	-	-	-	-
	-	28148	23729	-	-	-	-
	-	-	45108	83802	-	-	-
set11	-	-	43037	-	-	-	-
	-	-	-	52188	44460	-	-
	-	-	-	-	-	-	-
set12	-	-	-	-	-	-	-
	-	-	-	-	-	-	-

Tabla B.2: Resultados obtenidos por las heurísticas de coloración de grafos mediante la selección por ruleta

B.3. Ganadores de la competición ITC2007

En esta tabla se muestran los resultados medios de los ganadores de la competición ITC2007 y el número de veces que no encontraron solución factible. Estos datos se encuentran en el siguiente enlace.

	1^{er} Puesto	2º Puesto	3º Puesto	4º Puesto	5º Puesto
set1	4575 0	6064 0	9084 0	6671 0	12819 0
set2	414 0	1049 0	3669 0	623 0	3926 0
set3	10789 4	14134 0	19367 0	- 10	19812 0
set4	21639 4	20667 5	26347 5	- 10	25729 0
set5	3321 0	4229 0	4920 0	3858 0	11176 0
set6	27808 0	28078 2	29935 0	28155 7	34029 1
set7	4396 0	6760 0	11004 1	5432 0	19669 0
set8	7950 0	10809 0	14870 0	- 10	16721 0
set9	1085 0	1204 2	1936 0	1288 0	2277 0
set10	18581 0	- 10	15580 0	14778 0	20333 0
set11	34129 9	49861 0	- 10	- 10	44277 0
set12	6403 0	- 10	5542 0	- 10	7179 1

Tabla B.3: Resultados obtenidos por los ganadores de la competición ITC2007

B.4. Resultados de la hiperheurística

	Mejor Color		Ruleta	
	LeastSaturationDegree	Lista Aleatoria	LeastSaturationDegree	Lista Aleatoria
set1	6120	6347	6404	6139
set2	550	564	576	562
set3	11351	11494	11660	11659
set4	25359	26084	27763	28217
set5	4119	4065	4298	4232
set6	27185	27330	30195	29750
set7	5065	5196	5206	5448
set8	9782	9742	9977	9938
set9	1142	1122	1182	1200
set10	17175	17149	17248	17491
set11	32465	33384	36253	39256
set12	5897	5901	5830	5915

Tabla B.4: Resultados obtenidos por la hiperheurística según la lista inicial y la selección de color

Apéndice C

Resultados de la metaoptimización

C.1. Resultados para t = 0.1s

D	n	w	φ_1	φ_2	Rastrigin	Schweffel2-21	Esfera
3	19	0.2359	1.0806	1.8549	1.4214	6.34e-25	4.52e-80
	9	0.5699	1.6813	0.9039	1.0675	3.40e-30	9.18e-37
	18	0.3827	1.6366	1.3201	0.9153	3.13e-30	4.37e-62
	11	0.4631	2.0402	0.9813	0.6368	2.34e-05	3.74e-20
	25	0.1303	1.4545	1.7492	0.9983	7.08e-05	8.62e-58
	13	0.5548	1.1423	0.7646	1.0965	1.36e-05	7.15e-10
	10	0.4841	1.8191	1.2143	1.1817	4.20e-16	1.48e-81
	16	0.2134	1.1062	2.0343	1.7412	1.46e-09	1.28e-69
	60	0.0261	0.6483	1.7529	1.2735	1.90e-20	7.83e-43
	13	0.3458	1.7666	1.5961	1.1143	6.36e-30	1.70e-82
Media	19.4	0.3406	1.4376	1.4172	1.1447	1.08e-05	7.15e-11
5	51	0.4171	1.2718	1.0286	2.2683	1.55e-08	6.23e-18
	20	0.5354	1.6984	0.9452	1.6456	4.31e-4	1.17e-38
	43	0.3101	1.2711	1.3545	2.6197	0.0487	4.95e-29
	61	0.2766	1.3794	1.2730	2.0318	1.65e-3	1.97e-22
	64	0.2355	1.4994	1.3553	2.1151	0.0469	4.28e-13
	18	0.5672	1.5136	0.9412	2.2222	2.16e-4	1.45e-41
	20	0.5191	1.1737	1.4906	2.6622	3.87e-18	1.19e-34
	36	0.2685	1.4971	1.8766	2.0107	6.57e-13	1.13e-23
	26	0.3828	1.3514	1.4906	3.0595	3.71e-3	3.69e-31
	73	0.3083	1.4870	1.2326	1.7327	3.59e-05	8.59e-19
Media	36.6	0.3821	1.4125	1.2991	2.2368	0.0102	4.28e-14
10	101	0.4035	1.2001	1.1322	12.5645	0.6634	4.50e-3
	56	0.5397	1.8691	0.9726	9.3968	0.04138	2.69e-09
	75	0.5554	1.3581	0.8484	11.7164	0.1846	4.87e-07
	49	0.5443	1.6976	1.1078	8.8414	0.06671	1.18e-10
	47	0.4537	1.5082	1.4124	11.6794	0.4661	4.83e-06
	21	0.6999	1.3776	1.0370	11.9027	0.1158	3.77e-11
	33	0.5544	1.6997	1.1543	10.9193	0.3030	3.51e-07
	90	0.3992	1.1402	1.3406	12.2327	0.5384	5.88e-05
	22	0.5548	2.0549	1.3023	8.5410	0.09645	1.90e-13
	33	0.5575	1.6204	1.2419	10.5797	0.1176	6.69e-12
Media	52.7	0.5263	1.5526	1.1561	10.8373	0.2593	4.56e-4

Tabla C.1: Coeficientes y sus resultados sobre las funciones de muestra para $t = 0.1$

C.2. Resultados para t = 0.5s

D	n	w	φ_1	φ_2	Rastrigin	Schweffel2-21	Esfera
3	16	0.5759	1.1834	0.5895	0.2985	2.72e-4	1.56e-220
	19	0.4674	1.3565	0.9088	0.5771	3.30e-134	1.40e-313
	21	0.3808	1.9879	1.1050	0.3581	1.61e-77	9.49e-262
	20	0.5232	1.7851	0.8371	0.1591	3.54e-89	2.75e-194
	64	0.1592	2.1178	1.2775	0.1392	1.72e-30	2.11e-155
	39	0.2711	1.4912	1.1095	0.3234	4.86e-78	1.41e-173
	25	0.3745	1.6319	0.9536	0.1790	1.03e-93	6.05e-05
	16	0.5539	1.5169	0.5893	0.1396	1.74e-36	3.59e-254
	41	-0.0180	-0.8191	1.9333	1.8004	2.22e-10	2.45e-156
	15	0.5722	1.8455	0.5747	0.1193	4.80e-07	8.55e-227
Media	27.6	0.3859	1.4097	0.9878	0.4094	2.72e-05	6.05e-06
5	29	0.4009	1.6658	1.5623	2.3481	1.99e-58	3.43e-124
	18	0.6315	1.8235	0.9082	1.2735	5.33e-54	1.12e-129
	18	0.4685	2.2473	1.2886	1.9501	2.66e-07	7.97e-136
	22	0.6850	1.3494	0.8283	1.5322	1.74e-45	3.30e-93
	85	0.4323	1.2664	0.9225	1.2339	3.30e-29	2.16e-62
	39	0.5080	1.3619	1.1054	2.3879	3.36e-45	3.79e-91
	21	0.4783	1.7160	1.4114	2.2088	2.44e-13	1.25e-143
	54	0.1264	0.9584	2.2636	2.6688	9.75e-21	9.39e-109
	39	0.5205	1.1760	1.1315	3.3629	8.58e-59	1.70e-101
	23	0.5455	2.0185	1.0562	1.3531	4.28e-57	1.72e-133
Media	34.8	0.4797	1.5583	1.2478	2.0319	2.66e-08	2.16e-63
10	94	0.6583	1.2519	0.8079	8.0272	8.61e-12	6.55e-24
	66	0.7035	1.3681	0.6632	5.3066	4.41e-4	3.54e-32
	122	0.6121	1.7230	0.9379	5.3626	6.79e-09	1.11e-18
	80	0.6387	1.0402	1.1816	11.6211	2.36e-12	7.05e-28
	75	0.6082	1.4002	1.1604	7.9003	1.810e-13	3.79e-31
	49	0.7028	0.9783	1.1028	12.9543	1.21e-08	3.15e-33
	52	0.7065	1.0485	1.0348	12.0986	5.01e-10	7.14e-34
	112	0.5647	1.5776	0.8574	5.4601	5.51e-4	1.13e-32
	34	0.6658	1.5520	1.2037	8.6174	2.35e-16	8.61e-45
	25	0.8425	0.7102	0.8564	13.6323	2.76e-16	8.91e-42
Media	70.9	0.6064	1.2695	0.9806	9.0980	9.92e-05	1.11e-19

Tabla C.2: Coeficientes y sus resultados sobre las funciones de muestra para $t = 0.5$