UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Algoritmo metaheurístico para la formación de equipos dentro del aula: aplicación y análisis de GRASP

Trabajo Fin de Grado

Grado en Ciencia de Datos

AUTOR/A: Lanza Méndez, Dylan Adolfo

Tutor/a: Sánchez Anguix, Víctor

Cotutor/a: Alberola Oltra, Juan Miguel

CURSO ACADÉMICO: 2024/2025

Resumen

En el mundo laboral, el trabajo en equipo es fundamental para abordar proyectos de manera eficaz. Esto se debe a que, por lo general, una sola persona no es suficiente para manejar todas las facetas de un proyecto y, en consecuencia, la ejecución del mismo sería menos efectiva que la que se lograría con un grupo de personas trabajando juntas. Por ello, con el fin de adaptar a los estudiantes a las tendencias del mundo laboral en la resolución de problemas, muchas titulaciones de educación superior promueven la realización de actividades o proyectos grupales para desarrollar las competencias colaborativas necesarias en los alumnos. Sin embargo, para fomentar el desarrollo adecuado de estas competencias, lo ideal sería agrupar a los estudiantes de forma estratégica, siguiendo ciertos criterios (p.ej., la teoría de roles de Belbin, la diversidad de géneros, etc.) que aumenten la probabilidad de que los equipos funcionen bien y permitan que una experiencia colaborativa fructífera favorezca la adquisición de dichas competencias. Esto no quita que formar equipos sea una tarea compleja de realizar manualmente, debido a la gran cantidad de combinaciones posibles que habría que analizar para identificar los mejores equipos. De hecho, incluso desde un punto de vista algorítmico, la resolución del problema también puede ser computacionalmente costosa si no se emplean algoritmos eficientes. Por esta razón, en este trabajo se exploran alternativas como las metaheurísticas, que suelen ser métodos algorítmicos más rápidos a la hora de generar respuestas. Entre las metaheurísticas estudiadas se encuentran un algoritmo genético y GRASP, siendo esta última un método poco utilizado hasta ahora para abordar el problema. Además, parece haber una falta de estudios comparativos sobre el rendimiento de ambas metaheurísticas en la formación de equipos, por lo que en este trabajo se realiza dicha comparación. Para evaluar el rendimiento de estas metaheurísticas en la formación de equipos, se emplean distintos tipos de instancias (*i.e.*, tamaños de aula) con el objetivo de extraer conclusiones adecuadas sobre la calidad de las soluciones que son capaces de generar, así como sobre su escalabilidad temporal. Tras analizar su rendimiento, los resultados demuestran que ambos algoritmos generan soluciones igualmente satisfactorias en menos de un minuto, siendo GRASP la que las produce prácticamente de forma instantánea (*i.e.*, alrededor de cinco segundos), superando de manera significativa al algoritmo genético en términos de eficiencia.

Palabras clave: formación de equipos en el aula, metaheurísticas, GRASP, algoritmos genéticos

Abstract

In the world of work, teamwork is essential for dealing with projects effectively. This is because one person is usually not sufficient to handle all facets of a project and, as a result, project delivery would be less effective than would be achieved with a group of people working together. Therefore, in order to adapt students to the problem-solving trends of the world of work, many higher education degrees promote group activities or projects to develop the necessary collaborative competences in students. However, to foster the proper development of these competences, students should ideally be grouped strategically, following certain criteria (e.g. Belbin's role theory, gender diversity, etc.) that increase the likelihood of well-functioning teams and allow for a successful collaborative experience to foster the acquisition of these competences. This does not detract from the fact that team formation is a complex task to perform manually, due to the large number of possible combinations that would need to be analysed in order to

identify the best teams. In fact, even from an algorithmic point of view, solving the problem can also be computationally expensive if efficient algorithms are not used. For this reason, this paper explores alternatives such as metaheuristics, which tend to be faster algorithmic methods for generating answers. Among the metaheuristics studied are a genetic algorithm and GRASP, the latter being a method little used so far to address the problem. Moreover, there seems to be a lack of comparative studies on the performance of both metaheuristics in team formation, so this paper makes such a comparison. To evaluate the performance of these metaheuristics in team formation, different types of instances (i.e., classroom sizes) are used in order to draw appropriate conclusions about the quality of the solutions they are able to generate, as well as their temporal scalability. After analysing their performance, the results show that both algorithms generate equally satisfactory solutions in less than a minute, with GRASP producing them practically instantaneously (i.e., around five seconds), significantly outperforming the genetic algorithm in terms of efficiency.

Keywords : team formation in the classroom, metaheuristics, GRASP, genetic algorithms

Tabla de contenidos

1. Introducción

El trabajo en equipo constituye una práctica esencial para abordar proyectos de diversa complejidad que requieren la colaboración de múltiples personas. Lo que podría resultar complicado para un individuo trabajando solo se vuelve más manejable y eficiente cuando se cuenta con un equipo dedicado. Mejorar el rendimiento y la productividad en la ejecución de tareas son beneficios directos del trabajo en equipo, lo cual facilita que los objetivos establecidos se alcancen con éxito. Por todo ello, la formación de profesionales de cualquier ámbito o disciplina en titulaciones de educación superior suele enfocarse no solo en la enseñanza de conocimientos específicos y competencias técnicas (*hard skills*), sino también en el desarrollo de competencias personales (*soft skills*) tales como el trabajo en equipo. Un ejemplo destacado de institución cuya metodología docente está orientada también al desarrollo de dichas competencias es la Universidad Politécnica de Valencia (UPV). Entre las diferentes competencias que la UPV considera que sus estudiantes deben adquirir, el «trabajo en equipo y liderazgo» [1] es una de ellas, y para su desarrollo los alumnos son puestos en práctica para colaborar con otros compañeros de clase en diferentes tareas o proyectos a lo largo de su estancia en la universidad.

Al estar bajo a un ambiente de trabajo colaborativo, los alumnos aprenden a asumir responsabilidades y a contribuir a la mejora y desarrollo colectivo [1]. Al mismo tiempo, se van familiarizando con las dinámicas comunes del trabajo en equipo, como el intercambio de ideas, la división de tareas o la planificación del tiempo. La adaptación a estas prácticas por parte de los estudiantes a lo largo de su formación profesional es fundamental para que, una vez inmersos en el mundo laboral, estén preparados para colaborar con soltura y coordinación con otras personas, y así llevar a cabo diversas responsabilidades asignadas.

Para favorecer la adquisición de estas habilidades colaborativas, es importante agrupar adecuadamente a los alumnos en equipos. Por tanto, la definición de una estrategia o criterio para formar equipos eficaces es crucial. Aunque pueda ser discutible el hecho de que trabajar en equipos que no sean tan eficaces podría preparar a los estudiantes para su «vida laboral real», en contextos educacionales la agrupación no planificada de alumnos en equipos ineficaces podría comprometer su proceso de aprendizaje [2]. Esto se debe a que el trabajo en equipo, además de asociarse con la adquisición de habilidades colaborativas, también se relaciona con la aplicación práctica y el fortalecimiento de conocimientos específicos, e incluso con la adquisición de nuevos. En este sentido, si un alumno no puede desempeñarse correctamente debido a las malas dinámicas de un equipo ineficaz, esto podría impedir que se beneficie de una experiencia fructífera de colaboración en grupo. Por todo ello, queda fuera de lugar la aplicación de estrategias tradicionales y no elaboradas como, por ejemplo, dejar que los estudiantes se organicen en equipos por su cuenta, dado que este tipo de estrategias no aseguran la creación de equipos eficaces. En su lugar, la consideración de criterios más sofisticados, como la teoría de roles de Belbin [3] o la diversidad de géneros [4], representan estrategias más efectivas para aproximarse a la creación de equipos eficaces.

Sin embargo, a partir de un determinado número de alumnos en el aula, considerar todos los posibles equipos que se podrían crear con el objetivo de identificar cuáles cumplen mejor con los criterios establecidos supone un desafío considerable. Por un lado, realizar manualmente la búsqueda de los equipos más convenientes es inviable debido a la demora que implicaría todo el proceso. Por otro lado, la combinatoria ligada a la agrupación de los alumnos puede llegar a ser tan extensa que se convierte en un problema computacionalmente complejo para algunos métodos

algorítmicos, que encuentran dificultades al intentar encontrar la mejor solución en un tiempo razonable, es decir, la combinación óptima de equipos. Por ello, puede resultar conveniente estudiar la aplicación de metaheurísticas, las cuales son otros métodos algorítmicos ampliamente reconocidos por su capacidad para generar soluciones aceptables en tiempos no muy prolongados.

1.1. Objetivos

Los objetivos de este trabajo son:

- Definir los criterios que se utilizarán como base para guiar el proceso de formación de equipos potencialmente eficaces, en este caso, la teoría de roles de Belbin y la diversidad de géneros.
- Diseñar dos metaheurísticas distintas, en particular, GRASP y un algoritmo genético, encargadas de explorar y evaluar diferentes soluciones o combinaciones de equipos en función de los criterios establecidos.
- Estudiar el rendimiento de las metaheurísticas diseñadas en la resolución del problema, tanto por la calidad de las soluciones que son capaces de generar como por su escalabilidad temporal, considerando para ello distintos tipos de instancias (*i.e.*, tamaños de aula).
- A partir del estudio realizado, determinar, si es posible, aquel algoritmo cuyo rendimiento sea superior entre los dos, es decir, aquel que en general sea más efectivo (*i.e.*, genera soluciones de mayor calidad) y eficiente (*i.e.*, genera las soluciones en menor tiempo) para la formación de equipos en el aula.

1.2. Estructura del documento

Aparte de esta sección 1 de introducción, este documento contiene ocho secciones adicionales.

En la sección 2, se hace una revisión de diversos elementos de relevancia (optimización, metaheurísticas, criterios de formación de equipos, trabajos relacionados) que han contribuido al conocimiento necesario para poder desarrollar este trabajo y, por tanto, establecer la propuesta de estudio.

En la sección 3, se explica cómo se ha formulado el problema de la formación de equipos en el aula para poder abordarlo desde una perspectiva algorítmica, en este caso, mediante la aplicación de las dos metaheurísticas diseñadas (*i.e.*, GRASP y un algoritmo genético).

El diseño de ambas metaheurísticas, es decir, la descripción de cómo operan para generar soluciones al problema, se realiza en la sección 4.

En la sección 5 se describe el conjunto de datos disponible y qué tipos de instancias se han generado a partir de este. Estas instancias consisten en muestras de un número variado de alumnos que simulan el papel de aulas reales de distintos tamaños, utilizadas para evaluar el rendimiento de las metaheurísticas diseñadas en la formación de equipos.

El proceso experimental llevado a cabo con las metaheurísticas (utilizando las instancias generadas) para recopilar información relacionada con distintas medidas de su rendimiento (p.ej., la calidad de las soluciones que son capaces de generar para el problema) se detalla en la sección

6. Asimismo, en esta sección se analizan los resultados recopilados y finalmente se compara el rendimiento de ambas metaheurísticas.

En la sección 7 se reflexiona sobre el cumplimiento de los objetivos tras finalizar el trabajo, lo aprendido a partir de su desarrollo y la relación del trabajo con los estudios cursados. También se presentan propuestas de trabajo futuro.

Casi al final del documento, en la sección 8, se realiza un análisis del trabajo en relación con los aspectos legales y éticos asociados con la adquisición y el manejo de los datos disponibles para el desarrollo del trabajo, así como con la propiedad intelectual.

Por último, en la sección 9 correspondiente a las referencias, se proporcionan todas las fuentes de información e imágenes consultadas para la realización del documento.

2. Marco teórico

En esta sección se realiza una revisión de distintos temas, conceptos e investigaciones realizadas que fundamentan el estudio realizado en este trabajo. Primero, se trata el concepto de optimización de un problema (sección 2.1). Posteriormente, se explica qué son las metaheurísticas de optimización y cómo algunas de ellas funcionan específicamente (sección 2.2). Luego, se describen algunos de los criterios comunes que suelen considerarse para la creación de equipos potencialmente eficaces, destacando también que, independientemente del criterio elegido, el problema requiere ser resuelto mediante métodos algorítmicos de optimización eficientes (sección 2.3). Finalmente, se formaliza la propuesta de este trabajo en base a todo lo revisado (sección 2.4).

2.1. Optimización

Se entiende como optimización al proceso de búsqueda de la mejor solución a un problema. De acuerdo con la taxonomía de los problemas analíticos (figura 1), es decir, los problemas que pueden abordarse mediante el análisis de datos, la optimización es un proceso que permite resolver problemas clasificados dentro de la rama de la analítica prescriptiva. A los problemas de este tipo se pretende dar respuesta a una pregunta: ¿qué se debería hacer?

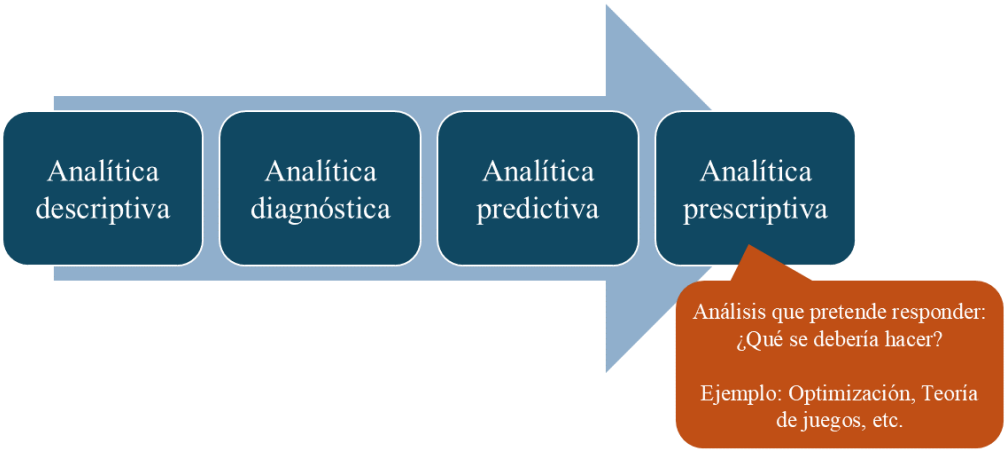

Figura 1. Taxonomía de los problemas analíticos [5]

Como ejemplo, se introduce a continuación el conocido problema de optimización combinatoria denominado problema de la mochila (*knapsack problem*) [6]. El problema parte de un conjunto de objetos o bienes que tienen un determinado valor y peso. El valor de un objeto puede ser una estimación numérica que la persona que tiene en su poder la mochila asigna al objeto, la cual puede reflejar lo importante que es dicho objeto para esa persona. Lo que se busca es incluir dentro de la mochila solamente aquellos objetos que en conjunto reporten un mayor valor a la persona, sin que la capacidad máxima de la mochila sea superada por el peso total de los objetos seleccionados. De este modo, no todos los objetos pueden ser incluidos en la mochila, por lo que se busca encontrar un subconjunto de todos los objetos que maximice el valor total y no sobrepase la capacidad de la mochila.

La figura 2 es una instancia sencilla del problema de la mochila. En este caso, los valores de los objetos están determinados mediante valores monetarios, y la capacidad total de la mochila es 15. Por tanto, una solución al problema podría ser incluir los objetos de color verde, azul y naranja, los cuales, en conjunto, no sobrepasan la capacidad máxima de la mochila. No obstante, la solución propuesta no tiene por qué ser necesariamente la mejor (*i.e.*, la que maximice el valor total). En realidad, dicha solución es simplemente una de las muchas combinaciones posibles de objetos que se podrían seleccionar y resolverían el problema (*i.e.*, no sobrepasarían la capacidad de la mochila). Por ello, el problema de la mochila es un problema combinatorio, e incluso uno de alta combinatoria, ya que el número total de combinaciones posibles aumenta exponencialmente con el número de objetos involucrados.

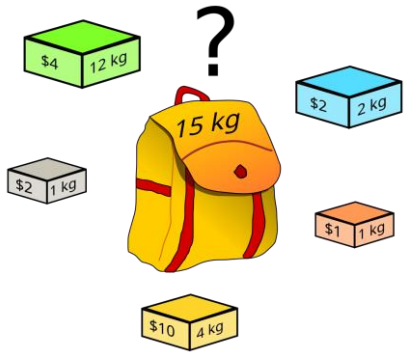

Figura 2. Instancia sencilla del problema de la mochila [7]

De este modo, problemas como el de la mochila son, precisamente, problemas en los que se quiere saber qué se debería hacer, o, en otras palabras, saber cuál es la mejor solución. La optimización da respuesta a este tipo de problemas. Realmente, la optimización también se puede ver como un proceso que da apoyo a la toma de decisiones, ya que encontrar la mejor solución a un problema permite, al mismo tiempo, tomar la mejor decisión. Alternativamente a mejor solución, el término «solución óptima» es un sinónimo del mismo. Llevar a cabo un proceso de optimización requiere de la modelización matemática del problema, para posteriormente emplear un algoritmo que se encargue de su resolución y, por tanto, encuentre la solución óptima al problema. La modelización matemática de un problema consta de los siguientes componentes:

- Parámetros: se tratan de los datos numéricos de entrada del modelo, sobre los cuales no se tiene control alguno.
- Variables de decisión: representan todas y cada una de las decisiones que se pueden tomar para la resolución del problema. Se expresan mediante letras. Estas, junto con los parámetros, se utilizan para diseñar otros componentes del modelo, como la función objetivo y las restricciones.
- Función objetivo: es una función matemática que define el aspecto del problema que se desea optimizar. Optimizar la función objetivo implica obtener el mayor o menor valor posible de esta. En cada caso, se habla respectivamente de maximización o minimización de la función objetivo.
- Restricciones: determinan cuáles decisiones son válidas y cuáles no. Se expresan mediante igualdades o desigualdades matemáticas ($=, \leq, \geq$).

A modo de ejemplo, a continuación se muestra cómo puede expresarse mediante un modelo matemático el problema de la mochila de la figura 2:

$$\max f = \sum_{i=1}^{n} v_i X_i$$

$$sujeto\ a:$$

$$\sum_{i=1}^{n} w_i X_i \leq W$$

$$con:$$

$$X_i: 1\ si\ se\ toma\ la\ decisión\ i, 0\ en\ el\ caso\ contrario;$$

$$X_i \in \{0,1\}, \qquad i = 1, \dots, n$$

Las variables de decisión están expresadas con la notación X_i, y cada una de ellas representa si la decisión i se toma o no. En otras palabras, una variable X_i representa si el objeto i se incluye en la mochila o no. Por ello, la variable está definida como una variable binaria que puede tomar únicamente los valores 0 (no se incluye) y 1 (se incluye). Los parámetros del modelo están expresados con las notaciones v_i, w_i y W, los cuales representan, respectivamente, el valor del objeto i, el peso del objeto i y la capacidad total de la mochila. A partir de X_i, w_i y W, se construye la única restricción que tiene el problema: $\sum_{i=1}^{n} w_i X_i \leq W$. Esta expresa correctamente que el peso total de los objetos seleccionados para ser incluidos en la mochila no debe sobrepasar la capacidad máxima de la misma. Por último, la función objetivo está definida mediante la fórmula $\max f = \sum_{i=1}^{n} v_i X_i$, a partir de la cual se busca maximizar el valor total de los objetos incluidos en la mochila. Por tanto, la mejor solución al problema estará compuesta por aquellas decisiones X_i que maximicen la función objetivo y, al mismo tiempo, cumplan con la restricción impuesta.

El modelo matemático planteado previamente es general, pero también se puede definir de una manera más directa, reduciendo levemente la notación utilizada. No obstante, en ambos casos se expresa exactamente lo mismo:

$$\max f = 4 \times X_1 + 2 \times X_2 + 1 \times X_3 + 10 \times X_4 + 2 \times X_5$$

$$sujeto\ a:$$

$$12 \times X_1 + 2 \times X_2 + 1 \times X_3 + 4 \times X_4 + 1 \times X_5 \leq W$$

$$con:$$

$$X_i: 1\ si\ se\ incluye\ el\ objeto\ i\ en\ la\ mochila, 0\ en\ el\ caso\ contrario;$$

$$X_i \in \{0,1\}, \qquad i = 1, \dots, 5$$

El problema de la mochila es conceptualmente sencillo, pero útil para introducir el concepto de optimización y el objetivo que se quiere lograr con este, es decir, obtener la solución óptima a un problema. Así, muchos de los problemas reales que se pueden presentar en las organizaciones pueden requerir de la aplicación de un proceso de optimización, por ejemplo, saber en qué invertir

en base a un determinado capital de manera que el beneficio de las opciones consideradas sea máximo; encontrar la manera de reducir el costo total de transporte de bienes desde varios puntos de suministro a varios puntos de demanda; planificar los turnos de trabajo del personal de manera que se cubran todas las necesidades y, al mismo tiempo, se minimicen los costos de pago; otros muchos problemas.

Una vez modelado matemáticamente un problema, un algoritmo es el encargado de su resolución. Existen diferentes tipos de algoritmos que pueden utilizarse. Algunos de ellos están diseñados únicamente para resolver problemas que, al modelizarse, cumplan con características específicas. Estas características están relacionadas con la estructura de la función objetivo definida, la estructura de las restricciones formuladas, el dominio de las variables de decisión (variables continuas, enteras, binarias, etc.) o, incluso, el número de funciones objetivo definidas, ya que puede haber casos en los que el problema que se aborda requiera de la definición de más de una función objetivo. El algoritmo *Simplex*, por ejemplo, está orientado a resolver modelos de programación lineal (modelos LP, *linear programming*), en los que la función objetivo debe expresarse como una combinación lineal, las restricciones como inecuaciones ($\leq, \geq$) y las variables de decisión pueden tomar valores continuos. Sin embargo, no está diseñado para resolver modelos de programación lineal entera (modelos ILP, *integer linear programming*), los cuales comparten características con los de programación lineal a excepción de las variables de decisión, dado que algunas o todas ellas deben tomar valores enteros. De resolver estos últimos modelos puede encargarse, por ejemplo, el algoritmo *Branch and Bound*, aunque también existen otros algoritmos. Por tanto, para resolver un determinado problema, es importante tener en cuenta sus características para elegir adecuadamente el algoritmo a utilizar.

Además, estos algoritmos de optimización, como *Simplex* o *Branch and Bound*, se encuentran implementados a través de lo que se conoce como *solvers*. Un *solver* es un *software* o librería matemática licenciada que puede contener diferentes algoritmos capaces de resolver problemas con diversas características. Existen tanto *solvers* comerciales como de código abierto. Google ORTools, por ejemplo, es una suite de código abierto que integra diferentes *solvers* para la resolución de problemas de optimización. Está escrita en C++, pero también se encuentra disponible en otros lenguajes de programación como Python, C# o Java [8].

Por último, cabe mencionar que existen dos tipos de métodos para llevar a cabo un proceso de optimización: exactos y aproximados. Los algoritmos implementados mediante *solvers* se catalogan como métodos exactos, ya que son capaces de obtener la solución óptima a un problema. Por otro lado, los métodos aproximados no garantizan obtener la solución óptima, pero suelen encontrar soluciones cercanas al óptimo, en un tiempo computacional generalmente menor que el requerido por un método exacto para obtener la solución óptima. Dentro de los métodos aproximados de optimización se encuentran las heurísticas y metaheurísticas.

2.2. Metaheurísticas

Previamente a definir lo que es una metaheurística, es necesario definir qué es una heurística. Una heurística es una estrategia no necesariamente racional que sirve para encontrar una solución a un problema, la cual no tiene por qué ser exactamente la mejor, pero que, sin embargo, puede ser decente. Por ejemplo, dado el problema de la mochila introducido en la sección 2.1, una estrategia para encontrar una solución podría ser seleccionar primero el objeto de mayor valor; posteriormente, elegir un segundo objeto cuyo valor sea el mayor entre todos los objetos restantes,

teniendo en cuenta que el peso de este, junto con el del primer objeto, no sobrepase la capacidad máxima de la mochila, y así sucesivamente. Una estrategia de este estilo se denomina concretamente heurística voraz, es decir, una heurística que, para crear una solución a un problema, consiste en empezar tomando la mejor decisión posible (incluir el objeto de mayor valor); luego, entre el resto de las decisiones disponibles, tomar la mejor de ellas sin incumplir las restricciones que puedan existir (de entre los objetos que aún no se han incluido, seleccionar el de mayor valor, sin sobrepasar la capacidad de la mochila), y así sucesivamente. Realmente, esta es una estrategia que cualquiera podría considerar para encontrar rápidamente una solución al problema. Se dice que nosotros las personas comúnmente aplicamos heurísticas en nuestro día a día para tomar decisiones o resolver problemas [5].

Definido el concepto de heurística, una metaheurística es simplemente una estrategia formada por la acción conjunta de varias heurísticas. Existen distintos tipos de metaheurísticas, de los cuales se hablará más adelante en esta misma sección. Estas, una vez implementadas, no dejan de ser algoritmos. Esta es una de las diferencias que existe entre las metaheurísticas y los métodos exactos de optimización. Recordemos que los métodos exactos son algoritmos (p.ej., Simplex, *Branch and Bound*, etc.) proporcionados a través de *solvers* para su uso, por lo que no es realmente necesario que una persona que desee aplicar un método exacto de optimización para resolver un problema desarrolle la estructura o esquema del algoritmo; únicamente es necesario que el usuario diseñe el modelo matemático del problema y lo transforme a un lenguaje entendible por el *solver*, de manera que este pueda resolver el modelo proporcionado como entrada. Por lo general, el esquema de operación de los algoritmos exactos es complejo y suelen encargarse grupos especializados de ingenieros para llevar a cabo su implementación [5], que finalmente se venden o proporcionan gratuitamente a través de los conocidos *solvers*. En cuanto a las metaheurísticas, muchas de ellas tienen un esquema de operación general, lo que hace que su implementación sea relativamente sencilla y las hace muy adaptables para su aplicación a prácticamente cualquier problema que se quiera resolver.

La modelización matemática de un problema para su resolución mediante el uso de una metaheurística es equivalente a la modelización que se podría realizar del mismo problema para que posteriormente sea resuelto por un algoritmo exacto. No obstante, dado que aplicar una metaheurística implica la codificación e implementación de un algoritmo, se deben tomar diferentes decisiones de diseño para poder resolver el problema planteado.

En primer lugar, la definición de una función objetivo se mantiene, que, como ya sabemos, recoge aquel aspecto del problema que se quiere maximizar o minimizar. Sin embargo, también puede recoger de alguna manera las restricciones que puedan existir para el problema y que determinan lo que es una solución factible (*i.e.*, válida) y lo que no lo es. Por ejemplo, dado el problema de la mochila, una manera de penalizar la generación de una solución no factible por parte de la metaheurística utilizada (cuando el peso de los objetos seleccionados supera la capacidad máxima de la mochila) puede ser determinar que el valor de la función objetivo de una solución así sea $-\infty$, el peor y más bajo valor que se puede obtener en un problema de maximización (∞ si el problema es de minimización). De este modo, se aseguraría que la metaheurística no proporcionara una solución no factible como resolución del problema.

Además de la función objetivo, también es necesario definir una forma de representar una solución al problema. Por ejemplo, una solución para el problema de la mochila se puede representar mediante un vector de ceros y unos como el que se muestra en la figura 3.

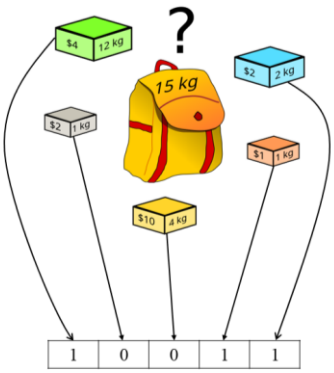

Figura 3. Representación binaria de una solución para el problema de la mochila. Elaboración propia parcial (imagen de la mochila extraída de [7])

Cada posición del vector se asocia con un objeto distinto, y el valor de una determinada posición indica si una decisión se toma o no, en este caso, si se incluye o no el objeto correspondiente. Por tanto, si el valor de una posición es 1, significa que ese objeto se selecciona para ser incluido en la mochila; si es 0, significa lo opuesto. De acuerdo con los objetos seleccionados que indica la solución de ejemplo, el valor de la función objetivo de esta es 7. Esta manera de representar una solución se conoce como representación binaria. Algunas otras representaciones pueden ser las que se muestran en la figura 4, entre las que se encuentran la representación mediante permutación (vector de elementos numéricos pertenecientes a $\mathbb{Z}$ que no se repiten), la representación entera (vector de elementos numéricos pertenecientes a $\mathbb{Z}$ que pueden aparecer repetidos más de una vez) y la representación flotante (vector de elementos numéricos pertenecientes a $\mathbb{R}$). La elección de una representación u otra debe hacerse dependiendo de las características del problema abordado (p.ej., la representación flotante no es adecuada para el problema de la mochila o no tiene mucho sentido).

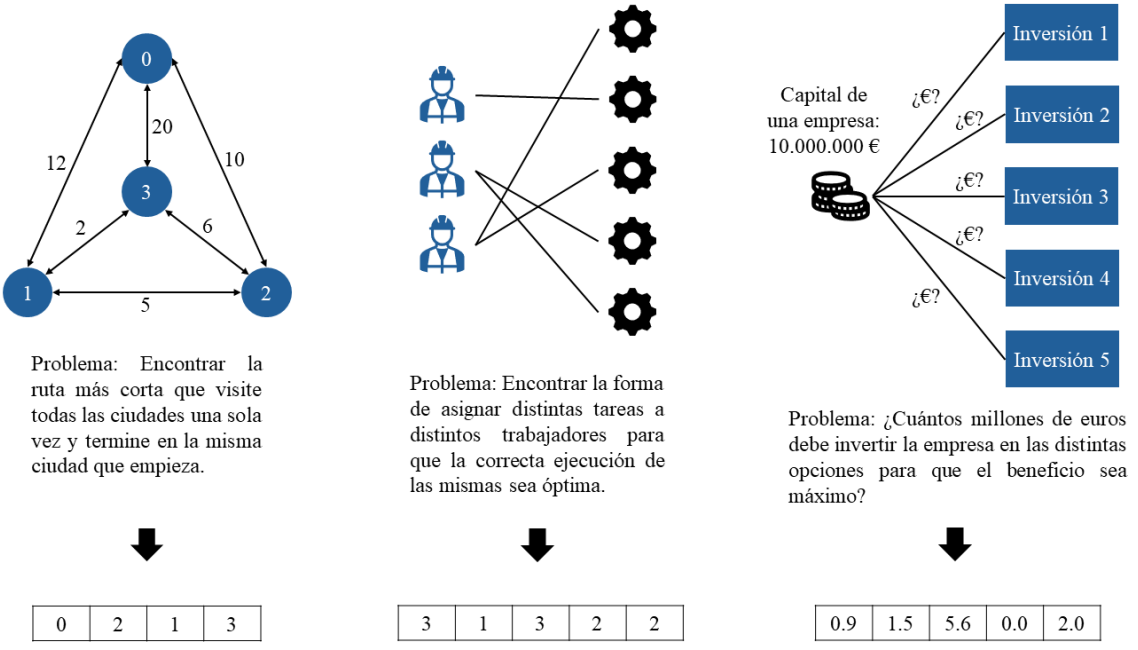

Figura 4. Representación permutación (izquierda), entera (centro) y flotante (derecha). Elaboración propia

Para la resolución de un problema, las metaheurísticas son capaces de generar una gran cantidad de soluciones a lo largo de su ejecución (recordemos que, al fin y al cabo, son algoritmos una vez implementadas). A esto también se le conoce como realizar una exploración del espacio de búsqueda o espacio de soluciones. La figura 5 ayuda a comprender mejor esto. Se trata de una representación gráfica hipotética de una función objetivo $f(x)$ definida para un problema cualquiera. El eje de abscisas representa las soluciones, es decir, supongamos que cada valor de x se corresponde con una solución distinta para el problema. Por tanto, cada solución tiene su propio valor de la función objetivo, representado por el eje de ordenadas. Los distintos puntos que se sitúan sobre la curvatura de la función son una muestra de todas las soluciones que pueden existir para el problema. Concretamente, los puntos azules se corresponden con mínimos o máximos globales, los naranjas con mínimos o máximos locales, y por último, los puntos grises con soluciones cuyo valor de la función objetivo no es ni mínimo ni máximo de ningún tipo. Por tanto, cuando una metaheurística es puesta en marcha para resolver un problema, lo más deseable es que, del muestreo que haga de todas las soluciones que existen para el problema, se encuentre aquella cuyo valor de la función objetivo sea máximo global si el problema es de maximización, o mínimo global si el problema es de minimización. En ambos casos, dicha solución sería óptima (*i.e.*, la mejor). No obstante, esto por lo general es difícilmente garantizable por una metaheurística, pero el hecho de que puede realizar un muestreo amplio (y con cierto grado de inteligencia) de todas las soluciones existentes para un problema es la razón por la cual la probabilidad de que termine encontrando una solución de buena o muy buena calidad (*i.e.*, el valor de la función objetivo que alcanza es mejor a la de muchas otras soluciones y puede ser cercano al óptimo) suele ser alta.

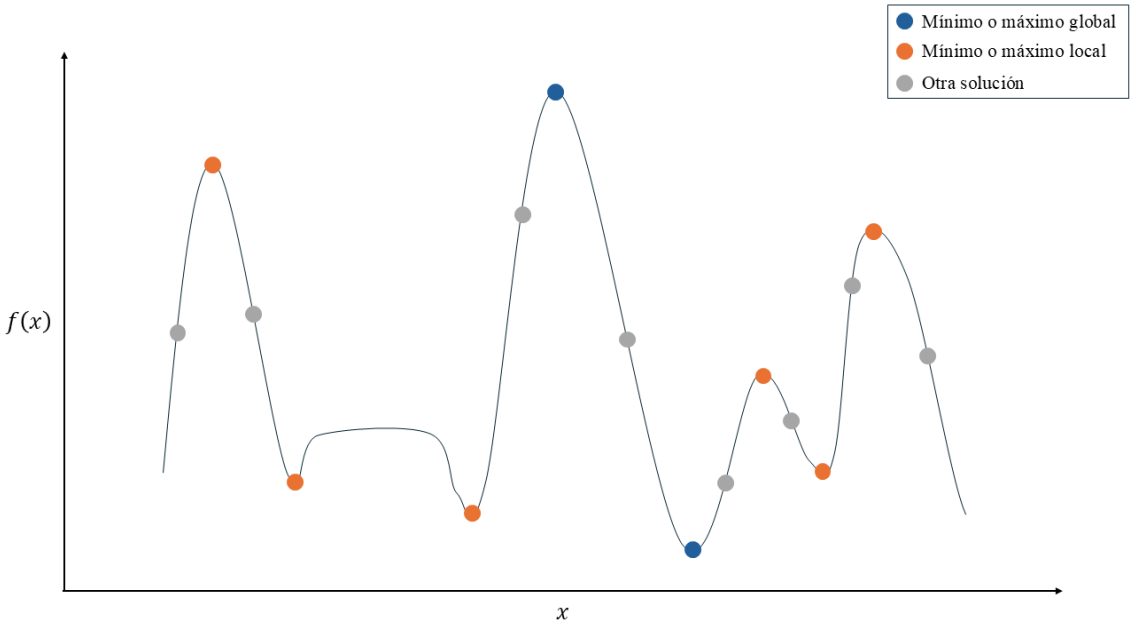

Figura 5. Exploración del espacio de búsqueda por parte de una metaheurística: muestreo de diferentes soluciones. Elaboración propia

Por último, a las metaheurísticas también se les atribuye el término *anytime*, ya que al momento de desarrollar el esquema de operación de uno de estos algoritmos, se puede establecer un criterio de parada a partir del cual su ejecución puede ser detenida para que proporcione la mejor solución entre todas las soluciones que ha podido encontrar hasta ese preciso momento. Dicho criterio de parada puede ser un número máximo de iteraciones o una medida de tiempo

(segundos, minutos, horas). Incluso, se puede determinar que el algoritmo se detenga cuando la mejor solución que haya encontrado no mejore tras un tiempo o número de iteraciones en concreto, o también es posible hacer que se detenga cuando encuentre una solución de una calidad preestablecida como buena para el problema tratado (a criterio del usuario). Detener la ejecución para obtener una solución en cualquier momento es una característica de la que carecen los algoritmos exactos de optimización, ya que es natural que encontrar la solución óptima a un problema requiera un tiempo de cómputo (*i.e.*, tiempo computacional) que resulta incierto. En ocasiones, los problemas que se presentan en una organización pueden requerir respuestas lo más rápidamente posible. El tiempo es, por tanto, un punto a favor de las metaheurísticas. En general, los algoritmos exactos tienden a escalar peor con la talla del problema, es decir, el tiempo que necesitan para obtener la solución óptima puede llegar a aumentar considerablemente en función de la cantidad de datos o elementos involucrados con el problema. Por ejemplo, dado el problema de la mochila, si el número de objetos implicados aumenta, el número de combinaciones posibles de objetos que se podrían seleccionar también aumenta, por lo que el tiempo computacional requerido por un algoritmo exacto para obtener la solución óptima podría incrementase de manera notoria.

Las características de las metaheurísticas que se han mencionado a lo largo de esta sección quedan resumidas en la tabla 1, donde también se realiza una comparación con las características de los algoritmos exactos.

	Algoritmos exactos	Metaheurísticas
Solución óptima	Garantizan obtener la solución óptima a un problema.	Difícilmente pueden llegar a obtener la solución óptima a un problema, pero no es totalmente descartable. En su lugar, suelen obtener soluciones cercanas al óptimo.
Tiempo computacional	El tiempo que necesitan para obtener la solución óptima a un problema tiende a incrementarse a medida que aumenta la talla del problema.	Generalmente son más rápidas para generar soluciones que den respuesta al problema y escalan mejor con la talla del mismo. Además, pueden proporcionar una solución en cualquier instante tras cumplirse un criterio de parada establecido.
Implementación	No es realmente necesario llevar a cabo su implementación, ya que se proporcionan a través de *solvers* para su uso. Sin embargo, si es necesario diseñar el modelo matemático del problema e introducirlo en el *solver*.	Aparte de la modelización del problema, llevar a cabo su implementación es relativamente sencillo. Son muy adaptables para su aplicación a casi cualquier problema.

Tabla 1. Resumen de las características de los algoritmos exactos y las metaheurísticas de optimización

Dichas características hacen que las metaheurísticas sean métodos de optimización muy atractivos para su aplicación en la industria.

A continuación, y para finalizar con esta sección, se hará una breve revisión de algunas de las metaheurísticas que existen y cómo estas se pueden clasificar.

La figura 6 propone una taxonomía de las metaheurísticas. La clasificación más general que se puede hacer de las metaheurísticas se basa en si existe o no una inspiración natural detrás de su creación. Aquellas metaheurísticas surgidas a partir de una inspiración natural intentan simular procesos o fenómenos de diversa naturaleza (*naturally inspired*). Por ejemplo, los algoritmos

genéticos (*Genetic Algorithms*) están inspirados en la teoría de la evolución natural de Darwin . Esta sostiene que aquellos individuos con características genéticas más favorables para adaptarse mejor al entorno que los rodea tienen más probabilidades de sobrevivir y, por tanto, de reproducirse que otros individuos con peor genética. Como resultado, las nuevas generaciones de individuos suelen mejorar con respecto a las generaciones predecesoras, debido a que heredan parte de sus genes, así como por las mutaciones que ocurren durante la reproducción.

El esquema de operación de un algoritmo genético es, por tanto, una simulación del proceso de evolución de los individuos descrito por la teoría de Darwin. El algoritmo inicia con una población de soluciones (un conjunto con cierta cantidad de soluciones que representa a una población de individuos) que va mejorando iteración tras iteración, ya que en cada una de ellas se realizan diferentes operaciones que simulan acontecimientos como la selección de individuos (generalmente los mejores), la reproducción entre los individuos seleccionados, las mutaciones que pueden presentar los nuevos individuos originados por la reproducción, y el reemplazo de la población actual de individuos por la nueva generación.

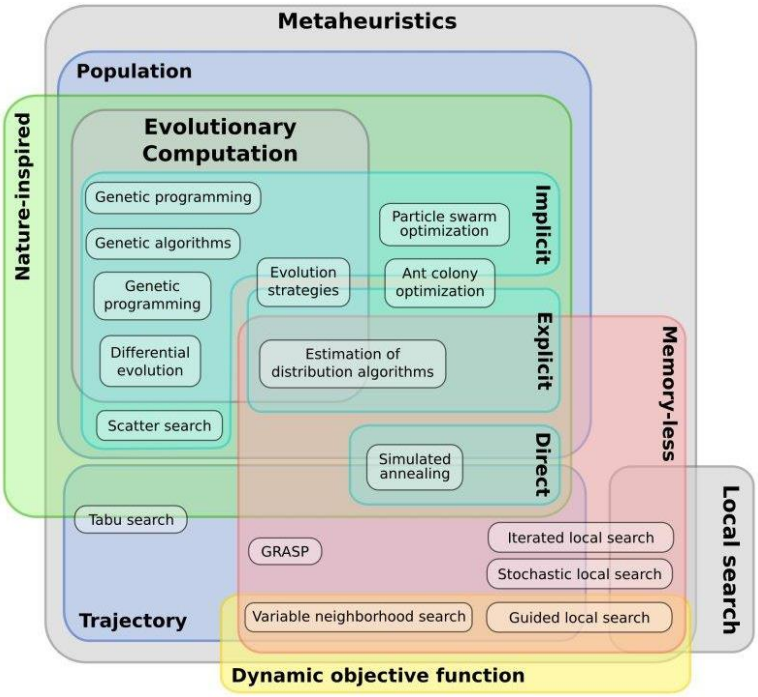

Figura 6. Taxonomía de las metaheurísticas [9]

De esta manera, aparece una clasificación de segundo nivel que divide a las metaheurísticas entre aquellas que trabajan con una población de soluciones (*population*) y las que no. Además de los algoritmos genéticos, otro ejemplo de metaheurística que trabaja con una población de soluciones es el algoritmo de optimización por colonias de hormigas (*Ant Colony Optimization*, ACO).

Los algoritmos genéticos, a su vez, se subclasifican como algoritmos evolutivos (*Evolutionary Algorithms*). Los algoritmos evolutivos constituyen un amplio grupo en el que, además de los algoritmos genéticos, se incluyen otras metaheurísticas como la programación genética (*Genetic Programming*), la programación evolutiva (*Evolutionary Programming*), la evolución diferencial (*Differential Evolution*) y la estrategia evolutiva (*Evolution Strategy*). Todas ellas se inspiran en

la teoría de Darwin y, por tanto, realizan operaciones que simulan diversos acontecimientos que ocurren durante la evolución natural de los individuos. No obstante, aunque todas ellas comparten una base común (*i.e.*, la teoría de Darwin), cada una presenta su propio enfoque.

El algoritmo ACO está inspirado en el comportamiento característico de las hormigas al desplazarse desde su nido hacia una fuente de alimento. A lo largo de la ruta que estas recorren, van depositando una sustancia química denominada feromona, que actúa como un mecanismo de retroalimentación. Esta sustancia permite que otras hormigas detecten el rastro y sigan la misma ruta para encontrar la misma fuente de alimento [10]. El esquema de operación del algoritmo simula este fenómeno, en el que la búsqueda de una solución óptima está influenciada por las soluciones de buena calidad que el algoritmo ya haya encontrado, las cuales aportan información sobre las áreas del espacio de búsqueda más prometedoras para dirigir la exploración hacia estas. Se puede decir que el algoritmo ACO es, por tanto, una metaheurística «bioinspirada», al igual que los algoritmos genéticos. No obstante, otras metaheurísticas, como el algoritmo de recocido simulado (*Simulated Annealing*) [11], también han surgido a partir de una inspiración natural, pero no de un proceso o fenómeno biológico, sino de la metalurgia y el proceso de enfriamiento en el horno.

Otras metaheurísticas se clasifican como de una sola trayectoria (*trajectory*). El algoritmo de recocido simulado o la búsqueda tabú (*Tabu Search*) [12] son algunos ejemplos de este tipo de metaheurísticas. Estas se centran en la mejora iterativa de una única solución, es decir, inician con una determinada solución en la primera iteración y luego tratan de mejorarla en las iteraciones posteriores. La búsqueda tabú, concretamente, comienza con una solución inicial en la primera iteración y explora su vecindario, es decir, investiga otras soluciones cercanas al lugar de donde dicha solución inicial se ubica dentro del espacio de búsqueda. A continuación, se desplaza por el espacio de búsqueda hacia una de las soluciones del vecindario. En las iteraciones siguientes, realiza lo mismo con la solución a la que se había desplazado en la iteración anterior, pero evitando moverse a soluciones que lleven al estancamiento en mínimos o máximos locales (dependiendo de la función objetivo del problema) e impidan encontrar un mínimo o máximo global. Para ello, utiliza una memoria tabú en la que se registran las soluciones ya encontradas que provocan dicho estancamiento, lo que le permite explorar otras zonas del espacio de búsqueda que podrían contener soluciones más prometedoras. De esta manera, la búsqueda tabú también se clasifica como una de las metaheurísticas que utilizan la técnica de búsqueda local (*local search*), la cual permite explorar el vecindario de una solución. Al mismo tiempo, la búsqueda tabú es una de las metaheurísticas cuya creación no se ha debido a ninguna inspiración natural.

La metaheurística GRASP (*Greedy Randomized Adaptive Search Procedure*), al igual que la búsqueda tabú, también utiliza la búsqueda local y no ha sido creada a partir de ninguna inspiración natural. Además, se caracteriza por ser una metaheurística sin memoria (*no memory*). Las metaheurísticas de este tipo no guardan información sobre las soluciones encontradas durante el proceso de búsqueda de una solución óptima. Por ello, cada vez que exploran el espacio de búsqueda, no tienen en cuenta si han visitado previamente determinadas zonas. En contraste, metaheurísticas como la búsqueda tabú o el algoritmo ACO sí almacenan información sobre las soluciones encontradas y, por tanto, guían la búsqueda de una solución óptima basándose en esa información.

En las secciones 2.2.1 y 2.2.2 se profundizará, respectivamente, en la metaheurística GRASP y en los algoritmos genéticos, que son los métodos diseñados en este trabajo para la formación de equipos y cuyo rendimiento en la resolución del problema es objeto de estudio.

2.2.1. GRASP

GRASP (*Greedy Randomized Adaptive Search Procedure*) [13] es una metaheurística de multiarranque, lo que significa que en cada iteración genera una solución desde cero y trabaja con ella. Este término no se debe confundir con el término atribuido a otras metaheurísticas que las clasifican como de una sola trayectoria, ya que las metaheurísticas de este tipo comienzan con una solución en la primera iteración para intentar mejorarla durante las iteraciones siguientes. GRASP, por su parte, en cada iteración arranca generando una solución inicial desde cero y luego aplica la técnica de búsqueda local para tratar de mejorarla. Debido a que no guarda información sobre las soluciones obtenidas en cada iteración, GRASP también se clasifica como una metaheurística sin memoria. Al mismo tiempo, se diferencia de otras metaheurísticas como, por ejemplo, los algoritmos genéticos, ya que no trabaja con una población de soluciones ni su esquema de operación simula ningún proceso o fenómeno natural.

El esquema general de operación de GRASP se describe en el pseudocódigo 1.

```
algoritmo GRASP(criterio_de_parada, α)
1    Leer_Instancia()
2    mientras criterio_de_parada no se cumpla hacer
3        Solucion ← Greedy_Randomized_Construction(α)
4        Solucion ← Busqueda_Local(Solucion)
5        Actualizar Mejor_Solucion con Solucion
6    devolver Mejor_Solucion
fin GRASP
```

Pseudocódigo 1. Esquema general de operación de GRASP [13]

El algoritmo, en primer lugar, recibe como entrada una instancia del problema que se quiere resolver para procesarla, lo que implica extraer y almacenar los datos involucrados con el problema. Por ejemplo, dada una instancia del problema de la mochila, el algoritmo extrae y guarda los datos correspondientes al valor y al peso de cada objeto, así como la capacidad máxima de la mochila. También recibe como entrada un valor específico del parámetro α, el cual se explicará un poco más adelante. Tras procesar la instancia, el algoritmo ejecuta a continuación una serie de iteraciones, tantas hasta completar un número máximo de ellas, alcanzar un tiempo límite de ejecución o cumplir con cualquier otro criterio de parada establecido. Cada iteración consta, principalmente, de dos fases:

- Fase de construcción: se crea una solución inicial al problema. La heurística que comúnmente se utiliza para construir una solución inicial se denomina *Greedy Randomized Construction*, que significa «construcción voraz y aleatoria». Esta heurística, como su nombre indica, permite crear una solución utilizando un enfoque totalmente voraz (el siguiente elemento que se añade a la solución es aquel que mejora en mayor medida el valor de la función objetivo, dependiendo de si el problema es de maximización o minimización) o utilizando un enfoque totalmente aleatorio (el siguiente elemento que se añade a la solución es elegido de manera aleatoria, sin tener en cuenta cómo cambia el valor de la función objetivo). Sin embargo, también permite una combinación de ambos enfoques. Realmente, el grado de voracidad o aleatoriedad con la que la heurística construye una solución está regulado por el parámetro α de entrada.

- Fase de búsqueda local: en esta fase se explora el vecindario de la solución inicial creada durante la fase de construcción previa. Como se mencionó en la sección 2.2, el vecindario de una determinada solución está formado por otras soluciones cercanas a la misma dentro del espacio de búsqueda. El objetivo de la búsqueda local es, por tanto, encontrar una solución dentro del vecindario de la solución inicial que mejore el valor de la función objetivo de esta última. Si la búsqueda tiene éxito, se repite el mismo proceso con la nueva solución encontrada, y así sucesivamente hasta hallar con una solución en cuyo vecindario no exista ninguna solución mejor. En tal caso, se habla de haber hallado con una solución óptima local o simplemente un óptimo local, ya que su valor de la función objetivo es potencialmente un máximo local (si el problema es de maximización) o un mínimo local (si el problema es de minimización).

Tras la fase de búsqueda local, una iteración del algoritmo finaliza actualizando la mejor solución encontrada hasta el momento con el óptimo local, si corresponde. Es decir, si el valor de la función objetivo del óptimo local obtenido en una determinada iteración mejora el valor de la mejor solución encontrada hasta el momento, entonces la mejor solución se actualiza con el óptimo local. Debido a que al comienzo el algoritmo no tiene ninguna solución guardada como la mejor solución, esta se actualiza en la primera iteración con el óptimo local obtenido durante la misma.

Cuando se cumple el criterio de parada, el algoritmo se detiene y termina devolviendo la mejor solución que ha podido encontrar para el problema.

A partir de aquí, se explicará de manera más detallada cómo funcionan concretamente las fases de construcción y búsqueda local. Para ello, se tomará como ejemplo el problema del viajante de comercio (*Travelling Salesman Problem*, TSP) [14]. Dicho problema se trata de un problema clásico de optimización en el cual el objetivo es encontrar la ruta más corta que permita visitar todas las ciudades una sola vez y que, al finalizar, regrese a la ciudad donde comenzó. En el contexto de la teoría de grafos, este objetivo se denomina encontrar un ciclo hamiltoniano. El mismo problema se puede ilustrar, por tanto, a través de un grafo ponderado como el de la figura 7. Esta figura muestra una instancia sencilla con solamente cuatro ciudades, donde los pesos de las aristas que las conectan se corresponden con la distancia que hay entre una ciudad y otra. Existen diferentes variantes del TSP, como una en la que no siempre existe una arista que conecte cada par de ciudades, u otra en la que hay aristas cuya dirección es en un solo sentido (solo se puede ir de una ciudad a otra, pero no al revés). Sin embargo, la simplicidad de la figura 7 será suficiente para facilitar la comprensión del funcionamiento de las fases de construcción y búsqueda local.

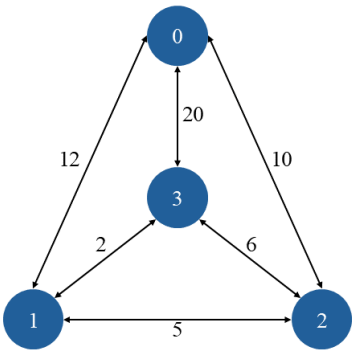

Figura 7. Instancia sencilla del TSP. Elaboración propia

Como ya sabemos, antes de resolver un determinado problema mediante la aplicación de una metaheurística como GRASP o cualquier otra, es necesario modelizarlo. Por tanto, una solución para la instancia del TSP de la figura 7 se puede representar mediante una permutación, como se muestra en la figura 8 (*i.e.*, mediante un vector de números enteros que no se repiten). Dicha solución representa una ruta que indica el orden de visita a las ciudades, asumiendo que la última ciudad visitada es la misma que la que se encuentra en la primera posición del vector, por lo que no se incluye al final.

0	2	1	3

Figura 8. Representación mediante permutación de una solución para el TSP

La función objetivo del problema debe expresar la distancia total de la ruta representada por una solución, la cual se busca que sea mínima. Por tanto, la función objetivo puede definirse de la siguiente manera:

$$\min f(S) = \sum_{i=1}^{3} distancia(C_i, C_{i+1}) + distancia(C_4, C_1)$$

$$S: Solución, \qquad C_i: Ciudad\ visitada\ en\ i\_ésimo\ lugar, \qquad i = 1 \dots 3$$

Durante la fase de construcción que tiene lugar en una iteración de GRASP, se puede crear una solución como la de la figura 8 para la instancia del TSP de la figura 7. Para ello, se emplea la heurística de construcción voraz y aleatoria. Básicamente, esta heurística construye una solución de manera iterativa, es decir, la solución se construye añadiendo uno a uno los elementos involucrados con el problema (p.ej., una solución para el TSP se construye añadiendo las ciudades una a una). La heurística también consta de un parámetro α que puede tomar valores dentro del intervalo $[0,1]$. El valor especificado para este parámetro permite a la heurística construir una solución de una u otra manera:

- Si $\alpha = 0$: en cada iteración, del conjunto de los elementos que falten por añadir a la solución, se selecciona aquel que mejore en mayor medida el valor de la función objetivo de la solución y se incluye en la misma.
- Si $0 < \alpha < 1$: en cada iteración, del conjunto de los elementos que falten por añadir a la solución, se toma en cuenta un subconjunto formado por los elementos que mejoren en mayor medida el valor de la función objetivo de la solución. Posteriormente, de dicho subconjunto, se selecciona un elemento de manera aleatoria y se incluye en la solución. Cuando el valor de α es más alto, el tamaño del subconjunto de elementos considerados es mayor.
- Si $\alpha = 1$: en cada iteración, del conjunto de los elementos que falten por añadir a la solución, se selecciona uno de manera aleatoria y se incluye en la solución.

Por tanto, cuando el valor de α es 0, la heurística construye una solución al problema siguiendo un enfoque totalmente voraz. Cuando α es superior a 0, la solución se construye con cierto grado de voracidad y, al mismo tiempo, de aleatoriedad. Realmente, cuando el valor de α es más alto, el grado de aleatoriedad con el que se construye una solución es mayor, mientras que el grado de

voracidad es menor. Por último, si el valor de α es 1, la solución se construye de manera totalmente aleatoria.

El esquema de operación que sigue la construcción voraz y aleatoria para generar una solución, en este caso, a un problema de minimización, se describe en el pseudocódigo 2.

heuristica $Greedy\_Randomized\_Construction(\alpha)$
1 $Solucion \leftarrow \emptyset$
2 Inicializar el conjunto de elementos candidatos: C
3 **mientras** $C \neq \emptyset$ **hacer**
4 Evaluar el coste incremental $c(e)$ para cada $e \in C$
5 $c^{\min} \leftarrow \min\{c(e) | e \in C\}$
6 $c^{\max} \leftarrow \max\{c(e) | e \in C\}$
7 $umbral \leftarrow c^{\min} + \alpha(c^{\max} - c^{\min})$
8 $RCL \leftarrow \{e \in C | c(e) \leq umbral\}$
9 Seleccionar un elemento e' de la RCL de manera aleatoria
10 $Solucion \leftarrow Solucion \cup \{e'\}$
11 Actualizar el conjunto de elementos candidatos C
12 **devolver** $Solucion$
fin $Greedy\_Randomized\_Construction$

Pseudocódigo 2. Fase de construcción de GRASP para un problema de minimización: heurística de construcción voraz y aleatoria [13]

A continuación, se explican las líneas de operación que componen el pseudocódigo 2, haciendo también analogías con la instancia del TSP de la figura 7 para ilustrar cómo se construye una solución para dicho problema:

- Línea 1: la solución se inicializa al conjunto vacío $\emptyset$.
  - Se trata de una solución sin ningún elemento todavía, es decir, sin ninguna ciudad añadida.
- Línea 2: se inicializa el conjunto C de elementos candidatos a ser incorporados a la solución.
  - El conjunto de elementos candidatos C contiene a las ciudades que todavía no se han añadido a la solución. Por tanto, al principio del procedimiento, dicho conjunto contiene a las ciudades 0, 1, 2 y 3.
- Línea 3: a partir de aquí, la solución comienza a construirse de manera iterativa. En cada iteración, se añade un nuevo elemento a la solución, hasta que no queden elementos por añadir del conjunto de candidatos C. Las operaciones que se realizan en cada iteración determinan cuál será el siguiente elemento que se incorporará a la solución.
- Línea 4: se evalúa el coste incremental $c(e)$ para cada elemento candidato $e \in C$ asociado con su incorporación a la solución. El coste incremental de un elemento define la cantidad en la que cambiaría el valor de la función objetivo de la solución actual si dicho elemento fuera el siguiente en incorporarse.
  - Como al principio del procedimiento la solución no contiene ninguna ciudad todavía, el coste incremental de cada ciudad asociado con su incorporación a la solución es nulo, ya que la distancia de la ruta no se incrementaría. Por tanto, en el caso del TSP, la primera ciudad que forme parte de la solución se añadiría de manera aleatoria. Sin embargo, suponiendo que la solución ya tuviera añadida, por ejemplo, la ciudad 0, entonces el coste incremental de añadir la ciudad 1 sería 12 unidades (distancia entre las ciudades 0 y 1), el de la ciudad 2 sería 10 unidades (distancia

entre las ciudades 0 y 2), y el de la ciudad 3 sería 20 unidades (distancia entre las ciudades 0 y 3).

- Líneas 5, 6, 7, 8, 9 y 10: de todos los costes incrementales evaluados para cada elemento candidato $e \in C$, a partir del mínimo y el máximo de todos ellos ($c^{\min}$ y $c^{\max}$, respectivamente), así como del valor especificado para el parámetro α, se calcula un umbral mediante la fórmula $c^{\min} + \alpha(c^{\max} - c^{\min})$. A continuación, se forma la lista restringida de candidatos RCL (*Restricted Candidate List*) con aquellos elementos del conjunto de candidatos C cuyo coste incremental sea menor o igual al umbral calculado. Posteriormente, de entre todos los elementos que se encuentren en la RCL, se selecciona un elemento e' de manera aleatoria y se añade a la solución.
    - Continuando con la hipotética solución en la que ya se ha añadido la ciudad 0 a la solución, cuyo valor de la función objetivo es 0, el coste incremental mínimo $c^{\min}$ sería 10 (coste incremental asociado a la ciudad 2) y el coste incremental máximo $c^{\max}$ sería 20 (coste incremental asociado a la ciudad 3). A partir de aquí, una ciudad u otra sería la siguiente en añadirse a la solución, dependiendo del umbral calculado, el cual está muy influenciado por el valor especificado de α:
        - Si $\alpha = 0$: el valor del umbral sería 10. De entre las ciudades 1 ($c = 12$), 2 ($c = 10$) y 3 ($c = 20$) que faltarían por añadir a la solución, la RCL estaría formada solamente por la ciudad 2, ya que es la única cuyo coste incremental es menor o igual al umbral de 10. Por tanto, de la RCL solo podría seleccionarse la ciudad 2, y sería esta la que finalmente se añadiría a la solución. Se puede notar que la ciudad que se añadiría es la que menos incrementaría el valor de la función objetivo de la solución, que es lo que se busca en un problema de minimización.
        - Si $0 < \alpha < 1$: en caso de que el valor de α fuera mayor que 0 pero menor que 1, por ejemplo, 0.5, entonces el valor del umbral sería 15. De entre las ciudades 1 ($c = 12$), 2 ($c = 10$) y 3 ($c = 20$) que faltarían por añadir a la solución, la RCL estaría formada por las ciudades 1 y 2, ya que el coste incremental de estas es igual o inferior al umbral de 15. Por tanto, de la RCL se seleccionaría aleatoriamente una de esas dos ciudades para añadirla a la solución. De este modo, se añadiría a la solución una de las ciudades que menos incrementaría el valor de la función objetivo, aunque no necesariamente la que menos lo haría.
        - Si $\alpha = 1$: en este caso, el valor del umbral sería 20, por lo que la RCL estaría formada por todas las ciudades que faltarían por añadir (1, 2 y 3), de donde se seleccionaría una de manera aleatoria para añadirla a la solución. Así, se añadiría cualquiera de las ciudades a la solución, sin tener en cuenta cómo cambiaría el valor de la función objetivo.
- Línea 11: se actualiza el conjunto de elementos candidatos C, ya que se ha añadido un elemento a la solución, quedando menos candidatos por incorporar.

Una vez que la solución se ha construido en su totalidad porque no quedan elementos por añadir, el procedimiento termina devolviendo dicha solución (línea 13). Cabe aclarar que, si se aborda un problema de maximización, el pseudocódigo 2 sería análogo. Lo único que cambiaría ligeramente sería el cálculo del umbral (línea 7), para el cual se utilizaría la fórmula $c^{\max} + \alpha(c^{\min} - c^{\max})$.

A partir de las analogías realizadas con la instancia del TSP de la figura 7, se ha podido verificar cómo un valor u otro de α permitiría a la heurística construir una solución para el problema de distintas maneras. La solución de la figura 8 se trata, concretamente, de una solución construida para el problema utilizando un valor de α igual a 0. Por tanto, dicha solución es una solución construida siguiendo un enfoque totalmente voraz, cuyo valor de la función objetivo es igual a 37 unidades (10 + 5 + 2 + 20).

Terminada la fase de construcción, se da paso a la fase de búsqueda local. Recordemos que en esta fase se explora el vecindario de la solución inicial creada durante la fase de construcción previa. Cada una de las soluciones del vecindario tiene su propia estructura de elementos, las cuales se pueden generar realizando algunas modificaciones sobre la solución actual cuyo vecindario se está explorando. Estas modificaciones consisten, por lo general, en intercambiar de lugar dos elementos de la solución. La figura 9 ayuda a comprender cómo son las soluciones pertenecientes al vecindario de una solución, en concreto, de la solución {0,2,1,3} de la figura 8. A partir de esta solución, una de las soluciones de su vecindario se puede generar intercambiando los elementos 0 y 2, que se encuentran, respectivamente, en la primera y segunda posición. Dicho intercambio produce una nueva solución: {2,0,1,3}. Así, mediante todos y cada uno de los intercambios de dos elementos que se pueden realizar sobre la solución {0,2,1,3}, se generan todas y cada una de las soluciones pertenecientes a su vecindario. A partir de ahora, las soluciones del vecindario de una determinada solución serán referidas como «soluciones vecinas».

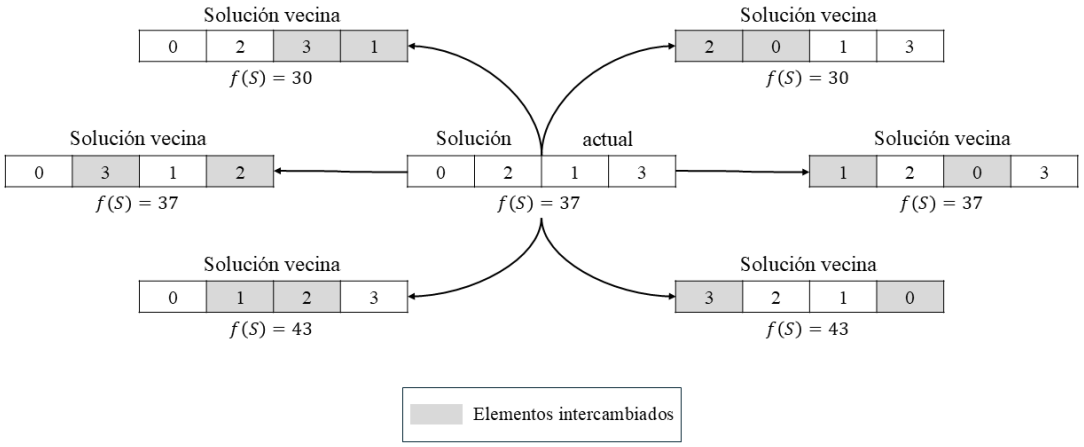

Figura 9. Vecindario de una solución para el TSP. Elaboración propia

Una vez entendido cómo son las soluciones vecinas de una determinada solución, el objetivo de la fase de búsqueda local es encontrar, dentro del vecindario de la solución inicial construida, una solución que la mejore, es decir, una solución cuyo valor de la función objetivo sea mejor. Para ello, se evalúa el valor de la función objetivo de cada una de las soluciones vecinas, en orden aleatorio. En la figura 9, la solución construida es {0,2,1,3}, cuyo valor de la función objetivo es 37. Las soluciones vecinas que mejoran este valor son {2,0,1,3} y {0,2,3,1}, ya que ambas tienen un valor de la función objetivo igual a 30. Generalmente, entre las soluciones vecinas que mejoren el valor de la función objetivo de la solución cuyo vecindario se está explorando, aquella que mejore en mayor medida dicho valor y se haya evaluado antes que cualquier otra que también lo mejore en la misma proporción, es elegida para continuar con el procedimiento, es decir, explorar su vecindario, con el mismo objetivo de encontrar una nueva solución que la mejore, y así sucesivamente hasta hallar con un óptimo local (*i.e.*, una solución en cuyo vecindario no exista

ninguna que mejore su valor de la función objetivo). En el caso de la figura 9, suponiendo que la solución vecina {2,0,1,3} se haya evaluado antes que {0,2,3,1} (con la que comparte el mismo valor de la función objetivo), entonces la primera de estas sería seleccionada para explorar su vecindario.

En general, existen dos estrategias de búsqueda local para determinar que solución vecina se selecciona para continuar con el procedimiento:

- *best-improving*: se evalúan todas las soluciones vecinas. Aquella que mejore en mayor medida el valor de la función objetivo es seleccionada para continuar con el procedimiento.
- *first-improving*: se van evaluando las soluciones vecinas, y al momento de encontrar una primera solución que mejore el valor de la función objetivo, esta resulta seleccionada para seguir con el procedimiento, sin importar si todavía quedaran otras soluciones vecinas por evaluar, entre las cuales no se descarta que incluso pudiera haber una mejor que la que se ha seleccionado.

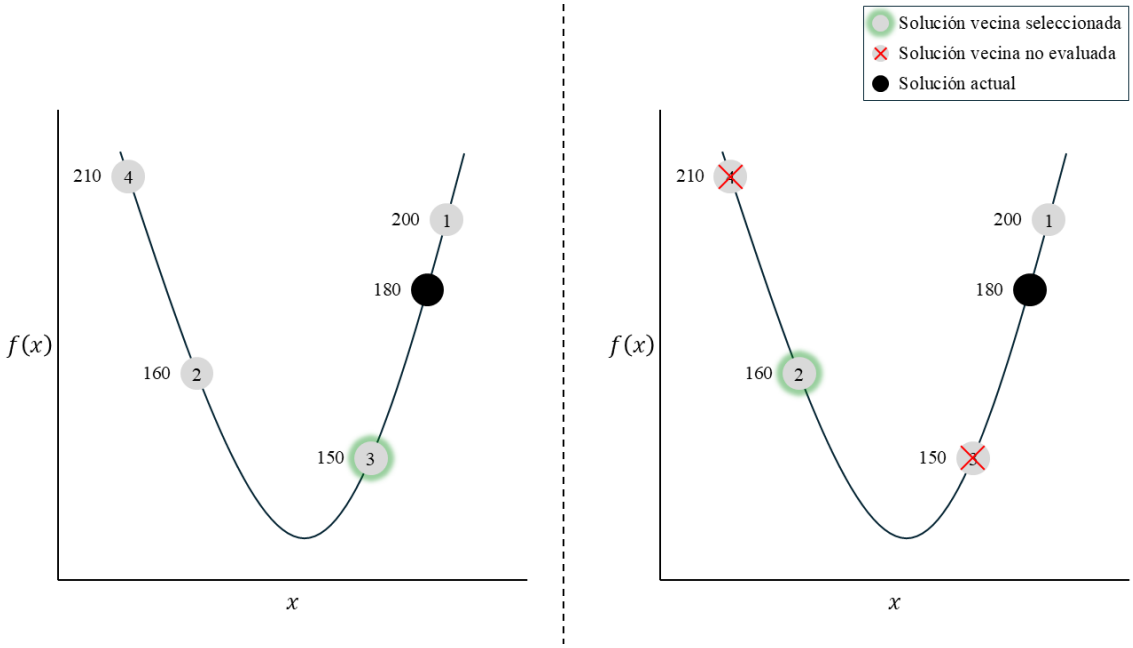

Figura 10. Estrategias de búsqueda local: *best-improving* (izquierda) y *first-improving* (derecha). Elaboración propia

La figura 10 ilustra ambas estrategias de búsqueda local para entender mejor cómo estas funcionan y se diferencian. Dejando a un lado la instancia del TSP de la figura 7, la figura 10 plantea un problema cualquiera de minimización de la función objetivo. Suponiendo que el nodo de color negro es una solución inicial creada para el problema durante la fase de construcción, los nodos grises se corresponden con sus soluciones vecinas. El valor de la función objetivo tanto de la solución inicial como de cada una de sus soluciones vecinas está representado por el número situado a la izquierda de los nodos. Además, los números dentro de los nodos que se corresponden con las soluciones vecinas indican el orden en que estas se han evaluado.

Comenzando con la estrategia *best-improving*, se puede observar que las únicas soluciones vecinas que mejoran el valor de la función objetivo de la solución inicial son las que se han evaluado en segundo y tercer lugar. De estas dos soluciones, la que mejora en mayor medida el

valor de la función objetivo es la evaluada en tercer lugar, por lo que esta última es la que se selecciona para continuar con el procedimiento.

Por otro lado, con la estrategia *first-improving*, suponiendo que las soluciones vecinas se han evaluado también en el mismo orden que con la estrategia *best-improving*, la primera solución que mejora el valor de la función objetivo de la solución inicial es la evaluada en segundo lugar. Por ello, se selecciona esta solución vecina para continuar con el procedimiento, sin que se evalúen las dos soluciones vecinas restantes, que con la estrategia *best-improving* sí se llegan a evaluar. Debido a esto, en este caso con la estrategia *first-improving* se pierde la oportunidad de encontrar una solución vecina mejor, que es la que se ha evaluado en tercer lugar con la estrategia *best-improving*. Sin embargo, esto no significa que la efectividad de la estrategia *first-improving* sea necesariamente peor que el de la estrategia *best-improving*. En muchas aplicaciones de GRASP, se ha observado que con ambas estrategias, en general, se suele encontrar el mismo óptimo local, aunque con la estrategia *first-improving* se logra hacerlo en un tiempo computacional menor. Además, la estrategia *best-improving* tiende a atascarse rápidamente en un óptimo local, lo que impide seguir explorando el espacio de búsqueda para encontrar una solución más cercana al óptimo global [13]. Este fenómeno se conoce como convergencia prematura.

La fase de búsqueda local finaliza cuando se encuentra un óptimo local. A continuación, la mejor solución encontrada hasta el momento por el algoritmo se actualiza con dicho óptimo local, siempre y cuando el valor de la función objetivo de este último sea mejor. Así concluye una iteración de GRASP. El algoritmo repite los mismos pasos en cada iteración (*i.e.*, fase de construcción, fase de búsqueda local y actualización de la mejor solución) y, cuando se cumple el criterio de parada establecido, se detiene para proporcionar la mejor solución que ha podido encontrar para el problema.

Con esto concluye la revisión del algoritmo GRASP. Puede resultar interesante destacar por qué la búsqueda local aplicada a una solución inicial que ha sido construida con cierta o total aleatoriedad ($\alpha > 0$) también puede llevar a la obtención de un buen óptimo local. Al principio, podría pensarse que una solución inicial creada de manera aleatoria puede ser de peor calidad que una creada siguiendo un enfoque totalmente voraz ($\alpha = 0$). Por lo general, esto tiene una alta probabilidad de ser así. Sin embargo, no olvidemos que, durante la fase de búsqueda local, la solución inicial es mejorada hasta encontrar un óptimo local. En este sentido, dado que no se sabe con certeza en qué zona del espacio de búsqueda podría encontrarse la solución inicial aleatoria o la voraz, podría suceder que el óptimo local obtenido al aplicar la búsqueda local a una solución inicial aleatoria sea mejor que el obtenido a partir de una solución inicial voraz.

La figura 11 ilustra esta posibilidad. En ella, se representa gráficamente la función objetivo de un problema hipotético que se busca minimizar, es decir, se representa el espacio de búsqueda. Se puede observar la presencia de una solución inicial voraz y una solución inicial aleatoria, ambas situadas en diferentes puntos del espacio de búsqueda. La solución voraz, por su parte, está ubicada en un punto donde el valor de la función objetivo es más bajo, mientras que la solución aleatoria se encuentra en un punto del espacio de búsqueda donde el valor de la función objetivo es más alto. Por tanto, la calidad de ambas soluciones es distinta, siendo la solución voraz de mejor calidad que la aleatoria. Sin embargo, tras aplicar la búsqueda local a ambas soluciones, los óptimos locales obtenidos son diferentes. El óptimo local obtenido a partir de la solución aleatoria es de mejor calidad que el obtenido a partir de la solución voraz. Esto se debe a que, aunque la solución aleatoria se encontraba inicialmente en una posición peor del espacio de búsqueda, estaba más cerca de un óptimo local de mejor calidad que el óptimo local más cercano

a la solución voraz. Por tanto, para resolver un problema, es importante probar y analizar qué configuraciones del parámetro α de la fase de construcción permiten a GRASP generar mejores resultados, es decir, maximizar su rendimiento.

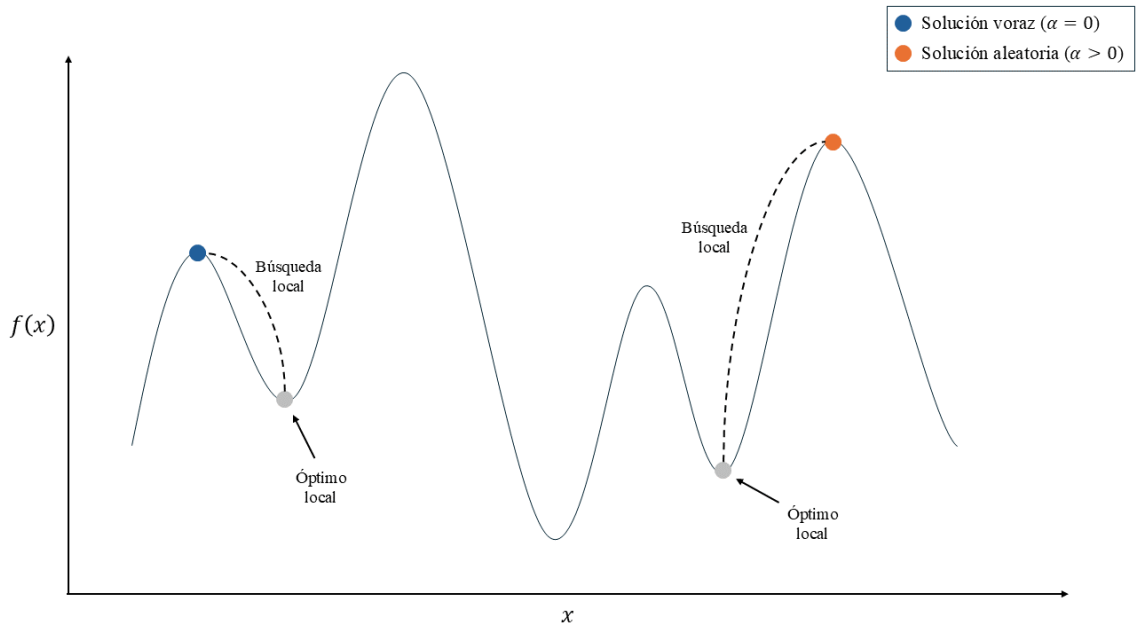

Figura 11. Diferentes óptimos locales obtenidos a partir de la aplicación de la búsqueda local a una solución voraz y a una aleatoria. Elaboración propia

2.2.2. Algoritmos Genéticos

Los algoritmos genéticos [5, 15] forman parte del amplio grupo conocido como algoritmos evolutivos, al cual pertenecen diversas metaheurísticas inspiradas en el proceso de evolución de los individuos en la naturaleza, descrito a través de la teoría desarrollada por el reconocido naturalista y científico inglés Charles Darwin.

Según Darwin [5, 16], el mundo es un espacio de recursos limitados, por lo que no puede haber infinitos individuos. Por tanto, los individuos compiten entre sí para conseguir estos recursos, con el objetivo de sobrevivir. Cada individuo tiene su propia composición genética, lo que se conoce como genotipo. El genotipo de un individuo determina cómo es su fenotipo, es decir, sus características observables, como sus propiedades físicas o su comportamiento. Aquellos individuos con mejor genética son los que suelen tener ventajas para adaptarse mejor al ambiente que los rodea y, en consecuencia, compiten de manera más eficaz por los recursos disponibles. Como resultado, la reproducción suele ocurrir entre individuos sobrevivientes cuya genética es destacada. Al combinarse y transferirse estos genes a los nuevos individuos, estos tienden a mejorar con respecto a sus progenitores. Durante la reproducción, también pueden ocurrir pequeñas mutaciones, las cuales pueden beneficiar a la genética de los nuevos individuos y, por ende, facilitar su supervivencia.

De este modo, todos los algoritmos evolutivos, incluidos los algoritmos genéticos, siguen los principios de la evolución natural en sus esquemas de operación para la búsqueda de soluciones optimizadas a un problema.

Antes de proceder con la descripción del esquema de operación de un algoritmo genético, es necesario establecer la equivalencia entre algunos términos específicos del campo de la genética y la evolución con los términos que se utilizan en el contexto de la optimización:

- Individuo: equivale a una posible solución al problema que se quiere resolver.
- Genotipo o cromosoma: un cromosoma contiene la información genética del individuo, codificada de alguna manera. En términos de optimización, un cromosoma define cómo se codifica o representa una solución (puede haber varias formas).
- Gen: es una parte específica del cromosoma, equivalente a una de las decisiones que componen una solución.
- Alelo: se trata del valor que puede tomar un gen, es decir, el valor asociado a una decisión.
- Fenotipo: es la manifestación externa de la información genética de un individuo, es decir, lo que se puede observar de él.

Para comprender mejor estos conceptos con un ejemplo, la figura 12 muestra una de las tantas soluciones posibles (no necesariamente la mejor) para el problema de la mochila introducido en la sección 2.1. Se trata de una mochila que contiene los objetos 1, 4 y 5 (*i.e.*, objetos de color verde, naranja y azul, respectivamente). Esta es la representación externa o fenotipo de la información genética de esa solución o individuo. Dicha información genética está contenida y codificada dentro del cromosoma, representado, en este caso, mediante un vector binario. Cada posición del vector corresponde a un solo gen del cromosoma, cuyo valor o alelo está asociado con la presencia o no de un determinado objeto dentro de la mochila. De esta manera, la solución o individuo mostrado es una mochila que incluye los objetos 1, 4 y 5, debido a que los alelos de los genes relativos a la presencia de estos objetos son 1, mientras que los de los otros objetos son 0.

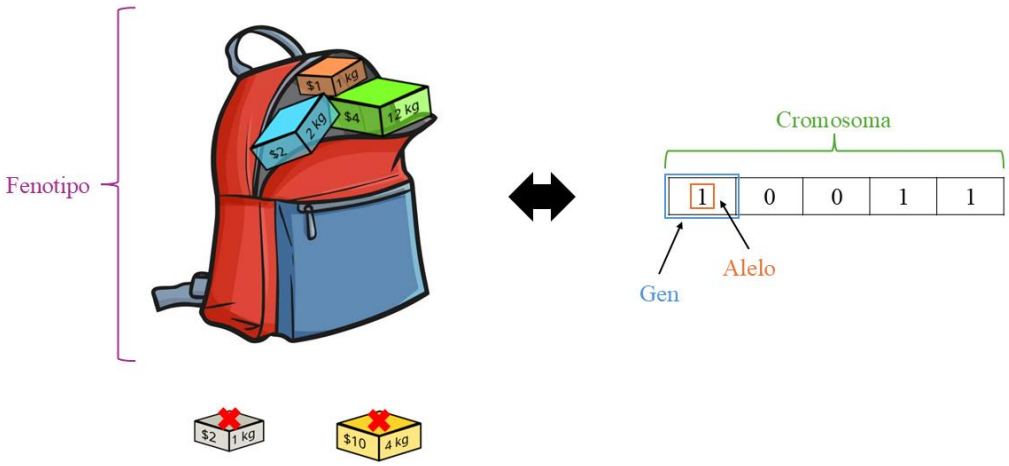

Figura 12. Solución para el problema de la mochila: cromosoma, gen, alelo y fenotipo. Elaboración propia parcial (imagen de la mochila extraída de [17])

Para evaluar qué tan apto es un individuo para el medio ambiente, es decir, que tan buena es una solución para el problema tratado, se emplea, como sabemos, una función objetivo, que en el contexto de los algoritmos evolutivos (y, por tanto, de los algoritmos genéticos) se denomina concretamente «función de *fitness*», traducido como función de aptitud. En el caso de la solución mostrada en la figura 12, su valor de la función objetivo o *fitness* es 7, suma de los valores de los objetos 1, 4 y 5 incluidos en la mochila.

También es importante entender qué ocurre exactamente durante la reproducción entre individuos:

- Cruce del material genético de los progenitores: los genes de dos individuos progenitores se combinan para formar el material genético o genotipo de nuevos individuos. Así, los descendientes resultantes de la reproducción tienen una mezcla de los genes de ambos progenitores.
- Pequeñas mutaciones: aunque el material genético de un nuevo individuo es una combinación del de sus progenitores, también puede sufrir pequeñas perturbaciones en uno o más genes.

La figura 13 muestra cómo, a partir del material genético de dos progenitores, se forma el genotipo de dos nuevos individuos. En este ejemplo, el Descendiente 1 hereda los dos primeros genes del Progenitor 1 y los dos últimos genes del Progenitor 2, mientras que el Descendiente 2 hereda los dos primeros genes del Progenitor 2 y los dos últimos genes del Progenitor 1. Sin embargo, debido a las mutaciones ocurridas durante la reproducción, el último gen del Descendiente 1 y el primer gen del Descendiente 2 sufren una modificación.

Figura 13. Generación de dos nuevos individuos a partir del cruce entre dos progenitores. Mutación de los dos nuevos individuos. Elaboración propia

Una vez explicado qué es un individuo, cómo su información genética puede ser codificada en un cromosoma y cómo la reproducción entre individuos da lugar a nuevos individuos, el esquema general de operación de un algoritmo genético se describe en el pseudocódigo 3.

```
algoritmo GA(criterio_de_parada)
1    Leer_Instancia()
2    Inicializar una población P de soluciones aleatorias
3    mientras criterio_de_parada no se cumpla hacer
4        Seleccionar progenitores de P
5        Cruzar parejas de progenitores
6        Mutar a los descendientes resultantes
7        Evaluar el valor de fitness de cada descendiente
8        Reemplazar la población P
9        Actualizar Mejor_Solucion con la solución de P que posea el mayor valor de fitness
10   devolver Mejor_Solucion
fin GA
```

Pseudocódigo 3. Esquema general de operación de un algoritmo genético. Elaboración propia

Primero, el algoritmo recibe como entrada una instancia del problema que se desea resolver y la procesa, extrayendo y almacenando todos los datos asociados con el problema.

A continuación, se inicializa una población de soluciones P. Generalmente, las soluciones que se generan e incluyen en la población inicial suelen ser aleatorias. Sin embargo, perfectamente

pueden utilizarse también otros tipos de heurísticas para la creación de las soluciones de la población inicial, como, por ejemplo, una heurística voraz. Por otro lado, fijar el tamaño de la población $|P|$ es totalmente flexible, aunque no se recomienda que sea demasiado pequeño para mantener una diversidad considerable de soluciones y, por tanto, de valores de *fitness*. Tampoco es recomendable establecer una población excesivamente grande por razones de eficiencia computacional.

Una vez creada una población inicial, el objetivo es mejorar dicha población de manera iterativa, hasta que se cumpla un criterio de parada establecido (p.ej., un número máximo de iteraciones completadas, un tiempo límite de ejecución, etc.). Para ello, en cada iteración se realizan los siguientes pasos:

- Selección de progenitores: de la población P, se seleccionan n soluciones, las cuales actuarán como progenitores para generar nuevas soluciones. Existen varias heurísticas para realizar la selección de progenitores. Por lo general, todas ellas consisten en seleccionar, principalmente, buenas soluciones. No obstante, también existe cierta aleatoriedad en el proceso, por lo que incluso soluciones de menor calidad tienen alguna probabilidad de ser elegidas, aunque normalmente pequeña.
- Cruce entre los progenitores: las soluciones elegidas como progenitores se cruzan para crear m nuevas soluciones o descendientes. Normalmente, se suelen tomar parejas de progenitores (*i.e.*, dos soluciones), y como resultado se suelen originar dos descendientes por pareja. Debido a que comúnmente se eligen buenos progenitores durante la etapa de selección, con la operación de cruce se asume que, si el material genético de los progenitores es bueno, la combinación de sus genes también será buena [5]. En otras palabras, si el valor de *fitness* de los progenitores es bueno, el de sus descendientes también será bueno. Por tanto, también hay probabilidades de que los descendientes sean mejores que sus progenitores. Para realizar el cruce entre progenitores, existen muchos tipos de heurísticas ya establecidas y conocidas en la literatura de los algoritmos genéticos. Algunas de ellas se caracterizan por ser más apropiadas para su aplicación a determinadas representaciones de una solución. No obstante, generalmente en cada una de ellas interviene cierta aleatoriedad.
- Mutación de los descendientes resultantes: se modifica el material genético de los descendientes resultantes del cruce entre los progenitores. Por lo general, la modificación que se aplica suele ser ligera, es decir, solo se cambian los valores de unos pocos genes y no de manera muy drástica. De este modo, con esta operación se asume que, si se altera de manera leve uno o algunos genes de un descendiente, este podría mejorar (*i.e.*, su valor de *fitness* podría mejorar) [5]. Cambios muy drásticos podrían tener el efecto contrario. Existen bastantes heurísticas para llevar a cabo la mutación de los descendientes, en las cuales también interviene cierto grado de aleatoriedad.
- Reemplazo de la población: se evalúa el valor de *fitness* de cada uno de los m descendientes, los cuales representan la nueva generación de individuos o soluciones. A continuación, se reemplaza la población de soluciones P por la nueva generación. Aunque para ello existen diferentes heurísticas, en general se puede realizar el reemplazo de la población de manera total o parcial. Esto significa que, por un lado, si $m \geq |P|$, se puede actualizar completamente la población con la nueva generación de soluciones, descartando todas las de la generación anterior (*i.e.*, las soluciones que ya se encontraban en la población). Por otro lado, si $m < |P|$, se pueden sustituir m soluciones de la generación anterior por las de la nueva generación, conservando el resto. En cualquier caso, se suelen

mantener buenas soluciones en la población, y también su tamaño, lo cual es ideal por motivos de eficiencia computacional. También puede haber cierta aleatoriedad en el proceso.

Cada iteración finaliza actualizando la mejor solución encontrada hasta el momento con aquella que tenga el mayor valor de *fitness* en la población *P* recientemente reemplazada, si corresponde. Es cierto que, a lo largo de todo el procedimiento, la población de soluciones evoluciona, es decir, el promedio de *fitness* de las soluciones que alberga va mejorando iteración tras iteración. Sin embargo, durante la etapa de reemplazo, puede suceder que la mejor solución generada por el algoritmo sea sustituida en la población por otra cuyo valor de *fitness* no sea superior, ya sea por el propio diseño de la heurística utilizada para el reemplazo o por la aleatoriedad asociada a esta. Debido a esto, la mejor solución no necesariamente pertenece a la última población generada por el algoritmo. Por tanto, para conservar el valor de *fitness* de la mejor solución y lo que esta representa para resolver el problema de manera óptima o casi óptima, es recomendable actualizar la mejor solución en cada iteración.

Tras cumplirse el criterio de parada, el algoritmo se detiene y termina devolviendo la mejor solución que ha podido encontrar para el problema.

Comparado con GRASP, se puede decir que un algoritmo genético, en general, tiene una naturaleza más aleatoria, mientras que GRASP es una metaheurística más adaptativa. Esto se debe a que, como se ha mencionado, existe cierta aleatoriedad durante la selección de progenitores, el cruce entre estos, la mutación de los descendientes resultantes y el reemplazo de la población. Sin embargo, esto no implica necesariamente que el rendimiento de un algoritmo genético en la resolución de un problema sea inferior al de GRASP. Para comparar el rendimiento de diferentes metaheurísticas, es necesario experimentar con ellas y realizar un análisis correcto de sus resultados que permita determinar cuál es más adecuada para resolver un problema específico.

2.3. Formación de equipos en el aula

Formar equipos eficaces no es una tarea fácil, ya que el rendimiento de uno depende de múltiples factores. No obstante, en la literatura se pueden encontrar diferentes criterios que se pueden utilizar para guiar dicho proceso. Algunos de estos criterios se basan en distintas investigaciones realizadas a lo largo del tiempo para entender e identificar qué aspectos influyen en el buen funcionamiento de un equipo. Uno de esos aspectos es el comportamiento característico de cada miembro del equipo, según señala la teoría de roles de Belbin. En la sección 2.3.1 se explica en detalle el principio que sustenta esta teoría para que el rendimiento de un equipo sea eficaz. Se comentan algunos casos en los que la aplicación de la teoría de Belbin para la formación de equipos en el aula ha proporcionado resultados satisfactorios. Además, se mencionan y comentan brevemente otros criterios de relevancia que se pueden tener en cuenta a la hora de crear equipos. Por último, en la sección 2.3.2 se resalta la importancia de utilizar métodos de optimización eficientes para la resolución del problema, y se comentan algunos de los que se han utilizado en diferentes trabajos del estado del arte.

2.3.1. Criterios para la formación de equipos en el aula

Como se ha mencionado, existen varios criterios en los que apoyarse para llevar a cabo la formación de equipos. Meredith Belbin [18, 19], un reconocido investigador británico en el campo

de la gestión de equipos, desarrolló una teoría en la que sostiene que, para garantizar el buen funcionamiento de un equipo, es importante que sus miembros sean capaces de desempeñar una serie de roles esenciales y diversos [3]. Según Belbin, un rol está relacionado con el comportamiento que exhibe una persona al trabajar en colaboración con otras, es decir, lo que es capaz de contribuir a un equipo. Así, un «rol Belbin» se distingue claramente de los conocimientos específicos y competencias técnicas que posee un individuo. En un principio, Belbin identificó y definió un total de ocho roles esenciales. En [20, 21] se pueden encontrar los ocho roles definidos de manera detallada. No obstante, a continuación se proporciona un breve resumen de cada uno:

- Implementador (*implementer*): trabajador/a disciplinado/a que hace lo que sea necesario para que el trabajo avance.
- Coordinador (*coordinator*): maduro/a, confiable y seguro/a, capaz de identificar las fortalezas de cada miembro del equipo y aprovecharlas para alcanzar los objetivos.
- Impulsor (*shaper*): enérgico/a y altamente motivado/a, fomenta la proactividad dentro del grupo.
- Cerebro (*plant*): muy creativo/a, genera nuevas ideas y resuelve problemas complejos.
- Investigador de Recursos (*resource investigator*): experto/a en identificar los recursos disponibles para llevar a cabo el trabajo y explorar alternativas.
- Monitor-Evaluador (*monitor-evaluator*): analiza de manera crítica las ideas y sugerencias de los compañeros.
- Cohesionador (*teamworker*): apoya a los demás miembros del equipo, manteniendo la armonía y previniendo conflictos.
- Finalizador (*completer-finisher*): presta gran atención a los detalles para finalizar y consolidar correctamente lo que se ha empezado.

Se puede notar que cada rol aporta algo único a un equipo, como la generación de ideas innovadoras, la búsqueda de recursos, la gestión de relaciones para mantener la armonía, etc. De este modo, la complementariedad de estos roles es lo que permite el buen funcionamiento del equipo. Es importante señalar que, dentro de un equipo, no es necesario que haya una persona distinta para cada rol. De hecho, una misma persona puede desempeñar no solo uno, sino que dos o más roles, siempre y cuando se caracterice por mostrar los patrones de comportamiento relacionados con estos roles. Lo ideal es que, en conjunto, todos los miembros del equipo sean capaces de cubrir con todos los roles.

Como se ha mencionado, Belbin identificó originalmente ocho roles. Sin embargo, al continuar con sus investigaciones, identificó un noveno rol: el «especialista» (*specialist*). Este rol es característico de aquellas personas especializadas en un área en particular, en la cual poseen un conocimiento profundo. Dentro de un equipo, la presencia de una persona capaz de desempeñar el rol de especialista puede ser crucial cuando los otros miembros carecen de la experiencia suficiente sobre un tema concreto. A diferencia de los ocho roles originales, el rol de especialista no solo define un patrón de comportamiento, sino que también está ligado a los conocimientos específicos que posee una persona.

Los roles característicos de un conjunto de individuos pueden identificarse mediante el Inventario de Autopercepción de Roles de Equipo Belbin (Belbin *Team Role Self-Perception Inventory*) [22]. Se trata de un cuestionario diseñado para evaluar cómo una persona se percibe a sí misma cuando trabaja en el contexto de un equipo. Consta de diversas secciones, cada una de las cuales ofrece una variedad de respuestas. En cada sección, los participantes deben distribuir un total de diez puntos entre las distintas respuestas, teniendo la opción de asignar todos los puntos

a una única respuesta. Una vez que todos los participantes completan el cuestionario, cada uno de ellos obtiene una puntuación numérica en cada uno de los ocho roles, en función de cómo hayan distribuido los puntos en cada pregunta. La tabla 2 indica las puntuaciones que un individuo debe alcanzar para saber qué roles le caracterizan. Si la puntuación conseguida por un individuo en un rol es alta o muy alta, se puede esperar que sea capaz de exhibir el comportamiento relacionado con dicho rol y, por ende, aportar al equipo lo correspondiente.

	Baja	Media	Alta	Muy Alta
Implementador	0-6	7-11	12-16	>16
Coordinador	0-6	7-10	11-13	>13
Impulsor	0-8	9-13	14-17	>17
Cerebro	0-4	5-8	9-12	>12
Investigador de Recursos	0-6	7-9	10-11	>11
Monitor-Evaluador	0-5	6-9	10-12	>12
Cohesionador	0-8	9-12	13-16	>16
Finalizador	0-3	4-6	7-9	>9

Tabla 2. Umbrales de cada rol Belbin [23]

De este modo, un criterio para formar equipos potencialmente eficaces es agrupar individuos en función de sus roles característicos. El objetivo sería, por tanto, asegurar la cobertura de todos los roles en cada equipo.

La aplicación de la teoría de Belbin en entornos educativos ha demostrado tener un impacto positivo, según avalan algunos trabajos. En [24] se desarrolló una herramienta informática que diera apoyo a la formación de equipos en el aula, considerando la teoría de roles de Belbin. La principal novedad de dicho trabajo residía en que, para identificar los roles de los alumnos, en lugar de utilizar el Inventario de Autopercepción de Belbin se tenía en cuenta la retroalimentación proporcionada por los compañeros, una práctica basada en la hoja de evaluación del observador (*Observer Assessment Sheet*) diseñada por Belbin para capturar la perspectiva de otras personas (observadores) sobre el rol o comportamiento que un determinado individuo exhibe al trabajar en equipo. Esto se debe a que, según se menciona, la principal desventaja del Inventario de Autopercepción es que los individuos que lo realizan pueden tener una imagen distorsionada de sí mismos, muy diferente de la que realmente proyectan hacia los demás. Para validar la efectividad de la herramienta, se llevó a cabo un experimento con estudiantes del Grado en Turismo de la Universidad Politécnica de Valencia durante dos módulos consecutivos (*i.e.*, primer y segundo semestre) de un mismo año académico. Inicialmente, la herramienta agrupaba a los estudiantes de manera aleatoria debido a la falta de retroalimentación. No obstante, a medida que se desarrollaban diferentes proyectos grupales a lo largo del año, cada alumno proporcionaba a la herramienta información sobre otros compañeros para determinar con mayor precisión sus roles característicos, utilizando para ello aprendizaje bayesiano. De este modo, se esperaba que, para los proyectos que quedaran por realizar hasta finalizar el curso, la herramienta fuera capaz de crear equipos más eficaces agrupando a los estudiantes de acuerdo con sus roles característicos, dejando de lado la aleatoriedad. Los alumnos fueron encuestados al final de cada proyecto para recopilar información sobre su experiencia colaborativa. Al concluir el experimento, los resultados de las encuestas demostraron la utilidad de la herramienta, la cual mejoraba aspectos como la dinámica de trabajo en equipo e incrementaba la satisfacción de los alumnos.

Los autores de [25] promovían el trabajo colaborativo y la realización de proyectos como un método de aprendizaje efectivo. Llevaron a cabo un experimento con estudiantes del Grado en Ingeniería Química de la Universidad del País Vasco, el cual consistía en crear dos tipos de equipos distintos: equipos organizados en función de los roles Belbin característicos de los estudiantes y equipos organizados aleatoriamente, en este caso, por cuenta propia de los alumnos. Cada equipo se encargaría de llevar a cabo un proyecto distinto. Tras finalizar todos los proyectos, el trabajo realizado por cada equipo sería evaluado, y sus miembros también serían evaluados individualmente con exámenes para medir los conocimientos adquiridos. Los resultados demostraron que aquellos equipos formados en base a la teoría de roles de Belbin obtuvieron mejores calificaciones en sus proyectos, y sus respectivos miembros también lograron mejores notas en los exámenes individuales en comparación con los miembros que pertenecían a equipos formados aleatoriamente.

Además de la tendencia de comportamiento de los individuos (*i.e.*, sus roles Belbin característicos), se ha demostrado que la presencia de distintas personalidades también puede influir de manera positiva en el rendimiento de un equipo [26]. El Indicador Tipológico de Myers-Briggs (Myers-Briggs *Type Indicator*, MBTI) [27] es una prueba psicológica en formato de cuestionario diseñada para evaluar cómo una persona percibe el mundo y cuáles son sus preferencias al tomar decisiones en diferentes situaciones. De este modo, la prueba permite identificar la personalidad de un individuo en base a cuatro dimensiones bipolares, es decir, dimensiones representadas por dos rasgos de personalidad totalmente opuestos. Estas dimensiones son las siguientes:

- Extroversión (E) / Introversión (I)
- Sensación (S) / Intuición (N)
- Pensamiento (T) / Sentimiento (F)
- Juicio (J) / Percepción (P)

Así, la personalidad de un individuo se compone de cuatro rasgos, uno por cada dimensión (p.ej., ENTP). En total, existen 16 tipos de personalidad como producto de todas las combinaciones posibles entre los ocho rasgos.

Otros estudios sugieren que la presencia de miembros de géneros diversos dentro de un equipo puede tener un impacto positivo en su rendimiento [4]. Esto se debe a que, independientemente de la genética, los hombres y las mujeres suelen diferenciarse por tener sus propios pensamientos, ideas o perspectivas a la hora de resolver problemas, que al complementarse contribuyen a la creatividad y al buen funcionamiento del equipo.

En general, se puede decir que los criterios mencionados hasta ahora comparten la idea de que la diversidad o heterogeneidad en los equipos (de roles, personalidades, géneros, etc.) es un factor ventajoso para su funcionamiento, ya que esta implica la mezcla de diferentes habilidades, perspectivas y tendencias. Esto es una creencia común en el contexto de la formación de equipos, donde en muchas ocasiones se busca evitar la creación de equipos homogéneos o con falta de características diversas.

2.3.2. Métodos de resolución para la formación de equipos en el aula

La optimización en la formación de equipos implica encontrar aquella combinación o manera de agrupar a los individuos que produzca los equipos más eficaces, de acuerdo con el criterio o los criterios considerados. Por ejemplo, si se tiene en cuenta la diversidad de personalidades, lo óptimo sería encontrar aquella combinación en la que los individuos con personalidades diferentes se agrupen en los mismos equipos. Sin embargo, evaluar manualmente todos los posibles equipos que se podrían formar para identificar cuáles cumplen mejor con el criterio considerado resulta inviable. Por ejemplo, si en un aula con 30 alumnos se desean formar equipos de cinco, utilizando el coeficiente binomial sabemos que se podrían crear un total de $\binom{30}{5} = 142.506$ equipos distintos de cinco alumnos [24], lo que representa una cantidad muy grande de equipos a analizar para seleccionar, en este caso, los seis equipos de cinco alumnos más convenientes y no solapados (*i.e.*, que no compartan miembros en común). Sin embargo, el verdadero problema reside en el análisis de aproximadamente $\binom{142.506}{6} = 1.16 \times 10^{28}$ posibilidades de selección de seis equipos, muchas de las cuales no son factibles debido al solapamiento entre equipos. Todo este proceso tomaría un tiempo desmesurado de llevarlo a cabo manualmente. El problema empeora aún más sabiendo que la combinatoria al agrupar estudiantes aumenta con el número de alumnos en el aula y/o el tamaño de los equipos. Por ello, es necesario recurrir al uso de algoritmos de optimización, ya sean exactos o aproximados, para automatizar la resolución del problema en un tiempo mucho menor.

Sin embargo, el problema de la formación de equipos, en muchas de sus variantes o formulaciones algorítmicas, está considerado como un problema «NP-completo» [28], de acuerdo con la teoría de la complejidad computacional. Esto significa que, debido a la alta combinatoria asociada con el problema, no se conoce todavía ningún algoritmo que lo resuelva de manera eficiente, es decir, que enumere todas o una cantidad considerable de las combinaciones posibles, las evalúe y encuentre la solución óptima en un tiempo razonable, considerando distintas tallas del problema (*i.e.*, tamaños de aula).

La complejidad temporal de un algoritmo [29, 30] hace referencia a la forma en que crece el tiempo que tarda en ejecutarse y, por ende, en resolver un problema a medida que aumenta la talla del mismo. Comúnmente, se expresa la complejidad temporal de un algoritmo usando la notación «O grande», como se muestra en la figura 14. Cada notación está asociada con un tipo de función distinta, que expresa la relación entre el tiempo de ejecución de un algoritmo y la talla del problema de entrada. Así, un algoritmo cuya complejidad temporal es $O(1)$ indica que, independientemente de la talla de un determinado problema, el tiempo que tarda en ejecutarse no varía. Si la complejidad temporal de un algoritmo es $O(N)$, significa que el tiempo que tarda en ejecutarse crece de manera proporcional o lineal con la talla del problema. En general, los algoritmos con una complejidad temporal situada en la zona verde son los más deseables, si bien pueden ser muy difíciles o incluso imposibles de encontrar o desarrollar. En contraste, los algoritmos cuya complejidad temporal se encuentra en la zona roja son los que peor escalan con la talla del problema. A la zona amarilla pertenecen los algoritmos con una complejidad temporal considerada intermedia. No obstante, debido a la dificultad de encontrar algoritmos con complejidad temporal $O(1)$ o $O(\log N)$, se suelen considerar eficientes los algoritmos con complejidad temporal polinómica, es decir, aquella expresada con la notación $O(N^x)$, donde x es una constante mayor o igual a 1.

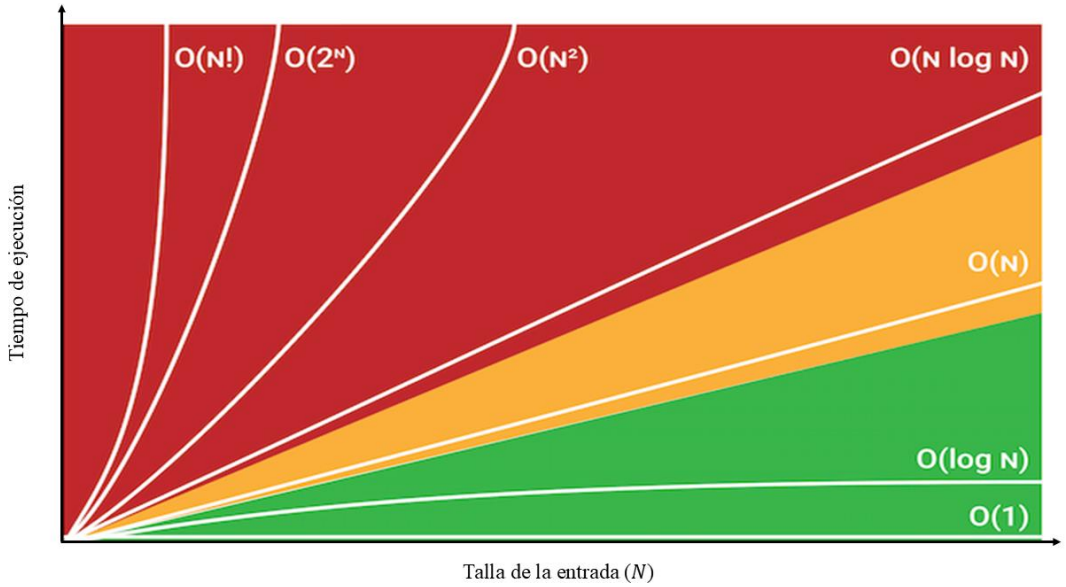

Figura 14. Clasificaciones de la complejidad temporal de un algoritmo según el tipo de función asociado con la notación O grande [31]

De este modo, un problema NP-completo es, de manera más precisa, uno para el que hasta la fecha no se conoce un algoritmo que lo resuelva en tiempo polinómico. Por tanto, para el problema de la formación de equipos, solo se conocen algoritmos exactos que encuentran la solución óptima, por ejemplo, en tiempo exponencial $O(2^N)$ o algún otro peor que el polinómico. Por ello, la aplicación de métodos aproximados, tales como las heurísticas o metaheurísticas, puede ser una alternativa más viable para resolver este problema, ya que el tiempo que tardan en ejecutarse escala mejor con la talla y, por ende, ofrecen respuestas más rápidas y de calidad probablemente aceptable, aunque no necesariamente óptimas.

Algunos trabajos del estado del arte de la formación de equipos han demostrado que los métodos exactos tardan considerablemente más en generar respuestas en comparación con los aproximados, los cuales también permiten obtener soluciones satisfactorias. Otros trabajos, sin realizar una comparativa entre los dos tipos de métodos, también destacan la validez de los métodos aproximados.

Los autores de [32] emplearon un algoritmo exacto y un algoritmo heurístico para crear equipos «sinérgicos». Según ellos, un equipo era sinérgico si, dada una determinada tarea, estaba formado por alumnos que cubrían con una serie de competencias técnicas necesarias para ejecutarla correctamente, y era diverso tanto en géneros como en personalidades, esto último de acuerdo con una variante reducida del MBTI (*i.e.*, para identificar las personalidades de los alumnos se empleó un cuestionario con muchas menos preguntas que las 93 de la versión estándar). Así, el primer factor mencionado (*i.e.*, las competencias técnicas cubiertas) medía la «proficiencia» de un equipo, mientras que el segundo y tercer factores (*i.e.*, la diversidad de géneros y de personalidades) medían su «congenialidad». La combinación lineal de las medidas de proficiencia y congenialidad definía el valor sinérgico de un equipo. Se buscaba, por tanto, que el valor sinérgico de los equipos a formar fuera máximo.

Para encontrar una solución óptima acorde con este objetivo, el primer método desarrollado consistía, primeramente, en enumerar todos los posibles equipos de un tamaño específico y calcular su valor sinérgico. Luego, al definir el modelo matemático, los equipos generados se

utilizarían como variables de decisión binarias, mientras que sus valores sinérgicos se emplearían como coeficientes. Las restricciones impuestas estaban relacionadas con el número total de equipos que se debían seleccionar y que cada alumno formara parte solamente de un equipo. Tras la creación del modelo, se emplearía un *solver* para resolverlo y determinar qué equipos serían seleccionados. El segundo método utilizado era un algoritmo basado en la búsqueda local, siguiendo un enfoque personalizado de esta técnica.

Con el objetivo de analizar y comparar el rendimiento de ambos algoritmos, se llevó a cabo un proceso experimental que permitiera recopilar los resultados computacionales de cada uno, asociados con la calidad de las soluciones que eran capaces de generar y el tiempo requerido para ello, en función de distintas instancias del problema (*i.e.*, diferentes tamaños de aula y de equipo). Los análisis realizados demostraron que el tiempo tomado por ambos algoritmos era similar para tamaños de aula y de equipo pequeños. Sin embargo, a medida que aumentaba el número de alumnos y/o el tamaño de equipo, el tiempo tomado por el algoritmo exacto crecía de manera más notoria que el del algoritmo heurístico. Por ejemplo, en el peor de los casos, el algoritmo exacto tardaba poco más de dos horas y media en encontrar la solución óptima, mientras que el algoritmo heurístico lograba una muy buena solución en aproximadamente cinco minutos. De hecho, en todos los casos, la calidad de las soluciones que este último proporcionaba representaban al menos un 95% de la calidad de las soluciones óptimas obtenidas con el algoritmo exacto. Finalmente, también se evidenció que en lo que más demoraba el algoritmo exacto para resolver las diferentes instancias era en la generación de todos los posibles equipos, una tarea que consumía más tiempo que la propia resolución del modelo matemático con el *solver*. Sin contar con eso, el rendimiento del algoritmo exacto en cuanto al tiempo de resolución mejoraba notablemente, e incluso se acercaba al del algoritmo heurístico en algunos casos. Sin embargo, esto solo es un aspecto informativo, ya que al resolver el problema mediante un algoritmo exacto siempre es necesario enumerar cierta cantidad de equipos en función del tamaño del aula y el tamaño deseado para los equipos, lo que a su vez significa que la demora adicional sea inevitable.

En [33] se implementó un algoritmo genético para crear equipos equilibrados de acuerdo con diversas características extraídas de los alumnos. Estas características estaban relacionadas, por ejemplo, con el nivel general de conocimiento de los estudiantes sobre diversos temas, así como sus habilidades comunicativas o de liderazgo. En general, el método desarrollado permitía la consideración de cualquier característica que se pudiera estimar cuantitativamente. El objetivo era crear equipos formados por alumnos cuyas características medias no se desviaran significativamente de las del aula (*i.e.*, de todo el conjunto de estudiantes). De esta manera, se promovía la igualdad o «interhomogeneidad» entre equipos.

Para estudiar el rendimiento del algoritmo genético, se llevó a cabo un proceso experimental comparándolo con dos métodos adicionales: uno aleatorio y otro exhaustivo. El método aleatorio consistía en un algoritmo que generaba equipos agrupando a los estudiantes de manera aleatoria, mientras que el exhaustivo enumeraba y evaluaba todas las posibles combinaciones para garantizar encontrar la solución óptima. Este último era equivalente a un método exacto.

Tras obtener los respectivos resultados computacionales, se demostró que el método exhaustivo lograba resolver instancias pequeñas del problema en un tiempo relativamente bajo, pero conforme aumentaba el tamaño del aula, el tiempo crecía de manera drástica. Por otro lado, el tiempo tomado por el algoritmo genético para encontrar soluciones de calidad bastante cercanas a las del método exhaustivo escalaba mucho mejor con el número de alumnos en el aula. El

algoritmo aleatorio, aunque era el más rápido de los tres, ofrecía las peores soluciones, debido a que no era un método tan sofisticado como los otros dos.

En [34] el objetivo era formar equipos y distribuirlos adecuadamente entre una serie de proyectos. Se buscaba que la eficiencia de cada proyecto fuera máxima, es decir, que el equipo encargado de su realización estuviera compuesto por personas predispuestas a colaborar juntas. Esto último estaba ligado a las relaciones sociales entre miembros, lo cual es un factor que los autores defendían y consideraban importante para el buen rendimiento de un equipo.

Para crear equipos que maximizaran la eficiencia de cada proyecto, se desarrollaron un algoritmo de búsqueda local, un algoritmo de búsqueda de vecindario variable (*Variable Neighbourhood Search*, VNS) y un modelo de programación de restricciones (*Constraint Programming*, CP). El algoritmo VNS es una metaheurística que deriva de la búsqueda local simple, pero que introduce mejoras, ya que a partir de una determinada solución explora múltiples y diversos vecindarios, lo que le permite evaluar una mayor cantidad de soluciones y aumentar las posibilidades de generar mejores resultados. Con la construcción de un modelo de CP, a diferencia de un modelo matemático de optimización, se pretende encontrar soluciones factibles que cumplan con determinadas restricciones, sin necesidad de optimizar la función objetivo. En este caso, a partir de dicho modelo se pretendía, para cada proyecto, formar equipos compuestos por personas con ciertas habilidades requeridas, y que el tiempo de dedicación al proyecto de cada persona no sobrepasara su disponibilidad máxima, sin tener en cuenta las relaciones sociales de los miembros (*i.e.*, lo que medía la eficiencia de un proyecto).

Tras experimentar con los tres algoritmos, los resultados obtenidos con cada uno demostraron la clara superioridad del algoritmo VNS sobre los otros dos para proporcionar soluciones de mayor calidad.

Los autores de [2] desarrollaron una herramienta informática que integraba un algoritmo genético para la formación de equipos en base a la teoría de roles de Belbin y la diversidad de géneros. Para ello, en primer lugar la herramienta permitía al usuario determinar el número de miembros por equipo o el número total de equipos a crear. Esta información, junto con la correspondiente a las «puntuaciones Belbin» y el género de los alumnos, sería recibida como entrada por el algoritmo genético para producir los equipos deseados. La utilidad de la herramienta fue validada en un aula real compuesta por 32 alumnos, de los cuales 16 eran hombres y 16 eran mujeres. Los resultados obtenidos fueron satisfactorios, ya que la herramienta lograba crear equipos que tuvieran cubiertos todos o la mayor parte de los roles, manteniendo al mismo tiempo un número equilibrado de hombres y mujeres.

La revisión realizada de los trabajos [32, 33, 34, 2], cuyas características se resumen en la tabla 3, ha permitido observar que el uso de métodos aproximados de optimización, como los algoritmos que emplean la búsqueda local o los algoritmos genéticos, resulta ser una opción efectiva (se obtienen soluciones de buena calidad) y eficiente (las soluciones se obtienen en tiempos razonables) para resolver el problema de la formación de equipos.

Por otro lado, también se han presentado múltiples criterios que pueden utilizarse para guiar el proceso de formación de equipos potencialmente eficaces: tendencia de comportamiento (roles Belbin), género, competencias de distinto tipo, etc. De hecho, la consideración simultánea de varios criterios podría aumentar las probabilidades de formar equipos verdaderamente más eficaces que teniendo en cuenta un solo criterio.

Título/Referencia	Criterios	Métodos de resolución	Resultados
Synergistic team composition: A computational approach to foster diversity in teams [32]	- Competencias técnicas (*hard skills*) - Diversidad de géneros - Diversidad de personalidades (MBTI)	- Exacto - Aproximado: algoritmo de búsqueda local	Soluciones óptimas obtenidas con el algoritmo exacto, y aproximadas a estas las obtenidas con el algoritmo de búsqueda local. Sin embargo, el tiempo de este último escalaba mejor con la talla del problema.
A genetic algorithm approach for group formation in collaborative learning considering multiple student characteristics [33]	- Competencias técnicas (*hard skills*) y personales (*soft skills*)	- Exacto - Aproximado: algoritmo genético; algoritmo aleatorio	Igual que [23], solo que en este caso un algoritmo genético era el que superaba al método exacto en cuestión de tiempo, y la calidad de las soluciones obtenidas con estos dos eran bastante similares. El algoritmo aleatorio era el peor de los tres en ambos aspectos.
The multiple team formation problem using sociometry [34]	- Relaciones sociales	- Aproximado: algoritmo de búsqueda local; algoritmo VNS - Modelo de programación de restricciones (CP)	El algoritmo VNS era el que mejores soluciones proporcionaba. El tiempo no fue evaluado.
Using the Behavioural Tendency of Students in a Team Environment for Team Formation [2]	- Tendencia de comportamiento (Belbin) - Diversidad de géneros	- Aproximado: algoritmo genético	Buenos resultados obtenidos con el algoritmo genético. El tiempo no fue evaluado.

Tabla 3. Resumen de las características de cuatro trabajos distintos del estado del arte de la formación de equipos

2.4. Propuesta

Además de los trabajos [33, 2] revisados en la sección anterior (2.3.2), se pueden encontrar muchos otros trabajos del estado del arte de la formación de equipos que han empleado algoritmos genéticos. En general, puede decirse que la aplicación de este tipo de metaheurísticas es muy popular y frecuente para resolver este problema. En los otros dos trabajos revisados en la sección anterior, [32, 34], se emplearon algoritmos basados en la búsqueda local, pero a diferencia de los algoritmos genéticos, no son un tipo de algoritmos que se hayan utilizado tanto en el contexto de la formación de equipos. Lo mismo ocurre con GRASP, una metaheurística que integra la búsqueda local pero cuya aplicación también ha sido prácticamente nula o muy escasa en dicho contexto. Además, hasta donde se tiene conocimiento, no parece haber trabajos publicados hasta ahora que hayan realizado una comparativa directa entre el rendimiento de GRASP y un algoritmo genético para la formación de equipos. Por ello, y considerando la efectividad comprobada tanto de los algoritmos basados en búsqueda local como de los algoritmos genéticos para abordar este problema (*i.e.*, los resultados demostrados en los trabajos [32, 33, 34, 2]), en el presente trabajo se lleva a cabo dicha comparación como una contribución novedosa.

En cuanto a ciertos aspectos de la modelización matemática que se realiza del problema de la formación de equipos en el aula y del diseño de los métodos propuestos (*i.e.*, GRASP y un algoritmo genético), se puede decir que, de los cuatro trabajos revisados en la sección anterior, el presente trabajo guarda más similitudes con [2], aunque al mismo tiempo presenta algunas diferencias con este.

En primer lugar, en el presente trabajo también se tiene en cuenta la tendencia de comportamiento de los alumnos (*i.e.*, sus roles Belbin característicos) y su género para guiar el proceso de formación de equipos potencialmente eficaces. Por ende, la estimación de la eficacia de un equipo se realiza en función de la cobertura de roles y del balance entre la proporción de miembros de cada género. En la sección 3, donde se modeliza matemáticamente el problema, se

detallan las métricas diseñadas para evaluar esos dos aspectos, las cuales difieren de las utilizadas en [2].

Por un lado, para medir el grado de cobertura de roles en un equipo, los autores de [2] calculaban las medias de las puntuaciones obtenidas por los miembros del equipo en cada rol, es decir, un total de nueve medias (consideraron el modelo de equipo con nueve roles propuesto por Belbin, que incluye el rol de especialista). Posteriormente, promediaban los resultados de esas medias. Sin embargo, la definición de esta métrica hace que su rango de valores varíe en función de las puntuaciones de los miembros de un equipo u otro, lo que implica que no exista un valor máximo ni mínimo específico. Como consecuencia, esta métrica no permite saber, para cualquier equipo, si todos o ninguno de los roles están cubiertos. En otras palabras, no permite identificar de manera precisa si un equipo es mejor o peor en ese aspecto. En contraste, en el presente trabajo se utiliza una métrica más efectiva e informativa para evaluar a un equipo en base a la cobertura de roles, la cual fue propuesta originalmente por los autores [23].

Por otro lado, para medir el grado de presencia de géneros diversos en un equipo, la fórmula que se utilizaba era $1 - \left(\frac{|f_j - m_j|}{N_j}\right)$, donde N_j representaba el número total de miembros del equipo, y tanto f_j como m_j denotaban, respectivamente, el número de mujeres y de hombres presentes en el mismo. La definición de esta métrica permite evaluar de manera más positiva a un equipo con mayor balance de géneros. Cuando existe un equilibrio total entre el número de hombres y mujeres dentro de un equipo, entonces el equipo es evaluado con un valor máximo de 1. Por el contrario, si un equipo está compuesto por miembros de un solo género, entonces el equipo es evaluado con un valor mínimo de 0. No obstante, esta métrica no considera ciertos escenarios. Por ejemplo, en un aula compuesta por estudiantes del mismo género o con una cantidad muy escasa de estudiantes de uno de los dos géneros, no sería posible formar equipos con el mismo número de hombres y mujeres o, al menos, que la cantidad fuera similar. Por ende, la métrica evaluaría de manera baja a esos equipos hipotéticos, lo cual no sería muy coherente debido a que las condiciones de esos tipos de aulas no favorecen la creación de equipos con la misma cantidad de alumnos de ambos géneros. Por ello, en lugar de evaluar a un equipo según la similitud numérica de hombres y mujeres, se puede valorar que la proporción de uno de los dos géneros (cualquiera) se aproxime a la proporción correspondiente en el aula. De este modo, en un aula compuesta por un 100% de hombres, no habría ninguna sola mujer en cualquier equipo que se formara, pero tampoco se evaluaría de manera negativa ese hecho, ya que la proporción de mujeres en cualquier equipo sería coherente con la del aula (0%). En cambio, un equipo compuesto únicamente por hombres en un aula con un 50% de alumnos de cada género no sería evaluado de manera positiva, dado que las condiciones del aula permitirían la inclusión de más mujeres. Por tanto, en el presente trabajo se propone una métrica de este estilo que evalúe con mayor precisión el balance de géneros en un equipo, teniendo en cuenta las condiciones del aula.

En segundo lugar, aunque los métodos propuestos en el presente trabajo para la formación de equipos (*i.e.*, GRASP y un algoritmo genético) no están integrados en una herramienta informática, se proporcionan para su uso como *scripts* de Python, el lenguaje de programación utilizado para su desarrollo. Estos algoritmos reciben como datos de entrada las puntuaciones Belbin y el género de los alumnos, así como el número de alumnos en el aula y el número de miembros por equipo, pero no están diseñados para recibir el número total de equipos a crear. Esto se debe a que, para la realización de tareas o proyectos en un contexto educativo, los docentes suelen pedir a sus alumnos que se agrupen en equipos de un tamaño específico, en lugar de lo contrario (*i.e.*, que formen un número predeterminado de equipos). Por ello, las metaheurísticas

propuestas están diseñadas para crear equipos con un tamaño especificado M, siempre y cuando el número de alumnos en el aula N sea divisible por M. Si no, el número total de equipos que se crean viene determinado mediante la fórmula $redondear\left(\frac{N}{M}\right)$, pero asegurando que sus tamaños sean lo más similares posible, de modo que la diferencia entre el número de miembros de un equipo y de otro no sea mayor a uno. El diseño de las metaheurísticas se detalla en la sección 4.

Por último, durante su ejecución, las metaheurísticas diseñadas generan y evalúan (en función de los dos criterios considerados) una variedad de soluciones, cada una de las cuales representa una combinación distinta para agrupar a los estudiantes en equipos. Para ello, cada solución consiste en un vector en el que cada posición se asocia con un alumno distinto, y su valor indica a qué equipo es asignado. Esta es la misma representación que utilizaron los autores de [2]. Sin embargo, su algoritmo genético empleaba una heurística de cruce (*Single-Point Crossover*) que podía producir algunas soluciones inválidas, es decir, equipos demasiado grandes o pequeños en comparación con otros. Para que el algoritmo descartara y no proporcionara soluciones generadas que no fueran factibles, se les asignaba un valor de *fitness* igual a cero. No obstante, si bien esta práctica es correcta, también puede provocar que el algoritmo tarde más en converger a una solución factible y de buena calidad, debido a que una parte de su tiempo la dedica a generar soluciones no válidas. En contraste, todas las soluciones que generan las metaheurísticas diseñadas en el presente trabajo son factibles, ya que la diferencia entre el número de miembros de un equipo y de otro es, como mucho, de uno. Por ende, la heurística de cruce integrada en el algoritmo genético desarrollado logra mantener la validez de las soluciones, por lo que se espera que el algoritmo demore menos en converger de lo que lo haría si generara soluciones no factibles. El diseño del algoritmo genético, así como la heurística de cruce empleada, se detalla concretamente a lo largo de la sección 4.2.

3. Modelización del problema

Para abordar el problema de la formación de equipos en el aula desde un enfoque algorítmico, en este caso mediante la aplicación de metaheurísticas, primero es necesario modelizar el problema de forma matemática.

Por tanto, se denota como $S = \{s_1, s_2, \dots, s_N\}$ al conjunto de N estudiantes que componen un aula. Asimismo, se denota como T a un conjunto de equipos factibles que podrían formarse a partir de S. Un equipo $t_i \in T$ se considera factible porque es disjunto de cualquier otro equipo del mismo conjunto (*i.e.*, no comparte ningún miembro) y porque además su tamaño $|t_i|$ es igual o difiere solo en una unidad (*i.e.*, por un único miembro) respecto al tamaño de cualquier otro equipo del conjunto.

Dada la notación indicada, la función objetivo del problema se define de la siguiente manera:

$$\max f(T) = \frac{1}{|T|} \times \sum_{t_i \in T} E(t_i)$$

En otras palabras, el problema de formar equipos en un aula consiste en encontrar, dentro de la gran cantidad de conjuntos o combinaciones de equipos con las características de un conjunto T que podrían existir, aquel que maximice la media de la eficacia de los equipos, según expresa la función objetivo definida. La eficacia de un equipo t_i viene dada, a su vez, por una función de aproximación E que estima el rendimiento que dicho equipo podría llegar a tener, es decir, qué tan eficaz podría ser. Se dice que esta función estima y no mide de forma exacta la eficacia de un equipo ya que no es posible evaluar con exactitud el grado de rendimiento de un equipo antes de que este se ponga finalmente en acción. Sin embargo, dicha función se relaciona con algunos criterios de formación de equipos descritos a lo largo de la sección 2.3, cuya consideración ha permitido crear equipos con buen rendimiento, según algunos de los trabajos revisados también en dicha sección. En particular, la función E se vincula con la teoría de roles de Belbin y la diversidad de géneros, expresando que la eficacia de un equipo está determinada por la cobertura de roles y el balance entre la proporción de miembros de cada género, tal y como se muestra a continuación:

$$E(t_i) = \lambda \times B_i + (1 - \lambda) \times G_i$$

B_i y G_i se consideran dos métricas que permiten, a partir de la información de los miembros de un equipo t_i (*i.e.*, sus puntuaciones Belbin y su género), evaluar respectivamente la cobertura de roles y el balance de géneros en dicho equipo. A continuación, se definen ambas métricas.

Comenzando con B_i, se denota cada rol Belbin como r_k, con $k = 1, \dots, 8$ (se considera el modelo de equipo con ocho roles propuesto por Belbin, en lugar del modelo que incluye el noveno rol de especialista). Además, se denota como $b_{j,k}$ la puntuación conseguida por un estudiante s_j en el rol r_k al completar el Inventario de Autopercepción de Belbin. Se sabe que un estudiante puede desempeñar un determinado rol dentro de un equipo siempre y cuando la puntuación que haya obtenido en ese rol sea alta o muy alta (ver tabla 2, sección 2.3.1). Por tanto, se denota como

u_k el umbral que es necesario sobrepasar o igualar para que la puntuación de un alumno en el rol r_k sea considerada, al menos, alta. Entonces, la puntuación obtenida por un equipo t_i en el rol r_k, representada como $b_{i,k}$, se define de la siguiente manera:

$$b_{i,k} = \begin{cases} 1, & si\ \exists s_j \in t_i\ tal\ que\ b_{j,k} \geq \mu_k \\ 0, & en\ otro\ caso \end{cases}$$

En otras palabras, si dentro de un equipo t_i hay un miembro cuya puntuación en el rol r_k es alta o muy alta, dicho equipo consigue un punto en ese rol. Entonces, la métrica que permite evaluar la cobertura de roles en un equipo, B_i, se define de la siguiente manera:

$$B_i = \frac{1}{8} \times \sum_{k=1}^{8} b_{i,k}$$

Esta métrica toma valores entre $[0,1]$. De este modo, cuanto mayor o menor sea B_i, mayor será la abundancia o escasez de roles únicos en un equipo, respectivamente. Cuando un equipo tiene los ocho roles cubiertos, es evaluado con un valor máximo de 1. Por el contrario, un equipo es evaluado con un valor mínimo de 0 si ninguno de sus miembros es capaz de desempeñar ningún rol.

Con respecto al balance de géneros en un equipo, se tiene en cuenta la proporción de uno de los dos géneros en el equipo, por ejemplo, el femenino, y también la proporción correspondiente en el aula. Por tanto, se denota como m_i la proporción de mujeres en un equipo t_i, y como m_c la proporción de mujeres en el aula. Entonces, la métrica que permite evaluar el balance de géneros en un equipo, G_i, se define de la siguiente manera:

$$G_i = 1 - |m_i - m_c|$$

El rango de valores de esta métrica también está definido en el intervalo $[0,1]$. De esta manera, si la proporción de mujeres en un equipo es la misma que la proporción de mujeres en el aula, por ejemplo, 50% o 70%, entonces el equipo es evaluado con un valor máximo de 1, ya que también contaría con una proporción de hombres igual a la del aula. Además, cuando la proporción de uno de los dos géneros en un equipo no sobrepasa ni queda por debajo de la proporción correspondiente en el aula, ese hecho permite que las proporciones de ambos géneros en otro equipo sean igualmente equilibradas. Por ello, se considera que esta es una forma adecuada de promover la diversidad de géneros en todos los equipos.

En la formulación de la función E, que estima la eficacia de un equipo, también aparece el símbolo λ. Este es un factor de peso que toma valores entre $[0,1]$ y representa la importancia relativa de la cobertura de roles y el balance de géneros en dicha estimación. Si el valor asignado a este factor es lo suficientemente alto ($\lambda > 0.5$), se consideraría que la cobertura de roles es un aspecto más influyente en la eficacia de un equipo. Por el contrario, si el valor asignado es lo suficientemente bajo ($\lambda < 0.5$), se consideraría que el balance de géneros tiene mayor influencia en la eficacia de un equipo. No obstante, si el valor asignado es exactamente 0.5, ambos aspectos se considerarían igualmente importantes. Cabría esperar que, dado un valor específico del factor

λ, la tendencia de las metaheurísticas diseñadas (encargadas de generar y evaluar diferentes combinaciones de equipos) esté orientada hacia la creación de equipos cuya estructura favorezca más la cobertura de roles que el balance de géneros o viceversa, o bien ambos aspectos en un grado similar. Todo dependerá de lo que permita maximizar la eficacia media de los equipos.

Por último, dado que la función E estima la eficacia de un equipo t_i como una combinación lineal de la cobertura de roles (B_i) y el balance de géneros (G_i), y ambos aspectos se evalúan con un valor entre 0 y 1, entonces la eficacia de un equipo también puede variar entre un mínimo de 0 y un máximo de 1. Por tanto, al formar equipos en un aula, se buscaría encontrar una combinación con las características de un conjunto T en la que la eficacia de cada equipo alcanzase o se aproximase lo más posible a 1, ya que esto implicaría que la eficacia media de los equipos también se acercase al máximo valor de 1.

4. Diseño de las metaheurísticas

El objetivo de esta sección es explicar, en primer lugar, cómo se representa una solución para el problema de la formación de equipos (sección 4.1). Posteriormente, se detalla cómo operan las dos metaheurísticas que se han diseñado específicamente para generar soluciones optimizadas al problema: GRASP (sección 4.2) y un algoritmo genético (sección 4.3).

4.1. Representación de una solución

Una solución representa una de las tantas combinaciones que podrían existir para agrupar a los estudiantes en equipos, utilizando para ello un vector donde cada posición se asocia con un alumno distinto y su valor indica el equipo al que es asignado. A modo de ejemplo, supongamos que en un aula con N alumnos, donde se quieren formar equipos de tamaño M, surgen un total de G equipos factibles, de acuerdo con la definición de factibilidad de un equipo realizada en la sección 3 de modelización del problema. Por tanto, una manera de agrupar los N alumnos la indica la solución de la figura 15. Esta solución señala que el alumno 1 pertenece al equipo 3, el alumno 2 es miembro del equipo 1, el alumno 3 es parte del equipo G, el alumno 4 pertenece al equipo 3, …, y el alumno N es miembro del equipo 2.

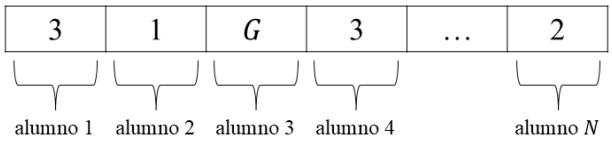

Figura 15. Representación de una solución para la formación de equipos

Las soluciones que generan las metaheurísticas diseñadas son del tipo representado en la figura 15. Dado que una solución representa una combinación de equipos, su calidad está determinada por el valor que alcance de la función objetivo definida en la sección 3, es decir, la eficacia media de los equipos. Asimismo, dado que la eficacia de un equipo se estima entre un valor mínimo de 0 y un valor máximo de 1, entonces la calidad de una solución también se evalúa dentro de ese rango de valores.

4.2. GRASP para la formación de equipos

Como se introdujo en la sección 2.2.1, GRASP es una metaheurística en la que en cada iteración se llevan a cabo dos fases principales: una fase de construcción de una solución inicial y una fase de búsqueda local para explorar el vecindario de dicha solución hasta encontrar un óptimo local. A continuación, se explica cómo se han diseñado ambas fases específicamente para la formación de equipos (secciones 4.2.1 y 4.2.2, respectivamente). Acto seguido, se presenta el esquema de operación de GRASP (sección 4.2.3), donde se integran las fases definidas.

4.2.1. Fase de construcción

Aunque en la literatura se pueden encontrar diferentes heurísticas que pueden utilizarse para llevar a cabo la fase de construcción, la que se emplea en este caso es la que generalmente se suele integrar en el esquema de operación de GRASP: la construcción voraz y aleatoria (*Greedy Randomized Construction*). En líneas generales, las operaciones que realiza esta heurística, descritas en el pseudocódigo 4, son similares a las del pseudocódigo 2 de la sección 2.2.1, pero adaptadas a la modelización realizada del problema de la formación de equipos en el aula.

heuristica $Greedy\_Randomized\_Construction(\alpha)$
Número de alumnos en el aula: N
Tamaño de los equipos: M
$Solucion \leftarrow$ Vector con N ceros
Número de equipos que se crearán: $G \leftarrow redondear(N/M)$
Inicializar el conjunto que contiene los identificadores de los N alumnos pendientes de asignar: $C = \{1,2,3,\dots,N\}$
mientras $C \neq \emptyset$ **hacer**
    para $t = 1,\dots,G$ **hacer**
        Evaluar el coste incremental $c(s,t)$ para cada alumno $s \in C$ asociado con su incorporación al equipo t
        $c^{\min} \leftarrow \min\{c(s,t) | s \in C\}$
        $c^{\max} \leftarrow \max\{c(s,t) | s \in C\}$
        $umbral \leftarrow c^{\max} + \alpha(c^{\min} - c^{\max})$
        $RCL \leftarrow \{s \in C | c(s,t) \geq umbral\}$
        Seleccionar un alumno s' de la RCL de manera aleatoria
        Sustituir por t el cero ubicado en la posición s' de $Solucion$
        Actualizar el conjunto de alumnos pendientes de asignar C eliminando del mismo el identificador s'
        si $C = \emptyset$ **entonces**
            interrumpir bucle **para**
devolver $Solucion$
fin $Greedy\_Randomized\_Construction$

Pseudocódigo 4. Fase de construcción de GRASP para la formación de equipos: heurística de construcción voraz y aleatoria

Básicamente, la heurística comienza inicializando una solución como un vector que contiene tantos ceros como alumnos en el aula haya (N). Esto se debe a que cada posición del vector se asocia con un alumno distinto de los N que hay en total y su valor indica el equipo al que es asignado, de acuerdo con la forma de representar una solución para la formación de equipos definida en la sección 4.1. Al principio, todos los valores del vector son ceros, insinuando que ningún estudiante forma parte de ningún equipo todavía.

Seguidamente, se calcula el número de equipos que se crearán G mediante la fórmula $redondear\left(\frac{N}{M}\right)$, donde N es el número de alumnos en el aula y M el tamaño que se desea que tengan los equipos a formar (según el docente o cualquier usuario). Los G equipos que se crearán estarán compuestos por el número de miembros M indicado, siempre y cuando N sea divisible por M. Si no, el tamaño de algunos o de todos los equipos puede diferir ligeramente de M, pero asegurando que entre ellos su tamaño sea lo más similar posible.

Acto seguido, los equipos finalmente comienzan a ser organizados mediante un patrón iterativo. Este patrón consiste en asignar primero un alumno al equipo 1, luego un alumno al equipo 2, y así sucesivamente hasta asignar un alumno al equipo G. Si todavía quedara algún alumno que no forme parte de ningún equipo, el patrón se repetiría nuevamente, asignando un nuevo alumno al equipo 1, otro al equipo 2, y así sucesivamente hasta que todos los alumnos pertenezcan a un equipo. Es este patrón lo que permite que, al final del procedimiento, todos los equipos que se hayan creado sean similares en tamaño, de modo que la diferencia entre el número

de miembros de dos equipos cualesquiera no sea mayor a uno. Al mismo tiempo, garantiza que los equipos sean disjuntos. Esto concuerda con la modelización realizada del problema en la sección 3, ya que el objetivo es encontrar aquel conjunto T de equipos factibles (*i.e.*, disjuntos y con una diferencia de tamaño máxima de una unidad entre ellos) que maximice la eficacia media de estos. Por tanto, una solución construida por esta heurística cumple con las características de factibilidad de un conjunto T.

En cada paso del patrón definido, el alumno que se incluye en un equipo t es seleccionado del conjunto C, que contiene los identificadores de cada estudiante s pendiente de asignar. Inicialmente, este conjunto contiene los identificadores de todos los estudiantes. La selección de un estudiante u otro para su incorporación a un equipo t varía en función del parámetro α:

- Si $\alpha = 0$: el estudiante s' que se incluye en el equipo t es aquel que, de entre todos los alumnos pendientes de asignar, tiene un coste incremental $c(s', t)$ que beneficia en mayor medida la eficacia del equipo. El coste incremental de un estudiante define cuánto variaría la eficacia de un determinado equipo con su incorporación. Cuanto más positiva sea esta variación, mayor será la cobertura de roles y/o el balance de géneros del equipo, lo que hará que su eficacia esté más cercana a 1.
- Si $\alpha > 0$: existe cierta aleatoriedad en la selección del estudiante s' que se incluye en el equipo t. Por ello, no necesariamente se elige el alumno que incrementa en mayor medida la eficacia del equipo. Lo que realmente sucede es que, del conjunto de alumnos que quedan por asignar, se toma en cuenta un subconjunto con aquellos alumnos que benefician más a la eficacia del equipo en comparación con otros. A continuación, se selecciona de manera aleatoria un alumno de dicho subconjunto para incluirlo en el equipo. El tamaño de este subconjunto aumenta conforme el valor de α es mayor, incrementando así el grado de aleatoriedad en la selección. Cuando α es igual a 1, la selección es totalmente aleatoria.

Tras seleccionar un estudiante s', para señalar que este se incluye en el equipo t, se sustituye por este último valor el cero ubicado en la posición s' de la solución. A continuación, se actualiza el conjunto de alumnos pendientes de asignar C eliminando del mismo el identificador s', indicando que faltan menos alumnos por incluir en algún equipo.

Cuando todos los alumnos ya pertenecen a un equipo ($C = \emptyset$), el procedimiento finaliza devolviendo la solución.

En resumen, cuando $\alpha = 0$, la heurística organiza los equipos paso a paso, priorizando la asignación del alumno que más beneficie a la eficacia de cada equipo. No obstante, si $\alpha > 0$, las asignaciones se realizan con cierto grado de aleatoriedad, sin enfocarse exclusivamente en lo que más beneficia a cada equipo en cada paso. Esta aleatoriedad aumenta a medida que α se aproxima a 1. Por tanto, diferentes valores de α permiten construir soluciones iniciales de calidad potencialmente distinta.

4.2.2. Fase de búsqueda local

Explorar el vecindario de una solución significa, conceptualmente, investigar otras soluciones situadas a su alrededor dentro del espacio de búsqueda. Por tanto, explorar el vecindario de una solución para la formación de equipos consiste en examinar otras soluciones que representan otras

combinaciones distintas para agrupar a los alumnos en equipos, pero igualmente factibles. Estas soluciones vecinas se pueden generar realizando pequeñas modificaciones sobre la solución actual cuyo vecindario se está explorando. En este caso, cada modificación consiste en intercambiar a dos alumnos de equipo.

Por ejemplo, la solución {2,1,2,1} de la figura 16 señala que el alumno 1 y el alumno 3 pertenecen al equipo 2, mientras que el alumno 2 y el alumno 4 forman parte del equipo 1. Una solución de su vecindario se puede generar intercambiando el alumno 1 y el alumno 2 de equipo, creando así la solución {1,2,2,1}. De este modo, mediante todos y cada uno de los posibles intercambios de alumnos pertenecientes a equipos distintos, se genera el vecindario completo de una solución. En el caso de la solución {2,1,2,1}, su vecindario está compuesto por cuatro soluciones diferentes. Todas ellas representan distintas combinaciones para agrupar a los cuatro alumnos en equipos de dos. La eficacia media de los equipos indicados por cada solución vecina podría ser mejor o peor que la de la solución actual, según el valor de la función objetivo de cada una.

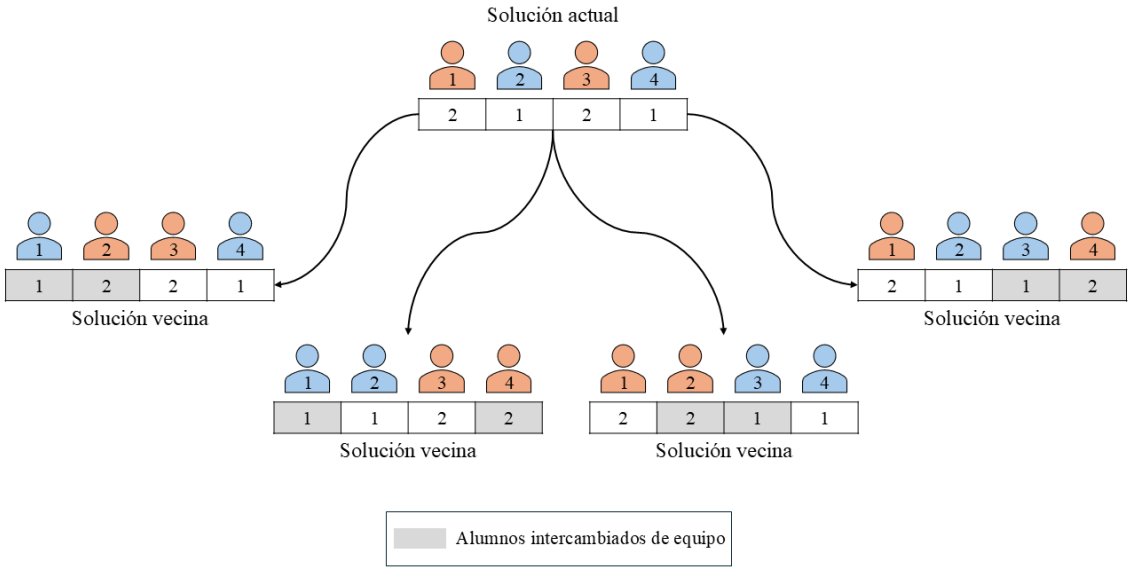

Figura 16. Vecindario de una solución para la formación de equipos. Elaboración propia

Después de introducir cómo se genera el vecindario de una solución para la formación de equipos, las operaciones que se realizan durante la fase de búsqueda local se describen en el pseudocódigo 5.

```
heuristica Busqueda_Local(Solucion, estrategia)
1    mientras Verdadero hacer
2        optimo_local ← 1
3        Generar vecindario de Solucion: V
4        para cada solucion_vecina ∈ V hacer
5            si f(solucion_vecina) > f(Solucion) entonces
6                optimo_local ← 0
7                Solucion ← solucion_vecina
8                si estrategia = "first-improving" entonces
9                    interrumpir bucle para
10       si optimo_local = 1 entonces
11           interrumpir bucle mientras
12   devolver Solucion
fin Busqueda_Local
```

Pseudocódigo 5. Fase de búsqueda local de GRASP para la formación de equipos

El pseudocódigo 5 representa correctamente el objetivo de la búsqueda local, el cual es explorar vecindarios de manera consecutiva hasta hallar con un óptimo local. Para ello, el procedimiento inicia asumiendo que la solución actual ($Solucion$) es un óptimo local ($optimo\_local \leftarrow 1$). Acto seguido, se genera el vecindario de la solución actual para explorarlo, es decir, evaluar las soluciones que forman parte de dicho vecindario con el objetivo de mejorar la solución actual. En caso de encontrar una solución vecina cuyo valor de la función objetivo es mejor que el de la solución actual, se niega que esta última sea un óptimo local ($optimo\_local \leftarrow 0$) y, por tanto, se actualizaría con la nueva solución que es mejor. A partir de ahí, la exploración del vecindario actual continuaría o no en función de si la estrategia de búsqueda local establecida es *first-improving* o no. Si la estrategia es *first-improving*, la exploración se detiene tras encontrar una primera solución mejor que la solución actual para continuar con el procedimiento, es decir, explorar el vecindario correspondiente a la nueva solución. En cambio, si la estrategia no es *first-improving*, se asume que es *best-improving*, por lo que la exploración continua hasta haber evaluado todas las soluciones.

Siempre que el vecindario de una nueva solución va a ser explorado, previamente se asume que es un óptimo local ($optimo\_local \leftarrow 1$). Si durante la exploración del vecindario de una determinada solución no se encuentra ninguna que la mejore, en ningún momento se negaría el hecho de que sea un óptimo local (el valor de $optimo\_local$ se mantendría en 1). Por ende, cuando esto sucede, significa que se ha encontrado un óptimo local, por lo que el procedimiento finaliza devolviendo esta solución.

4.2.3. Esquema de operación de GRASP

En el pseudocódigo 6 se describe el esquema de operación de la metaheurística GRASP diseñada para generar soluciones optimizadas al problema de la formación de equipos en el aula.

```
algoritmo GRASP(Tiempo_Limite, α, estrategia)
1    Leer_Instancia()
2    Valor de la función objetivo de la mejor solución: f* ← −∞
3    mientras Tiempo_Limite no sea alcanzado hacer
4        Solucion ← Greedy_Randomized_Construction(α)
5        Solucion ← Busqueda_Local(Solucion, estrategia)
6        si f(Solucion) > f* entonces
7            Mejor_Solucion ← Solucion
8            f* ← f(Solucion)
9    devolver Mejor_Solucion
fin GRASP
```

Pseudocódigo 6. Esquema de operación de GRASP para la formación de equipos

Realmente, se trata del esquema general descrito en el pseudocódigo 1 de la sección 2.2.1. Lo único que cabe aclarar es que, nada más arrancar el algoritmo, los datos que extrae y almacena al procesar una instancia del problema proporcionada como entrada están relacionados con el número de alumnos en el aula N, el tamaño de los equipos M, las puntuaciones Belbin de los alumnos y su género. Estos datos, posteriormente, son los que se utilizan durante la fase de construcción para generar una solución inicial al problema.

Además, la repetición de las fases de construcción y búsqueda local, así como la actualización de la mejor solución, se detiene cuando se alcanza un tiempo límite de ejecución establecido. Tras ello, el algoritmo termina devolviendo la mejor solución de todas las que ha podido generar para el problema.

4.3. Algoritmo genético para la formación de equipos

Dado que un algoritmo genético consiste en la evolución progresiva de una población de soluciones, a continuación se define la población inicial que será objeto de dicha evolución (sección 4.3.1). Posteriormente, se describen las heurísticas utilizadas para llevar a cabo cada una de las operaciones que permiten la evolución periódica de dicha población: la selección de progenitores (sección 4.3.2), el cruce entre progenitores (sección 4.3.3), la mutación de los descendientes (sección 4.3.4) y el reemplazo de la población (sección 4.3.5). Finalmente, se presenta el esquema de operación del algoritmo genético (sección 4.3.6), donde se integran todos los componentes mencionados.

4.3.1. Población inicial

La población inicial P que se pretende evolucionar se compone de n soluciones creadas de manera aleatoria. El tamaño de la población, n, podría ser cualquier valor, pero en el caso del algoritmo genético diseñado se establece en 100 soluciones. Para la generación de cada una de estas soluciones, se emplea una heurística de diseño propio denominada *Randomized Construction*, que está basada en la heurística de construcción voraz y aleatoria utilizada en la fase de construcción de GRASP, pero con un valor de α igual a 1. Recordemos que, cuando se usa este valor de α, la heurística de construcción voraz y aleatoria forma equipos de alumnos asignándolos aleatoriamente, sin tener en cuenta la variación en la eficacia de los equipos. Por ende, la solución que se construye se considera aleatoria.

Dicho esto, las operaciones que realiza la heurística *Randomized Construction*, descritas en el pseudocódigo 7, son similares a las del pseudocódigo 4 de la sección 4.2.1 correspondiente a la heurística de construcción voraz y aleatoria, pero sin calcular costes incrementales ya que asigna aleatoriamente a los alumnos sin importar cómo varía la eficacia de los equipos. Asimismo, dado que está basada en la heurística de construcción voraz y aleatoria, genera soluciones factibles (*i.e.*, equipos disjuntos y con una diferencia de tamaño máxima de una unidad entre ellos) cuya representación corresponde a la que se muestra en la figura 15 de la sección 4.1.

```
heuristica Randomized_Construction
1     Número de alumnos en el aula: N
2     Tamaño de los equipos: M
3     Solucion ← Vector con N ceros
4     Número de equipos que se crearán: G ← redondear(N/M)
5     Inicializar el conjunto que contiene los identificadores de los N alumnos pendientes de asignar: C = {1,2,3, ..., N}
6     mientras C ≠ ∅ hacer
7           para t = 1, ..., G hacer
8                 Seleccionar un alumno s' de C de manera aleatoria
9                 Sustituir por t el cero ubicado en la posición s' de Solucion
10                Actualizar el conjunto de alumnos pendientes de asignar C eliminando del mismo el identificador s'
11                si C = ∅ entonces
12                      interrumpir bucle para
13    devolver Solucion
fin Randomized_Construction
```

Pseudocódigo 7. Creación de una solución aleatoria para la población: heurística *Randomized Construction*

4.3.2. Selección de progenitores

Para seleccionar un conjunto de soluciones de la población que actúen como progenitores, la heurística que se emplea consiste básicamente en elegir cualquier solución con una probabilidad proporcional a su valor de *fitness* (*i.e.*, su valor de la función objetivo). Esta estrategia se denomina selección proporcional al *fitness*, o más conocida en inglés como *Roulette Wheel Selection* (selección por ruleta) [35].

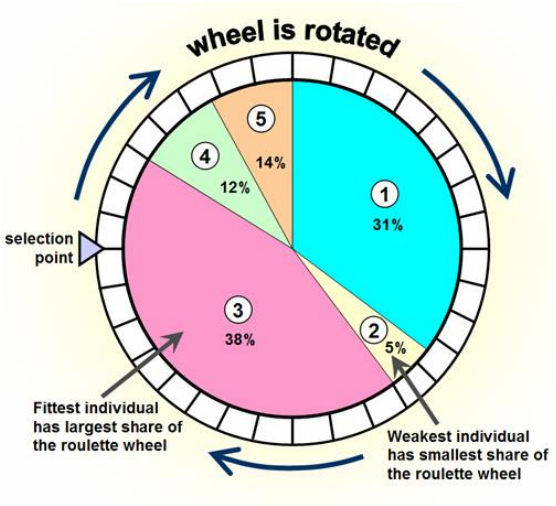

Figura 17. Selección de progenitores: heurística de selección proporcional al *fitness* (*Roulette Wheel Selection*) [36]

La lógica de esta estrategia se puede entender mejor observando la figura 17. Cada sector de la ruleta hace referencia a una solución diferente de la población, y el área que ocupa cada sector es proporcional al valor de *fitness* de la solución a la que hace referencia con respecto al valor total de *fitness* de la población (*i.e.*, dividido entre la suma de los valores de *fitness* de todas las soluciones de la población). Si la ruleta se girara, habría más posibilidades de que el indicador se detuviera apuntando hacia los sectores rosa o azul, ya que estos abarcan una mayor parte del área total de la ruleta. Por ende, las soluciones con mejores valores de *fitness* son las que tienen más probabilidades de ser elegidas que aquellas con valores inferiores. Por esta misma razón, si la ruleta se girara múltiples veces, las soluciones con mejores valores de *fitness* tenderían a seleccionarse en más de una ocasión.

4.3.3. Cruce entre progenitores

La heurística que se utiliza para cruzar progenitores es una adaptación de la heurística que en la literatura se conoce como *Cut-and-Crossfill*. Se trata de una heurística que, en principio, está diseñada para ser aplicada a permutaciones, es decir, un tipo de representación de una solución que consiste en un vector de números enteros que no se repiten (como se introdujo en la sección 2.2). En este trabajo, una solución para la formación de equipos también se representa con un vector de números enteros, solo que estos números sí pueden repetirse dos o más veces, lógicamente porque diferentes alumnos pueden pertenecer al mismo equipo (ver figura 15, sección 4.1). Debido a que esta representación difiere de una permutación, la heurística mencionada no es adecuada para realizar el cruce entre dos soluciones para la formación de equipos, ya que produciría soluciones no factibles. Más adelante se mencionará exactamente la razón por la que esto ocurriría.

Por tanto, para llevar a cabo el cruce entre dos progenitores cualesquiera y generar dos nuevas soluciones o descendientes también factibles, se ha realizado una ligera modificación de la heurística original *Cut-and-Crossfill*, la cual se ha nombrado como *Cut-and-Crossfill-TF*. Esta heurística propuesta, derivada de la heurística *Cut-and-Crossfill*, constituye una innovación de este trabajo, ya que no existe como tal en la literatura de los algoritmos genéticos. Además, la creación de esta heurística responde a la carencia de estrategias alternativas que, a partir de dos soluciones factibles representadas como la figura 15 de la sección 4.1, produzcan nuevas soluciones o descendientes igualmente factibles.

Las operaciones que realiza la heurística propuesta se describen en el pseudocódigo 8.

heuristica $Cut\_and\_Crossfill\_TF(Progenitor_1,\ Progenitor_2)$
1 $L \leftarrow$ Número total de genes de $Progenitor_1/Progenitor_2$
2 Contar el número de veces v_k que cada valor o alelo k aparece en $Progenitor_1/Progenitor_2$
3 Generar un número aleatorio $i \in \{1, \dots, L-1\}$
4 $Descendiente_1 \leftarrow$ Hereda los i primeros genes de $Progenitor_1$
5 $Descendiente_2 \leftarrow$ Hereda los i primeros genes de $Progenitor_2$
6 Escanear de izquierda a derecha los valores de $Progenitor_2$ desde la posición $i+1$ hasta acabar en la posición i, e ir rellenando el material genético de $Descendiente_1$ con cada valor k escaneado, si es que dicho valor no aparece todavía v_k veces en $Descendiente_1$
7 Hacer lo mismo que el paso anterior pero con el $Progenitor_1$ y el $Descendiente_2$
8 **devolver** $Descendiente_1,\ Descendiente_2$
fin $Cut\_and\_Crossfill\_TF$

Pseudocódigo 8. Cruce entre progenitores: heurística *Cut-and-Crossfill-TF*

De manera complementaria al pseudocódigo 8 y para facilitar la comprensión del funcionamiento de la heurística, se presenta un ejemplo de su aplicación en la figura 18.

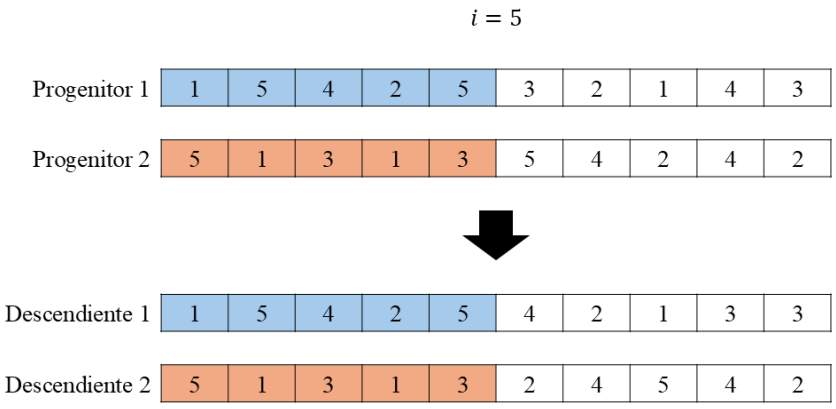

Figura 18. Ejemplo de cruce entre progenitores utilizando la heurística *Cut-and-Crossfill-TF*

Siguiendo los pasos definidos en el pseudocódigo 8, en primer lugar se genera un número aleatorio i entre $\{1, \dots, L-1\}$, siendo L la longitud o el número total de genes de cualquiera de los dos progenitores, ya que ambos siempre tendrán la misma cantidad de genes. En este caso, el número aleatorio resultante es $i = 5$. Dicho número indica que el Descendiente 1 hereda los cinco primeros genes del Progenitor 1, mientras que el Descendiente 2 hereda los cinco primeros genes del Progenitor 2.

A continuación, se procede a rellenar el material genético de los dos descendientes, ya que, lógicamente, ambos deben tener la misma cantidad de genes que sus progenitores. Para ello, primero hay que tener en cuenta el número de veces que cada valor o alelo aparece en cualquiera de los dos progenitores. En este caso, los valores 1, 2, 3, 4 y 5 aparecen en dos ocasiones cada uno.

Luego, se escanean de izquierda a derecha los valores del Progenitor 2, iniciando desde la posición $i + 1 = 6$ hasta finalizar en la posición $i = 5$. A medida que se escanean los valores, se va rellenando el material genético del Descendiente 1 con cada uno de ellos, siempre y cuando no aparezcan ya dos veces en dicho descendiente. Así, al escanear de izquierda a derecha los valores del Progenitor 2 desde la posición 6, el primer valor que se encuentra es el 5, el cual no se añade al Descendiente 1 debido a que ya aparece dos veces. Sin embargo, el siguiente valor es el 4, que sí se añade porque no aparece ni siquiera una vez en el Descendiente 1. Continuando de esta manera, se terminaría de rellenar completamente el material genético del Descendiente 1. El mismo proceso se realizaría con el Progenitor 1 y el Descendiente 2 para rellenar el material genético de este último.

De este modo, la heurística *Cut-and-Crossfill-TF* permite que los descendientes que se producen por el cruce entre los progenitores representen otras combinaciones distintas pero válidas para agrupar a los estudiantes en equipos, ya que estos mantienen la similitud en tamaño sin que el número de miembros de ningún equipo exceda en dos o más al de otro, y también se mantienen disjuntos.

Dado que la heurística propuesta deriva de la heurística original *Cut-and-Crossfill* que aparece en la literatura, ambas funcionan de manera similar. En lo único que difiere la heurística original

de la propuesta es que, para rellenar el material genético de un descendiente, solo se añaden los valores escaneados del progenitor correspondiente que no aparezcan ni una sola vez. En otras palabras, aquellos valores escaneados que ya aparezcan en el descendiente no se añaden. Por ello, si la heurística original *Cut-and-Crossfill* se aplicara para cruzar progenitores, se generarían soluciones no factibles en las que habría alumnos que no pertenecerían a ningún equipo. Por ejemplo, sabiendo que el Descendiente 1 de la figura 18 hereda los cinco primeros genes del Progenitor 1, al escanear los valores del Progenitor 2 para rellenar el material genético del Descendiente 1, este último se quedaría como $\{1,5,4,2,5,3,-,-,-,-\}$. Por tanto, en este caso habría cuatro alumnos que no pertenecerían a ningún equipo. Este es el motivo por el cual realizar una modificación de la heurística original ha resultado necesario para poder generar nuevas soluciones o descendientes factibles.

4.3.4. Mutación de los descendientes

La mutación de un descendiente, en este caso, consiste en intercambiar los valores o alelos de dos genes seleccionados aleatoriamente. Esta estrategia o heurística de mutación se conoce como *Swap Mutation*.

Sin embargo, aunque es cierto que modificar o alterar ligeramente los genes que componen una solución puede llevar a una mejora en su calidad, también puede tener el efecto contrario. Por ejemplo, si se obtuviera un descendiente cuya calidad fuera máxima de acuerdo con la función objetivo definida, llevar a cabo una mutación de sus genes podría deshacer su impecabilidad, lo que al mismo tiempo implicaría perder su capacidad para resolver el problema de forma óptima. Por ello, para promover en cierto grado la variación en las soluciones pero también la conservación de la integridad de algunas, se ha establecido una condición implícita en la codificación de la heurística *Swap Mutation*: la probabilidad de que ocurra el intercambio de valores entre dos genes cualesquiera debe ser mayor a un umbral p_mut, que podría ser cualquier valor, aunque en este caso se establece en 0.5. De este modo, un descendiente tiene prácticamente las mismas probabilidades de experimentar una mutación que de no hacerlo.

Por tanto, el funcionamiento explicado de la heurística *Swap Mutation* se describe también en el pseudocódigo 9. Además, se muestra un ejemplo de la aplicación de dicha heurística en la figura 19.

heuristica $Swap\_Mutation(Descendiente)$
1 $L \leftarrow$ Número total de genes de $Descendiente$
2 Generar un número aleatorio $i \in \{1, \dots, L\}$
3 Generar un número aleatorio $j \in \{1, \dots, i-1, i+1, \dots, L\}$
4 Generar una probabilidad aleatoria p entre [0,1]
5 $p\_mut \leftarrow 0.5$
6 **si** $p > p\_mut$ **entonces**
7 $Descendiente \leftarrow$ Intercambiar los valores de los genes i y j
8 **devolver** $Descendiente$
fin $Swap\_Mutation$

Pseudocódigo 9. Mutación de un descendiente: heurística *Swap Mutation*

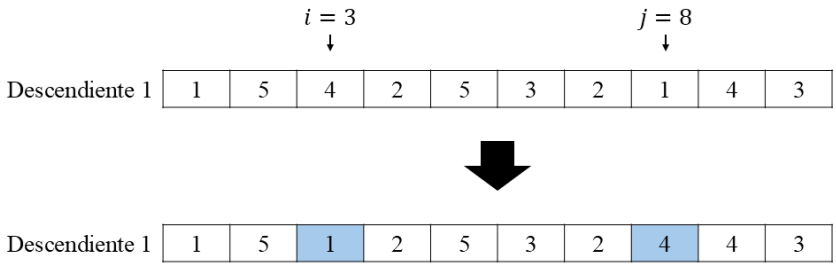

Figura 19. Ejemplo de mutación de un descendiente utilizando la heurística *Swap Mutation* (si es que $p > p\_mut$)

Por último, cabe resaltar que, al aplicar esta heurística de mutación, también se mantiene la factibilidad del descendiente, independientemente de si se produce o no el intercambio de valores entre genes. Esto significa que todos los equipos conservan su tamaño y permanecen disjuntos en cualquier caso, además de que todos los alumnos se mantienen formando parte de un equipo.

4.3.5. Reemplazo de la población

El reemplazo de la población P consiste en conservar dentro de ella la misma cantidad de soluciones que su tamaño fijado inicialmente (n), pero solamente aquellas con mejores valores de *fitness*, considerando todos los descendientes que se hubieran generado previamente y todas las soluciones que pertenecieran actualmente a dicha población. Dado que en el algoritmo genético diseñado el tamaño de la población inicial se establece en 100 soluciones, este tamaño se mantiene en cada iteración del algoritmo hasta que finaliza su ejecución.

4.3.6. Esquema de operación del algoritmo genético

En el pseudocódigo 10 se describe el esquema de operación del algoritmo genético diseñado para generar soluciones optimizadas al problema de la formación de equipos en el aula.

```
algoritmo GA(Tiempo_Limite)
1    Leer_Instancia()
2    Valor de fitness de la mejor solución: f* ← −∞
3    n ← 100
4    Inicializar una población P de n soluciones aleatorias generadas con Randomized Construction
5    mientras Tiempo_Limite no sea alcanzado hacer
6        Seleccionar 50 progenitores de P utilizando la estrategia Roulette Wheel Selection
7        Cruzar parejas de progenitores consecutivos aplicando Cut_and_Crossfill_TF
8        Mutar a los descendientes resultantes aplicando Swap_Mutation
9        Evaluar el valor de fitness de los descendientes y de las soluciones en P
10       Reemplazar la población P conservando las n soluciones con mejores valores de fitness que existen,
         considerando todos los descendientes generados y todas las soluciones actuales en P
11       S' ← Solución de P con el mayor valor de fitness
12       si fitness(S') > f* entonces
13           Mejor_Solucion ← S'
14           f* ← fitness(S')
15   devolver Mejor_Solucion
fin GA
```

Pseudocódigo 10. Esquema de operación del algoritmo genético para la formación de equipos

El algoritmo, lógicamente, primero recibe como entrada una instancia del problema para procesarla y almacenar los datos sobre el número de alumnos en el aula N, el tamaño de los equipos M, las puntuaciones Belbin de los alumnos y su género.

A continuación, se inicializa una población de partida P compuesta por $n = 100$ soluciones aleatorias generadas con la heurística *Randomized Construction*, como se indicó en la sección 4.3.1. Acto seguido, y con el objetivo de mejorar dicha población, se repiten de manera iterativa los procesos de selección de progenitores, el cruce entre los progenitores, la mutación de los descendientes resultantes del cruce y el reemplazo de la población, hasta alcanzar un tiempo límite de ejecución establecido. En este sentido, las operaciones que realiza el algoritmo coinciden con las del esquema común de un algoritmo genético, descrito en el pseudocódigo 3 de la sección 2.2.2. Sin embargo, para llevar a cabo cada una de dichas operaciones, se emplean específicamente las heurísticas definidas en las cinco secciones anteriores a esta.

Por tanto, primero se seleccionan 50 progenitores de la población utilizando la estrategia de selección proporcional al *fitness* o *Roulette Wheel Selection*. Seguidamente, se procede a realizar el cruce entre parejas de progenitores consecutivas aplicando la heurística *Cut-and-Crossfill-TF*. Esto quiere decir que, suponiendo que los progenitores están organizados dentro de una estructura de datos ordenada, se cruza el primer progenitor con el segundo, luego el segundo con el tercero, y así sucesivamente hasta cruzar el progenitor número 49 con el progenitor número 50. Siguiendo este patrón, y dado que por cada cruce se generan dos nuevas soluciones o descendientes, entonces se generarían en total 98 descendientes en cada iteración.

Luego, se lleva a cabo la mutación de los descendientes con la heurística *Swap Mutation*. De acuerdo con el funcionamiento de esta heurística, los genes de algunos de los descendientes sufrirán una modificación o no, en función de una probabilidad aleatoria.

Tras la mutación de los descendientes, se evalúa el valor de *fitness* de cada uno de estos, y también de las soluciones que pertenecen actualmente a la población P. Realizar esto resulta necesario antes de reemplazar la población. Esto se debe a que, como se indicó en la sección 4.3.5, el reemplazo de la población consiste en conservar dentro de la misma las n soluciones con mejores valores de *fitness* que existen, en este caso las 100 mejores, considerando tanto los descendientes generados como las soluciones que pertenecen actualmente a la población. De este modo, el tamaño de la población se mantiene constante en cada iteración, es decir, permanece en 100 soluciones.

Finalmente, una iteración finaliza actualizando la mejor solución encontrada hasta el momento con aquella solución que pertenece a la población recientemente actualizada y posee el mayor valor de *fitness*, siempre y cuando este valor sea mayor que el correspondiente al de la mejor solución.

Una vez alcanzado el tiempo límite de ejecución después de haber repetido periódicamente todas las operaciones que permiten la evolución de la población, el algoritmo termina devolviendo la mejor solución que todas las que ha podido generar para el problema.

5. Conjunto de datos

Para estudiar el rendimiento de las metaheurísticas diseñadas en la formación de equipos, es necesario contar con unos datos que permitan evaluar dicho rendimiento. A continuación, se describe el conjunto de datos del que se ha tenido disponibilidad para tal propósito (sección 5.1). Seguidamente, se detalla qué tipos de instancias se han generado a partir de dicho conjunto de datos (sección 5.2), que son las que en última instancia se han utilizado para evaluar el rendimiento de las metaheurísticas.

5.1. Descripción de los datos

Los datos de los que se ha tenido disponibilidad fueron extraídos del alumnado de tercer curso del Grado en Turismo de la Universidad Politécnica de Valencia durante los cursos académicos comprendidos entre 2014-2015 hasta 2018-2019. Estos datos fueron recopilados y proporcionados por los autores de [23].

Dicho esto, el conjunto de datos se compone de 383 filas y 18 columnas. La tabla 4 es una pequeña muestra del conjunto total de los datos, en la que se visualizan tanto las primeras como las últimas cinco filas.

student_id	gender	CW	CH	SH	PL	RI	ME	TW	CF	E/I	S/N	T/F	J/P	group_1	group_2	has_belbin	has_mbti
1	M	6	0	44	6	6	10	7	7	-	-	-	-	3	-	1	0
2	F	17	10	10	3	10	0	7	13	-	-	-	-	-	-	1	0
3	F	11	16	14	26	3	6	10	10	-	-	-	-	1	-	1	0
4	M	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0
5	F	7	9	10	11	9	4	10	7	-	-	-	-	17	-	1	0
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
379	F	14	13	11	8	5	8	9	22	2	-6	-14	10	4	8	1	1
380	F	8	12	10	15	13	3	10	7	6	0	-8	2	3	-	1	1
381	F	11	8	9	18	12	6	16	7	8	-2	-6	0	8	5	1	1
382	F	15	13	14	3	10	4	8	10	8	-4	-2	0	12	8	1	1
383	F	4	9	20	11	7	9	19	19	4	-6	12	2	12	9	1	1

Tabla 4. Muestra de los datos extraídos del alumnado del Grado en Turismo de la Universidad Politécnica de Valencia

Cada fila corresponde a un alumno distinto, identificado por un número entero en la columna *student_id*. Esto significa que el conjunto de datos está anonimizado.

Todos los alumnos realizaron el Inventario de Autopercepción de Belbin y/o el Indicador Tipológico de Myers-Briggs. La columna *has_belbin* indica si un alumno realizó el Inventario de Autopercepción de Belbin (1) o no (0). De manera análoga, la columna *has_mbti* indica si un alumno realizó o no el Indicador Tipológico de Myers-Briggs.

En el caso de los alumnos que realizaron el Inventario de Autopercepción de Belbin, las columnas que contienen las puntuaciones que obtuvieron en cada uno de los ocho roles están denominadas con las abreviaturas comunes que se asocian a estos roles: CW (Implementador), CH (Coordinador), SH (Impulsor), PL (Cerebro), RI (Investigador de Recursos), ME (Monitor-Evaluador), TW (Cohesionador) y CF (Finalizador).

Por otro lado, en el caso de los alumnos que realizaron el Indicador Tipológico de Myers-Briggs, las columnas con el nombre E/I, S/N, T/F y J/P indican el rasgo predominante de cada alumno en cada una de las cuatro dimensiones de personalidad. Por ejemplo, si la puntuación numérica de un alumno en la columna E/I es positiva, significa que su personalidad se inclina más hacia la extroversión. Si es negativa, entonces quiere decir que su personalidad se inclina más hacia la introversión. Si es 0, no existe un rasgo predominante, sino que su personalidad mantiene un equilibrio entre la extroversión y la introversión. La misma lógica se aplica con las otras tres columnas para identificar el rasgo predominante de un alumno en cada una de las tres dimensiones restantes, o si mantiene un equilibrio de rasgos.

La columna *gender* indica el género de cada alumno: «M» si es un hombre o «F» si es una mujer.

Las columnas *group_1* y *group_2* se pueden ignorar, ya que la información que contienen no es relevante para el proceso de formación de equipos.

Como ya se ha mencionado, y de acuerdo con la modelización realizada del problema de la formación de equipos en el aula en la sección 3, en este trabajo se tienen en cuenta los roles Belbin característicos y el género de los alumnos para su agrupación en equipos potencialmente eficaces. Por tanto, la única información del conjunto de datos que se ha utilizado para tal propósito es la correspondiente a las columnas asociadas con las puntuaciones Belbin y el género de los alumnos. Esto supuso el filtrado del conjunto de datos para conservar solamente la información de aquellos alumnos que realizaron el Inventario de Autopercepción de Belbin (*has_belbin* = 1), los cuales fueron en total 331.

5.2. Instancias generadas

A partir del conjunto de datos filtrado, se han generado instancias de diferentes tamaños de aula, las cuales se pueden entender como muestras más pequeñas que contienen información sobre un número variado de alumnos seleccionados aleatoriamente. Concretamente, se han generado 30 instancias, que se dividen en tres tipos: de 40, 50 y 60 alumnos. Así, diez de las 30 instancias contienen información sobre 40 alumnos seleccionados aleatoriamente del conjunto de datos (quienes no necesariamente son los mismos en cada instancia), otras diez contienen información sobre 50 alumnos, y las diez últimas restantes contienen información sobre 60 alumnos.

Cada una de las 30 instancias generadas consiste, específicamente, en un archivo de texto que alberga información relacionada con el número total de alumnos (según el tipo de instancia), un tamaño particular para la agrupación de los alumnos en equipos de dicho tamaño, las puntuaciones Belbin de los alumnos y, por último, el género de estos. La tabla 5 muestra el contenido (reducido) de una de las 30 instancias, concretamente, de una instancia con 40 alumnos.

tamano_aula	40								
tamano_equipos	5								
1	F	8	10	16	3	3	7	13	10
2	F	20	4	7	9	6	10	6	7
3	F	7	13	22	12	0	5	8	3
4	M	14	7	11	13	11	4	31	7
5	F	10	12	13	4	6	6	10	9
…	…	…	…	…	…	…	…	…	…
37	F	11	8	22	7	3	3	1	15
38	M	4	13	14	11	10	3	8	7
39	F	3	8	10	8	12	7	9	11
40	F	18	4	11	8	7	12	8	2

Tabla 5. Contenido (reducido) de una de las 30 instancias utilizadas para evaluar el rendimiento de las metaheurísticas

De este modo, cada instancia «ejerce» el papel de un aula real y común que puede haber en cualquier titulación universitaria o centro de educación superior, donde puede iniciarse un proceso de formación de equipos para la realización de proyectos grupales. En este sentido, las instancias generadas sirven de apoyo para poner a prueba las metaheurísticas diseñadas en la resolución de las mismas y, así, evaluar su rendimiento en la formación de equipos. Este proceso se denomina experimentar con las metaheurísticas, y puede tomar varias formas (en la sección 6.1 se describe el proceso experimental llevado a cabo con las metaheurísticas diseñadas). Para la resolución de cada una de las instancias, se podrían crear equipos de cualquier tamaño. Sin embargo, en este caso, las metaheurísticas formarán equipos de cinco alumnos por instancia, ya que se trata de un tamaño ideal para que exista una posibilidad considerable de que todos los equipos cuenten con la mayor cobertura posible de los ocho roles Belbin [37]. No obstante, al formar equipos se recomienda, generalmente, mantener un tamaño relativamente pequeño, entre tres hasta cinco o seis alumnos [38]. Tamaños de equipos superiores a este intervalo podrían causar, por ejemplo, la existencia de problemas de comunicación entre miembros de un mismo equipo.

Por último, cabe explicar que la generación de instancias de distinto tipo se debe a que el rendimiento de una metaheurística puede variar dependiendo del tamaño del problema abordado. Esto sucede porque, generalmente, las instancias de mayor tamaño en cualquier problema de optimización combinatoria suelen representar una mayor complejidad de resolución, dado que el número de combinaciones posibles aumenta y ello puede dificultar la búsqueda de una solución óptima o casi óptima en un tiempo eficiente. En el caso del problema de la formación de equipos en el aula, la agrupación de alumnos en equipos podría complicarse en aulas más grandes, es decir, encontrar una combinación de equipos eficaces en un tiempo razonable podría resultar más desafiante.

6. Experimentación y resultados

En esta sección, finalmente se analiza y compara el rendimiento de las dos metaheurísticas diseñadas. Previamente a dicho análisis (sección 6.2), se proporcionan detalles sobre el proceso experimental llevado a cabo con las metaheurísticas (sección 6.1), el cual ha permitido recopilar información relacionada con distintas medidas de su rendimiento.

6.1. Proceso experimental

Ambas metaheurísticas, GRASP y el algoritmo genético, se han ejecutado diez veces durante un minuto por cada una de las 30 instancias proporcionadas como entrada (*i.e.*, las descritas en la sección 5.2). En otras palabras, cada algoritmo ha resuelto diez veces cada una de las instancias. El objetivo de ejecutar múltiples veces una metaheurística para resolver cada instancia es recopilar suficiente información relacionada con distintos aspectos que miden su rendimiento, como la calidad de las soluciones que genera o el tiempo que tarda en converger a una solución de una determinada calidad. Estos aspectos del rendimiento de una metaheurística pueden variar no solo en función de distintos tipos de instancias, sino también en cada ejecución, debido a la naturaleza estocástica que caracteriza a estos métodos (*i.e.*, que para buscar soluciones optimizadas pueden basarse en cierto grado de aleatoriedad). Por ello, es necesario considerar todos estos factores para obtener una visión más general y confiable del verdadero rendimiento de una metaheurística.

Sin embargo, en el caso de GRASP, diferentes valores del parámetro α de la fase de construcción pueden generar distintas soluciones iniciales, lo que significa que su rendimiento también puede verse afectado por dicho parámetro. Por tanto, es conveniente determinar que configuración maximiza el rendimiento de GRASP (*i.e.*, que configuración le permite generar mejores resultados para el problema abordado). Para ello, en realidad se ha ejecutado dicho algoritmo utilizando 11 configuraciones distintas de α: $\{0.0, 0.1, 0.2, \ldots, 0.9, 1.0\}$. El número de ejecuciones realizadas con cada configuración se ha mantenido en diez para resolver cada una de las 30 instancias. En cuanto a la fase de búsqueda local, se ha utilizado la estrategia *first-improving* en lugar de *best-improving*. Esto se debe a que, como se mencionó en la sección 2.2.1, en muchas aplicaciones de GRASP se ha observado que la calidad de las soluciones que el algoritmo genera no varía al emplear una estrategia u otra, aunque el tiempo que tarda en generar estas soluciones es menor cuando se utiliza la estrategia *first-improving*. De este modo, usar la estrategia *first-improving* beneficia más al rendimiento del algoritmo.

Además, se han considerado dos valores específicos del factor λ implícito en la función objetivo definida: 0.5 y 0.8. El propósito de esto es averiguar cómo varía la eficacia de los diferentes equipos creados por las metaheurísticas al considerar que la cobertura de roles y el balance de géneros son igual de importantes para la estructura de los equipos ($\lambda = 0.5$), o que la cobertura de roles es más relevante que el balance de géneros ($\lambda = 0.8$).

Por último, es importante señalar que todo el proceso experimental, es decir, todas las ejecuciones de los algoritmos y la extracción de resultados, se ha llevado a cabo en una máquina con procesador Intel Core i7, 6 núcleos y 8 GB de memoria RAM. Esta información se proporciona porque los resultados computacionales obtenidos con cada algoritmo (los cuales se mostrarán y analizaran en la siguiente sección) pueden variar dependiendo del *hardware* utilizado, aunque se anticipa que dicha variación no sea significativa.

6.2. Análisis del rendimiento de las metaheurísticas

El estudio del rendimiento de las metaheurísticas diseñadas se divide en dos partes. En primer lugar, se analiza qué configuración del parámetro α maximiza el rendimiento de GRASP (sección 6.2.1). Una vez identificada la configuración que optimiza el rendimiento de dicho algoritmo, se comparan los resultados obtenidos a partir de esta configuración con los del algoritmo genético (sección 6.2.2).

6.2.1. Análisis del rendimiento de GRASP

En la figura 20 se presenta la distribución de la calidad de las mejores soluciones obtenidas con cada configuración de α para las instancias con 40, 50 y 60 alumnos, considerando que $\lambda = 0.5$. En otras palabras, y tomando como ejemplo la primera caja de la izquierda, esta representa la distribución de diez valores. Cada uno de estos diez valores, a su vez, corresponde a la media de las mejores soluciones obtenidas para una determinada instancia de 40 alumnos en las distintas ejecuciones del algoritmo al utilizar un valor de α y λ concreto. Lo mismo aplica para el resto de las cajas y, en general, para cualquier otro gráfico del mismo tipo que se muestre en adelante, teniendo en cuenta que la medida de rendimiento representada en el eje de ordenadas puede variar, al igual que el eje de abscisas.

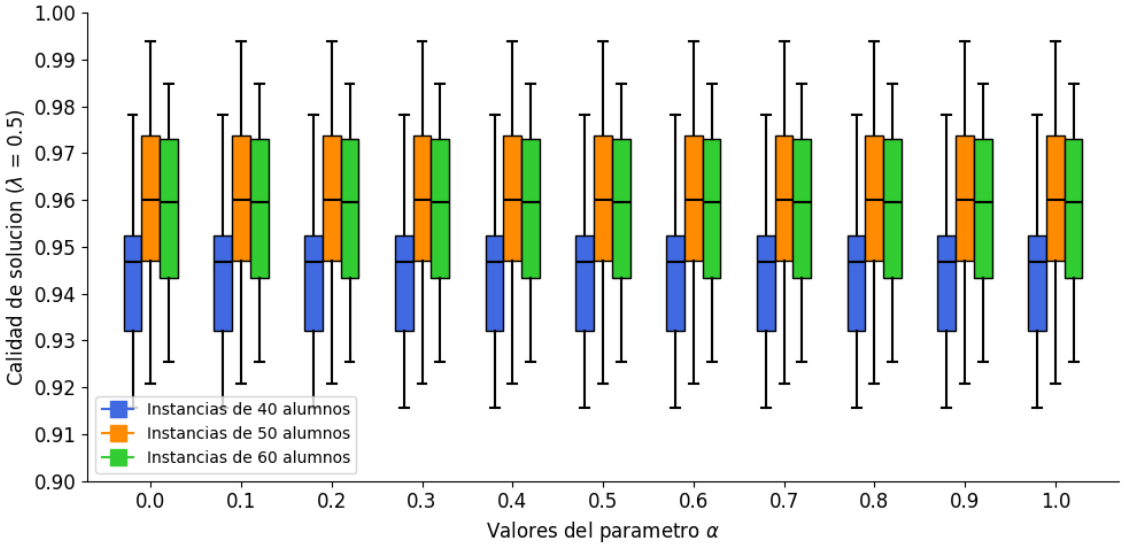

Figura 20. Distribución de la calidad de las mejores soluciones obtenidas según distintas configuraciones del parámetro α de GRASP y los tres tipos de instancias (40, 50 y 60 alumnos), considerando que $\lambda = 0.5$

Aclarado ello, lo que sorprende de la figura 20 a simple vista es que, entre distintas configuraciones, la eficacia de los equipos generados con cada una no varía en absoluto, independientemente de los tipos de instancias. Sí que es cierto que la eficacia de los equipos obtenidos con cualquier configuración varía entre los tres tipos de instancias. De hecho, la tendencia que más o menos se observa es que, cuando hay más alumnos en el aula, el algoritmo crea equipos más eficaces. Esto podría deberse a que, al haber un mayor número de alumnos, la variedad de características disponibles aumenta, lo que favorece la formación de equipos más equilibrados y, por tanto, más eficaces. No obstante, esta tendencia se rompe al pasar de 50 a 60 alumnos, ya que la eficacia de los equipos generados en estas últimas instancias es inferior a la

de los equipos creados en las de 50 alumnos, aunque la diferencia no parece significativa. De cualquier forma, en términos generales se puede decir que la eficacia de la mayoría de los equipos creados para los tres tipos de instancias oscila con mayor frecuencia entre 0.93 y 0.97.

Por otro lado, la figura 21 muestra lo mismo que la figura 20, pero en este caso considerando que $\lambda = 0.8$.

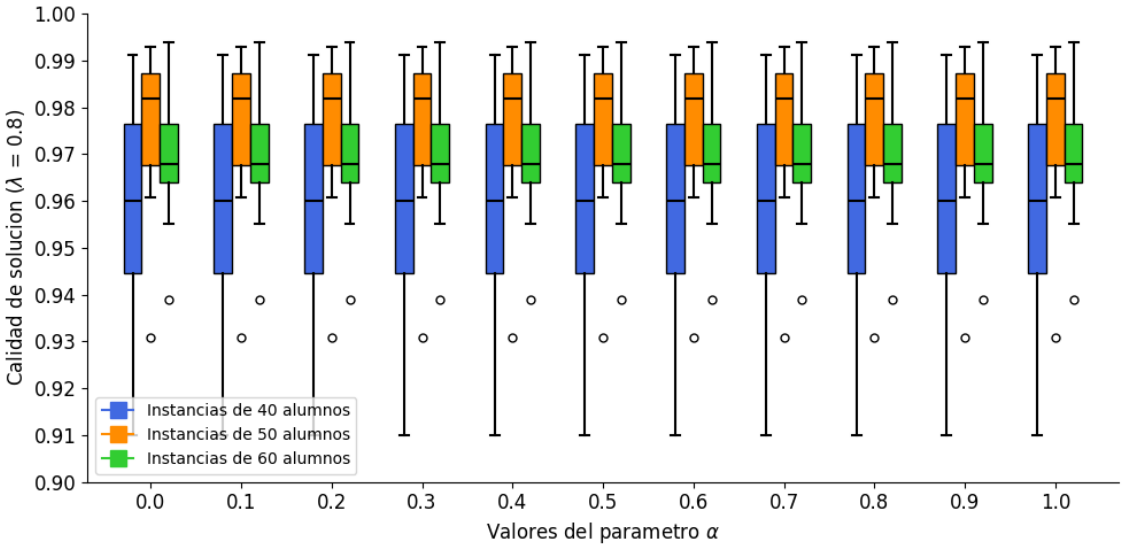

Figura 21. Distribución de la calidad de las mejores soluciones obtenidas según distintas configuraciones del parámetro α de GRASP y los tres tipos de instancias (40, 50 y 60 alumnos), considerando que $\lambda = 0.8$

Nuevamente, se observa que la eficacia de los equipos generados para los distintos tipos de instancias no varía entre las diferentes configuraciones utilizadas. Sin embargo, la tendencia del algoritmo en crear equipos más eficaces a medida que aumenta el número de alumnos en el aula ya no es tan clara, pues la diferencia en eficacia de los equipos generados para las instancias de 50 y 60 alumnos es más notable que antes. Aun así, dado que las distribuciones correspondientes a los tres tipos de instancias se mantienen idénticas al cambiar de una configuración a otra, esto podría indicar que el rendimiento del algoritmo no es necesariamente mejor o peor al resolver un tipo de instancia u otro, sino que para cualquier instancia logra obtener la combinación óptima de equipos de acuerdo con los criterios considerados (*i.e.*, la cobertura de roles y el balance de géneros). Por tanto, la diferencia en la eficacia de los equipos creados para unas instancias u otras probablemente esté más relacionada con las condiciones propias de cada una, es decir, que la distribución de roles y/o géneros sea más favorable en unas instancias que en otras para formar equipos más equilibrados y, por tanto, más eficaces. Por ejemplo, aunque no se muestra aquí, realizando un análisis exploratorio de algunas instancias (una de cada tipo) se ha observado que algunos roles solo pueden ser desempeñados por muy pocos alumnos, como el de Monitor-Evaluador. Esto dificulta que la cobertura de dicho rol se cumpla en todos los equipos formados en esas instancias, lo cual se refleja en que la eficacia de algunos de ellos no alcanza el valor máximo de 1. Igualmente, la eficacia de la mayoría de los equipos creados para los tres tipos de instancias ha aumentado con el incremento del valor de λ, ya que ahora dicha eficacia oscila con mayor frecuencia entre poco más de 0.94 y casi 0.99.

Después de realizar una investigación en profundidad para entender por qué la eficacia de los equipos varía con el valor asignado al factor λ, se han encontrado los siguientes hallazgos:

- El valor de la cobertura de roles de los equipos creados para las instancias de 40 alumnos oscila dentro del intervalo [0.9,1.0], mientras que el balance de géneros oscila alrededor de [0.9,0.95].
- El valor de la cobertura de roles de los equipos creados para las instancias de 50 alumnos oscila entre [0.93,1.0], y el balance de géneros alrededor de [0.9,1.0].
- El valor de la cobertura de roles de los equipos creados para las instancias de 60 alumnos oscila dentro del intervalo [0.94,1.0], mientras que el balance de géneros oscila alrededor de [0.9,1.0].

Los rangos de valores de ambos aspectos, los cuales influyen en la eficacia de un equipo, son los mismos que se han obtenido con cualquiera de los dos valores de λ considerados, así como con cualquier configuración. Por tanto, dado que los valores de la cobertura de roles tienden a ser mayores que los del balance de géneros (*i.e.*, es más fácil formar equipos con estructuras que favorezcan más la cobertura de roles que el balance de géneros), esto explica que la eficacia de los equipos sea mayor cuando $\lambda = 0.8$, ya que en este caso el valor de la cobertura de roles tiene más peso y contribuye más al valor final de la eficacia de los equipos. De este modo, si el valor de λ fuera 0.2, se anticipa que la eficacia de los equipos sería inferior, ya que el valor del balance de géneros tendría más peso, pero su valor probablemente se seguiría manteniendo inferior al de la cobertura de roles, debido a que se trata de un aspecto más complejo de favorecer en los equipos.

Cabe mencionar también que, durante las múltiples ejecuciones del algoritmo para resolver cada una de las 30 instancias utilizando las 11 configuraciones distintas del parámetro α, se recopiló por separado información sobre la calidad de todas y cada una de las soluciones generadas por el algoritmo antes y después de aplicar la búsqueda local. Posteriormente, los valores obtenidos se promediaron según la configuración de α y los tres tipos de instancias correspondientes. Los resultados se muestran en la figura 22, considerando que $\lambda = 0.8$.

Por un lado, se puede apreciar que, independientemente de los tres tipos de instancias, la calidad de las soluciones que el algoritmo es capaz de generar sin la aplicación de la búsqueda local se mantiene más o menos constante cuando los valores de α son 0.0, 0.1, 0.2 y 0.3. Sin embargo, una vez que α supera este último valor, la calidad de las soluciones decae con el incremento de dicho parámetro. Esta tendencia resulta comprensible dado que, a medida que el valor de α aumenta, se introduce mayor aleatoriedad en el proceso de asignación de los estudiantes a los distintos equipos, considerando cada vez menos cómo varía la eficacia de estos.

Por otro lado, cuando el algoritmo aplica la búsqueda local, naturalmente la calidad de las soluciones que genera aumenta, ya que esta es una fase fundamental que garantiza obtener, al menos, soluciones iguales o superiores a las que se pueden lograr solamente durante la fase de construcción del algoritmo. Sin embargo, resulta sorprendente que, independientemente de las diferencias en la calidad de las soluciones que se obtienen durante la fase de construcción con cada configuración de α, estas mejoran exactamente hasta el mismo punto cuando interviene la búsqueda local. Esto explica por qué los resultados vistos en la figura 21 son, en última instancia, idénticos con cualquiera de las 11 configuraciones de α, ya que no importa en sí la configuración que se utilice, pues la capacidad del algoritmo para generar soluciones de calidad radica principalmente en la búsqueda local.

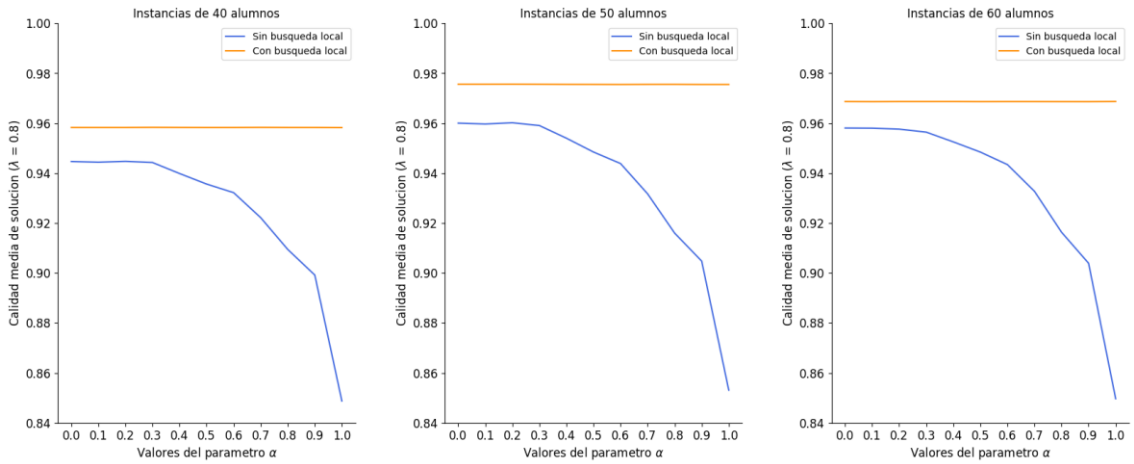

Figura 22. Calidad media de las soluciones obtenidas con GRASP sin aplicar la búsqueda local y aplicándola, considerando que $\lambda = 0.8$

Aunque no se muestra aquí, se informa que las tendencias observadas en la figura 22 son las mismas cuando $\lambda = 0.5$.

Por último y para finalizar con el análisis del rendimiento de GRASP, en la figura 23 se presenta la distribución de los tiempos de convergencia alcanzados con cada configuración de α para las instancias de 40, 50 y 60 alumnos, considerando que $\lambda = 0.8$. El tiempo de convergencia se puede entender de diferentes maneras. En este caso, dicho término hace referencia al tiempo que le ha tomado al algoritmo encontrar las mejores soluciones que ha podido, es decir, las soluciones vistas en la figura 21.

Aclarado esto, en dicha figura se pueden apreciar diferentes tendencias. En primer lugar, aparentemente el algoritmo tarda más en obtener las mejores soluciones posibles para las instancias de 60 alumnos a medida que el valor de α se incrementa. Esta tendencia también parece observarse en las instancias de 50 alumnos, aunque en este caso no es tan notable. En cambio, para las instancias de 40 alumnos, el tiempo aparenta mantenerse constante independientemente del valor de α. Por otro lado, con cualquier configuración que se utilice de dicho parámetro, el tiempo de convergencia del algoritmo tiende a incrementarse a medida que aumenta el número de alumnos en el aula.

Independientemente de estas tendencias observadas (las cuales también coinciden cuando $\lambda = 0.5$), con ninguna configuración de α el tiempo que el algoritmo ha tardado en obtener la mejor solución para cualquier instancia supera los 12 segundos, salvo en algunos casos atípicos en los que ha llegado como mucho a los 20 segundos. Estos resultados son bastante satisfactorios, ya que el algoritmo ni siquiera ha requerido el 50% del minuto de ejecución establecido para obtener dichas soluciones, lo cual destaca su gran rapidez.

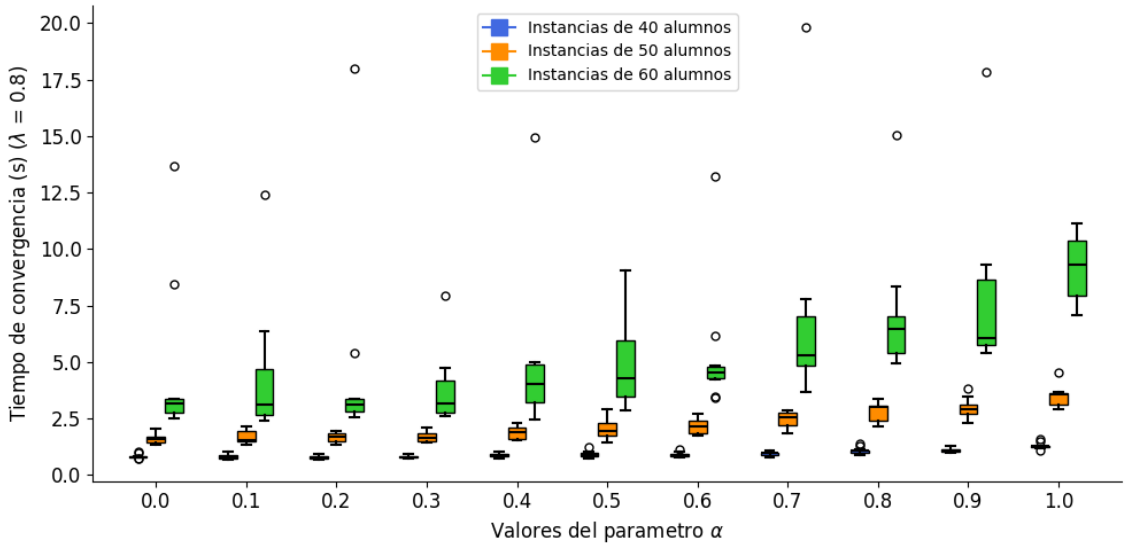

Figura 23. Distribución de los tiempos de convergencia alcanzados según distintas configuraciones del parámetro α de GRASP y los tres tipos de instancias (40, 50 y 60 alumnos), considerando que $\lambda = 0.8$

Teniendo en cuenta todos los resultados observados a lo largo del análisis, se puede decir que el rendimiento de GRASP se maximiza cuando $\alpha = 0.0$. La razón es sencilla, ya que si bien la calidad final de las soluciones que genera el algoritmo se debe a la efectividad de la búsqueda local, sí que se observan diferencias en los tiempos de convergencia alcanzados con cada configuración del parámetro. Aunque estas diferencias no parezcan ser significativas entre algunas configuraciones y, en general, el tiempo de convergencia sea bastante bajo con cualquiera de ellas, el tiempo mínimo se obtiene con el valor mencionado.

6.2.2. Comparación entre GRASP y el algoritmo genético

Para comparar el rendimiento de ambos algoritmos, se realizará un análisis análogo al de la sección anterior para identificar qué configuración del parámetro α maximiza el rendimiento de GRASP. Es decir, primero se comparará la calidad de las soluciones obtenidas con ambos algoritmos y, posteriormente, se comparará su escalabilidad temporal, es decir, los tiempos de convergencia alcanzados por ambos algoritmos para obtener dichas soluciones, en función de los tres tipos de instancias. En el caso de GRASP, los resultados que se presentarán serán los mismos observados en la sección anterior, pero solo aquellos correspondientes al valor de α que maximiza su rendimiento, es decir, 0.0. Sin embargo, en esta ocasión se mostrarán al lado de los resultados obtenidos con el algoritmo genético para analizar si existen diferencias entre ambos algoritmos.

Por tanto, observando la figura 24 se puede apreciar que, inesperadamente, los dos algoritmos han conseguido crear equipos con exactamente la misma eficacia según los tres tipos de instancias. Claro está que, cuando $\lambda = 0.8$, dicha eficacia es mayor (por la razón ya comentada). Todo ello, al mismo tiempo, se puede interpretar como que ambos algoritmos han logrado obtener la combinación óptima de equipos para cada una de las 30 instancias de prueba.

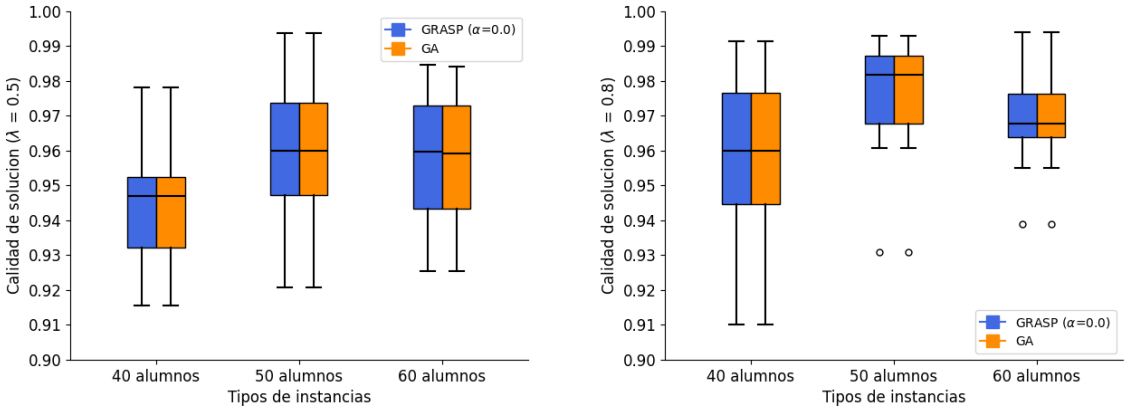

Figura 24. Distribución de la calidad de las mejores soluciones obtenidas con GRASP y el algoritmo genético para los tres tipos de instancias (40, 50 y 60 alumnos), considerando que $\lambda = 0.5$ y $\lambda = 0.8$

No obstante, en la figura 25 se puede notar claramente que, a pesar de que los dos algoritmos consiguen converger a la mejor solución posible antes de cumplirse el minuto de ejecución establecido, el algoritmo genético es el que más tarda en general. Este, inclusive, ha llegado a demorar entre 20 y 35 segundos en los peores casos, es decir, en las instancias de 60 alumnos. GRASP, por su parte, no ha superado ni los 5 segundos para converger en ningún caso, excepto en algunos casos atípicos en los que ha alcanzado 10 o 15 segundos. Además, para cada uno de los tres tipos de instancias, la variabilidad en los tiempos de convergencia de GRASP aparenta ser casi nula, lo que destaca su robustez en resolver diferentes instancias del mismo tipo prácticamente en el mismo tiempo. En cambio, el algoritmo genético muestra menos consistencia en sus tiempos de resolución. Cabe mencionar también que, aunque no se muestran aquí, resultados casi idénticos a los de dicha figura se pueden observar cuando $\lambda = 0.5$.

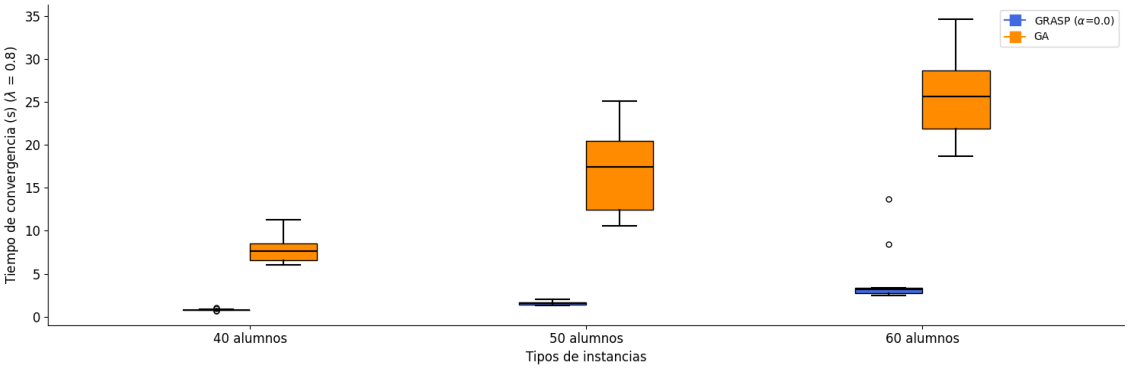

Figura 25. Distribución de los tiempos de convergencia alcanzados con GRASP y el algoritmo genético para los tres tipos de instancias (40, 50 y 60 alumnos), considerando que $\lambda = 0.8$

Aunque visualmente se han podido encontrar diferencias claras entre los tiempos de convergencia de ambos algoritmos, estos no dejan de ser bastante bajos en cualquier caso. Se puede decir incluso que el rendimiento de ambos en este aspecto es más que aceptable, dado que ambos convergen antes de finalizar el minuto de ejecución. Sin embargo, la tendencia del algoritmo genético a incrementar su tiempo de convergencia de manera mucho más notoria que GRASP conforme aumenta el número de alumnos en el aula sugiere que, con instancias de mayor tamaño que las consideradas, tardaría incluso más en resolver estas últimas. Esto representa una desventaja para el algoritmo genético, ya que en esas instancias hipotéticas podría no llegar a

converger dentro del minuto de ejecución. Por este motivo, y también porque la calidad de las soluciones que el algoritmo genético es capaz de obtener equivale a la de GRASP, este último podría considerarse en última instancia superior, ya que es más consistente, escala mucho mejor con el tamaño del aula y genera respuestas casi de forma instantánea.

Sin embargo, para constatar realmente que GRASP es más rápido y, por tanto, que sus tiempos de convergencia son significativamente menores que los del algoritmo genético, se realizó una serie de pruebas de significación estadística.

Antes de ello, se verificó si las seis distribuciones de la figura 25 seguían una distribución normal. Para ello, se utilizaron gráficos Q-Q, en los cuales si se observa que los puntos correspondientes a una distribución de datos siguen una línea recta, entonces es probable que dicha distribución sea normal. No obstante, también se aplicó la prueba de Shapiro-Wilk para comparar resultados y asegurarse de realizar conclusiones adecuadas. La figura 26 muestra dichos resultados. En primer lugar, se observa que, para las instancias de 40 alumnos, solo los tiempos de convergencia del algoritmo genético siguen una distribución normal, ya que así lo demuestran la alineación de los puntos y el p-valor de la prueba de Shapiro-Wilk (> 0.05). Sin embargo, los datos de GRASP reflejan todo lo contrario, por lo que en este caso no sería posible compararlos con los del algoritmo genético utilizando una prueba paramétrica. Lo mismo ocurre con las instancias de 60 alumnos. No obstante, en las instancias de 50 alumnos, los tiempos de convergencia de ambos algoritmos sí siguen una distribución normal (puntos alineados y p-valor menor a 0.05), lo que significa que en este caso se podría aplicar una prueba paramétrica.

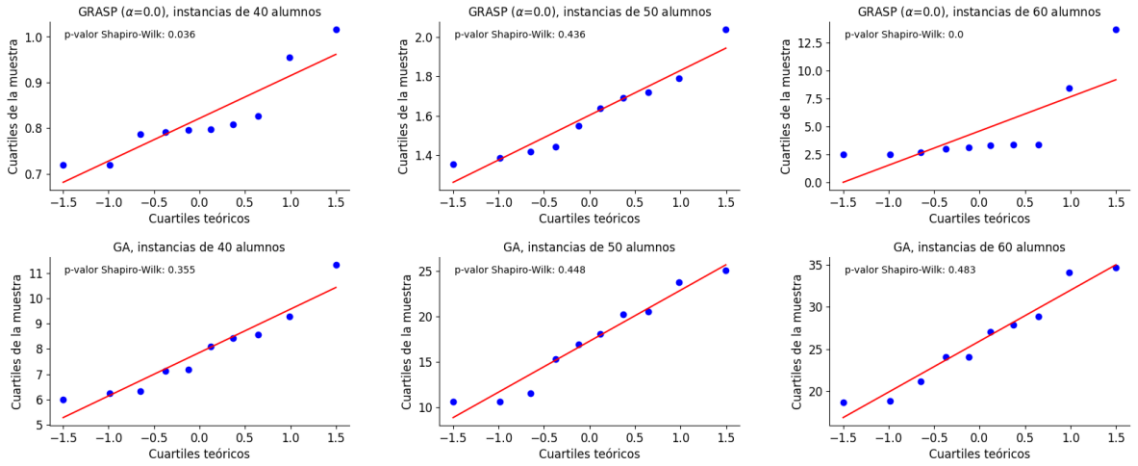

Figura 26. Gráficos Q-Q y prueba de Shapiro-Wilk para estudiar la normalidad de los tiempos de convergencia de GRASP y del algoritmo genético

En base a los resultados de la figura 26, se utilizó la prueba no paramétrica de rangos con signo de Wilcoxon [39] para comparar los tiempos de convergencia alcanzados por ambos algoritmos en las instancias de 40 y 60 alumnos, respectivamente. Por otro lado, se utilizó la prueba t de muestras relacionadas [40] para hacer lo propio con las instancias de 50 alumnos. Los resultados de estas pruebas se presentan en la tabla 6. Teniendo en cuenta que la comparación realizada (hipótesis alternativa) es que los tiempos de convergencia de GRASP son menores que los del algoritmo genético en los tres tipos de instancias, los resultados de todas las pruebas han sido positivos, ya que el p-valor es claramente inferior a 0.05 en todos los casos. Todo esto supone

evidencia suficiente para afirmar que GRASP es significativamente más veloz que el algoritmo genético para generar soluciones optimizadas.

Alg. 1	Alg. 2	Tipo de instancias	Prueba	Comparación	p-valor
GRASP ($\alpha = 0.0$)	GA	40 alumnos	Rangos con signo de Wilcoxon	Alg. 1 < Alg. 2	$9.7656e^{-04}$
GRASP ($\alpha = 0.0$)	GA	50 alumnos	t para muestras relacionadas	Alg. 1 < Alg. 2	$2.3915e^{-06}$
GRASP ($\alpha = 0.0$)	GA	60 alumnos	Rangos con signo de Wilcoxon	Alg. 1 < Alg. 2	$9.7656e^{-04}$

Tabla 6. Resultados de las pruebas de significación estadística realizadas para comprobar la existencia de diferencias significativas entre los tiempos de convergencia de GRASP y los del algoritmo genético

Por tanto, finalmente se puede determinar con total certeza que la metaheurística GRASP diseñada representa un método más adecuado que el algoritmo genético diseñado para la formación de equipos en el aula, tal y como este problema se ha modelizado. Esta conclusión se basa en que, si bien ambos algoritmos son igualmente efectivos porque la calidad de las soluciones que generan es la misma, GRASP es más rápido en producir dichas soluciones, lo que lo convierte en un algoritmo más eficiente.

7. Conclusiones

Todo lo desarrollado en este trabajo, hasta obtener los resultados finales presentados en la sección anterior, responde satisfactoriamente a los objetivos fijados en un principio.

En primer lugar, para guiar el proceso de formación de equipos potencialmente eficaces, se lograron formular correctamente unos criterios sólidos que establecen lo que es crucial para su buen funcionamiento, en este caso, la presencia equilibrada de miembros de diversos géneros y también con diferentes tendencias de comportamiento (roles Belbin). Esto constituyó la base para poder abordar el problema desde un enfoque algorítmico, debido a la inviabilidad e ineficiencia que resultaría evaluar manualmente una enorme cantidad de equipos en busca de los más convenientes.

Sin embargo, promovido también por las complicaciones que los algoritmos exactos de optimización suelen encontrar para dar respuesta al problema en tiempos razonables, se estudiaron como alternativas dos metaheurísticas de optimización distintas: un algoritmo genético y GRASP, siendo esta última un método poco utilizado hasta ahora para la formación de equipos, además de que hasta la fecha no parecen existir públicamente estudios que comparen el rendimiento de ambas metaheurísticas en la resolución del problema. Estas metaheurísticas fueron diseñadas de acuerdo con el esquema común que las caracteriza y la modelización realizada del problema. Llevar a cabo correctamente el estudio de su rendimiento en la formación de equipos requirió de la experimentación con ambas, considerando diferentes tipos de instancias del problema y múltiples ejecuciones para obtener resultados confiables que permitieran extraer conclusiones adecuadas sobre su rendimiento.

Los resultados finales del estudio parecen indicar que ambos algoritmos logran obtener la combinación óptima de equipos para cada una de las instancias de prueba, dado que la calidad de las soluciones generadas por ambos es exactamente la misma. No obstante, aunque los dos algoritmos producen soluciones igualmente satisfactorias en menos de un minuto, GRASP lo hace en un tiempo significativamente menor y prácticamente de forma instantánea, requiriendo únicamente cinco segundos o alrededor de tal cantidad con mayor probabilidad. Esto lo convierte en un algoritmo más eficiente y, por tanto, más adecuado para la formación de equipos en el aula, según la modelización realizada de este problema. Se anticipa que esta tendencia de resolución óptima en pocos segundos se mantenga incluso en aulas con un mayor o menor número de alumnos, e incluso considerando otros tamaños de equipos distintos al analizado en este estudio (*i.e.*, cinco miembros). Sin embargo, considerar tamaños menores podría reducir la eficacia media de los equipos, ya que la escasez de alumnos por equipo dificultaría satisfacer en mayor medida los criterios definidos para su eficacia, en especial la cobertura de roles.

Asimismo, a la luz de los resultados finales obtenidos, el estudio realizado puede contribuir al conocimiento sobre el buen rendimiento de GRASP en la formación de equipos y fomentar una consideración más frecuente de esta metaheurística para su aplicación en dicho contexto, especialmente porque la mayoría de las aproximaciones existentes hasta ahora para abordar el problema se basan en la aplicación de algoritmos genéticos. Incluso, podría promover un uso más amplio de los algoritmos basados en la búsqueda local, dada la potencia demostrada de esta técnica, que constituye la principal razón por la cual la metaheurística GRASP diseñada en este trabajo logra obtener soluciones de calidad satisfactoria, independientemente de la configuración utilizada del parámetro α que la caracteriza.

Por último, cabe mencionar que durante el desarrollo de este trabajo no se encontraron mayores dificultades para completarlo. No obstante, su realización ha significado la adquisición de nuevos conocimientos, sin los cuales no hubiera sido posible llevarlo a cabo.

En primer lugar, la creación de equipos eficaces requiere de la definición de ciertos criterios que establezcan unas bases o requisitos a cumplir por cualquier equipo para que su potencial rendimiento sea, de manera aproximada, eficaz. A lo largo de la sección 2.3, se realizó una revisión de distintos criterios comúnmente utilizados para este fin. Sin embargo, estos criterios no se conocían de antemano, debido a la inexperiencia en el contexto de la formación de equipos. Por ende, fue necesario realizar las investigaciones oportunas para obtener conocimientos sobre diferentes criterios de formación de equipos eficaces.

En segundo lugar, aunque se tenía conocimiento sobre la optimización y las metaheurísticas, además de cierta experiencia en su aplicación, no se conocía de manera específica el funcionamiento de GRASP, por lo que fue necesario investigar este método en particular.

Por otro lado, la realización de este trabajo también ha permitido adquirir nuevos hábitos beneficiosos al trabajar con metaheurísticas de optimización y, en general, al trabajar con código. Debido a que cualquier trabajo que implique la experimentación con metaheurísticas suele tomar una cantidad considerable de tiempo por las múltiples ejecuciones (en este caso, varios días), se ha aprendido a organizar mejor dicho proceso, de modo que los resultados se almacenen en archivos. De hecho, se creó un *script* específicamente para llevar a cabo la experimentación y la extracción de los resultados computacionales vistos, además de para almacenar dichos resultados en archivos. Así, para cualquier análisis que se realizara en diferentes momentos, los resultados siempre estarían disponibles.

Además, se ha aprendido a distribuir de manera más eficiente todo el código de trabajo en diferentes módulos, ya que el desarrollo de cada una de las metaheurísticas se llevó a cabo en *scripts* separados. Al mismo tiempo, se ha reforzado el conocimiento sobre cómo establecer dependencias comunes entre los diferentes *scripts*, evitando así la repetición innecesaria de código. Por ejemplo, el *script* correspondiente a la experimentación hereda todo el código dedicado a las dos metaheurísticas, y solo fue necesario modificar ligeramente el esquema de operación de estas para poder extraer diferentes medidas de su rendimiento.

7.1. Legado

Todo el código de trabajo desarrollado se pone a disposición en el enlace https://github.com/dlaaann/Formacion-De-Equipos para su libre uso. En él se incluyen los *scripts* correspondientes a las dos metaheurísticas diseñadas, junto con instrucciones sobre el uso de ambos *scripts*. También se dejan disponibles las 30 instancias utilizadas para evaluar el rendimiento de las metaheurísticas. Finalmente, se proporciona el archivo en el que se ha realizado el análisis de los resultados, que es un *notebook* de Jupyter (software web que integra el lenguaje de programación Python). En este archivo es donde se muestran los resultados obtenidos cuando $\lambda = 0.5$ pero que no se mostraron en la sección 6.2 correspondiente al análisis del rendimiento de las metaheurísticas. Sin embargo, para poder reproducir correctamente el análisis, también se incluyen los archivos asociados a los resultados computacionales obtenidos a partir de la experimentación.

7.2. Relación del trabajo con los estudios cursados

Diversos conocimientos específicos adquiridos durante los cuatro cursos del Grado en Ciencia de Datos de la Universidad Politécnica de Valencia han sido de utilidad y se han aplicado para el desarrollo de este trabajo. Por tanto, es posible establecer relaciones entre lo realizado y el contenido de algunas asignaturas.

Empezando por la asignatura «Optimización», puede decirse que la mayor parte del trabajo está relacionada con los contenidos de esta materia. La asignatura, como su nombre indica, estaba dedicada a introducir lo que era la optimización de problemas y a explicar los diferentes métodos que pueden utilizarse para llevarla a cabo, tanto exactos como aproximados. Entre los métodos aproximados, solo se hacía hincapié en el funcionamiento de los algoritmos genéticos, debido a la limitación de horas asignadas para el aprendizaje de la asignatura. Por esta misma razón es que fue necesario investigar sobre el funcionamiento específico de GRASP. No obstante, gracias a la introducción que la asignatura ofrecía de las metaheurísticas y tomando como ejemplo los algoritmos genéticos, comprender el funcionamiento de GRASP no resultó especialmente complicado. Además, cabe destacar que, para poner en práctica el desarrollo de los diferentes métodos estudiados en la asignatura para la resolución de diversos problemas de ejemplo (p.ej., problema de la mochila, del viajante de comercio o problemas más complejos con aplicaciones más realistas), el lenguaje de programación que se utilizaba era Python, el cual es el que finalmente se ha utilizado para desarrollar las metaheurísticas empleadas en este trabajo.

El aprendizaje en el manejo de Python puede atribuirse a las enseñanzas que se impartían en asignaturas como «Fundamentos de programación», «Programación», «Estructuras de datos» y «Algorítmica», destacando la primera de ellas como el punto de partida en el aprendizaje más básico de dicho lenguaje. En cuanto a las otras tres asignaturas mencionadas, sus contenidos estaban asociados a la adquisición de un dominio más avanzado del lenguaje, incluyendo una mayor variedad de funciones y estructuras de datos comúnmente utilizadas por los profesionales de la Ciencia de Datos al emplear Python para su manejo. Además, en las dos últimas asignaturas, se introdujo al análisis de la complejidad temporal de los algoritmos, es decir, la notación O grande. En este sentido, dichas materias destacaban la importancia de crear algoritmos lo más eficientes posible para la resolución de problemas, lo cual se pudo comprender mejor en la práctica a través de la asignatura «Optimización».

Una práctica fundamental para el estudio de diferencias significativas en el rendimiento de distintos algoritmos de optimización, como las metaheurísticas, es la aplicación de pruebas de significación estadística, tal y como se ha realizado en este trabajo. Los conocimientos sobre estas pruebas (tanto paramétricas como no paramétricas) para la comparación entre muestras se adquirieron en la asignatura «Modelos Estadísticos para la Toma de Decisiones II».

La elección de los gráficos presentados para transmitir los hallazgos encontrados en los resultados de las metaheurísticas se fundamenta en los contenidos vistos en las asignaturas «Análisis exploratorio de datos» y «Visualización». En estas asignaturas se enseñaban diversos tipos de visualizaciones, desde las más sencillas y comunes hasta otras más complejas, que pueden ser necesarias si lo que se pretende es transmitir una gran cantidad de información en un solo gráfico.

7.3. Trabajos futuros

A continuación, se presentan algunas propuestas a partir de las cuales el trabajo desarrollado podría ampliarse.

En primer lugar, dado que el conjunto de datos disponible también contiene información sobre las puntuaciones obtenidas por los alumnos al completar el Inventario Tipológico de Myers-Briggs, podría plantearse la formulación de otro enfoque alternativo en el cual, para la formación de equipos eficaces, se considere la presencia equilibrada tanto de géneros diversos como de personalidades diversas. Posteriormente, se podría estudiar cómo varía el rendimiento de las metaheurísticas diseñadas al considerar este nuevo enfoque, y compararlo con el rendimiento obtenido a partir del enfoque original empleado en este trabajo (*i.e.*, la presencia equilibrada de géneros diversos y de diferentes tendencias de comportamiento en los equipos).

Además, como se ha hecho en algunos estudios para comparar en la práctica enfoques distintos de formación de equipos (véase [24]), podría diseñarse un experimento con estudiantes de un aula existente en el cual, para la ejecución de ciertos proyectos, se agrupase a una mitad de los alumnos utilizando un enfoque y a la otra mitad empleando el otro. Al finalizar el periodo de realización de los proyectos, se encuestaría a los estudiantes sobre su grado de satisfacción con la dinámica y el funcionamiento de los equipos que les hubieran tocado, así como sobre otros aspectos que pudieran proporcionar información sobre su experiencia de colaboración en grupo. Las respuestas podrían recopilarse mediante una escala de Likert, lo cual facilitaría el análisis de los resultados de las encuestas y la aplicación de pruebas estadísticas para identificar diferencias significativas entre ambos enfoques. Tras ello, el enfoque que reflejara una cantidad significativamente mayor de respuestas positivas por parte de los alumnos podría considerase más efectivo.

Por último, y derivado de la propuesta anterior, el enfoque con resultados más positivos podría considerarse para su integración en una herramienta informática, dado que se habría validado su utilidad real para formar equipos de manera satisfactoria. Sin embargo, antes de proceder con ello, sería necesario aplicar algunos ajustes en los *scripts* correspondientes a las metaheurísticas diseñadas para controlar cualquier tipo de error (p.ej., introducción de valores incorrectos, instancias con una estructura distinta a la definida, etc.).

8. Análisis del marco legal y ético

Dado que para el desarrollo de este trabajo se tuvo acceso a unos datos personales, es importante remarcar algunos aspectos asociados con la legalidad y la ética en su adquisición y uso.

En primer lugar, como se mostró en la sección 5.1 (tabla 4), la mayor parte del conjunto de datos proporcionado para la realización de este trabajo contiene información relacionada con los resultados de las pruebas de Belbin y Myers-Briggs realizadas por alumnos del Grado en Turismo de la UPV. Previamente a la recopilación de estos resultados, es decir, las respectivas puntuaciones de los alumnos para identificar sus perfiles de acuerdo con la teoría de Belbin y el MBTI, se les informó de que participarían en un experimento relacionado con el estudio de distintos métodos para la formación de equipos. Por tanto, todos los datos utilizados pertenecen solamente a alumnos de los cuales se obtuvo un consentimiento previo para la recopilación de su información. Esta declaración puede verificarse en [23], cuyos autores fueron los responsables de recopilar dichos datos y también de proporcionarlos para hacer posible el desarrollo de este trabajo.

Además, debido a que los datos se encuentran anonimizados y no contienen información sensible de ningún tipo, su acceso imposibilita la realización de prácticas indebidas que pudieran poner en riesgo la identificación de los alumnos. De hecho, todos los datos extraídos se asocian exclusivamente con información necesaria para llevar a cabo la finalidad por la que fueron recopilados, es decir, el estudio de diferentes métodos para la formación de equipos. El propósito de este trabajo y todo lo expuesto sobre el mismo en el presente documento también ha coincidido con dicha finalidad.

De este modo, el proceso descrito de recolección y tratamiento de los datos ha cumplido con diferentes requisitos que, en muchos casos, son de debido cumplimiento y se recogen en la Ley Orgánica de Protección de Datos Personales y garantía de los derechos digitales [41], tales como la obtención de consentimiento para la recopilación de los datos, la transparencia en la finalidad por la que se recogen, la limitación a esa finalidad y, por último, la seguridad de los datos, ya que aunque están anonimizados y no contienen información sensible, no se han hecho públicos.

En segundo lugar, se cree firmemente que la aplicación de los métodos diseñados para la formación de equipos no introduzca sesgos discriminatorios. La razón de esto es porque dichos algoritmos se basan en criterios como la teoría de roles de Belbin, que no se fundamenta en factores como por ejemplo la raza para determinar cómo deben agruparse unos individuos. En este sentido, los algoritmos no discriminan por raza y, en teoría, la creación de equipos más o menos eficaces no estaría ligada a la segregación racial, evitando así comprometer el desarrollo y el aprendizaje de los estudiantes por su origen étnico. Por otro lado, aunque los algoritmos también tienen en cuenta el factor sexo para agrupar a los alumnos, este criterio se utiliza con el propósito de fomentar el balance de hombres y mujeres en los equipos. Nuevamente, la creación de mejores o peores equipos no estaría vinculada a la separación de hombres y mujeres para que estos no puedan ser incluidos en un mismo grupo, evitando así comprometer el desarrollo y el aprendizaje de los estudiantes de un género o del otro.

Por último, también es fundamental remarcar otros aspectos legales asociados, en este caso, con la propiedad intelectual. Como ya se ha mencionado, el lenguaje de programación utilizado

para desarrollar los métodos estudiados para la formación de equipos es Python. Este es un lenguaje distribuido bajo la licencia de software libre PSFL (Python Software Foundation License) [42], la cual es una licencia permisiva que no impone mayores restricciones para compartir o distribuir de forma abierta con otros usuarios proyectos desarrollados a partir de dicho lenguaje. Por tanto, utilizar este lenguaje supone una ventaja en el sentido de que los *scripts* correspondientes a los algoritmos desarrollados para la formación de equipos pueden ser distribuidos para su libre uso, tal y como se ha hecho.

En cuanto a las figuras o cualquier tipo de visualizaciones presentadas a lo largo de este documento, aquellas extraídas de fuentes externas han sido debidamente referenciadas. En el caso de las de elaboración propia, cuando pudiera haber surgido alguna duda sobre su autoría, se ha indicado explícitamente que lo son.

9. Referencias

[1] «Actualización de las competencias transversales de la UPV,» [En línea]. Disponible: https://www.upv.es/entidades/vecal/proyecto-de-actualizacion-de-competencias-transversales/. [Último acceso: 25 de junio de 2024].

[2] N. Ugarte, A. Aranzabal, A. Arruarte y M. Larrañaga, «Using the Behavioural Tendency of Students in a Team Environment for Team Formation,» de *2022 IEEE Frontiers in Education Conference (FIE)*, Uppsala, Suecia, 2022, pp. 1-9, doi: 10.1109/FIE56618.2022.9962648.

[3] R. M. Belbin, Team Roles at Work, Routledge, 2012.

[4] S. Takeda y F. Homberg, «The effects of gender on group work process and achievement: an analysis through self- and peer-assessment,» *British Educational Research Journal*, vol. 40, nº 2, p. 373–396, 2014, doi: 10.1002/berj.3088.

[5] V. Sánchez Anguix, *Contenido de la asignatura "Optimización" del Grado en Ciencia de Datos de la Universidad Politécnica de Valencia*.

[6] M. Assi y R. A. Haraty, «A Survey of the Knapsack Problem,» de *2018 International Arab Conference on Information Technology (ACIT)*, Werdanye, Líbano, 2018, pp. 1-6, doi: 10.1109/ACIT.2018.8672677.

[7] «Problema de la mochila,» [En línea]. Disponible: https://es.wikipedia.org/wiki/Problema_de_la_mochila. [Último acceso: 11 de agosto de 2024].

[8] «OR-Tools - Google Optimization Tools,» [En línea]. Disponible: https://github.com/google/or-tools. [Último acceso: 11 de agosto de 2024].

[9] P. Game, Dr. Vaze y M. Emmanuel, «Bio-inspired Optimization: metaheuristic algorithms for optimization,» 2020, doi: 10.48550/arXiv.2003.11637.

[10] S. Katiyar, I. Nasiruddin y A. Q. Ansari, «Ant Colony Optimization: A Tutorial Review,» de *National Conference on Advances in Power and Control*, 2015, pp. 99-110.

[11] A. E. Ezugwu, A. K. Shukla, R. Nath et al., «Metaheuristics: a comprehensive overview and classification along with bibliometric analysis,» *Artificial Intelligence Review*, vol. 54, p. 4237–4316, 2021, doi: 10.1007/s10462-020-09952-0.

[12] H. H. Hoos y T. Stützle, «2 - SLS METHODS,» de *Stochastic Local Search*, Eds: H. H. Hoos y T. Stützle, en *The Morgan Kaufmann Series in Artificial Intelligence*, Morgan Kaufmann, 2005, pp. 61-112, ISBN 9781558608726, doi: 10.1016/B978-155860872-6/50019-6.

[13] M. G. C. Resende y C. C. Ribeiro, «Greedy Randomized Adaptive Search Procedures: Advances and Extensions,» de *Handbook of Metaheuristics*, Eds: M. Gendreau y J. Y. Potvin, en *International Series in Operations Research & Management Science*, vol. 272, Springer, Cham, 2019, pp. 169-220, doi: 10.1007/978-3-319-91086-4_6.

[14] S. Goyal, «A Survey on Travelling Salesman Problem,» de *Midwest Instruction and Computing Symposium*, 2010, pp. 1-9.

[15] A. Lambora, K. Gupta y K. Chopra, «Genetic Algorithm- A Literature Review,» de *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, Faridabad, India, 2019, pp. 380-384, doi: 10.1109/COMITCon.2019.8862255.

[16] «Los tres puntos clave para entender la Teoría de la Evolución de Darwin,» [En línea]. Disponible: https://www.nationalgeographic.com.es/ciencia/tres-puntos-clave-para-entender-teoria-evolucion-darwin_21543. [Último acceso: 13 de agosto de 2024].

[17] [En línea]. Disponible: https://www.istockphoto.com/es/ilustraciones/mochila-abierta. [Último acceso: 14 de septiembre de 2024].

[18] «Dr. Meredith Belbin,» [En línea]. Disponible: https://www.belbin.es/acerca-de-belbin/dr-meredith-belbin. [Último acceso: 25 de agosto de 2024].

[19] «Meredith Belbin,» [En línea]. Disponible: https://en.wikipedia.org/wiki/Meredith_Belbin. [Último acceso: 25 de agosto de 2024].

[20] «Belbin Team Roles,» [En línea]. Disponible: https://wiki.uia.no/images/success/b/bc/Belbin-team-roles-handout.pdf. [Último acceso: 25 de agosto de 2024].

[21] «Los nueve Roles de Equipo Belbin,» [En línea]. Disponible: https://www.belbin.es/acerca-de-belbin/roles-de-equipo-belbin. [Último acceso: 25 de agosto de 2024].

[22] «Self-Perception Inventory,» [En línea]. Disponible: https://www.belbin.ie/wp-content/uploads/2014/08/BELBINUK-Self-PerceptionInventory+CompletionGrid.pdf. [Último acceso: 25 de agosto de 2024].

[23] V. Sánchez Anguix, J. M. Alberola, E. del Val et al., «Comparing computational algorithms for team formation in the classroom: a classroom experience,» *Applied Intelligence*, vol. 53, pp. 23883-23904, 2023, doi: 10.1007/s10489-023-04748-5.

[24] J. M. Alberola, E. del Val, V. Sánchez Anguix, A. Palomares y M. D. Teruel, «An artificial intelligence tool for heterogeneous team formation in the classroom,» *Knowledge-Based Systems*, vol. 101, pp. 1-14, 2016, doi: 10.1016/j.knosys.2016.02.010.

[25] A. Aranzabal, E. Epelde y M. Artetxe, «Team formation on the basis of Belbin's roles to enhance students' performance in project-based learning,» *Education for Chemical Engineers*, vol. 38, pp. 22-37, 2022, doi: 10.1016/j.ece.2021.09.001.

[26] I. Lykourentzou, A. Antoniou, Y. Naudet y S. P. Dow, «Personality matters: Balancing for personality types leads to better outcomes for crowd teams,» de *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*, 2016, pp. 260-273.

[27] I. B. Myers, The Myers-Briggs Type Indicator: Manual, 1962.

[28] J. Juárez, C. Santos y C. A. Brizuela, «A comprehensive review and a taxonomy proposal of team formation problems,» *ACM Computing Surveys (CSUR)*, vol. 54, nº 7, pp. 1-33, 2021.

[29] S. España Boquera, *Contenido de la asignatura "Estructura de datos" del Grado en Ciencia de Datos de la Universidad Politécnica de Valencia.*

[30] «Complejidad temporal,» [En línea]. Disponible: https://es.wikipedia.org/wiki/Complejidad_temporal. [Último acceso: 21 de septiembre de 2024].

[31] «Big-O Notation Explained,» [En línea]. Disponible: https://danielmiessler.com/p/big-o-notation/. [Último acceso: 21 de septiembre de 2024].

[32] E. Andrejczuk, F. Bistaffa, C. Blum, J. A. Rodríguez Aguilar y C. Sierra, «Synergistic team composition: A computational approach to foster diversity in teams,» *Knowledge-Based Systems*, vol. 182, nº 104799, 2019, doi: 10.1016/j.knosys.2019.06.007.

[33] J. Moreno, D. A. Ovalle y R. M. Vicari, «A genetic algorithm approach for group formation in collaborative learning considering multiple student characteristics,» *Computers & Education*, vol. 58, nº 1, pp. 560-569, 2012, doi: 10.1016/j.compedu.2011.09.011.

[34] J. H. Gutiérrez, C. A. Astudillo, P. Ballesteros Pérez, D. Mora Melià y A. Candia Véjar, «The multiple team formation problem using sociometry,» *Computers & Operations Research*, vol. 75, pp. 150-162, 2016, doi: 10.1016/j.cor.2016.05.012.

[35] A. Shukla, H. M. Pandey y D. Mehrotra, «Comparative review of selection techniques in genetic algorithm,» de *2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE)*, Greater Noida, India, 2015, pp. 515-519, doi: 10.1109/ABLAZE.2015.7154916.

[36] «Roulette wheel selection,» [En línea]. Disponible: http://www.edc.ncl.ac.uk/highlight/rhjanuary2007g02.php. [Último acceso: 16 de octubre de 2024].

[37] F. Bullen y D. Wood, «The construction of undergraduate student engineering design teams using the MBTI and Belbin Test,» de *Proc. 17th Annual Conference of the Australasian Association for Engineering Education*, 2006, pp. 111-120.

[38] R. J. Breitenecker, «A methodological contribution to team-role-balance measurement based on Belbin's self-perception inventory,» de *12th European*, 2014 doi: 10.13140/2.1.5034.1446.

[39] J. Derrac, S. García, D. Molina y F. Herrera, «A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithm,» *Swarm and Evolutionary Computation*, vol. 1, pp. 3-18, 2011, doi: 10.1016/j.swevo.2011.02.002.

[40] «Prueba t para muestras relacionadas,» [En línea]. Disponible: https://datatab.es/tutorial/paired-t-test. [Último acceso: 28 de octubre de 2024].

[41] «Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales,» [En línea]. Disponible: https://www.boe.es/buscar/act.php?id=BOE-A-2018-16673. [Último acceso: 31 de octubre de 2024].

[42] «Python Software Foundation License,» [En línea]. Disponible: https://es-academic.com/dic.nsf/eswiki/1351984. [Último acceso: 31 de octubre de 2024].

ANEXO

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	**Alto**	**Medio**	**Bajo**	**No Procede**
ODS 1. **Fin de la pobreza.**				**X**
ODS 2. **Hambre cero.**				**X**
ODS 3. **Salud y bienestar.**				**X**
ODS 4. **Educación de calidad.**	**X**			
ODS 5. **Igualdad de género.**	**X**			
ODS 6. **Agua limpia y saneamiento.**				**X**
ODS 7. **Energía asequible y no contaminante.**			**X**	
ODS 8. **Trabajo decente y crecimiento económico.**			**X**	
ODS 9. **Industria, innovación e infraestructuras.**				**X**
ODS 10. **Reducción de las desigualdades**.				**X**
ODS 11. **Ciudades y comunidades sostenibles**.				**X**
ODS 12. **Producción y consumo responsables.**				**X**
ODS 13. **Acción por el clima.**				**X**
ODS 14. **Vida submarina.**				**X**
ODS 15. **Vida de ecosistemas terrestres.**				**X**
ODS 16. **Paz, justicia e instituciones sólidas.**				**X**
ODS 17. **Alianzas para lograr objetivos.**				**X**

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

De los anteriores objetivos de desarrollo sostenibles mencionados, este trabajo puede relacionarse con:

- **Educación de calidad**, ya que mediante la formación de equipos eficaces se busca, en primer lugar, que los estudiantes desarrollen adecuadamente competencias colaborativas, y en segundo, que refuercen y adquieran conocimientos, lo que se conoce como metodología de aprendizaje colaborativo. Todo ello para que, en sus futuros empleos, puedan desenvolverse correctamente y sean capaces de colaborar con soltura con otras personas. Por todo ello, se puede considerar que este trabajo está altamente relacionado con el objetivo de elevar la calidad de la educación de los estudiantes.
- **Igualdad de género**, porque se promueve la creación de equipos diversos en géneros. La inclusión de tanto hombres como mujeres en un equipo, además de que puede contribuir significativamente a su buen funcionamiento, también puede ayudar a fomentar el respeto, la igualdad y a romper estereotipos. En este sentido, los estudiantes pueden observar que las habilidades de una persona no dependen del género, y que cualquiera puede destacar en cualquier área de estudio (p.ej., mujeres en ingenierías) con esfuerzo y dedicación.
- **Energía asequible y no contaminante**, dado que al presentar métodos eficientes para la formación de equipos, en este caso algoritmos que generan respuestas de calidad en tiempos muy reducidos, se promueve la reducción en el gasto de energía por cómputo prolongado. En otras palabras, el uso de los métodos estudiados en este trabajo implica un consumo más eficiente y responsable de electricidad.
- **Trabajo decente y crecimiento económico**, ya que al formar equipos eficaces y por tanto favorecer el aprendizaje y desarrollo de los alumnos, se espera que estos sean más competentes en el mundo laboral y contribuyan con sus conocimientos al éxito de las empresas en las que trabajen. Esto podría derivar en el crecimiento económico de dichas empresas y, por extensión, en un crecimiento económico potencialmente global.