UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Estudio de modelos de optimización exactos y metaheurísticos para la formación de equipos en el aula

Trabajo Fin de Grado

Grado en Ciencia de Datos

AUTOR/A: Candel Peiró, Gonzalo

Tutor/a: Sánchez Anguix, Víctor

Cotutor/a: Alberola Oltra, Juan Miguel

CURSO ACADÉMICO: 2021/2022

Resumen

A día de hoy, el trabajo en equipo se ha vuelto una parte central de muchas titulaciones de educación superior. Esto responde a una tendencia laboral, en la que una gran mayoría de proyectos complejos son resueltos en equipo. Para desarrollar correctamente las competencias relacionadas con el trabajo en equipo, es deseable que los equipos formados en el aula permitan el buen desarrollo de estas competencias. Aunque existen diferentes guías basadas en la psicología organizacional para la formación de estos equipos (e.g., Belbin, Myer-Briggs, Big Five, etc.), la formación óptima de equipos en el aula es una tarea que puede ser cognitivamente muy costosa para el docente, pues los tamaños de aula son grandes y la combinatoria explota rápidamente. De hecho, el problema es una instancia especial del conocido como *Coalition Structure Generation Problem*, un problema altamente combinatorio que consiste en dividir un conjunto de agentes en una partición completa y disjunta. En este trabajo final de grado (TFG) proponemos un modelo entero para la formación de equipos en el aula, y también exploramos el comportamiento de varios *solvers* de código libre a la hora de resolver dicho modelo. Debido a la pobre escalabilidad de los *solvers* exactos con tamaños de aula grande, también diseñamos algoritmos metaheurísticos, más asequibles para la resolución de instancias grandes del problema y estudiamos su rendimiento comparado con los métodos exactos.

Palabras clave: optimización; analítica prescriptiva; algoritmos genéticos; programación matemática.

Abstract

Nowadays teamwork has become an essential part of higher education degrees. This fact comes as a response of a trend in the labor market, where most of the complex projects are solved by teams. To correctly develop the skills related to teamwork, it is desirable for teams formed in the classroom to allow a great development of those skills. Although there exist different guidelines based on organizational psychology (e.g, Belbin, Myers-Briggs, Big Five, etc.), optimal team formation in the aula can be, cognitively, a very difficult task for the teacher, as class sizes are large, and the combinatorics quickly explodes. In fact, the problem is a particular instance of the well-known *Coalition Structure Generation Problem*, a highly combinatorial problem which consists of splitting a set of agents into a complete and disjoint set. In this final degree dissertation, we propose an integer model for team formation in the classroom, and we also explore the behavior of several open-source solvers when solving the aforementioned model. Due to the poor scalability of exact solvers with large class sizes, we also design metaheuristic algorithms, which are more affordable for solving large instances of the problem, and we study their performance compared to the exact methods.

Keywords: optimization; prescriptive analytics; genetic algorithms; mathematical programming.

Índice de contenido

Índice de figuras

Índice de tablas

Índice de algoritmos

1. Introducción

El trabajo en equipo en aulas de estudiantes es una práctica que se ha hecho más común dentro de muchas titulaciones de educación superior. Sin ir más lejos, la Universidad Politécnica de Valencia (UPV) [1] considera el "trabajo en equipo y liderazgo" como una competencia transversal, esto es, una competencia vinculada al crecimiento de uno mismo. Desde la UPV se busca la aplicación y evaluación de estas competencias [2], adquiriendo diversos niveles de dominio, y cuya importancia se recoge en el proyecto de competencias transversales de la UPV.

Las actividades orientadas al trabajo en equipo permiten el aprendizaje de muchos aspectos como consecuencia de estar bajo un clima de colaboración entre personas, en el que para progresar, resulta necesario compartir los conocimientos o puntos de vista que se tienen acerca de cuestiones asociadas a la tarea que le sea encomendada al equipo. Con ello, entra en juego una hibridación entre la demostración de los conocimientos técnicos al resto de componentes del equipo y habilidades más relacionadas con la personalidad de estos, como podrían ser la capacidad de apoyo o resolución de conflictos internos, la comunicación efectiva o el liderazgo. La demanda de la combinación de este tipo de capacidades es constante en las industrias. En la industria, a menudo deben enfrentarse a proyectos de gran calibre que no pueden abordarse individualmente y pueden requerir la participación de individuos de incluso departamentos distintos.

Para favorecer el desarrollo personal de los alumnos, los equipos se deben estructurar de una forma adecuada que permita una experiencia fructífera. Para ello existen muchos criterios posibles a la hora de formar equipos en el aula, pasando de los más simples como hacer las agrupaciones teniendo en cuenta la nota media de los alumnos o el horario que tienen disponible para el trabajo, hasta otros más profundos como la teoría de roles de Belbin [3] o las pruebas de personalidad de Myers-Briggs [4]. En cualquiera de los casos, normalmente es el instructor quien debe aplicar el criterio deseado, y esto supone un gran esfuerzo pues la cantidad de equipos que se deberían considerar, así como la forma en la que estos se conjugan para particionar el aula, entra dentro de la combinatoria. Encontrar una partición óptima de estudiantes en equipos es un problema altamente combinatorio como para ser resuelto manualmente, por lo que resulta necesario la utilización de alguna herramienta informática para este fin.

Este problema entraría dentro del conjunto de problemas de *Coalition Structure Generation* [5], y, a pesar de que puede tener numerosas variantes que pueden depender del tamaño de los grupos que se quieran formar, el criterio de formación a utilizar u otras restricciones adicionales, comparte con estas variantes el hecho de que se busca crear la estructura óptima de equipos según el/los criterio/s mencionado/s. La resolución de este tipo de problemas de forma manual queda fuera de lugar, por lo que se debe recurrir a algoritmos para lidiar con la complejidad computacional asociada a este tipo de problemas. De hecho, son problemas que pertenecen al tipo NP-completo [5], lo que se traduce en gran complejidad computacional y la existencia de dificultades en su resolución por técnicas existentes. Por ello, es necesario estudiar algunos algoritmos dedicados a la resolución de este tipo de problemas.

1.1. Objetivos

Sabiendo que existe actualmente un modelo matemático para la resolución de este tipo de instancias (se verá en la sección 3), los objetivos de este Trabajo Final de Grado (TFG) son fundamentalmente tres:

- Implementar el modelo matemático empleando herramientas de código libre para favorecer la difusión del modelo y su incorporación en herramientas gratuitas.
- Implementar soluciones algorítmicas para la resolución de problemas con tallas grandes y que permitan la resolución exacta o aproximada del problema en tiempos razonables para su incorporación en herramientas informáticas.
- Analizar el rendimiento a nivel de optimalidad y a nivel temporal de las soluciones implementadas para determinar las soluciones algorítmicas más apropiadas.

1.2. Estructura

El contenido de la memoria se distribuirá en las siguientes secciones:

- **1. Introducción**. En ella se ha descrito de forma holística el contexto del problema a tratar y su relación con el uso de modelos de optimización, además de los objetivos perseguidos con este trabajo.
- **2. Revisión bibliográfica**, donde se hace un repaso teórico de conceptos a conocer en el trabajo y se profundiza en aspectos vistos en la introducción. Concretamente se comienza por presentar la optimización y la formulación de modelos matemáticos de optimización (2.1) entre otras cosas, poniendo el foco sobre métodos metaheurísticos para la resolución de problemas e incidiendo en los algoritmos evolutivos (2.2). Finalmente se habla más en detalle del contexto de formación de equipos, y se justifica la utilización de algoritmos para lidiar con este tipo de problemas (2.3). Esta última subsección incluye la mayor parte del estado del arte del problema, al dar como ejemplos alternativas vistas en la literatura para la creación de equipos.
- **3. Análisis del problema**. Aquí se habla brevemente del problema de la creación de equipos en el aula. Concretamente, se proporcionará un modelo matemático para su formalización y correspondiente modelado. Además, se realizará un análisis del marco legal y ético.
- **4. Diseño de las soluciones**. Son dos los tipos de soluciones que se han implementado para dar solución al problema, una en relación con el uso de métodos exactos (4.1) y otra con el uso de metaheurísticas (4.2). En el primer punto se describen los algoritmos a utilizar, mientras que en el segundo se detallan todos los componentes que compondrán el algoritmo genético (metaheurística planteada) así como su estructura.
- **5. Desarrollo de las soluciones**. Esta sección describe cómo se ha implementado en Python las soluciones diseñadas de la sección anterior. Consta de varias subsecciones, en las que habla de fragmentos de código comunes a todas las soluciones (5.2), la estructura seguida en la utilización de los métodos exactos (5.3) o la clase creada para el algoritmo genético y sus diferentes configuraciones (5.4).
- **6**. **Experimentación.** Además de la descripción de los datos de partida y la generación de instancias para resolver (6.1), se explican las pruebas realizadas para el análisis del rendimiento de los métodos exactos (6.2), así como las pruebas realizadas para el

análisis de rendimiento de las metaheurísticas propuestas, una vez observado que los métodos exactos no son capaces de resolver en tiempos razonables las instancias más grandes del problema (6.3).

- **7. Conclusiones**, donde se reflexiona sobre los resultados obtenidos con las soluciones planteadas, la relación del trabajo con los estudios cursados y los trabajos futuros.
- **8. Bibliografía**, que contiene todas las referencias a libros, artículos, etc. empleados para la realización de la memoria.

2. Revisión bibliográfica

Este apartado se describe el marco teórico del trabajo. En primer lugar, se introduce el campo de la optimización matemática, las fases típicas que conlleva encontrarse ante un problema de este tipo o los componentes que se pueden encontrar en los modelos matemáticos de optimización (sección 2.1); posteriormente se analiza el concepto de metaheurísticas, aportando una descripción de una tipología de clasificación encontrada y detallando el caso de los algoritmos evolutivos, entre los cuales se encuentran los algoritmos genéticos (sección 2.2); finalmente, se describe el contexto de formación de equipos y se dan ejemplos de algoritmos o herramientas de formación de equipos vistas hasta ahora (sección 2.3).

2.1. Optimización

La optimización es un campo multidisciplinar que busca hallar las mejores decisiones a considerar en un problema que pueda estar restringido por ciertos factores. Es un concepto amplio que se puede aplicar en ocasiones bien diversas, como el ámbito empresarial, en la ingeniería, en el ámbito educativo, en las ciencias de la computación, o en relación con sucesos que ocurren en la naturaleza.

La optimización matemática forma parte de la disciplina de la investigación operativa. Ésta y otros componentes de la investigación operativa quedan enmarcados en lo que se conoce como la analítica prescriptiva. Ésta busca anticiparse a los riesgos que puedan surgir en un futuro para acercarse lo máximo posible a los objetivos definidos en el caso de una organización, tomando las mejores decisiones según los datos procedentes de una anterior etapa de analítica predictiva, como se puede ver en la figura 1, en la que se pueden utilizar modelos de aprendizaje automático para determinar una situación futura.

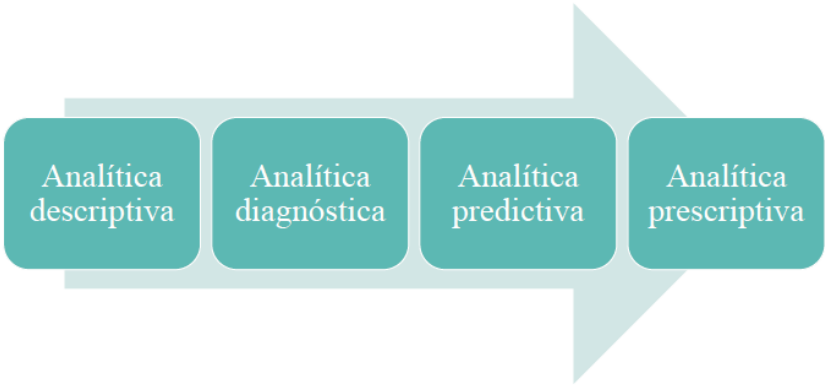


Figura 1. Tipos de analítica. [6].

Según [7, pp 1-7], suelen seguirse los siguientes pasos a la hora de modelar escenarios en los que se debe optimizar las decisiones tomadas en una organización:

1) **Formulación del problema a resolver**, el cual estará relacionado con alguna de las metas a cumplir por la organización.
2) **Observación del sistema**. Esta fase comprende la recopilación de datos del sistema en observación para la estimación de parámetros que utilizará el modelo de optimización en las fases siguientes.
3) **Formulación de un modelo matemático de optimización para el problema**. Se comentará en detalle la estructura de los modelos más adelante, por ahora hay que conocer que un modelo de optimización es una formulación que trata de representar matemáticamente los factores asociados al problema a resolver.
4) **Verificación del modelo**. Se trata de una fase que puede darse en múltiples ocasiones, ya que el modelo planteado puede no recoger algún comportamiento del proceso o en general presentar fallos en su definición.
5) **Selección de una alternativa que se ajuste**. Una vez se ha definido el modelo de optimización, se personaliza con los datos de los parámetros obtenidos y se resuelve el problema asociado, devolviendo la mejor decisión encontrada. Esta parte requiere del uso de programas informáticos o librerías para resolver problemas matemáticos, denominados *solvers*. Un *solver* se encargaría de tomar el modelo matemático y devolver la solución al problema, así como posibles análisis sobre ella.
6) **Presentación de los resultados**. Se pueden crear informes explicativos de la respuesta obtenida por el algoritmo de optimización, y discutir si estas son coherentes con el juicio experto. En caso de no ser correctas estaríamos ante problemas de cómo se formula o modela el problema, o bien en la propia recogida de los datos utilizados en las estimaciones.
7) **Implementación y evaluación de recomendaciones**. El sistema puede ponerse en producción una vez cumple con las expectativas de la organización, aunque debe ser constantemente monitorizado ya que con el paso del tiempo los parámetros que lo conforman (y que dependen de los datos del sistema) pueden cambiar.

Tal y como se mencionaba en las fases anteriores es posible emplear una representación matemática de un escenario observado en la realidad, lo que se conoce como un modelo matemático. Si lo que se desea es tomar las decisiones adecuadas sobre elementos que intervienen en la realidad con tal de alcanzar el mejor resultado posible, puede definirse un modelo matemático de optimización siguiendo la anterior metodología. Un modelo de optimización [6] está formado por los siguientes componentes:

- **Parámetros:** equivalente a los datos de entrada del problema, y los que se estiman inicialmente. Representan, generalmente, valores o hechos sobre los que no se tiene poder de decisión.
- **Variables:** representan las decisiones que se quieren efectuar en el problema.
- **Función objetivo:** función matemática que expresa qué es lo que se desea optimizar (ya sea maximizar o minimizar) y que depende de las variables y los parámetros definidos. En otras palabras, indica el rendimiento del sistema que se desea crear de forma cuantitativa, y es lo que se emplea para evaluar la calidad de una posible solución al problema.
- **Restricciones:** definen qué relaciones existen entre las variables además de restringir los posibles valores que pueden tomar ellas. Por tanto, determinan los hechos que se deben cumplir en una solución válida al problema.

Existen numerosos tipos de modelos de optimización, la categorización de unos u otros depende de factores como la precisión con la que se hayan estimado los parámetros (se puede tener modelos deterministas o inciertos), el dominio de las variables de decisión empleadas, la forma de la función objetivo y/o las restricciones del problema o el uso de una o varias

funciones objetivo, entre otros. La figura 2 muestra una taxonomía de los tipos de modelos de optimización:

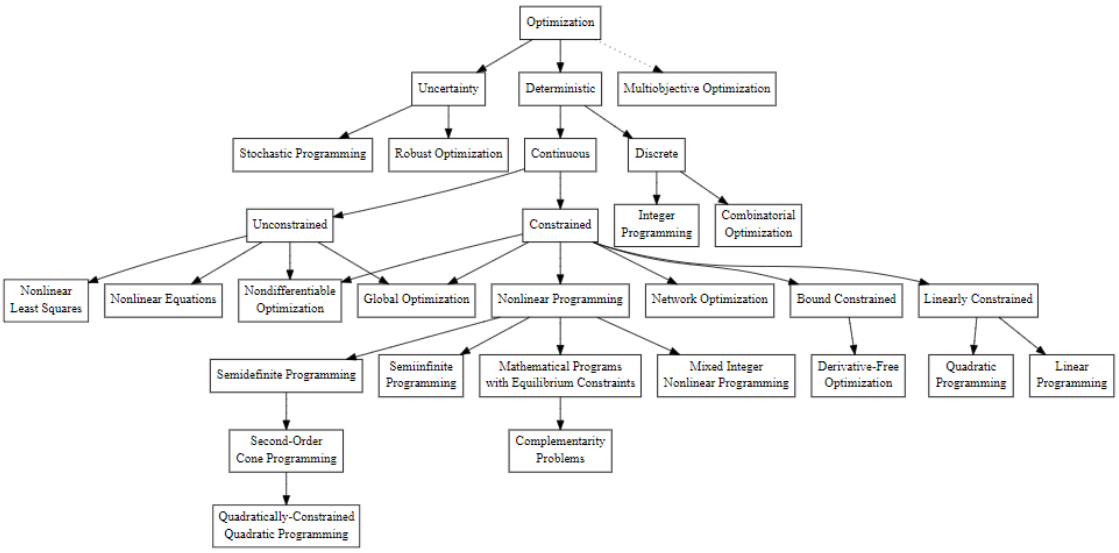


Figura 2. Taxonomía de los modelos matemáticos de optimización. [8].

Un tipo de modelo a destacar es el asociado al problema planteado y resuelto en este trabajo: los modelos de programación entera lineal. En estos modelos, todas las variables de decisión cobran valores enteros y la función objetivo y restricciones son lineales. Además, se debe cumplir que los valores de las variables de decisión no sean negativos (restricción de no negatividad) y los coeficientes del modelo sean deterministas. Su definición matemática general sería la siguiente:

$$min\ /\ \max Z = \sum_{i=1}^{n} c_i\, X_i$$

Sujeto a:

$$\sum_{i=1}^{n} a_{1,i}\, X_i \leq (\geq)(=)\ b_1$$

$$\dots$$

$$\sum_{i=1}^{n} a_{m,i}\, X_i \leq (\geq)(=)\ b_m$$

Con:

$$X_i \geq 0, X_i \in Z$$

Para ayudar a su comprensión, se muestra en la siguiente figura (figura 3) un ejemplo de modelo de programación lineal entera con la mayoría de los elementos comentados hasta ahora:

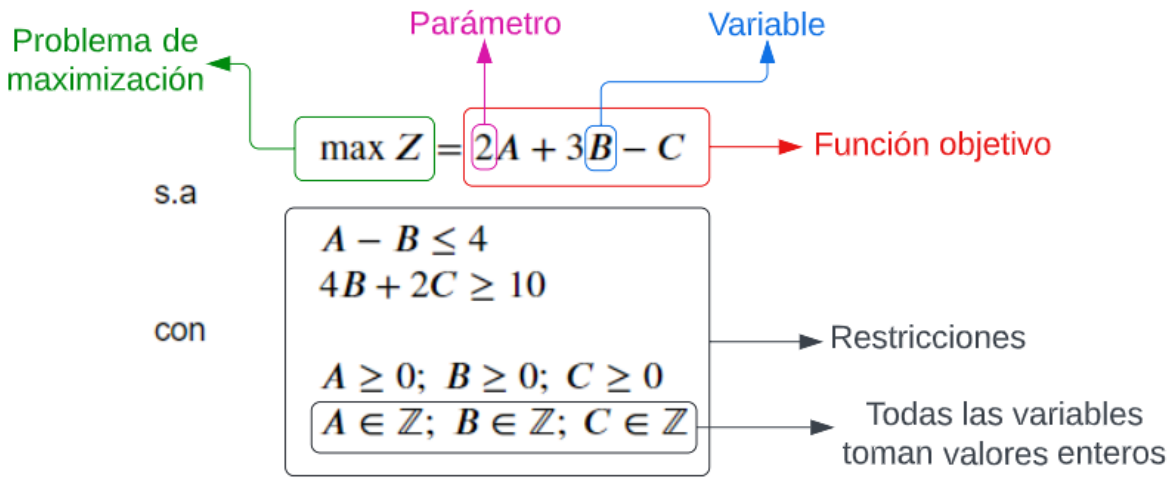


Figura 3. Ejemplo de modelo de programación lineal entera. Elaboración propia.

Por último, en cuanto a los algoritmos de optimización, podríamos destacar la clasificación dada según la calidad de la solución obtenida [6]. Si el algoritmo se encarga de garantizar la optimalidad de las soluciones, entonces, se trata de un método exacto, cuyo principal inconveniente es el tiempo que puede llegar a tardar cuando la talla de los problemas crece. Se pueden encontrar varios métodos exactos encargados de resolver problemas de programación lineal entera. El esquema de ramificación y poda (Branch and Bound), de forma simple, se encarga de resolver el problema relajado de programación lineal asociado a un modelo lineal entero (esto es, restricciones y función objetivo lineales y variables que pueden tomar valores reales) y de expandir un árbol de búsqueda que particiona el espacio de búsqueda, imponiendo nuevas restricciones sobre las variables de decisión hasta dar con soluciones enteras. En el proceso se permite podar ramas o áreas del espacio de búsqueda que ya no sean prometedoras con el fin de poder encontrar antes la solución final. El método de ramificación y corte (Branch and Cut) [9] combina el proceso de ramificación y poda con el uso de un algoritmo de imposición de planos de corte, el cual se encarga de imponer nuevas restricciones lineales al problema relajado en caso de encontrarse una variable con valor no entero tras su resolución. También encontramos el algoritmo Branch and Price [10], en el que en cada nodo del algoritmo de Branch and Bound se aplica el algoritmo de generación de columnas, consistente en resolver el problema relajado con un subconjunto de variables e ir incrementando la cantidad de estas según la “recompensa” que devuelven, siendo útil para problemas con gran número de variables donde no es viable resolver el problema con todas ellas a la vez. Finalmente, se tendrían los métodos aproximados, donde se encuentran las heurísticas y las metaheurísticas, y que suelen escalar de forma adecuada en problemas complejos, pero a costa de no garantizar encontrar la mejor solución del problema.

2.2. Metaheurísticas

Una heurística es, según una de las varias acepciones dadas por la RAE, “en algunas ciencias, manera de buscar la solución de un problema mediante métodos no rigurosos, como por tanteo, reglas empíricas, etc” [11], mientras que una metaheurística se califica como una

estrategia formada por la acción conjunta de diversas heurísticas para el mismo fin. Los métodos aproximados confirman su base en el uso de alguna de estas dos grandes categorías, ya que a pesar de que no tienen una definición que garantice la mejor calidad en la solución de un problema, está probado que en la práctica pueden llegar a dar resultados muy próximos (o iguales) a los óptimos. Las heurísticas en concreto tienen un mejor coste computacional que las metaheurísticas, aunque estas últimas no hacen un uso tan intensivo en comparación a como lo demuestran muchos de los métodos exactos. Además, otra diferencia clave entre ambas es que las heurísticas son ad-hoc para problemas en concreto, mientras que las metaheurísticas son técnicas genéricas, aunque fácilmente adaptables y de las que se puede obtener un mejor rendimiento si esta adaptación se hace teniendo en cuenta el conocimiento del dominio o problema.

Existen infinidad de metaheurísticas, muchas incluso se crean para problemas específicos y están inspiradas por reglas o sucesos que ocurren en la naturaleza. En [12] se realiza una interesante recopilación de metaheurísticas bio-inspiradas, las cuales se pueden basar en la teoría de la evolución darwiniana, leyes físicas o en la inteligencia encontrada en enjambres. Para el primer tipo (los llamados algoritmos evolutivos) encontramos muchas variantes, como los algoritmos genéticos, la programación genética o la evolución diferencial; para el tercer tipo, basado en inteligencia de enjambres, se puede dar como ejemplos el algoritmo de colonia de hormigas o la búsqueda de cuco.

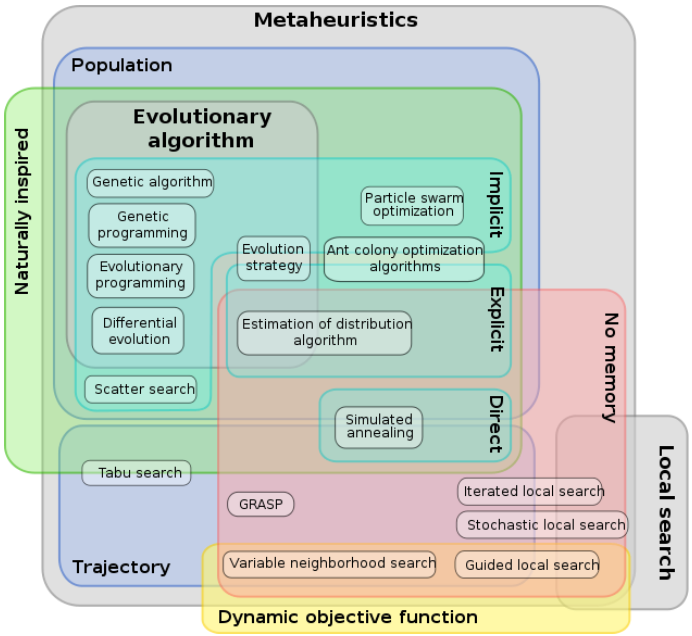


Figura 4. Diagrama de Euler de las diferentes clasificaciones de metaheurísticas. [12].

La figura 4 es una propuesta de clasificación de las metaheurísticas, donde se incorporan algunos ejemplos. Un primer grupo grande es el de las técnicas metaheurísticas que se inspiran en procesos naturales, como puede ser el proceso de recocido del acero para mejorar sus propiedades (*Simulated annealing*) o el proceso de liberación de feromonas por parte de las hormigas a la hora de encontrar recursos para guiar a otras (algoritmo de colonia de hormigas). Un subconjunto de este son los algoritmos evolutivos, sobre los cuales se centrará la sección 2.2.1. Otros dos grupos grandes por destacar serían los métodos basados en población, que

consisten en mantener en todo momento un conjunto de soluciones sobre las que realizar determinadas acciones, y los basados en trayectoria, que forman una trayectoria de búsqueda mediante la mejora de una única solución. Los métodos explícitos, implícitos y directos se diferencian en que los primeros consiguen utilizar una aproximación de la distribución de puntos que se utiliza para muestrear la función objetivo, los segundos deben aprenderla de forma implícita mediante varias formas y los últimos usan directamente la función objetivo. El resto de las clasificaciones serían los métodos basados en encontrar óptimos locales en vez del global, los que se utilizan en sistemas dinámicos o los sin-memoria, que no tienen en cuenta soluciones vistas anteriormente en el proceso de optimización [13]. Por ejemplo, la búsqueda tabú sería un algoritmo basado en la trayectoria y que utiliza memoria, ya que en cada iteración se trabaja sobre una solución que proviene del vecindario de soluciones de la iteración anterior y se emplea una memoria tabú, que está formada por soluciones que ya se han visitado en el proceso de búsqueda y son una guía para que el algoritmo no las vuelva a considerar y explore nuevas áreas posiblemente prometedoras.

Centraremos a continuación la explicación en los algoritmos evolutivos, pues una solución propuesta en este trabajo será el uso de algoritmos genéticos para la resolución del problema planteado.

2.2.1. Algoritmos evolutivos

Los algoritmos evolutivos inspiran su funcionamiento en la teoría de la evolución de Darwin. En ésta, se tiene en cuenta que los seres que conforman las especies o poblaciones de individuos tienen rasgos físicos (determinados por los genes) que pueden traspasarse a generaciones posteriores, y la existencia de recursos limitados en la naturaleza hará que prevalezcan aquellos grupos que mejor consigan adaptarse al medio [14], [15]. Principalmente, tras un periodo en que las especies se escogen entre sí, estas se aparean para dar lugar a nuevos seres que comparten cierta información genética de sus predecesores. Si esta información nueva resulta valiosa para cumplir con la mencionada selección natural, la población que comparte este tipo de características tenderá a crecer conforme suceden las generaciones o ciclos evolutivos. No obstante, puede ocurrir que alguno de los descendientes presente algún tipo de alteración en su material genético ya sea por alguna imperfección hereditaria de alguno de los padres, ciertos cambios en el ADN o alguno de los genes, o bien efectos posteriores como el impacto de los rayos ultravioleta. Si bien la mayoría no tienen efecto en los individuos afectados y ayudan a generar diversidad en la población, otras pueden derivar en enfermedades como el cáncer o en mejoras como una mayor inteligencia. [16], [17]. El traspaso de características como éstas depende de factores como la selección hecha por los predecesores o si estos genes son dominantes o no.

Podemos encontrar un símil entre la explicación anterior y los componentes principales de los algoritmos evolutivos, que se resumen en las siguientes entradas:

- **Representación:** una solución a un problema determinado puede asociarse con un individuo de una población considerada. Una población constaría de un conjunto de soluciones. Un cromosoma o genotipo de un individuo nos indica cómo representar internamente una solución y contiene todos los genes de éste, donde cada uno de ellos sirve para representar los distintos rasgos que definen al ser en concreto, o en este caso

a la solución en concreto. Un gen se referiría a las decisiones atómicas a tomar en el problema, y los alelos, que indican los valores que toma un gen en concreto, se traducirían a los posibles valores que podrían darse a dichas decisiones. El fenotipo consistiría en la representación externa (situación real) de la solución al problema. Podríamos tomar como ejemplo de lo anterior el problema típico de las N-reinas, el cual consiste en situar N reinas en un tablero de N x N casillas de forma que no se amenacen entre sí, sabiendo que tienen movimientos de tantas casillas como se quiera tanto horizontal como verticalmente. El fenotipo sería el propio tablero con la disposición de las piezas, mientras que el genotipo podría ser una matriz de ceros y unos donde un 1 indica que en esa casilla estaría una reina, aunque por motivos de espacio (se trataría de una matriz muy dispersa) se prefiere un único vector de N elementos (genes) donde el elemento en la posición i-*ésima* indicaría en qué fila se encuentra la reina puesta en la columna *i* (esta representación además evitaría colocar reinas en una misma columna). En la figura 5 se ve precisamente esta diferencia entre un tablero donde se han colocado 4 reinas (fenotipo) y la representación vectorial de este (genotipo), siendo este caso una solución no factible pues las reinas de las columnas 3 y 4 se amenazarían diagonalmente.

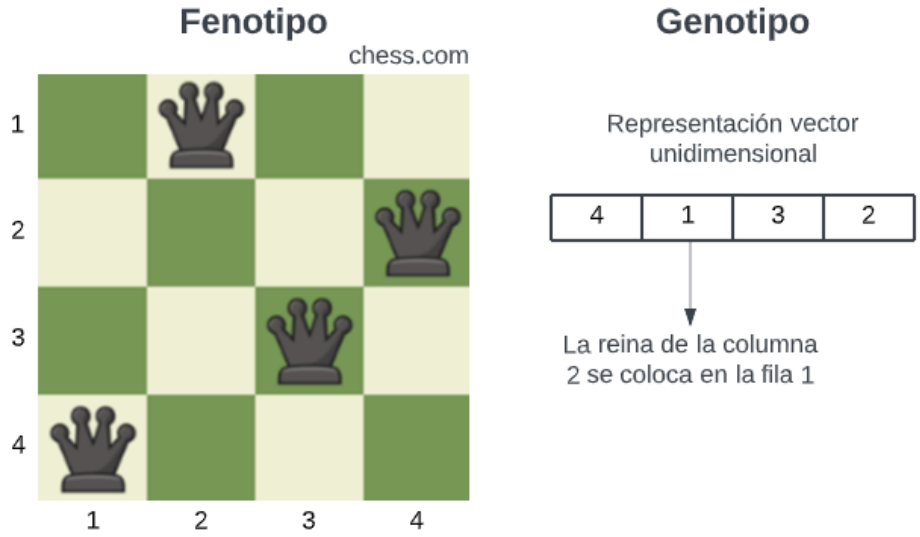


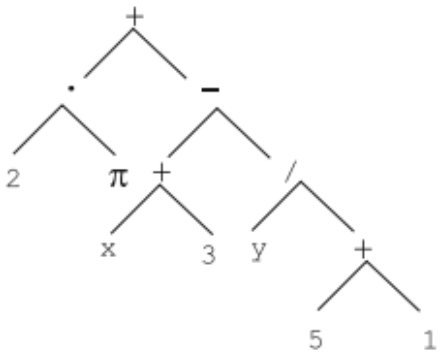
Figura 5. Fenotipo y genotipo de ejemplo en problema de N-reinas con N=4. Elaboración propia.

De esto también se deduce que, dado un fenotipo, puede haber diferentes tipos de genotipos que lo representen. La forma en que se representa una solución o, dicho de otra forma, el diseño del cromosoma asociado a un fenotipo es clave para un funcionamiento correcto y eficaz de un algoritmo evolutivo, y en general, de cualquier metaheurística, y es dependiente del problema en cuestión. Existen codificaciones que pueden ofrecer un resultado no válido o factible en la realidad, así como soluciones que no puedan representarse según el tipo de cromosoma escogido[1]. Otro ejemplo sería el hecho de que un fenotipo pueda dar lugar a cromosomas distintos del mismo tipo, esto es, redundancias en la representación. Ambos casos ralentizarían el rendimiento del algoritmo en cuestión, pues serían necesarios métodos para lidiar con las

[1] El denominado "problema de la ceguera" surge como consecuencia de utilizar una representación que no puede interpretar todas las posibilidades que se pueden dar en el problema real.

soluciones no factibles o se correría el riesgo de explorar espacios de búsqueda con la misma información.

Atendiendo a los alelos presentes en los cromosomas, se conocen diferentes tipos de representaciones bien establecidas dentro de los algoritmos genéticos: i) de tipo vector binario (bits o valores 0-1); ii) de tipo vector entero (genes que son ocupados por cualquier número entero); iii) de tipo vector real o iv) de tipo vector permutación (usualmente, números enteros positivos no repetidos). También existe la representación en forma de árbol, en la que se podrían plasmar expresiones más complejas como fórmulas aritméticas o lógicas o programas de ordenador, constituidas por un conjunto terminal (valores o variables que forman las hojas) y un conjunto de funciones (operaciones aritmeticológicas o funciones de programa que crean las dependencias entre el resto de los valores). La figura 6 muestra los conjuntos mencionados y la representación equivalente a la ecuación $2 \cdot \pi + \left((x+3) - \frac{y}{5+1}\right)$ en forma de árbol:



Conjunto de funciones	$\{+, -, \cdot, /\}$
Conjunto terminal	$\mathbb{R} \cup \{x, y\}$

Figura 6. Ejemplo de representación en árbol y conjuntos de funciones y terminal. [18, p 76].

Las anteriores son las representaciones más comunes, aunque se puede ver en la literatura el uso de representaciones mixtas (por ejemplo, poder combinar una parte de números enteros y otra de bits) u otras que son específicas del problema a resolver y que difieren en cierta manera de las ya comentadas. En el punto 4.2.2. se podrán ejemplos de algunos de estas representaciones para el problema de la formación de equipos en el aula.

Si consideramos las soluciones de un problema en concreto, para determinar cuáles sobreviven en el ciclo evolutivo se utiliza una función de aptitud o *fitness*. Ésta es común para las metaheurísticas y mide lo bien que se ajusta una solución al problema. Sería equivalente a la función objetivo en los métodos exactos.

Los operadores genéticos corresponderían a los distintos procesos dados en la naturaleza relacionados con la teoría de la evolución. En el ámbito de los algoritmos genéticos (descripción más adelante), estos operadores hacen referencia a parte de las heurísticas seguidas por el algoritmo para guiar el proceso de búsqueda. Entre ellas destacamos:

- **Selección** hace referencia al primer proceso mencionado al comienzo del apartado, donde se debe elegir qué individuos de la población deberían tenerse en cuenta para la generación de descendientes. Para ello este método deberá basarse en la aptitud (valor

de *fitness* de ahora en adelante) de las soluciones dentro del problema, de forma que aquellas que se califiquen de mejor manera en base a sus características tengan una mayor probabilidad de ser escogidas.

- **Recombinación**, denominado usualmente también como **cruce,** consistiría en escoger típicamente dos de los individuos obtenidos mediante la selección (padres) para la generación de dos nuevos individuos (hijos) con una cierta probabilidad de cruce, donde cada uno comparte información genética de cada uno de los padres. En los algoritmos genéticos (ver más adelante) puede verse como el principal operador para la búsqueda de soluciones, dado que la idea de su uso radica en que, si ambos padres tienen buen material genético, la mezcla de ambos será también buena y permitirá la exploración de espacios de búsqueda con material prometedor.
- **Mutación** sería lo siguiente en el proceso evolutivo, donde dado un descendiente como consecuencia del cruce entre los padres, se produce una variación ligera en el contenido genético del mismo con una determinada probabilidad, la cual suele ser baja (por ejemplo, un 10%). Esto último se debe a que modificaciones constantes del contenido de los descendientes evitarían centrar la búsqueda del algoritmo, que tendría más bien un comportamiento aleatorio. La ventaja del uso de esta operación radica en que se puede inyectar información útil que no necesariamente se encuentre en los padres (o incluso la población) y que dé paso a la exploración de zonas cercanas a una solución prometedora.
- **Reemplazamiento**, o también referido como selección de supervivientes. Basa su funcionamiento en la selección de aquellos individuos que conformarán la población en la siguiente generación. Dependiendo de los hijos generados en la generación actual, esta elección puede realizarse considerando el fitness de todas las soluciones tal como se veía en el método de selección, o bien otros como su edad (esto es, cuántas generaciones están incluidas dentro de la población).

Existen numerosos métodos evolutivos, donde se podría comentar algunos de los más populares [18, pp 99-112]:

- ***Evolution Strategies (ES)*** es un método propuesto inicialmente en 1960 por Rechenberg y Schwefel con la característica principal que, dado un individuo, la mutación del mismo para la obtención de uno nuevo se produce mediante la adición a cada uno de sus genes solución de un valor que proviene de una distribución normal de media 0 y desviación típica σ, la cual es otro parámetro del evolutivo denominado tamaño del paso de mutación, que se adapta automáticamente según la información recibida en el proceso de búsqueda. Típicamente se usa en problemas con decisiones representadas mediante variables reales o continuas. En cuanto al resto de operadores se destaca la selección uniforme de padres, una recombinación que puede ser local o global (esta última es más común e implica el uso de más de dos padres en la creación de un hijo) o reemplazamiento completo de la población.
- **Programación genética** es más bien un método aplicado para optimizar modelos de aprendizaje automático y consiste en determinar la fórmula (generalmente es un conjunto de reglas sobre las variables de decisión del modelo) que maximice el *fitness* obtenido, que podría ser la precisión del modelo en caso de ser este de clasificación. Es común pues utilizar la representación en árbol, algo más compleja y para la que no se utilizan métodos de recombinación o mutación habituales.
- Otros métodos podrían ser la **programación evolutiva**, la cual se ha vinculado bastante con la generación de inteligencia mediante elementos como autómatas finitos, los **Learning Classifier Systems (LCS)** ampliamente usados en el aprendizaje por refuerzo o los **algoritmos genéticos**, en los que se profundizará a partir del siguiente párrafo.

Conociendo un poco más algunos métodos evolutivos, nos centramos finalmente en los **algoritmos genéticos** (AG), que son los métodos más utilizados dentro del conjunto de los algoritmos evolutivos. Un AG comienza mediante la inicialización de una población de individuos generalmente de forma aleatoria y cuyo tamaño se puede parametrizar. Tras evaluar la calidad de las soluciones obtenidas se decide si continuar con la ejecución o no. En caso afirmativo, se hace una selección de padres de la población actual, los cuales se organizan típicamente por pares (por ejemplo, aleatoriamente) para proceder con su reproducción (cruce), cuyos resultados generan los hijos. Cabe la posibilidad de que algunos de estos individuos reciban una mutación genética. Tras disponer de todos los hijos se aplica el operador de reemplazamiento, que devolverá la población que será tenida en cuenta (i.e, evaluada) para la siguiente generación. La condición que implica la continuación en la ejecución es importante y se conoce como el criterio de parada. El conocer la solución óptima del problema facilita detenerse una vez alcanzada esta (ya sea de forma exacta o aproximada) pero lo más común es que ésta no se conozca debido a la complejidad que resulta en su cálculo. Por ello suelen plantearse otros criterios como el tiempo, número de iteraciones alcanzadas o el número de iteraciones en las cuales la mejor solución no presenta ninguna mejora. Un resumen de este proceso puede verse en la figura 7.

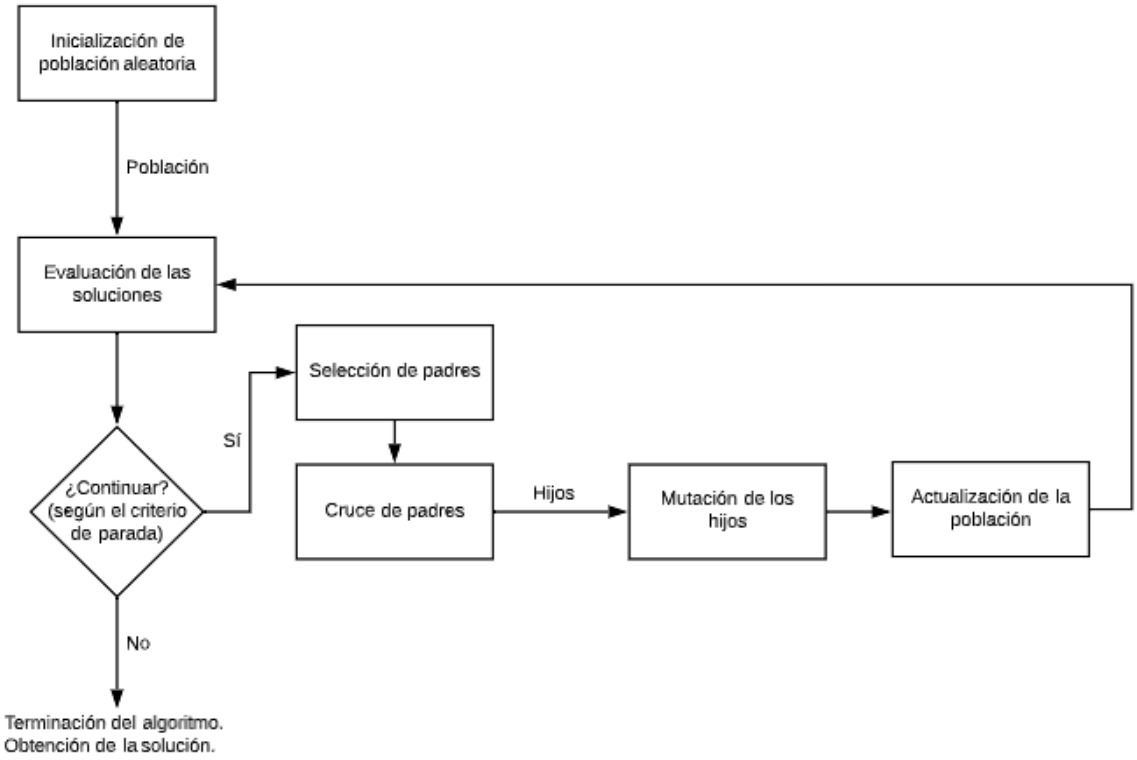


Figura 7. Esquema general de un algoritmo genético. Elaboración propia.

Dado este esquema (que utiliza una definición similar a un varAnd de cruce y mutación), también cabría comentar que existen ciertas variaciones en cómo se usan los distintos operadores durante las generaciones. En su defecto se puede indicar el caso de varOr, [19], donde para generar la población de sucesores se escoge una de estas 3 opciones: cruce,

mutación y reproducción. En el cruce solo se añadiría a los sucesores uno de los hijos, descartándose el otro, y la reproducción no es más que copiar el padre escogido a la nueva población.

2.3. Formación de equipos

En este último apartado de la revisión bibliográfica se hará hincapié, en primer lugar, en las estrategias de formación de equipos, las cuales pueden basarse en diferentes teorías. Tras ello, se discutirá algún trabajo relacionado que indica la aplicación de estas teorías en entornos de trabajo (apartado 2.3.1) y la complejidad de distribuir los individuos en los equipos manualmente. Posteriormente, en 2.3.2 se habla de en qué consisten los problemas de formación de estructuras de coalición, y se dan ejemplos de métodos exactos y metaheurísticas utilizados para resolver este tipo de problemas tanto a nivel general como en el caso particular de creación de grupos en aulas universitarias o en base a ciertas teorías.

2.3.1. Introducción

Cada vez más en las industrias prevalece el trabajo en equipo con objeto de abordar proyectos complejos que de por sí pueden consumir muchos recursos y requerir de gran tiempo de planificación y dirección, algo que llevan replicando los centros de educación superior [20] para formar a sus estudiantes en este tipo de prácticas.

Por ello, resulta interesante desarrollar el trabajo en equipo dentro de las universidades, ya que cobra una relación directa con el contenido de este proyecto. De hecho, la importancia de los futuros profesionales no recae únicamente en sus habilidades técnicas (*hard skills*) sino también en sus habilidades blandas (*soft skills*), entre las que se encuentra el trabajo en equipo [21]. Esta última permite además el desarrollo de otras competencias personales, como son la capacidad de expresar y defender ideas frente al grupo de compañeros (en general habilidades para la comunicación), el liderazgo o el manejo eficiente del tiempo (entre otros hechos para cumplir con las metas impuestas por el propio equipo o el instructor) [22], [23] entre infinidad de otras. Estas habilidades blandas pueden ayudar a las empresas en su búsqueda de perfiles.

Para que haya una evolución en este tipo de competencias, se debe potenciar un buen trabajo en equipo en todas sus fases. Tomando como referencia las etapas de formación de grupos de Tuckman [24], ya resulta clave el cómo realizar la formación de los mismos (primera etapa), mientras que etapas posteriores (asalto, normalización y actuación) dependerían no solamente de la interacción del grupo sino también del instructor en caso necesario. Claros ejemplos del primer caso podrían ser la resolución de conflictos internos que permitan que los integrantes maduren y se centren en la distribución de las tareas y roles, o la coordinación de estos en la resolución de estas, mientras que el segundo podría considerar la monitorización del instructor de los grupos, imponiendo metas a corto plazo para mantener la motivación de estos [20].

Cabe centrarse en la propia formación de los equipos, señalado la existencia de numerosas estrategias para ello, en general relacionadas con las ideas de diversidad o heterogeneidad entre los integrantes de un equipo. Los autores de [25] establecen un interesante debate de las

perspectivas que se pueden ofrecer al término de "diversidad" desde un conjunto de dimensiones formados por diversas variables (por ejemplo, raza, sexo, rendimiento académico, experiencia laboral, relaciones de amistad…) contemplando algunas opiniones extremas o teorías. En el mejor caso, estas opiniones ayudan a aportar puntos de vista distintos al trabajo, y en el peor solo producen divisiones en el equipo. En cuanto al desarrollo personal, podríamos poner el ejemplo de [26], donde se concluye que, aunque las estrategias y resultados no puedan ser replicados para otro tipo de cursos, el emplear una estrategia de formación de equipos basada en habilidades complementarias (esto es, cada estudiante aporta algo distinto al equipo) garantiza las habilidades necesarias para el ámbito industrial en los estudiantes de ingeniería considerados en comparación con otras estrategias que impliquen la diversidad en el género de estos o su nota media.

Además de este tipo de criterios, también surgen teorías psicológicas que buscan entender cómo se pueden estructurar los equipos. Estas teorías basadas en la psicología organizacional pueden aplicarse en ámbitos reales. Son de vital importancia para el proyecto las dos primeras teorías que se describirán a continuación, ya que las heurísticas implementadas se basan en estas dos teorías (Belbin y Myers-Briggs):

- Teoría de roles de Belbin. Belbin realizó numerosos estudios registrados en su libro *Management Teams: Why they succeed or fail* [3] sobre qué factores influyen en el buen funcionamiento de equipos de trabajo. Por ejemplo, todo equipo según la actividad y otras variables tendrá un tamaño (esto es, el número de personas que lo forman) idóneo. Un buen clima de trabajo se puede propiciar por miembros que faciliten oportunidades al resto de sus compañeros o sepan cuándo es el mejor momento de intervenir dada una decisión. No obstante, su salto a la fama se debe a la definición de roles dentro de un equipo. Estos roles describen patrones de comportamiento que son necesarios en sus integrantes para alcanzar a ser un equipo con éxito. Estos roles no son excluyentes en un individuo, al tratarse realmente de patrones de comportamiento. Una persona podría jugar diferentes roles dentro del equipo como consecuencia de la emersión de diferentes comportamientos en diferentes momentos. Esto es, no existe una limitación de un rol por persona. Belbin demostró que un buen balance en la distribución de roles dentro de un equipo presentaba resultados a nivel de equipo mucho más satisfactorios que equipos con roles sobrerrepresentados o con falta de estos. Inicialmente definió los siguientes 8 roles:

- Implementador (Co-worker, CW) que lleva a cabo las tareas predefinidas;
- Coordinador (Chairman, CH) que, con actitud calmada, busca la contribución de todos los miembros del equipo;
- Impulsor (Shaper, SH), persona de gran determinación y energía que ayuda a todos a superar los obstáculos;
- Cerebro (Plant, PL), la mente innovadora y creativa;
- Investigador de recursos (Resource Investigator, RI), destaca en su extroversión y creación de contactos necesarios para el equipo;
- Monitor-evaluador (Monitor-Evaluator, ME), que con actitud crítica piensa en la viabilidad de las ideas surgidas;
- Cohesionador (Team Worker, TW) que trata de prevenir la fragmentación del equipo y busca la creación de un buen clima de trabajo;
- Finalizador (Completer-finisher, CF) indispensable para el "control de calidad" de los resultados.

Finalmente, cabe indicar que se añadió posteriormente un noveno rol, el especialista, con gran conocimiento técnico sobre algún tema y que busca asesorar a otros miembros [27]. La notoriedad de todos los roles anteriores en un individuo puede conocerse a partir del uso del *Self Perception Inventory*, un cuestionario cuya estructura se resumirá más adelante.

- **Inventario tipológico de Myers-Briggs** (Myers-Briggs Type Indicator o MBTI por abreviar). Está basado en el trabajo *Psychological Types* de Jung, quien consideraba que todas las personas eran o bien extrovertidas o introvertidas y podían existir hasta 8 personalidades distintas dentro de cada grupo al combinar 3 escalas dicotómicas más, cada uno con 2 rasgos opuestos [4]. Este indicador definido por Katharine Cook Briggs y su hija Isabel Briggs Myers considera la existencia de 4 dimensiones o dicotomías, cada una formada por 2 rasgos que son opuestos entre sí, y que la combinación de los rasgos de cada dimensión conforma hasta 16 personalidades distintas. Las dicotomías son Extroversión/Introversión, Sensorial/Intuitivo, Pensamiento/Emocional y Calificador/Perceptivo. Por ejemplo, el calificado como "logista" corresponde con la abreviatura ISTJ (introvertido, sensorial, pensamiento o thinking y calificador o judging) y es un "individuo práctico y enfocado en los hechos, de cuya confiabilidad no puede dudarse" [28]. En la web de la anterior referencia se hace una división de los 16 tipos de personalidad en 4 grupos: los analistas, formados por las personalidades de arquitecto, lógico, comandante e innovador y que comparten las dimensiones de intuitivo y pensamiento; los diplomáticos, con la intuición y lo emocional en común, donde están los perfiles de abogado, mediador, protagonista y activista; los centinelas, formados por el logista, el defensor, el ejecutivo y el cónsul y que comparten lo sensorial y lo calificador; y los exploradores, que guardan relación en lo sensorial y lo perceptivo y lo forman los que nombran como el aventurero, el virtuoso, el emprendedor y el animador.

- **Big Five**. La teoría proviene de los resultados de un conjunto de investigadores, y es consecuencia de la aplicación del análisis de factores sobre datos de personas de cuestionarios relativos a rasgos de personalidad [29]. Con el análisis de factores se busca obtener rasgos ocultos derivados de correlaciones que puedan existir en un conjunto de variables, y en este caso el resultado de la aplicación de éste sobre cuestionarios de personalidad revela la aparición de palabras que pueden asociarse a la personalidad de los individuos. La consideración de estas asociaciones da lugar a la definición de 5 dimensiones de personalidad que cubren en gran medida los comportamientos que se pueden esperar en las personas. Estos son: extraversión, agrado, apertura a la experiencia, neuroticismo y escrupulosidad. Cada una de estas dimensiones se basa en la conjunción de rasgos relacionados. Por ejemplo, el neuroticismo implica la observación de sentimientos negativos en una persona como la ansiedad o la depresión.

Algunas de estas teorías se han probado en ámbitos educativos, viendo casos que resultan satisfactorios. En todo el trabajo comprendido en [30] se implanta una metodología basada en proyectos. Esto es, la asignatura en cuestión se basa en el desarrollo de un caso o proyecto base en el que los estudiantes trabajan en equipo para su resolución) en diversos cursos de ingeniería de la universidad del País Vasco. Se comparó la evolución de aquellos estudiantes que han trabajado en grupos basados en la teoría de roles de Belbin frente a aquellos creados sin ningún criterio definido. Se siguió un procedimiento bien definido para los equipos de Belbin, que pasa por actividades y formaciones sobre esta teoría hasta el ofrecimiento de informes (con la herramienta Belbin GETSET) sobre los resultados de los alumnos en los roles. El seguimiento y resultados finales concluyen que esta teoría ha contribuido de mejor manera al aprendizaje de dichos alumnos, lo cual se puede medir no solamente por la satisfacción de estos en el trabajo en equipo sino también por las notas finales obtenidas, mejores que los equipos formados de la

otra manera y además suponiendo un menor tiempo de estudio por los alumnos fuera del aula. Rodríguez Montequín et al. [31] conforman un análisis del rendimiento de equipos para un curso de gestión de proyectos en la carrera de ingeniería industrial en el que también se lleva a cabo una metodología basada en proyectos en relación con las personalidades de MBTI. Aunque la creación de los equipos es de forma aleatoria, se estudia cómo afecta la distribución de personalidades en cada uno así como el papel del coordinador del equipo (fundamental para distribuir las tareas y gestionar las metas pendientes) en las calificaciones recibidas. Se concluye que los equipos que más han prosperado son los que han seleccionado a un coordinador con un perfil con grandes capacidades de liderazgo y comunicación, como el ESTP, al contrario de lo que podría ocurrir con otros coordinadores con roles más autoritarios. Otro aspecto destacable es que los conflictos en los grupos se han agravado más (afectando al rendimiento de estos) al combinar perfiles emocionales con otros que tienen más en cuenta los objetivos del proyecto.

Un inconveniente en la creación de equipos es que el proceso puede llegar a ser algo tedioso para el instructor si se hace de forma manual. En el primer ejemplo del párrafo anterior, los grupos basados en Belbin eran formados por el tutor, quien buscaba encontrar cierta diversidad de roles considerando los 2 o 3 más prominentes en cada alumno. La cantidad total de equipos (de tamaño entre l y u) posibles que pueden surgir de una clase con n alumnos sigue la combinatoria $\sum_{i=l}^{u}\binom{n}{i}$, que crece enormemente para problemas grandes. Por ejemplo, una clase típica de 60 alumnos en educación superior en la que se creen grupos de 4 implicaría analizar $\binom{60}{4} = 487.635$ equipos diferentes y escoger los 15 que en principio funcionen mejor, teniendo además en cuenta que los equipos no deben ser solapados. Realizar este proceso manualmente no es viable. Es por ello por lo que cada vez más aparecen herramientas o algoritmos que, con los datos necesarios, ayudan no solamente a esta tarea sino a encontrar la mejor estructura de equipos posible, los cuales se pueden aplicar a cualquier problema que consista en dividir un conjunto de agentes de forma separada (ver apartado 2.3.2).

En resumen, el trabajo en equipo es importante para el desarrollo de habilidades que sirvan en cualquier aspecto del mundo laboral, y el cómo se lleve a cabo determina el aprendizaje de estas. Sin embargo, la formación de equipos es un proceso que requiere un esfuerzo computacional alto, con lo que se necesitan herramientas que sean capaces de encontrar soluciones satisfactorias en un tiempo razonable. A continuación, se verá que es posible utilizar algoritmos que, basados en ciertas estrategias, permitan obtener la mejor estructura de equipos de un conjunto de personas. El uso de estos será el centro de este proyecto.

2.3.2. Algoritmos de formación de equipos

Siempre ha existido interés por parte de los investigadores en sistemas multiagente en la formación de coaliciones de agentes, las cuales se unen con el objetivo de obtener una mayor recompensa que trabajando individualmente. Por ejemplo, uno de estos objetivos puede ser el de reducir la utilización de recursos de un cierto proceso o el de realizar compras a escala a precios más reducidos. El problema de dividir un conjunto de agentes en subconjuntos o coaliciones de forma que se optimiza la función objetivo global del sistema se conoce como el problema de la formación de estructuras de coaliciones (*Coalition Structure Generation*, CSG) [5].

Principalmente, el problema de crear estructuras de coalición consiste en obtener una partición de un conjunto finito de agentes en coaliciones disjuntas de forma que se logre

maximizar el incentivo obtenido por la actuación de éstas o, en su defecto, minimizar el coste derivado de dichas actividades. Según esto, la creación de equipos en el aula (problema en el que se basa este trabajo) puede verse como una instancia del problema de CSG, pues se busca obtener la mejor organización de los alumnos para llevar a cabo los proyectos que se les plantean.

Existen diferentes versiones de este problema. Una primera clasificación depende de la tipología de funciones empleadas para valorar la utilidad aportada por una coalición. Los *Characteristic Function Games* (*CFGs*) evalúan la utilidad de una coalición mediante una función que únicamente tiene en cuenta la estructura de la propia coalición. Los *Partition Function Games (PFGs)* valoran la utilidad de una coalición teniendo en cuenta a la propia coalición así como al resto de las coaliciones. Por otro lado, se pueden tener instancias estructuradas e irregulares, donde la primera define una estructura simple en el espacio de estructuras de coalición mientras que la segunda utiliza una función objetivo donde los valores asociados a cada coalición son estocásticos [32].

En cualquier caso, se trata de un problema NP-completo[2], que permitiría su resolución mediante fuerza bruta pero que es prácticamente inviable dado que la cantidad total de posibles estructuras de coalición sigue el número de Bell (B_n), encontrándose en el rango de $\alpha n^{n/2} \leq B_n \leq n^n$, siendo n el número de agentes y α una cierta constante positiva [5]. Si estamos interesados en encontrar la solución óptima al problema, existen algoritmos que realizan una búsqueda sistemática del espacio de coaliciones y permiten, por lo general, evitar una exploración exhaustiva de todo el espacio de búsqueda en el caso de CFGs. No obstante, de igual forma, el problema sigue siendo, en el peor de los casos, NP-completo.

Relativo a los algoritmos para la resolución de este problema, existen algoritmos que se basan en la idea que todas las estructuras de coalición se pueden representar mediante un grafo donde, tras la partición más fina de los agentes (coaliciones de 1 miembro), en cada nivel se muestra, dado un nodo, la fusión de dos coaliciones de otro nodo en el nivel anterior hasta llegar a la coalición más grande a formar (ver figura 8). En [33] se propone el método determinista SPLIT, que hace una búsqueda primero en anchura desde el nodo más bajo y continúa con ella hasta encontrar con una solución que no es más que *n* veces peor que la óptima (este *n* lo establece el algoritmo) después de explorar 2^{n-1} nodos. También existe la posibilidad de crear modelos de programación entera [5] utilizando restricciones que aseguren la creación de grupos disjuntos. Otras propuestas implican el uso de la programación dinámica. Por ejemplo, en [5] se indica el algoritmo ODP (*Optimal DP*) que es una mejora del primer algoritmo de programación dinámica que surgió para este tipo de problemas (DP) y minimiza el número de evaluaciones que tienen lugar.

[2] Un problema NP-completo presenta un gran grado de dificultad dentro de los problemas NP, cuya definición se basa en que pueden resolverse por algún algoritmo sin conocer a priori el tiempo de ejecución y sus soluciones verificarse en tiempo polinómico [18, pp 9-13].

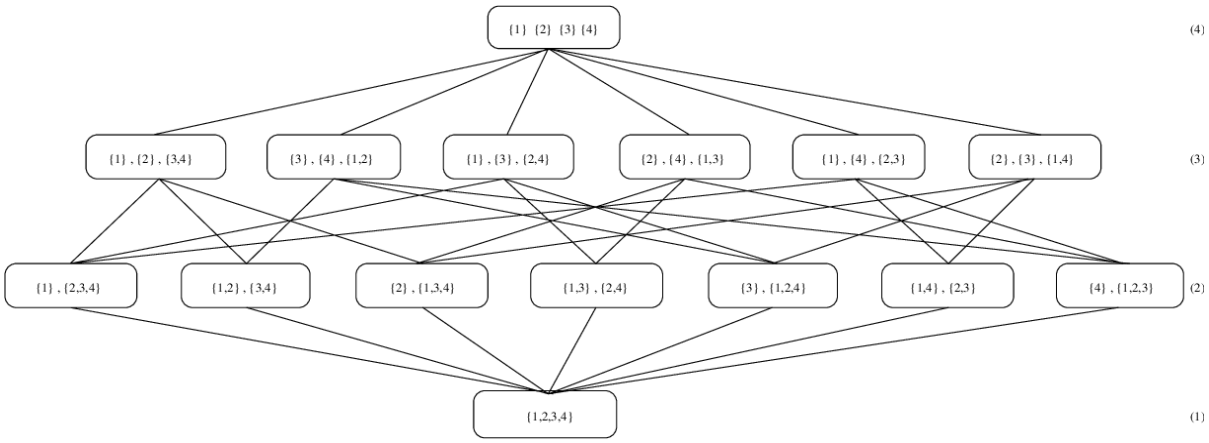


Figura 8. Representación en grafo de las estructuras de coaliciones en problema de 4 agentes. [34]

A costa de no garantizar la mejor solución, pero con gran mejora en tiempo de ejecución respecto de los métodos anteriores, se definen cada vez más numerosos algoritmos metaheurísticos. En [34] se crea el OBGA (*Order-Based Genetic Algorithm*), un algoritmo de búsqueda estocástica (algoritmo genético) que escala bien con el número de agentes debido a su innovadora representación de los cromosomas (representación basada en el orden, de ahí el nombre de OBGA) y en la experimentación realizada se concluye que logra tiempos hasta un orden de magnitud inferiores a los de SPLIT. En [35] se propone el uso de recocido simulado y búsqueda tabú en problemas de CSG, creando 4 operadores específicos para la generación del vecindario de una solución (combinación, extracción, partición y desplazamiento) y comparando el rendimiento de estos algoritmos para instancias pequeñas de CFGs y PFGs, siendo búsqueda tabú el mejor de ellos. Otro ejemplo sería el uso de GRASP (*Greedy Randomised Adaptative Search Procedure*) [36] con sus operadores específicos de exploración del vecindario de una estructura de coalición, el cual se compara con algoritmos de programación dinámica y el de recocido simulado comentado.

Por último, cabría destacar algoritmos o herramientas creadas para problemas que busquen crear coaliciones o estructuras de coalición óptimas en base alguna de las teorías (Belbin, MBTI…) desarrolladas en el apartado anterior u otro tipo de criterios. CATME dispone de una herramienta [37] para la creación y evaluación de equipos de estudiantes universitarios pasando como parámetros el tamaño de estos, las variables que se quieran tener en cuenta (hay un total de 16, como el sexo, nota media o el horario que tienen, y cuantas más se seleccionen menor impacto tendrán individualmente) y si se desea hacer la agrupación por la semejanza o disimilitud en los valores de estas. El algoritmo de optimización subyacente [38] es el algoritmo de ascenso de colinas (*Hill Climbing Algorithm*) el cual no garantiza un óptimo global pero sí uno local. En [39] se propone un modelo matemático de optimización para la generación de grupos de trabajo con distintas funciones objetivo que busca maximizar la diversidad dentro de los grupos a la vez que se minimiza la heterogeneidad entre estos. Permite la utilización de cualquier tipo de característica, teoría, etc. de los agentes siempre y cuando se pueda disponer en un formato matricial en el que una celda indica si un agente (fila) cumple o no (esto es, existe un valor 0 o 1) con la característica (columna), y las funciones objetivo tienen que ver con la maximización de la proporción promedio de características comunes en el equipo o con minimizar la desviación total absoluta entre equipos según estas. El algoritmo propuesto es un

pool-based metaheuristic, un híbrido que comienza con un método basado en la población y aplica posteriormente varias técnicas de optimización local, entre otros aspectos. Alberola et al [40] introducen una herramienta para crear equipos en el aula basados en la metodología de roles de Belbin mediante la resolución de un modelo de programación lineal entera, donde se utiliza aprendizaje bayesiano para estimar la probabilidad de que un agente pertenezca a cualquiera de los roles. Concretamente, se utiliza el historial de valoraciones que hacen los estudiantes sobre el de referencia para estimar dichas probabilidades (en caso de que no exista, se propondría una formación aleatoria), las cuales participan en el cálculo del valor esperado de un cierto equipo dadas las valoraciones y, por tanto, también a la hora de maximizar el valor de toda la estructura de coalición. En [41] se propone un algoritmo de optimización multiobjetivo de enjambre de partículas para la estructuración de un equipo en el ámbito de desarrollo de nuevos productos en la empresa, que pretende obtener el conjunto de trabajadores con las mejores capacidades para desempeñar el proyecto en mente en base a unos pesos derivados de las calificaciones de expertos basadas en una serie de criterios como la experiencia o la resolución de problemas (primer objetivo), y aquellos trabajadores que mantienen a su vez las mejores relaciones interpersonales, las cuales se estiman numéricamente según las interacciones entre las diferentes personalidades de MBTI (segundo objetivo). Este caso sería algo distinto a los anteriores pues se desea únicamente crear una coalición que contenga los mejores agentes para la tarea.

3. Descripción del problema

El problema que se desea abordar está relacionado con la creación de las "mejores" estructuras de coalición en aulas de universidad, de forma que el trabajo en equipo desempeñado por los alumnos no solo sea satisfactorio, sino que derive en el fortalecimiento de habilidades que puedan ser útiles para el futuro de los estudiantes. Principalmente, el trabajo se basa en el artículo *Assessing the use of intelligent tools to group tourism students into teams: a comparison between team formation strategies* [42], donde se presenta una infraestructura completa para la obtención de coaliciones óptimas en base a las teorías de Belbin y Myers-Briggs, además de la comparación de los resultados obtenidos para un conjunto de datos de estudiantes y la discusión de las estrategias de formación de equipos empleada según factores como el *feedback* recibido de los profesores/ponentes.

Actualmente la arquitectura presentada en esta infraestructura utiliza un programa comercial (CPLEX) encargado de resolver la asignación de personas a los equipos de trabajo maximizando la "puntuación" obtenida con las estrategias mencionadas. La palabra "comercial" implica un pago por el uso dado a la aplicación que centraliza la resolución de este tipo de problemas, así que una primera fase de este proyecto reside en la utilización de *solvers* de software libre (*open source*) que faciliten la diseminación y uso de la herramienta que se ha creado. Básicamente una primera solución será implementar *solvers* de este tipo y medir, mediante un estudio comparativo, cuál puede ser el más adecuado para qué casos concretos teniendo en cuenta criterios como el tiempo de ejecución.

Aun así, se ha conocido que en los problemas de generación de estructuras de coalición tiene gran presencia la combinatoria, haciendo que el número de soluciones crezca de forma exponencial hasta niveles en los que obtener una solución al problema para un número de agentes relativamente moderado signifique dedicar una gran cantidad de horas de cómputo utilizando un algoritmo exacto. De hecho, la experiencia en el uso de la herramienta original indica ejecuciones de varias horas para un conjunto de alumnos de tamaño relativamente moderado. Por todo esto, una segunda solución es el uso de metaheurísticas como los algoritmos genéticos que, aunque no garanticen la solución óptima al menos la pueden devolver en un tiempo mucho más ajustado, además de ser algoritmos *anytime* (esto es, se pueden detener y ver la mejor solución alcanzada hasta el momento).

Por último, aunque se hablará de ello en su respectivo apartado, se busca que las soluciones implementadas produzcan una reducción en los costes actuales, al dedicar menor tiempo de cómputo, y también al utilizar *solvers* gratuitos en el caso de la resolución por algoritmos exactos.

3.1. Modelización matemática

En este apartado se describe la formulación ofrecida en [42] como problema de optimización matemática para la formación óptima de equipos. Se presenta en este artículo un modelo de programación entera lineal que busca maximizar la suma de las puntuaciones de los

equipos formados en el aula, y es en cierto modo flexible dado que permite la utilización de diversas métricas para la asignación de puntos a equipos/coaliciones.

Teniendo un conjunto $S = \{s_1, s_2, \dots, s_N\}$ de N alumnos, donde s_i denota al alumno i-ésimo, se desea formar equipos de trabajo cuyo tamaño oscila entre un mínimo l y un máximo u integrantes. Estos parámetros son definidos por el tutor a fin de evitar formaciones incoherentes con el proyecto que desee que estos aborden. Un equipo t_i será factible cuando $l \leq |t_i| \leq u$ y además se cumpla que $\forall s_j, s_k \in t_i, (s_j, s_k) \notin \mathcal{N}$. De la anterior expresión, $\mathcal{N}$ es el conjunto de todos los pares de estudiantes que son "incompatibles", recibiendo esa definición aquellos a los cuales el profesor decida separar por motivos culturales o de conflicto entre ellos, entre otros. En un equipo válido no coincidirán estos. Considerando $\mathcal{T}$ como el conjunto de todos los equipos factibles (i.e., no contienen pares de estudiantes incompatibles y con tamaño comprendido entre l y u), el modelo matemático quedaría de la siguiente manera:

$$\max Z = \sum_{t_i \in \mathcal{T}} f(t_i) \times \delta_i$$

Sujeto a:

[complete partition]: $\sum_{t_i \in \mathcal{T}} |t_i| \times \delta_i = n$

[student j in a team]: $\sum_{t_i \in \mathcal{T}, s_j \in t_i} \delta_i = 1 \quad ; j = 1, \dots, n$

[compulsory pair j and k]: $\sum_{t_i \in \mathcal{T}; s_j, s_k \in t_i} \delta_i = 1 \quad ; \forall (s_j, s_k) \in \mathcal{C}$

Se explicará a continuación los componentes de dicho modelo:

3.1.1. Variables

El modelo especifica un único tipo de variable de decisión δ_i. Es una variable de tipo binario (esto es, toma valores 0 o 1) que indica si el equipo t_i es escogido o no para la estructura de coaliciones. Concretamente, su valor será 1 cuando dicho equipo participe en la solución (estructura de coaliciones) y 0 en caso contrario.

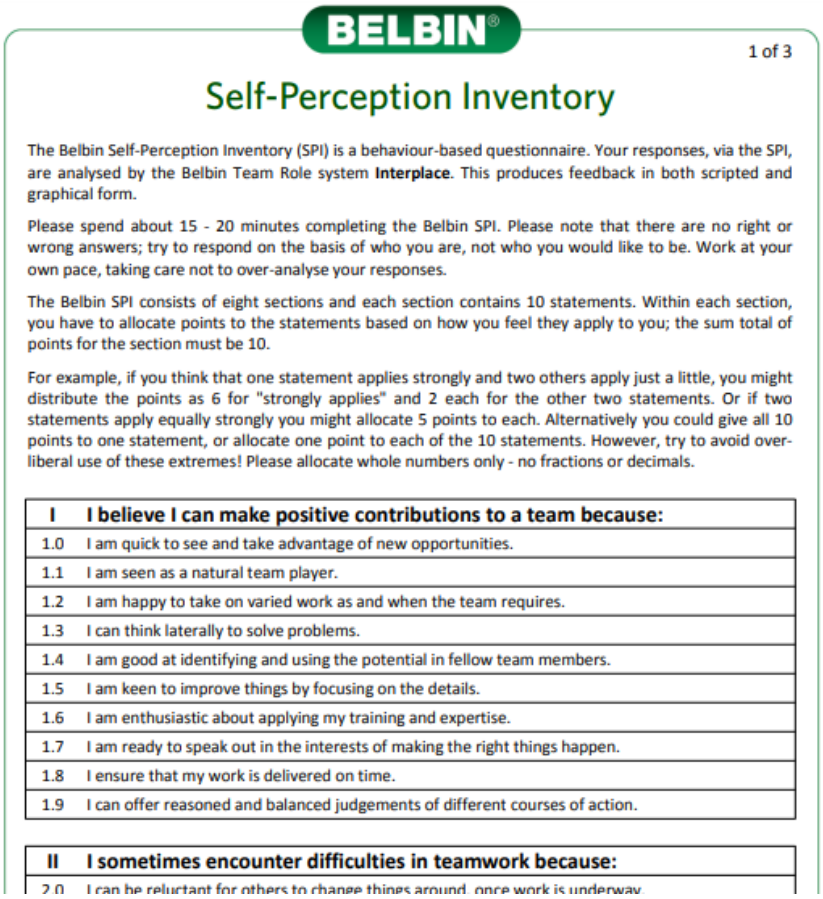
3.1.2. Función objetivo

Se define $f(.)$ como una función que evalúa numéricamente la calidad de los equipos en base a los patrones de comportamiento de sus integrantes. Por tanto, la función objetivo no es más que la suma de las puntuaciones obtenidas por los equipos seleccionados. Esto se debe a que si una variable binaria es 0 entonces ese equipo no se elige y por tanto el producto $f(t_i) \times \delta_i$ es 0. Esta función se pretende maximizar. Para $f(.)$ se puede definir cualquier tipo de función que mida el rendimiento de un equipo o que sirva de aproximación al mismo. En el artículo de referencia se crean dos basadas en las teorías de roles y personalidades comentadas

anteriormente, y se refiere a ellas como “heurísticas” pues solo permiten obtener una aproximación al rendimiento real de los equipos cuya única manera de cuantificar es una vez estos entran en acción. Se proporcionará una descripción de dichas heurísticas:

3.1.2.1. Heurística de Belbin

El Belbin *Team-Role Self Perception Inventory* [43] es una prueba que permite identificar los patrones de comportamiento salientes de un individuo dentro de un equipo. Se trata de un cuestionario con 8 apartados en los que el sujeto debe distribuir 10 puntos para cada uno de los apartados del cuestionario. Dentro de cada apartado, el individuo debe dividir los 10 puntos correspondientes entre una serie de afirmaciones. La forma en que estos se distribuyan (algunos asignan muchos a ciertos enunciados, otros lo hacen de forma más equitativa…) determina unas puntuaciones en valor relativo para los 8 roles disponibles.



BELBIN®

1 of 3

Self-Perception Inventory

The Belbin Self-Perception Inventory (SPI) is a behaviour-based questionnaire. Your responses, via the SPI, are analysed by the Belbin Team Role system **Interplace**. This produces feedback in both scripted and graphical form.

Please spend about 15 - 20 minutes completing the Belbin SPI. Please note that there are no right or wrong answers; try to respond on the basis of who you are, not who you would like to be. Work at your own pace, taking care not to over-analyse your responses.

The Belbin SPI consists of eight sections and each section contains 10 statements. Within each section, you have to allocate points to the statements based on how you feel they apply to you; the sum total of points for the section must be 10.

For example, if you think that one statement applies strongly and two others apply just a little, you might distribute the points as 6 for "strongly applies" and 2 each for the other two statements. Or if two statements apply equally strongly you might allocate 5 points to each. Alternatively you could give all 10 points to one statement, or allocate one point to each of the 10 statements. However, try to avoid over-liberal use of these extremes! Please allocate whole numbers only - no fractions or decimals.

I	**I believe I can make positive contributions to a team because:**
1.0	I am quick to see and take advantage of new opportunities.
1.1	I am seen as a natural team player.
1.2	I am happy to take on varied work as and when the team requires.
1.3	I can think laterally to solve problems.
1.4	I am good at identifying and using the potential in fellow team members.
1.5	I am keen to improve things by focusing on the details.
1.6	I am enthusiastic about applying my training and expertise.
1.7	I am ready to speak out in the interests of making the right things happen.
1.8	I ensure that my work is delivered on time.
1.9	I can offer reasoned and balanced judgements of different courses of action.

II	**I sometimes encounter difficulties in teamwork because:**
2.0	I can be reluctant for others to change things around, once work is underway.

Figura 9. Captura del inventario del Belbin *Team-Role Self Perception Inventory*. [43].

La heurística definida considera que un individuo domina un rol cuando el valor asociado a este rol sea “alto” o “muy alto”, categorizaciones que se pueden observar en la tabla 1. La puntuación requerida para alcanzar este primer nivel puede ser mayor en algunos roles respecto de otros debido a que los primeros se dan de forma más frecuente entre las personas, hecho que Belbin tuvo en cuenta a la hora de diseñar esta tabla.

Rol / calificación	Bajo	Medio	Alto	Muy alto
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 1. Umbrales necesarios para alcanzar cada calificación en los roles de Belbin según el cuestionario de autopercepción.

Si $b_{j,k}$ es la puntuación obtenida por el estudiante j en el rol k, y β_k es el umbral para el cual ésta se considere como "alta", entonces decimos que el equipo adquiere puntuación positiva en ese rol si alguno de los estudiantes que lo forman sobresale en este (puntuación superior al umbral), y el valor total de la heurística será la suma del valor obtenido para cada uno de los roles normalizada por el máximo a alcanzar.

$$f_k(t_i) = \begin{cases} 1 & \exists s_j \in t_i, b_{j,k} \geq \beta_k \\ 0 & en\ otro\ caso \end{cases} \qquad f(t_i) = \frac{1}{8} \times \sum_{k=1}^{8} f_k(t_i)$$

3.1.2.2. Heurística de Myers-Briggs Type-Indicator (MBTI)

La prueba de MBTI obtiene un valor para cada una de las cuatro dimensiones definidas, y el rasgo o personalidad asignada a la persona será uno si este es positivo (por ejemplo, extroversión) u otro si es negativo (introversión). La heurística utilizada es una versión normalizada de otra ya existente, habiendo otras variantes en la literatura.

Para la definición de esta heurística, asumiremos que *k* es una de las cuatro dimensiones establecidas por Myer-Briggs, y k_1 y k_2 son los rasgos opuestos en dicha dimensión. Además, $\gamma_k(s_j, k_l)$ es una función que devuelve un 1 si un estudiante tiene asignado el rasgo k_l de esta dimensión. La heurística aplicada sobre una dimensión y equipo concretos devolverá: 0 si todos los integrantes mantienen el mismo rasgo de personalidad; 1 si al menos uno de ellos presenta un rasgo distinto; 2 en caso contrario. Se busca pues una mayor heterogeneidad en los rasgos obtenidos al otorgar la máxima puntuación al último caso. La puntuación final del equipo será la suma de los valores obtenidos para cada dimensión normalizada por la puntuación máxima a alcanzar:

$$f_k(t_i) = \begin{cases} 0\ si\ \exists k_l, \sum_{s_j \in t_i} \gamma_k(s_j, k_l) = |t_i| \\ 1 \quad si\ \exists k_l, \sum_{s_j \in t_i} \gamma_k(s_j, k_l) = 1 \\ 2 \quad en\ otro\ caso \end{cases} \qquad \mathrm{f(t_i)} = \frac{1}{8} \times \sum_{\mathrm{k=1}}^{4} \mathrm{f_k(t_i)}$$

3.1.3. Restricciones

Tal como se ve, existen 3 restricciones asociadas al problema:

- [*complete partition*] asegura que la suma del número de alumnos de todos los equipos escogidos es igual al tamaño del conjunto de alumnos. Es evidente que si esta suma fuera mayor entonces un alumno estaría en varios equipos, y si fuera menor, entonces alguno de ellos habría quedado fuera en la elección de equipos.
- [*student j in team*] es una familia de restricciones que indica que, para cada alumno, la suma de las variables binarias asociadas a equipos en los que este aparezca sea 1. Dicho de otra manera, se busca que, de los equipos seleccionados en la solución final, una persona solo esté asociada a uno de ellos.
- [*compulsory pair j and k*]. Un "*compulsory pair*" o "par obligatorio" es un par de estudiantes que deben ir en el mismo equipo a elección del tutor por motivos diversos, y $\mathcal{C}$ es el conjunto de todos esos pares. Esta familia de restricciones implica que, para cada par de estos estudiantes, la suma de las variables binarias asociadas a equipos en los que esté alguno de ellos sea 1, lo que se traduce en que la cantidad de equipos seleccionados donde esté alguno sea 1 y, por ende, el mismo equipo para ambos.

3.2. Análisis del marco legal y ético

Aquí se describe brevemente algunas condiciones relativas a los datos utilizados o la propiedad intelectual.

En primer lugar, se ha de remarcar que este trabajo no tiene como objetivo extraer conocimiento o utilizar modelos que tomen decisiones sobre los datos con los que se trabaja, sino que estos datos son un medio para poder medir cuál sería el rendimiento de algoritmos de optimización. Estos datos provienen de alumnos que han hecho, dando un consentimiento, el cuestionario de autopercepción de Belbin o la prueba de MBTI (se detallará su origen en secciones posteriores) y, si bien estos algoritmos serán los que se utilizarán posteriormente para tomar una decisión (i.e, formación de equipos óptima), dicha decisión no depende de cómo funcionen estos algoritmos sino del modelo matemático planteado, algo que ya venía definido en [42]. En cualquiera de los casos, las heurísticas definidas para la resolución de este tipo de problemas se basan en teorías psicológicas que han sido ampliamente estudiadas, por lo que se cree que las soluciones aportadas no darán lugar a la creación de equipos sesgados en base a factores como el sexo o la raza.

Los datos, proporcionados por los tutores, se consideran además como "no personales", pues se ha eliminado cualquier tipo de referencia que permita identificar a los individuos como puede ser su nombre y apellidos, disponiéndose solo de registros identificados con un código y almacenándose para cada uno de ellos el sexo de la persona correspondiente y las puntuaciones en ambas pruebas. En resumen, se experimentará utilizando datos reales anonimizados. En cuanto a estos datos también se ha de aclarar que fueron recogidos en base a un consentimiento informado para el cual los alumnos estuvieron de acuerdo.

Finalmente, habría que comentar algunos aspectos de propiedad intelectual en el software usado o las imágenes del trabajo. El software se describirá en la sección 5.1, aunque por ahora

se podría adelantar que se utiliza el lenguaje de programación Python, que utiliza la licencia PSFL (*Python Software Foundation License*) [44] y que es una licencia de software libre permisiva que, a diferencia de la GPL, no obliga a que los trabajos derivados de esta tengan que ser necesariamente de código abierto. Librerías o *toolkits* de dicho lenguaje como *matplotlib* utilizan licencias basadas en la anterior. Otras como *ORtools* [45] utilizan la licencia Apache, que también es permisiva (y por tanto usar, modificar o redistribuir ese software libremente) y que solamente exige que en el software distribuido se añada un aviso que especifique el uso de código con esta licencia. En cuanto a las imágenes, todas aquellas extraídas de artículos, páginas web, etc. han sido debidamente referenciadas.

4. Diseño de las soluciones

Tal y como se ha introducido en previos apartados, a continuación se aportan diversas soluciones al problema planteado en 3.1. La primera de ellas es la aplicación y comparación de *solvers* de código abierto (apartado 4.1) para dar variedad a la herramienta de formación de equipos, mientras que la segunda es el uso de metaheurísticas (apartado 4.2.) con diferentes configuraciones para encontrar cuáles se adaptan mejor a qué problemas y determinar si realmente suponen grandes mejoras respecto a los métodos exactos. Comenzaremos con estos últimos.

4.1. Algoritmos exactos

Según lo comentado anteriormente, una primera fase del proyecto consiste en establecer una comparación entre distintos *solvers* no comerciales, no solamente para conocer el rendimiento de cada uno de ellos en diferentes instancias del problema, sino también para disponer de las soluciones óptimas (en caso de poder encontrarse) que sirvan como referencia para las posteriores comparaciones realizadas con métodos metaheurísticos. Los *solvers* a utilizar serán tres:

- **SCIP:** se define como "uno de los *solvers* desarrollados en el ámbito académico más rápidos para programación entera mixta y programación entera no lineal mixta" [46].
- **COIN-OR Branch and Cut (CBC):** [47] es un *solver open-source* que permite resolver problemas de programación lineal (LP) y programación entera mixta (MIP) y es una variante de la técnica de ramificación y poda, el cual entre su funcionamiento pone planos de corte para buscar la solución de forma más rápida.
- **CP-SAT:** [48] es un *solver* diseñado para resolver problemas de programación entera que consta de un solver de tipo *Lazy Clause Generation* sobre un *solver* de tipo SAT. *Lazy Clause Generation* es una técnica de búsqueda en programación con restricciones o CP que añade explicación y aprendizaje a un *solver* basado en propagación, que se encarga de ir estrechando el rango que toman las variables de decisión.

Para la resolución del modelo por parte de los *solvers* anteriores se programa directamente el modelo planteado en la Sección 3.1, donde el esquema seguido se verá en la sección 5.3. Estos *solvers* consumen directamente el modelo de optimización matemática para la obtención de una solución óptima. Por ello, en este caso simplemente es necesaria la introducción, mediante programación y automatización, del modelo matemático planteado.

4.2. Diseño del algoritmo genético

La solución desarrollada sigue el esquema general de los algoritmos genéticos, pero presenta una serie de matices distintos como consecuencia de la definición y restricciones del problema. En esta sección nos centraremos en justificar cuál sería una forma adecuada de codificar los individuos de la población según las clasificaciones vistas en la literatura, en

conocer los operadores de selección, cruce, mutación y reemplazamiento que se emplearán y en presentar la estructura final del algoritmo.

4.2.1. Función de *fitness*

La función de fitness corresponde con las heurísticas de Belbin y Myers-Briggs, definidas en el apartado 3.1.2.

4.2.2. Representación de una solución

En CSG, existen numerosas propuestas en la literatura para representar una solución del problema. Los autores de [32] establecen una interesante recopilación de estas representaciones, aunque no necesariamente todas estas representaciones son originarias del problema de CSG. Además, los autores proceden a su comparación a partir del análisis de diversos criterios, como el tamaño del espacio genotípico explorado, la generación o no de soluciones factibles o la cantidad de redundancias que se llegan a generar. El propósito de este apartado consistirá en describir en brevedad la estructura de algunas de estas representaciones, destacar algunas de las ventajas e inconvenientes que presentan y ver cómo se pueden adaptar a las restricciones de nuestro problema en concreto sin quitar crédito a las conclusiones extraídas en el mencionado artículo.

1. Codificación basada en filas (Row-based encoding): se trata de una representación binaria donde se tienen tantos elementos como posibles coaliciones y un 1 indica su elección en la estructura colectiva, como se ve en la figura 10. Si bien resulta la más sencilla de entender y tiene su parecido con el funcionamiento del modelo exacto de nuestro problema, su tamaño en memoria puede resultar prohibitivo, ya que el tamaño en memoria de una solución crece de forma exponencial con el número de alumnos. Esto es debido a que se tienen en cuenta todas las combinaciones de alumnos dentro de los tamaños establecidos. Además, se generan fácilmente soluciones no factibles debido a que o bien se podrían escoger grupos distintos en los cuales aparezca un mismo alumno o bien se podría hacer una selección de grupos de forma que no se escojan a todos los alumnos o se escojan individuos de más. Casos anteriores se añaden a las restricciones del problema de que alumnos podrían ir en un mismo grupo a elección del tutor o deberían estar separados por cualquier razón, aunque estas últimas son restricciones que afectan de igual manera al resto de codificaciones a comentar. Por todo ello, otras representaciones pueden ser mejores.

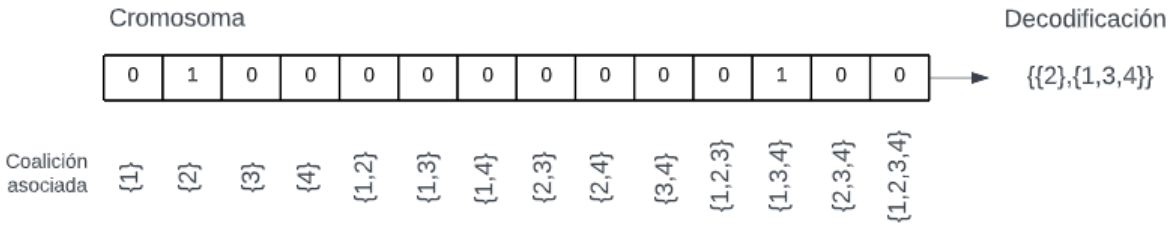


Figura 10. Ejemplo de Row-based encoding y su decodificación en problema de 4 agentes. Elaboración propia

2. Codificación basada en columnas (Column-based encoding): se tienen tantos elementos como agentes presentes en el problema, y en cada posición del vector que los contiene se indica a qué coalición pertenece cada uno de ellos. Se trata de una mejora respecto a la anterior codificación en cuanto al tamaño en memoria de una solución, ya que es lineal con el número de alumnos. Ésta (y las siguientes) además siempre generará soluciones factibles (al menos en el problema genérico de formación de coaliciones) y aporta una mejora también en el tamaño del espacio genotípico por el hecho de que existen en el caso anterior muchas decodificaciones inválidas de los cromosomas generados. No obstante, se puede generar numerosas soluciones redundantes, lo cual se puede observar en el ejemplo de la figura 11, donde a pesar de que a los agentes se les asigna un identificador de grupo distinto, en realidad la decodificación de ambas soluciones resulta en la misma estructura colectiva. Además, cabe comentar que esta representación no garantiza que se cumpla la condición relacionada con el tamaño de los equipos, si bien hace cumplir aquella que indica que un agente no esté en varias coaliciones distintas.

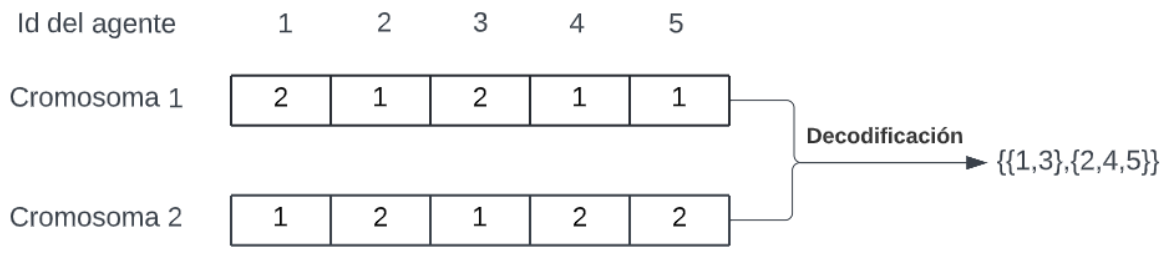


Figura 11. Representación basada en columnas, donde cromosomas distintos se decodifican en un mismo fenotipo. Elaboración propia.

3. Codificación basada en el orden (Order-Based Encoding): se introduce por primera vez en el año 2000 por Sen y Dutta [34], y consiste en una representación de permutación de 2*n* - 1 elementos, longitud que viene dada por los *n* agentes del problema y las *n-1* divisiones que se podrían hacer a lo sumo de dichos agentes. Un ejemplo de esta está en la figura 12. Dicho tamaño de representación se podría afinar un poco más para nuestro caso dada la restricción de tamaño mínimo de equipo l, obteniendo una con $n + \frac{n}{l} - 1$ elementos. Por ejemplo, si se deseara obtener grupos cuyo tamaño esté entre 3 y 5 alumnos en una clase de 60, como mucho se podrían formar 20 grupos de 3 y, por tanto, hacer 19 divisiones sobre dicha clase. Este tipo de codificación permite el uso de operadores sencillos de permutación, obteniendo siempre soluciones que son factibles (en el caso general). En el caso de que se desee obtener coaliciones de un cierto tamaño, se puede utilizar una función de penalización que guíe al algoritmo puntuando aquellas coaliciones buscadas con mayor puntuación tal y como indica el artículo que introduce esta nueva representación. Para nuestro caso las soluciones no factibles se encontrarán en aquellos casos donde los marcadores estén tan cercanos entre sí que delimiten grupos de tamaño menor al mínimo especificado, o grupos demasiado grandes. Además, las redundancias aquí se encuentran en el hecho de que cualquier marcador es válido para separar los grupos, en el orden en que pueden aparecer los individuos entre cada par de marcadores, y

en la posición que cada coalición cobra en el vector. En cualquier caso, el tamaño en memoria de la representación de una solución sigue siendo lineal con respecto al número de alumnos.

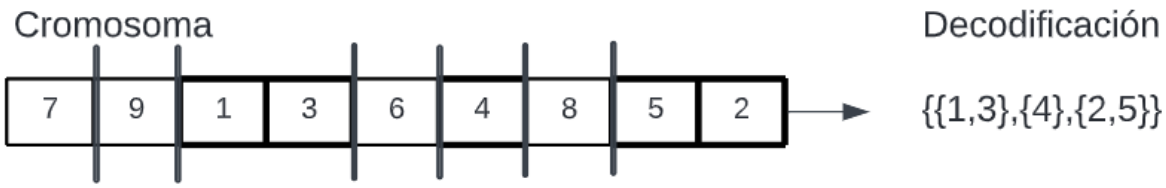


Figura 12. Order-Based Encoding y decodificación para un ejemplo con 5 agentes. Elaboración propia.

4. Codificación basada en el orden con llave de bits (Order-Based Bit Key / OBBK): es una representación dual o híbrida que surge como mejora de las anteriores en [32]. Consta de un vector codificado según el orden (caso anterior) cuya longitud es el número de agentes, y de una llave de bits en la que un 1 indica que se establece una partición del vector anterior por la posición especificada. Esto se puede observar gráficamente mediante el ejemplo de la figura 13:

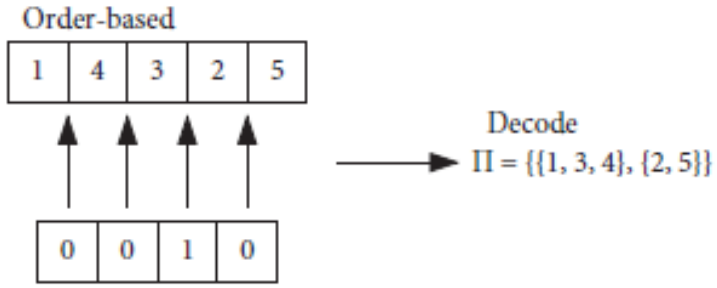


Figura 13. Ejemplo de representación y decodificación de Order-Based Bit Key (5 agentes). [32].

A pesar de su similitud con el anterior caso, esta representación evita una serie de redundancias y permite un uso más controlado de los operadores de cruce y mutación, pues el vector basado en el orden es de menor tamaño (solamente incluye a los agentes, no a los marcadores). Además, se permite el uso de operadores simples como mutación por bit en el caso de la llave o máscara de bits, tal y como menciona y demuestra mediante expresiones matemáticas el señalado artículo.

Existen también versiones que utilizan números reales para los casos 2 y el 3, denominados respectivamente *Frac-row based encoding* y *Random-key encoding*. Básicamente se juega con valores reales entre 0 y 1 y mediante ciertas operaciones y/o elementos adicionales se vuelve a la correspondiente representación entera. Los principales riesgos adicionales de este tipo de representaciones se encuentran en el gran número de redundancias que pueden llegar a presentar, pues el mismo vector de números enteros se puede haber conseguido a partir de muchas representaciones reales distintas. Pondremos el ejemplo de la decodificación del Random-key. La figura 14 extraída del artículo anterior muestra el proceso de decodificación para un problema con 5 agentes. El vector tiene 9 elementos dado que se incluyen los 4

separadores de coaliciones a los 5 agentes. Los valores entre 0 y 1 son ordenados crecientemente, y las posiciones en esta ordenación indican el valor entero que tendría el elemento asociado. A partir de aquí el resto del proceso de decodificación se resuelve como se haría con un vector basado en el orden. Las redundancias que se mencionan se pueden observar en el hecho de que, con cualquier valor real menor a 0.38 (valor del segundo elemento en la ordenación) el primer elemento actual (que tiene valor 0.23) seguiría siendo el primero, y esto se puede replicar para todos los elementos de dicho vector. Para este caso concreto no solamente esto se trata de un problema, sino también el hecho de tener que ordenar el vector para obtener finalmente el fenotipo, ya que es una operación que puede resultar costosa.

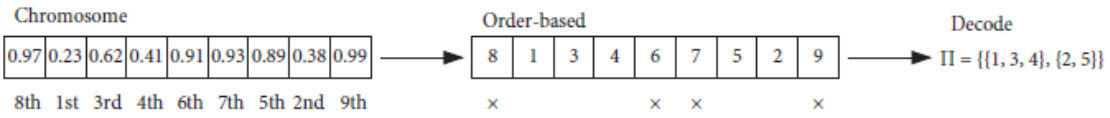


Figura 14. Proceso de decodificación para un ejemplo de 5 agentes representado con Random-key encoding. [32].

Cabe destacar que también existen variantes del problema en cuestión, y que a menudo se definen estructuras adaptadas a los requisitos de dichos problemas. Un ejemplo son los problemas de *task allocation*, donde se debe formar aquellas coaliciones de agentes de forma que desempeñen mejor su trabajo en un conjunto de tareas. En [49] se definen codificaciones bidimensionales de bits para indicar qué robots deberían asignarse a cada tarea definida para que se maximice la eficiencia en su realización, tal como se muestra en la figura 15; otros problemas de la misma naturaleza son las actividades encomendadas en comunidades de ancianos [50] o la gestión de redes a gran escala de *Unmaned Aerial Vehicles* (*UAVs*) [51], entre otros.

Robot	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9	R_{10}
T_1	1	0	0	1	1	0	0	1	0	1
T_2	0	0	0	0	0	1	0	0	1	0
T_3	0	1	0	0	0	0	0	0	0	0
T_4	0	0	1	0	0	0	1	0	0	0

Figura 15. Cromosoma bidimensional binario que muestra la coalición de robots necesarios para cada tarea. [49].

Tras el conocimiento y la comparación de las anteriores representaciones se decide finalmente el uso de OBBK dado que, según el artículo de referencia, supera por lo general al resto de representaciones y ofrece resultados bastante robustos para problemas con un número de agentes elevado (más de 25). Concretamente, para medir la calidad de las soluciones obtenidas en los distintos problemas de formación de coaliciones (PFG, CFG, irregulares, regulares) se utiliza el GAP relativo en dicho artículo, que tiene la expresión $GAP = \frac{v-v*}{v*}100\%$, donde $v*$ es la solución óptima para el problema y v la solución obtenida con el algoritmo genético. Los experimentos diseñados implican el uso de las mismas poblaciones iniciales para cada representación, así como operadores clásicos, y finalmente los autores concluyen que OBBK es la mejor opción al conseguir el mejor valor de GAP para gran parte de

las instancias generadas, el cual suele estar por debajo del 5% al menos para un número de agentes inferior a 20.

Una facilidad que se manifiesta con el uso de la representación OBBK es el uso de operadores sencillos de permutación y de intercambio de bits, si bien será necesario definir al menos el doble de operadores que en un genético con representación única. Esto se justifica por el hecho de que una solución depende no solamente de explorar las distintas formas en que se pueden organizar los alumnos hablando de nuestro problema en concreto (esto es, la posición que cobran dentro del vector de permutación y junto a qué otros alumnos están cercanos) sino también en explorar los cambios en el tamaño de los grupos y el número de grupos con determinado tamaño. En resumen, la información conjunta sobre la distribución secuencial de los alumnos y las divisiones que se deben efectuar es la que ayudará en la búsqueda de la solución óptima, por lo que se requerirá definir operadores de cruce y mutación que actúen tanto para el vector de alumnos como el que indica las particiones de estos en grupos disjuntos.

Otra ventaja que se explotará es la posibilidad en la reducción de redundancias y en la generación de soluciones no factibles producidas por las máscaras de bits.

- Respecto al primer caso, por ejemplo, los cromosomas [0,1,0,0] y [0,0,1,0] refieren en ambos casos a la creación de equipos de 2 y 3 alumnos (independientemente de quiénes sean), por lo que lo ideal sería utilizar una única secuencia de bits que refiriera esta estructura de coalición en concreto. Este número de redundancias crecerá cuanto mayor sea el tamaño de la clase y más diversidad haya en el tamaño de grupos, por lo que se propone limitar las máscaras utilizadas a una por tipo de estructura de coalición. Esto supone una mejora, pero también implica que el peso de situar a los alumnos en los equipos adecuados recae enteramente en los métodos que actúan sobre la parte de permutación.
- Respecto al segundo, puede darse que el algoritmo genético genere máscaras que no cumplen con las restricciones del problema y en consecuencia equipos inválidos. Para un problema de 6 agentes, por ejemplo, crear soluciones con la máscara [0,1,0,1,0] no sería aceptable si se busca un mínimo tamaño de equipo de 3, ya que dichas soluciones implicarían la formación de 3 equipos con 2 integrantes cada uno.

La combinación de ambas problemáticas deriva en dos soluciones que se han diseñado de forma especial para la resolución de este tipo de problemas. La primera de ellas es tener una referencia de las máscaras de bits que se ajustan a las restricciones impuestas y, a su vez, corresponden a un único fenotipo (hablando en términos de los tamaños de equipos creados). Este conjunto serán las “máscaras válidas” del problema, y la idea es que de forma previa a la ejecución del algoritmo genético se haga un cómputo de estas según la información disponible (número de alumnos, tamaño mínimo y máximo de equipos…). Definir estas máscaras válidas no tiene sentido si luego no se liga su uso al proceso de generación de máscaras en el algoritmo genético, y por tanto la segunda solución se basa en la creación de operadores personalizados que actúen sobre dichas máscaras, y que se comentarán a continuación. Estas propuestas por lo general agilizarán el proceso de búsqueda, ya que solo se transitará entre divisiones de alumnos que resultan factibles.

4.2.3. Operadores de selección

El operador encargado de seleccionar aquellos individuos que participarán en el cruce para la creación de nuevos individuos será la *selección por ruleta con rango por ránking lineal.*

Su funcionamiento sigue el siguiente esquema:

- Ordenación de los individuos de la población en función del *fitness*, de peor a mejor valor. En nuestro caso, la ordenación es de forma creciente al estar ante problemas de maximización.
- Se guarda la posición que ocupa cada individuo en este vector ordenado (comenzando en 1), y se calcula el ránking mediante la siguiente fórmula:

$$Ránking(Pos) = 2 - SP + \left(2 \cdot (SP - 1) \cdot \frac{Pos - 1}{n - 1}\right)$$

  La anterior fórmula depende del hiperparámetro SP, tomando valores $1 \leq SP \leq 2$. Éste se encarga de controlar la presión en la selección, de forma que cuanto mayor sea el valor de éste, mayor probabilidad tendrán los mejores individuos en ser seleccionados, mientras que un valor igual al mínimo equivaldría a realizar una selección de forma uniforme de los individuos.

- Los individuos son seleccionados con probabilidad $\mathrm{P(Pos)} = \frac{\mathrm{Ránking(Pos)}}{\sum_{i=1}^{n} \mathrm{Ránking(i)}}$.

Mediante el uso de este tipo de selección se busca evitar la causa de una convergencia prematura[3], ya que la probabilidad de selección de los padres no depende directamente del valor de fitness (como ocurriría en la selección por ruleta tradicional) sino del rango obtenido, de forma que independientemente de dicho valor el mejor individuo siempre se escogerá con la misma probabilidad.

4.2.4. Operadores de cruce

Esta sección, al igual que la relativa a los operadores de mutación, se divide en dos subsecciones, una para comentar los operadores utilizados para la representación de permutación y otra para las máscaras de bits. En este caso, son dos tipos de cruce los definidos para la primera y uno para la segunda. El uso de varios operadores de cruce o mutación para una misma parte del genotipo tiene que ver con la experimentación que se describirá en el apartado 6.3.

4.2.4.1. Cruces para permutación

Cycle Crossover (CX) es un operador de cruce definido por Oliver et al [52] para abordar problemas diferentes al uso de representaciones de cromosomas de tipo binario como el viajante

[3] La convergencia prematura refiere a la rápida convergencia de un algoritmo hacia un valor extremo dentro del espacio de soluciones que puede no ser el óptimo.

de comercio[4] y que logra preservar el orden absoluto de alrededor de la mitad de los elementos de ambos padres de media.

Su funcionamiento está basado en la localización de "ciclos" en los distintos padres. Tras la creación de un hijo sin valores asociados, un ciclo comienza con la selección de un elemento cualquiera de ambos padres cuya posición es la primera que está vacía en el hijo. Dicho elemento se copia en el hijo en la misma posición, y el siguiente paso buscaría este elemento en el otro padre, obteniendo una nueva posición. Se escoge ahora un valor cualquiera de los padres dada esa posición, y puede ocurrir: si este no se encuentra en el hijo, se copia y repite el proceso; si se encuentra, verificamos que el otro no se encuentre ya, en cuyo caso volvemos al caso anterior; si ambos se encuentran en el hijo, el ciclo termina. Si un ciclo termina pero el hijo no está completo aún, se busca la siguiente posición vacía en el hijo y se trata de crear un nuevo ciclo, repitiendo esto hasta completarlo finalmente, siendo de forma análoga para el segundo hijo.

La siguiente figura (figura 16) muestra un ejemplo del funcionamiento de este método de cruce:

[4] El viajante de comercio (*Travelling Salesman Problem*) es un problema muy conocido que busca encontrar el trayecto más corto entre un conjunto de localizaciones sin pasar dos veces por una misma (a excepción de la de origen).

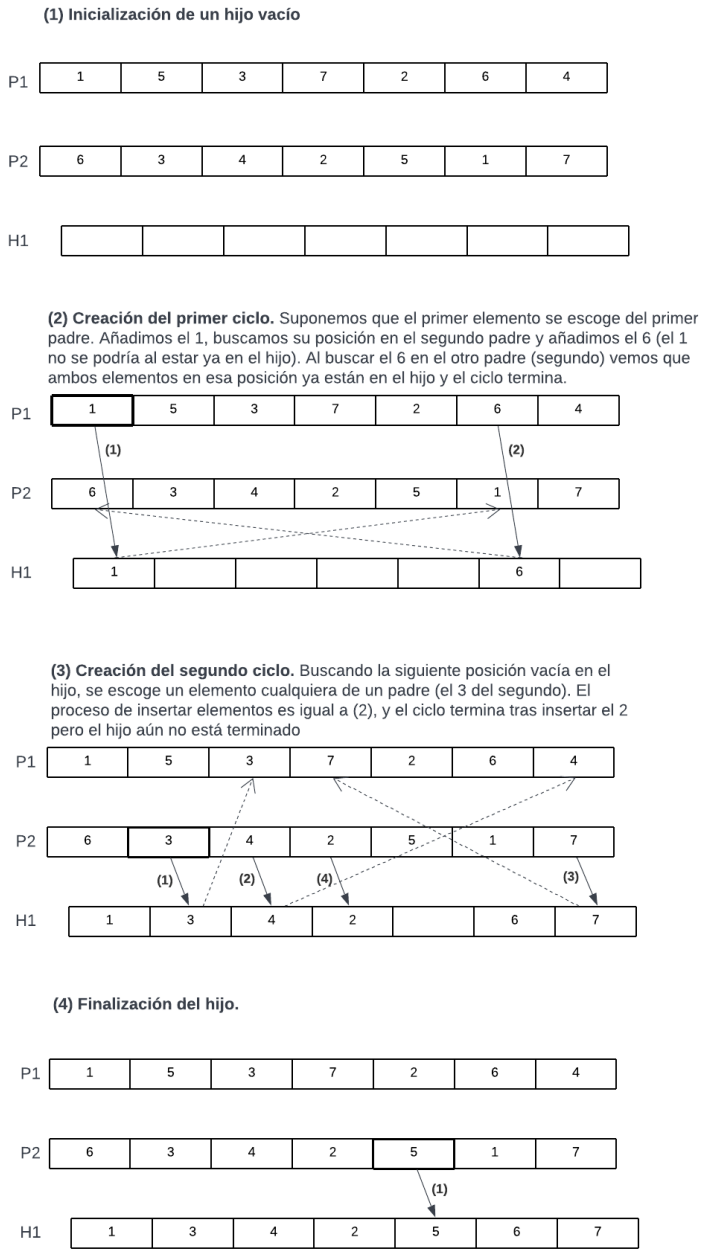


Figura 16. Traza de ejemplo de Cycle Crossover. Elaboración propia.

Non-Wrapping Order Crossover (NWOX) [53] es un tipo de cruce diseñado para el problema de la programación de tardanzas con ponderación y configuraciones que dependen de la secuencia[5], surgiendo como alternativa del clásico *Ordered Crossover (OX)* que mantiene las posiciones absolutas de los valores en las permutaciones que constituyen los padres. La experimentación con distintas configuraciones y comparaciones con *solvers* "estado del arte" vistas en el mencionado artículo resulta satisfactoria, de hecho, se puede encontrar aplicado a otros problemas como el viajante del comercio. Por ejemplo, en [54] se concluye que aquellos

[5] *Weighted Tardiness Scheduling Problem with Sequence-Dependent Setups*, se busca secuenciar un conjunto de trabajos con sus respectivos tiempos de procesamiento y ponderaciones en el trabajo de forma que se minimice la tardanza total dedicada a estos.

operadores basados en "crear y rellenar agujeros" como el NWOX suelen obtener mejores resultados en comparación a otros como *Partially Mapped Crossover*.

El método genera en primer lugar los hijos como copias de los padres, además de dos índices de forma aleatoria *i, j* tal que $i \leq j$. Del primer hijo se eliminan aquellos elementos que caen dentro del bloque de valores comprendidos entre las posiciones *i, j* (se convierten en "agujeros"), y todos aquellos valores no vacíos son desplazados hacia la izquierda del vector hasta que se tengan todos ellos de forma contigua. En caso de haber posiciones no vacías dentro del bloque entre *i, j*, estas se desplazan hacia la derecha hasta que todos los "agujeros" se reúnan dentro de dicho bloque, el cual finalmente es reemplazado por los valores que ocupan las mismas posiciones en el segundo padre. El segundo hijo se crea de forma análoga. Un ejemplo de funcionamiento de este método de cruce puede observarse en la figura 17.

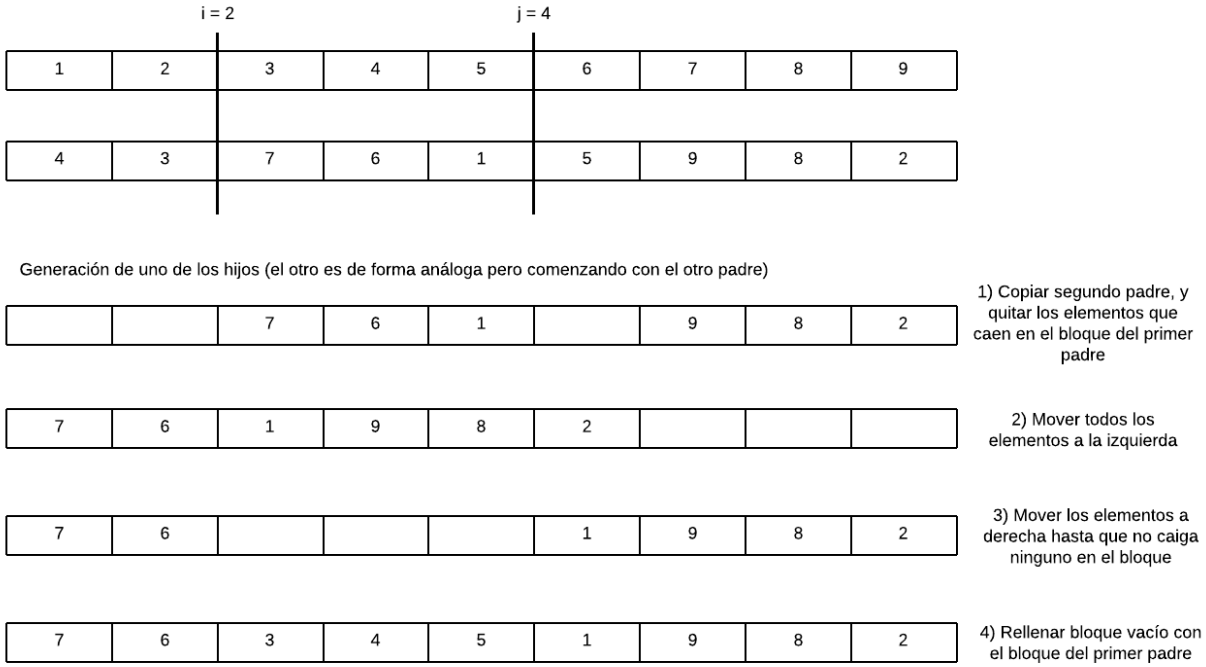


Figura 17. Traza para la generación de un hijo con NWOX. Elaboración propia.

4.2.4.2. Cruces para la llave de bits

Finalmente, la operación de cruce aplicada sobre la máscara de bits se podría definir como **"herencia"**. Numerosos operadores de cruce pueden ser aplicados en vectores de bits, tales como cruce por un punto o multipunto, cruce uniforme, entre otros. El problema de su aplicación atendiendo a este problema en específico implica la generación de máscaras que, junto con cualquier permutación de estudiantes, daría lugar a un fenotipo que no es el deseado.

La opción más sencilla en una primera versión de este algoritmo genético ha sido no aplicar ningún tipo de cruce, lo que se resume en que los hijos heredarán la misma llave de bits que tenían uno de sus padres, solo que no necesariamente el "primer padre" cederá su máscara al "primer hijo", sino que este último podría obtener la máscara del segundo padre y de forma análoga para el segundo hijo.

En resumen, con este operador podría asociarse una estructura de coaliciones de tamaño distinto ya existente a otro conjunto de estudiantes, pero no se produce una modificación del material genético en estos cromosomas binarios. Es necesario que se dé esto último para poder explorar estructuras de coaliciones de diferentes tamaños, y si bien se podría haber definido un cruce algo más sofisticado, se ha decidido en la solución que esta responsabilidad venga a través de los operadores de mutación de máscaras de bits, como se verá en la sección 4.2.5.2.

4.2.5. Operadores de mutación

Al igual que en la sección anterior, esta se divide en dos partes, la primera en relación con los métodos que trabajan sobre la parte de permutación de la representación y la segunda con los que lo hacen sobre el vector de bits. En ambos casos se introducen 2 operadores distintos que serán probados durante la experimentación con el algoritmo genético.

4.2.5.1. Mutaciones para permutación

Para este primer caso destacamos el uso de los siguientes dos métodos:

Simple swap es una forma bastante utilizada en algoritmos genéticos para mutar un cromosoma de cualquier tipo, quizás debido a su sencillez en la implementación, la cual se basa únicamente en intercambiar dos valores escogidos aleatoriamente del vector en cuestión.

Reverse Sequence Mutation (RSM) es mostrado en [55] como parte de un listado de operadores de mutación a comparar para la resolución del problema del viajante de comercio, donde se concluye que, tras unos experimentos realizados sobre 50 poblaciones distintas de una misma instancia del problema, es uno de los más eficientes y de los que mejor solución final reporta.

Inicialmente se escogen dos posiciones aleatorias del vector i y j tal que $0 \leq i \leq j \leq N-1$. Los elementos asociados a esas posiciones son intercambiados, y se desplaza el índice i una posición a la derecha y el j una a la izquierda. Este intercambio y reasignación de posiciones se produce siempre y cuando $i < j$. La siguiente figura (figura 18) detalla un ejemplo de este funcionamiento:

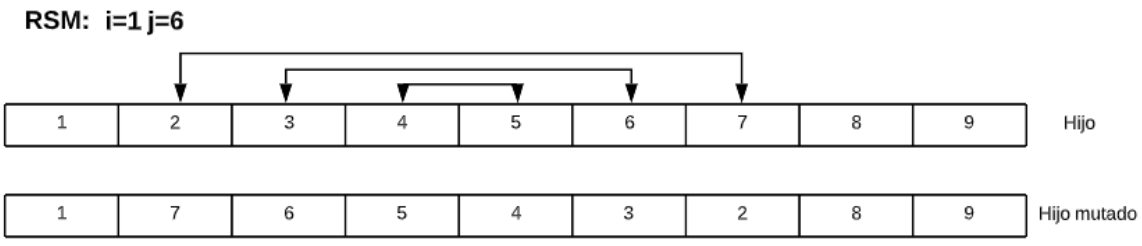


Figura 18. Ejemplo de funcionamiento de RSM. Elaboración propia.

4.2.5.2. Mutaciones para máscara de bits

Las mutaciones para la máscara o llave de bits son, tal como se había comentado anteriormente, creadas específicamente para el problema en cuestión. Si bien se podrían utilizar operadores específicos para cadenas de bits como mutación uniforme, que invierte cada bit del vector con una determinada probabilidad, en gran parte de las ocasiones esta operación generaría una división no factible atendiendo a las restricciones como la del tamaño de los grupos, razón por la cual se propone la definición de operadores que consideren únicamente divisiones válidas y que tienen comportamientos muy similares a lo que se conoce como "mutación". Son dos las nuevas contribuciones ofrecidas con este TFG, las cuales se listan más adelante:

Mutación selectiva de divisiones es el nombre que hemos dado a uno de los operadores de mutación aplicado sobre la máscara de bits, y que se define a medida para este algoritmo. Básicamente consiste en seleccionar, con una cierta probabilidad, la máscara "más parecida" a la que se está considerando, y las posibles opciones a seleccionar son únicamente las máscaras válidas del problema. El proceso de selección es similar a una selección lineal por ruleta con rango por lo que todas las máscaras son posibles para la elección. Un ejemplo de este operador se da en la figura 19:

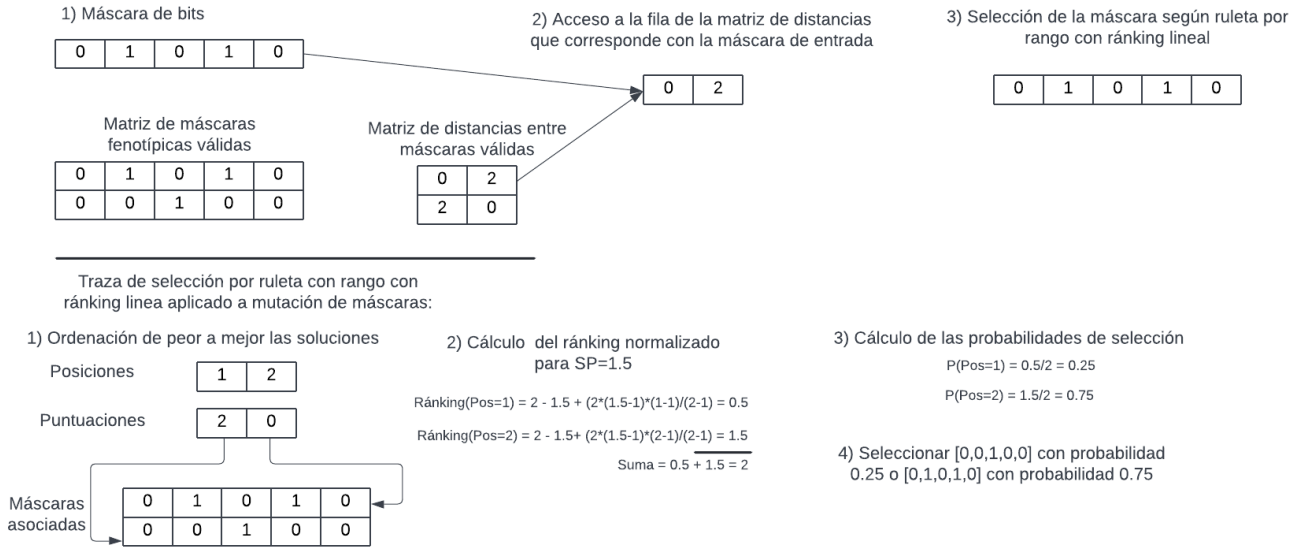


Figura 19. Traza de mutación selectiva de divisiones para ejemplo con 6 agentes. Elaboración propia.

El término de similitud entre máscaras viene dado en este caso por la distancia de edición[6] entre máscaras, y como se ve en el ejemplo superior, aquellas máscaras con menor distancia de edición tendrán una mayor probabilidad de ser escogidas como resultado de la mutación.

Siguiendo el hilo sobre la motivación de la creación del operador anterior, hemos definido también el llamado ***Randomised Mask Mutation***. En éste, se utiliza una *bitwise mutation* o mutación por bit, que consiste en invertir cada bit del vector con una determinada probabilidad, y el resultado se intercambia por la máscara más similar de entre aquellas factibles. En caso de

[6] Distancia de edición: menor número de operaciones de edición (errores) con las que se puede transformar una cadena de texto en otra.

que el mismo mejor valor de similitud lo compartan varias máscaras, se devolverá cualquiera de ellas con la misma probabilidad. Una traza de este tipo de mutación se observa en la figura 20:

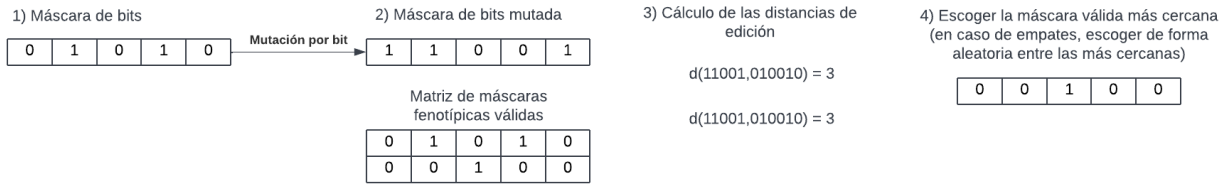


Figura 20. Traza de Randomised Mask Mutation para ejemplo con 6 agentes. Elaboración propia.

4.2.6. Operador de reemplazamiento

Para mantener la población estable en cada iteración se decide utilizar el *reemplazamiento elitista*. Este consiste en seleccionar, como población para la siguiente generación, los |P| mejores individuos del conjunto de soluciones formado por la población y los hijos nacidos en la generación actual. Mediante este método se logra mantener la mejor solución encontrada hasta el momento siempre para futuras iteraciones del algoritmo, pero se puede correr el riesgo de llegar a la convergencia prematura.

4.2.7. Esquema específico del algoritmo genético

Según lo mencionado en previos apartados, el uso de una codificación dual deriva en tener que considerar un operador de cruce y mutación para cada una de las partes. Cabe recalcar que, a pesar de esta división en la representación, el cruce y mutación son a nivel de solución. Esto es, cuando se decide aplicar la mutación, por ejemplo, tanto la representación de los alumnos como la que indica las divisiones cambian a la vez según los operadores correspondientes usados (de igual manera ocurre con el cruce), si bien también podría ser interesante asignar probabilidades de cruce o mutación distintas y encontrar las que mejor resuelven el problema. Esto último se deja como trabajo a futuro, lo que permitiría explorar si realmente puede encontrarse una mejoría al "personalizar" las probabilidades de cruce y mutación según la representación (de permutación frente a binaria).

Aquellos operadores definidos para la máscara de bits son específicos del problema en cuestión, con la intención de recibir siempre una partición que resulte factible. No obstante, estos últimos no garantizan que se cumplan el resto de las restricciones del problema; esto es, podrían encontrarse en un mismo equipo alumnos que no deberían ir juntos o no cumplirse las restricciones *compulsory_pair_j_and_k* del modelo matemático. Una primera idea consistía en reparar estas soluciones, aunque esto implicaría extraer y/o incluir alumnos de determinadas coaliciones, lo cual no parece algo trivial (y posiblemente eficiente). Una alternativa dada en este caso constaría de 2 partes:

- Típicamente 2 padres generan 2 hijos. Si alguno de ellos no resulta factible, se repite el proceso de cruce/mutación para los mismos padres hasta un máximo de 5 veces,

guardando los hijos factibles y parando cuando se tengan 2 soluciones válidas. Todos los hijos no válidos en estos intentos son registrados, y si por cualquier razón no hay dos que cumplan las restricciones tras los 5 intentos, entonces se completa el resultado del cruce/mutación con alguno de ellos (escogidos de forma aleatoria) hasta obtener 2 hijos.

- Modificación de la función de fitness para que se comporte como una función de penalización. Si lo anterior no resuelve el problema y termina almacenándose alguna solución no factible (consecuencia de no obtener soluciones factibles tras los 5 intentos), la evaluación de estos individuos en los métodos de reemplazamiento de la población y/o selección de padres será la peor posible (en nuestro caso un valor de 0, por ejemplo).

Con esto, lo que se busca es que se genere el máximo número de soluciones factibles posible de forma que eventualmente la población evolucione en base al resto de individuos que sí cumplen las condiciones. No obstante, como se verá más adelante, a pesar de que se haya implementado lo anterior, la experimentación no tendrá en cuenta el uso de estudiantes que sean incompatibles o que deban ir en conjunto de manera obligada, lo que implica que no habrá opción de generación de soluciones no factibles en este estudio. Más bien, esta sección pretende satisfacer de alguna manera el resto de las condiciones que se planteaban en el apartado del análisis del problema, para que en un caso real el algoritmo no llegue a ofrecer soluciones incoherentes según los parámetros que un usuario quiera especificar, aunque un trabajo a futuro podría comprender el estudio de esta aproximación para resolver soluciones no factibles con otras que puedan implicar la reparación de estas, entre otras.

Con todas las consideraciones indicadas, el esquema del algoritmo genético se puede ver como sigue (figura 21):

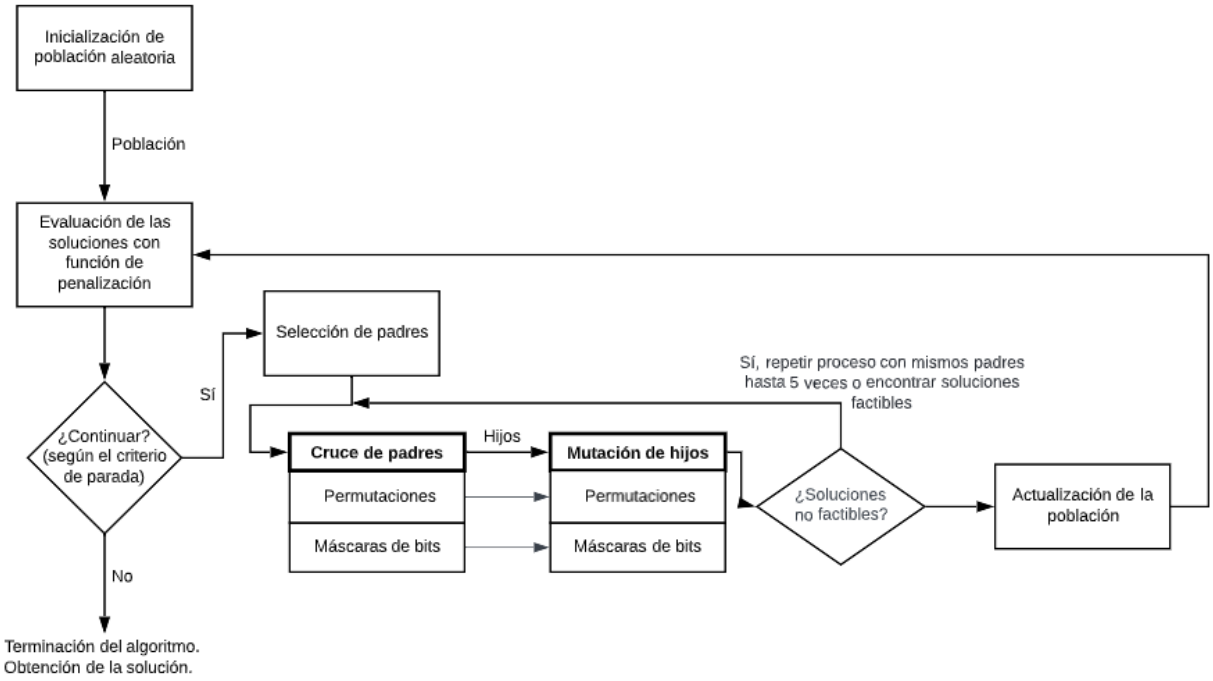


Figura 21. Esquema específico del algoritmo genético diseñado. Elaboración propia

El funcionamiento del anterior esquema se puede resumir de la siguiente manera: tras inicializar una población de soluciones de forma aleatoria, estas se evalúan, lo que permite saber si se debe continuar o no con la ejecución del algoritmo. En un problema donde se usen restricciones de inclusión/exclusión de alumnos es posible que se generen soluciones no factibles (tanto tras esta inicialización como tras el cruce y mutación) las cuales serán penalizadas. En caso de continuar con la ejecución del algoritmo, se hace la selección por ruleta con rango lineal de los padres, el cruce de estos y la mutación de hijos según las probabilidades de cruce y mutación (en la parte de permutación y la de bits de la representación) y se repite este proceso hasta 5 veces en caso de no encontrarse a 2 soluciones factibles. Finalmente se aplica el reemplazamiento elitista y evalúa la población, volviendo al ciclo inicial.

5. Desarrollo de las soluciones

En esta sección se indica cómo se ha implementado cada una de las soluciones descritas en el cuarto apartado. 5.1 habla sobre el lenguaje de programación empleado, así como algunas librerías básicas; a partir de este se comienza por hablar de algunos elementos comunes para todas las soluciones propuestas (5.2) lo que deriva en introducir la programación de los modelos exactos (5.3) así como la del algoritmo genético, la cual se desglosa según los operadores definidos.

La estructuración de este punto tiene que ver con cómo se ha dispuesto el código del trabajo. De este modo, y como puede verse en la figura 22, existe un directorio padre que contiene los scripts de los que dependen todas las soluciones. Éste contiene 3 directorios y un script con funciones que son comunes al funcionamiento de todas las soluciones. En la primera carpeta se encontraría de forma aislada los datos anonimizados de los alumnos de los que se basará la generación de instancias del problema, las cuales también se almacenan aquí. El segundo (*exact_model*) contiene dos ficheros de código con la implementación de los modelos exactos y el diseño de la experimentación, así como otro directorio en el que se guardan los resultados de la anterior y se realiza el análisis estadístico para comparar el rendimiento de los distintos *solvers*. El tercer y último subdirectorio (*genetic_algorithm*) contiene la programación del algoritmo genético y el diseño de los experimentos asociados a evaluar su rendimiento en el problema, además de los análisis estadísticos necesarios y otros elementos como directorios para almacenar las poblaciones generadas aleatoriamente, los resultados con los conjuntos de entrenamiento o test y ficheros para la definición de las configuraciones usadas. Detalles de la experimentación se verán en el punto 6 de la memoria.

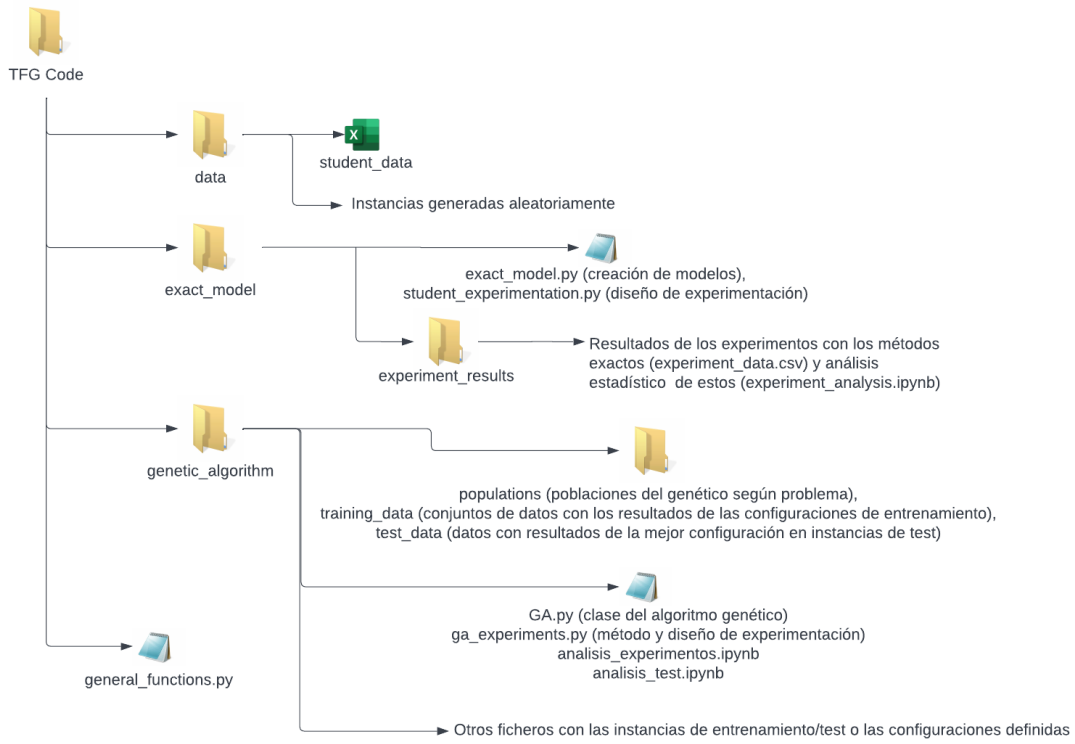


Figura 22. Estructuración del código desarrollado. Elaboración propia.

5.1. Software utilizado

Para la implementación de todas las soluciones mencionadas se ha utilizado el lenguaje de programación Python, lenguaje interpretado multiparadigma con una escritura de código muy parecida al lenguaje natural. Además de las funcionalidades básicas que éste ofrece, se utilizan numerosas librerías, de entre las que podemos destacar: *numpy*, librería *open source* que permite la creación de *arrays* de tantas dimensiones a desear y que dispone de infinidad de funciones matemáticas para cálculos matriciales o generación de números y muestras aleatorias entre otros; *itertools*, cuya importancia reside en poder iterar eficientemente sobre elementos procedentes de problemas combinatorios (permutaciones, combinaciones con o sin repetición…); *matplotlib* y *seaborn* para todos los tipos de visualización de datos necesarios; y en general otro tipo de librerías que se introducirán en apartados en específico.

5.2. Elementos generales

Todas las propuestas del proyecto tienen una parte en común, que consiste en la resolución de instancias del problema de formación de equipos. Esto implica definir funciones que permitan la generación y lectura de instancias aleatorias del mismo dado un conjunto global de estudiantes, además de aquellas que produzcan la evaluación de los equipos según las heurísticas comentadas en 3.1.2. Para el primer caso, se define las funciones *generate_random_selection()*, que crea un conjunto de clases de alumnos seleccionados aleatoriamente sin reemplazamiento y los guarda en un fichero de texto, y *load_student_selection()*, que permite la carga del anterior fichero; para el segundo caso, primero se define *get_scores_subset_students()*, que devuelve un diccionario cuyas claves son los roles/personalidades y los valores son listas que incluyen los valores asociados para cada alumno, estructura que es usada por *belbin_score()* y *mbti_score()* para calcular la puntuación de un determinado equipo que se le pase como argumento, la cual se calcula mediante la iteración sobre los alumnos de dicho equipo y la comparación de sus puntuaciones obtenidas. La decisión de guardar en un diccionario todas las puntuaciones se hace a modo de no acceder constantemente a la tabla de datos de los estudiantes, lo cual sería más costoso en tiempo.

5.3. Métodos exactos

El uso de métodos exactos se ha realizado mediante la librería de Google ORtools [45], desarrollada en C++, aunque se puede llamar desde otros lenguajes de programación como Python. Es un proyecto de software libre que ofrece una interfaz común a *solvers* tanto comerciales (CPLEX, GUROBI) como no comerciales (SCIP, GLOP…) para la resolución de problemas de optimización de programación lineal, entera o con restricciones. La creación de los modelos resueltos con SCIP y CBC se realiza mediante el módulo *linear_solver.pywraplp*, mientras que CP-SAT es usado a partir del módulo *sat.python.cp_model*. Las funciones para crear restricciones o la función objetivo varían un poco, aunque realmente el esquema seguido para crear y resolver el modelo en ambos casos sigue la misma idea. Dos funciones distintas son definidas para este fin, recibiendo todos los equipos a considerar en el problema sin los estudiantes rechazables (previamente se define otra que retorna todas las combinaciones de alumnos escogidas para el total de tamaños de grupos disponibles utilizando *itertools*) y los

estudiantes obligatorios entre otros, y el proceso para crear los modelos sigue los pasos vistos en el pseudoalgoritmo 1, los cuales se pueden resumir en inicializar el *solver* en cuestión, crear las variables de decisión, restricciones y función objetivo y proceder a su resolución:

Pseudoalgoritmo 1. Creación y resolución de un problema con un *solver*

- Inicialización del solver en cuestión
- Creación de una variable binaria por cada equipo, que indica si este se incluye en la solución final o no
- Creación de las restricciones:
    - 1 restricción para garantizar que el tamaño de la clase solución es el adecuado.
    - 1 restricción por alumno para garantizar que solo participa en un equipo.
    - Tantas restricciones como conjuntos obligatorios de alumnos, iterando sobre todos ellos para encontrar alguno de cada par en algún equipo.
- Creación de la función objetivo de tipo maximización. Se recupera la puntuación de cada equipo según la heurística a desear y se añade a dicha función multiplicando a la variable correspondiente.
- Resolución del modelo
- Evaluación del tipo de solución obtenida: óptima, factible, no factible, resultado anormal…

Todo el código relativo a la generación de los equipos factibles y a la creación y resolución del modelo exacto queda almacenado en el script situado en *TFG_Code/exact_model /exact_model.py.*

5.4. Algoritmo genético

Aunque existen numerosas librerías en Python para la creación de algoritmos genéticos, como DEAP, una “infraestructura de computación evolutiva para el rápido prototipado y testeo de ideas” [19] se ha optado por realizar una implementación “desde cero” dada el nivel de personalización que requieren algunos operadores y la propia representación además de la sencillez del propio esquema.

Principalmente, se establece la creación de una clase StudentGA en Python que permita recopilar todos aquellos métodos específicos que requiere el problema, como son los nuevos operadores definidos o la forma de computar las máscaras válidas, entre otros. Una clase permite abstraer toda la lógica del programa haciendo que el usuario solo necesite llamar a los métodos que quiera, además de establecer una dependencia más sencilla entre los distintos parámetros y métodos que la conforman. Otros operadores genéricos como pueden ser NWOX o mutación por bit se definen fuera de ella, pues no son específicos de este problema, sino que pueden aplicarse a muchos en el caso de utilizar una representación adecuada de las soluciones, aunque sí que se dejan ubicados en el mismo módulo para ser importados de manera conjunta. En global, el código queda registrado en *TFG_Code/genetic_algorithm/GA.py*.

La mencionada clase recibe como argumentos una instancia del problema (información del aula), los estudiantes obligatorios y rechazables, los tamaños mínimo y máximo de los equipos, así como el tipo y las puntuaciones de todos ellos en los respectivos roles/personalidades. El

constructor, además de considerar los anteriores, inicializa en base a estos las máscaras de bits que serían válidas en el problema y realiza un precálculo de la matriz de distancias de edición entre estas máscaras.

El método *generate_valid_masks* se encarga de esto primero, y resulta interesante mostrar un esquema general de dicho cálculo con el pseudoalgoritmo 2, con el que se crearía una máscara por tipo de estructuración de equipos según sus tamaños:

Pseudoalgoritmo 2. Generación de máscaras válidas

- Generación de todas las combinaciones con repetición de tamaño $\frac{n}{l}$ de números entre l y u.
- Se recorta cada combinación para que la suma de alumnos sea igual al tamaño de la clase (a ser posible, en caso contrario no se tiene en cuenta). Estos constituyen los posibles fenotipos del problema.
- Todos los fenotipos válidos se convierten a la máscara de bits asociada. Para ello se itera sobre cada valor *v* del fenotipo y se añaden *v*-1 0s y un 1 (este 1 no se añade en caso de encontrarse ante el último valor del fenotipo). Recordemos de la sección 4.2.2 que, para evitar redundancias, solo se tiene una máscara de bits por fenotipo distinto.

Ilustremos este algoritmo con un ejemplo. Imaginemos que tenemos una clase de 6 alumnos, y se decide agruparlos en grupos de 2 y 3. Como mucho se pueden formar 3 grupos de 2. Las combinaciones con reemplazamiento generadas son: [(2,2,2), (2,2,3), (2,3,3), (3,3,3)]. Al tratar de recortar estas combinaciones se termina teniendo los fenotipos [(2,2,2), (3,3)], mientras que (2,2,3) ni (2,3,3) no participan en este último conjunto dado que al sumar de izquierda a derecha no se consigue el total de 6 alumnos. Finalmente, (2,2,3) se traduce a la máscara [0,1,0,1,0,0], por ejemplo.

Una vez teniendo las máscaras válidas del problema, solo basta con utilizar un doble bucle en Python para calcular la distancia entre cada par de ellas (previamente transformándolas a cadenas de texto de 0s y 1s) y almacenarlas en una matriz de *numpy*, proceso que hace el método *get_distance_matrix*. El cálculo de esta distancia se realiza mediante la llamada de la función *distance* de la librería *Levenshtein*. Cabe señalar que este proceso se puede precalcular y su resultado almacenar para futuras ocasiones, dado que será el mismo para cualquier instancia del problema con el mismo tamaño de aula y con el mismo tamaño de equipos válidos.

5.4.1. Representación de las soluciones, codificación, decodificación, validación …

Una solución se codifica mediante una lista en Python que contiene en su interior otras 2 listas, la primera relativa a los identificadores de los alumnos y la segunda a la llave de bits. Un ejemplo para el caso anterior sería el genotipo [[3,5,4,2,1,6], [0,0,1,0,0]].

La generación de una solución aleatoria se basaría en hacer un barajado (*random.shuffle*) de la lista de alumnos y una selección aleatoria (*random.choice*) de una máscara de bits válida,

mientras que una población aleatoria incluiría en una nueva lista tantas soluciones aleatorias como el tamaño de población que se le indique. Cabe mencionar la implementación de funciones que permitan guardar poblaciones en ficheros de texto con el objetivo de poder replicar experimentos en el futuro.

Se debe remarcar la existencia de procesos de codificación y decodificación de las soluciones, entrando en especial detalle del segundo. Una solución se decodifica mediante el esquema del pseudoalgoritmo 3:

Pseudoalgoritmo 3. Proceso de decodificación de una solución del AG

1) Creación de un grupo vacío a partir de un conjunto de Python que contiene el primer alumno.
2) Para cada bit de la máscara:
    a. Si es un 1, el grupo actual está cerrado, se añade este a la lista de grupos y se vuelve a 1)
    b. Si no, se añade el alumno al grupo actual.
3) Se añade el último grupo formado a la lista, devolviéndola como resultado del algoritmo.

Por último, y según lo comentado en apartados anteriores, de igual manera un cromosoma puede no cumplir con ciertas de las condiciones impuestas. Se define en la clase un método que permita verificar este hecho, el cual se encarga de iterar sobre todas las coaliciones del cromosoma decodificado, comprobando para cada una de ellas que no haya un par de alumnos que se rechacen o que sí vayan juntos aquellos que deben irlo por "obligación".

5.4.2. Función de fitness

La función de fitness decodifica una solución del problema y evalúa para cada equipo formado la puntuación alcanzada según la heurística a desear. El fitness de una solución será pues la suma de dichos valores, a no ser que previamente se detecte que esta no sea factible, en cuyo caso se devuelve un valor de 0.

5.4.3. Selección

La selección por ruleta con rango lineal queda implementada en el método *ranked_wheel_selection*. Básicamente se obtiene una copia ordenada ascendentemente por el fitness de la población (función *sorted* con clave *get_fitness*), se asigna el rango a cada elemento y se escala mediante la fórmula descrita en 4.2.3, se obtiene la probabilidad de selección dividiendo cada rango escalado por la suma de todos ellos y finalmente se hace un una selección aleatoria con reemplazamiento sobre la población de tantos elementos como hijos se quieran generar con *np.random.choice* teniendo en cuenta las probabilidades calculadas.

5.4.4. Cruce

Una alternativa igualmente válida a la hora de implementar el NWOX que evita los desplazamientos de elementos a izquierda y derecha del vector ha consistido en, para el primer hijo, crear un vector de valores nulos exceptuando los que caen dentro del bloque de elementos entre las posiciones generadas aleatoriamente (con *random.randint*), que se rellenan con los esos valores del segundo padre, y posteriormente recorrer el primer padre y ocupar en el mismo orden las casillas que estén vacías del primer hijo.

En el caso de CX se ha seguido la descripción detallada en el apartado del diseño del algoritmo genético. El siguiente pseudocódigo (pseudoalgoritmo 4) resume su implementación, la cual está basada en un bucle principal que se repite mientras la suma de los elementos del hijo sea distinta a la del primer padre:

Pseudoalgoritmo 4. Proceso de creación de hijos con Cycle Crossover

- Creación de hijo de la misma longitud de los padres y relleno con valores -1.
- Mientras que la suma de los elementos del hijo sea distinta a la suma del primer padre (falta algún elemento por insertar)
    - Si un ciclo ha terminado, indexamos la siguiente posición con valor -1 del hijo.
    - Si no:
        - Se escoge un valor cualquiera (*random.choice*) de los padres considerando la posición actual
        - Si este está en el hijo:
            - Miramos el otro valor, si este también está, dar el ciclo por terminado. Si no, nos guardamos este junto con su posición en el otro padre.
        - Si el valor no está en el hijo:
            - Añadimos este en la posición actual del hijo, haciendo que ahora la posición actual sea la posición de este en el otro padre.

5.4.5. Mutación

Los operadores de Simple Swap y RSM son relativamente sencillos de implementar, para el primero se utiliza dos veces *random.randint* para generar dos posiciones aleatorias en el vector cuyos elementos son intercambiados; el segundo comienza con la generación de las posiciones aleatorias, cambia esos elementos y hace avanzar dichas posiciones mediante un bucle *while* (hacia izquierda y derecha del vector) hasta que ambas coinciden.

Sobre todo, en este apartado es útil destacar cómo se han desarrollado los nuevos operadores que mantienen la factibilidad de las claves de bits de los cromosomas.

La mutación selectiva de divisiones (método *mask_mutation*) recibe la máscara a mutar y busca en primer lugar la fila de la matriz de distancias asociada a esta (se obtiene el índice de la máscara a partir del conjunto de las máscaras válidas y se accede a la matriz con él). Esta fila contiene la distancia de edición entre esta máscara y el resto de aquellas factibles en el problema (una distinta por fenotipo). La mutación se basará en hacer una selección por ruleta con ránking

lineal atendiendo al problema de la minimización de la distancia entre máscaras, método cuya implementación ya se ha detallado más arriba.

Randomised masked mutation (método *randomised_mask_mutation*) utiliza inicialmente una mutación convencional para representaciones de tipo binario, la mutación por bit. Esta se implementa recorriendo cada elemento de dicho vector binario, generando un valor real aleatorio entre 0 y 1 e invirtiendo ese elemento (poner a 0 el que estaba a 1 y viceversa) si este valor es menor que una cierta probabilidad (para nuestro ejemplo se invierte cada bit con una probabilidad de 0,5). Habiendo descrito esta fase, el algoritmo (pseudoalgoritmo 5) se puede resumir en los siguientes pasos:

Pseudoalgoritmo 5. Proceso de mutación de máscaras con Randomised Masked Mutation

- Obtener hijo mutado mediante una mutación por bit.
- Recorrer todas las máscaras válidas del problema y calcular la distancia de Levenshtein de cada una con la anterior.
- Cálculo de la distancia mínima, y anotación de todas aquellas máscaras que están a esa misma distancia de la máscara mutada.
- Se retorna de forma aleatoria una máscara de todas las del conjunto creado en el paso anterior.

Un aspecto de este último que no se ha implementado (no ha existido la necesidad) pero que sería interesante es, en caso de encontrarse en problemas con gran número de máscaras válidas, se podría hacer un muestreo previo de estas en la mutación para optimizar el tiempo empleado en su recorrido, aunque no se garantizase obtener la que mejor se ajusta a la solución del problema.

5.4.6. Reemplazamiento

En la función *elitist_replacement* se desarrolla el funcionamiento del reemplazamiento elitista. Tras recibir la población y los hijos generados en la generación actual, concatena ambos (los cuales son listas de Python) mediante el operador suma, resultado el cual se ordena descendentemente según el fitness y terminan escogiéndose un número de elementos igual al tamaño de la población.

5.4.7. Estructuración del algoritmo genético

Con todos los operadores definidos, además de la mención de otro tipo de funciones que se utilizan en el genético como aquellas necesarias para evaluar las soluciones, se describe finalmente la estructura de dicho algoritmo, la cual se incluye en otro método de la clase StudentGA. El pseudoalgoritmo 6 consistiría en lo siguiente, definiendo antes algunas variables:

Pseudoalgoritmo 6. Funcionamiento del algoritmo genético diseñado

- *num_iter*: número de iteraciones realizadas por el algoritmo hasta el momento. Inicialmente a 0.

- *max_iter*: número máximo de iteraciones definidas por el usuario. Es un hiperparámetro.

- *rep_iter*: número de iteraciones en los que el valor de la mejor solución es el mismo. Definido por el usuario (hiperpárametro).

-actual_pop: población actual.

- Generación de una población de soluciones aleatorias, almacenada en *actual_pop*.
- Mientras que *num_iter* < *max_iter*:
    - Evaluar soluciones de *actual_pop*. Si el valor de la mejor es óptimo, o este no ha cambiado tras *rep_iter*, terminar. En caso contrario, continuar.
    - Seleccionar padres, y barajarlos.
    - *new_pop* = [ ]
    - Para cada par de padres:
        - *provisional_children, feasible_children* = [ ], [ ]
        - *feasible_iter_rep* = 0
        - Mientras que |*feasible_children*| < 2 o *feasible_iter_rep* < 5:
            - Cruzar los padres con probabilidad p_{cross}, dando lugar a los hijos.
            - Mutar a los hijos con probabilidad p_{mut}.
            - Añadir los hijos a *provisional_children* y solo los factibles a *feasible_children*.
            - *feasible_iter_rep* = *feasible_iter_rep* + 1
        - Añadir 2-|*feasible_children*| hijos de *provisional_children* a los hijos factibles.
        - Añadir los 2 hijos resultantes a *new_pop*.
    - *actual_pop* = reemplazamiento elitista de *actual_pop* y *new_pop*.
    - *num_iter* = *num_iter* + 1

Detalles del algoritmo comprenderían mencionar la implementación de mecanismos para ir contando las iteraciones donde se repite el mejor valor de fitness o de almacenado de los resultados (tanto los finales como los obtenidos en cada iteración) que puedan servir para analizar el rendimiento con distintas configuraciones de operadores genéticos.

6. Experimentación

Este apartado comprende el diseño y resultados de las pruebas realizadas sobre las soluciones implementadas. Todas ellas se llevan a cabo en una máquina virtual con 4 núcleos y 8 GB de RAM proporcionada por VRAIN (*Valencian Research Institute for Artificial Intelligence*). En 6.1 se describe el conjunto de datos del que se parte, las instancias de problemas generadas a partir de este y algunas condiciones generales de todos los experimentos. En 6.2 se muestran los resultados obtenidos mediante los algoritmos exactos y algunas consideraciones sobre el modelo de programación lineal. La optimización monobjetivo mediante metaheurísticas (algoritmos genéticos) se describe en 6.3, donde se comparan diversas configuraciones de los algoritmos utilizados, así como las diferencias con las soluciones óptimas.

6.1. Creación de instancias

En los proyectos de investigación en optimización es habitual la creación de instancias de problemas para probar cómo los *solvers* o algoritmos de optimización diseñados proceden a su resolución y poder medir el rendimiento de los mismos. El objetivo en esta sección pues será describir la manera en que se han creado instancias de problemas que permitan posteriormente evaluar las soluciones aportadas en este trabajo.

Se parte originalmente de un conjunto de datos anonimizado formado por un total de 384 alumnos del grado de Turismo en la Universidad Politécnica de Valencia, datos que han sido recogidos desde los cursos académicos comprendidos entre 2014-2015 hasta 2018-2019 y que contienen los resultados de la aplicación de las pruebas de Belbin y MBTI.

En primer lugar, se excluyen aquellos individuos que no tengan puntuaciones en alguno de dichos tests, pues se desea hacer una comparación de las puntuaciones que se obtendrían de la aplicación de ambas heurísticas para un mismo conjunto de estudiantes. No todos presentan valores de ambas pruebas ya que éstas se hacen semestralmente y algunos alumnos solo permanecen uno de los dos periodos o directamente no realizan alguna de estas. Teniendo en cuenta este conjunto filtrado, formado por 260 alumnos, el siguiente paso consiste en la generación de clases de forma artificial mediante muestreos del conjunto de datos de partida. Esto nos permite crear clases artificiales, y por tanto instancias de problema, con características similares a las que uno se encontraría en el aula. La necesidad de tener diferentes instancias de problema viene motivada por la posibilidad de realizar experimentos y evaluar las distintas soluciones en una diversidad de condiciones.

Concretamente, las pruebas se realizan sobre 30 instancias generadas aleatoriamente para 4 tamaños de aula distintos, siendo estas aulas de 20, 30, 40 y 60 alumnos en total. Por tanto, se observará el rendimiento de los distintos algoritmos sobre un total de 120 instancias distintas en términos del tiempo de ejecución y los valores de solución obtenidos principalmente. Cada una de las 120 instancias generadas puede ser resuelta atendiendo a diferentes restricciones en cuanto al tamaño de los equipos. No obstante, se definió que los posibles grupos de trabajo a

formar tendrán un mínimo de 3 alumnos y un número máximo de 5. Estos valores se han escogido en base a la experiencia de mis tutores a la hora de formar equipos en aulas de tamaños similares. En principio no se impondrán restricciones que tengan que ver con la obligatoriedad o exclusión de ciertos pares de alumnos en los equipos. También se ha probado a forzar grupos de mismo tamaño dentro de este rango entre 3 y 5 para notar la diferencia en el rendimiento de las soluciones indicadas (por ejemplo, para aulas de tamaño 20, o bien hacer grupos de solo 4 o de solo 5 alumnos, frente a configuraciones que permiten tanto grupos de 3, 4 o 5).

6.2. Pruebas con métodos exactos

Si bien se sabe mediante la teoría que un algoritmo exacto encuentra la solución óptima en caso de que exista, se ha propuesto la ejecución de cada tipo de *solver* (SCIP, CBC, CP-SAT) un total de 5 veces sobre cada instancia del problema a fin de evitar algún tipo de error inesperado en la ejecución o la devolución de alguna solución que sea anormal, así como para capturar diferencias estadísticas en el tiempo de ejecución de los diferentes *solvers*.

Una primera incidencia en los resultados obtenidos proviene del hecho de que ninguno de los algoritmos ha sido capaz de resolver los problemas con tamaños de aula de 60 alumnos y formación de equipos de entre 3 y 5. El número de combinaciones que se deben tener en cuenta escala enormemente al pasar de aulas de 40 a 60, por lo que el programa explota rápidamente en memoria RAM y termina por detenerse. CP-SAT también presenta este mismo problema para alguna instancia de tamaño 40, aunque en general es capaz de resolver las de este tipo y en este caso se trata más bien de alguna limitación de la máquina de pruebas, mientras que CBC no llega a tolerar este tipo de pruebas. En cuanto al resto de soluciones, todas han sido calificadas como óptimas, y una forma sencilla de verificar que realmente son todas la misma y no hay ninguna inconsistencia es comparar cada valor de solución con el promedio de todas las repeticiones agrupado por la heurística, la instancia y el *solver* utilizados, lo cual efectivamente da unos resultados positivos. En resumen, de todas las instancias que se han podido obtener solución (esto es, no ha habido problemas con el *solver* o el uso de recursos) se ha conseguido encontrar la solución óptima, lo cual resulta algo positivo dentro de los análisis, y los fallos en el cómputo con algunos *solvers* comienzan cuando la talla del problema crece.

Resulta interesante hacer un análisis cualitativo de los valores de solución obtenidos a fin de comprender cómo se pueden asignar los grupos en base a las heurísticas definidas, sabiendo que los valores de solución son óptimos en todas las instancias de cada tipo de problema. En primer lugar, resulta comprensible que a medida que aumenta el número de alumnos a considerar, la distribución en las puntuaciones obtenidas para ambas heurísticas (figura 23) sea por lo general mayor, dado que se pueden llegar a crear más equipos de trabajo que contribuyen al valor de la solución final (i.e., suma de las heurísticas para cada equipo). Al forzar el tamaño de los grupos se ve además que las puntuaciones parecen ligeramente menores respecto de utilizar los grupos entre 3 y 5. En la figura mencionada se observa que para el eje de abscisas no existen cajas con todos los colores de la leyenda: en el caso de grupos de 3 a 5 faltaría la roja referente a problemas de clases de 60 alumnos, las cuales no ha sido posible su compleción según lo descrito en el párrafo anterior; por otro lado, faltarían las cajas que tienen que ver con la imposibilidad de partir un tamaño de aula únicamente en grupos de un determinado tamaño, como puede ser el caso de dividir 40 alumnos en solo equipos de 3.

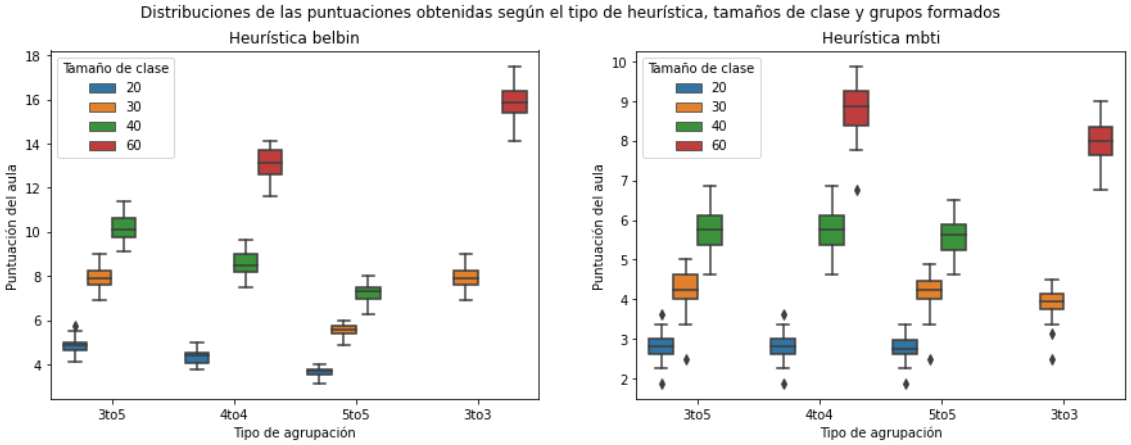


Figura 23. Distribuciones de las puntuaciones obtenidas según el tipo de heurística, tamaños de clase y grupos formados.

Se ha observado que existe un ligero descenso (aunque no del todo significativo) en la distribución de la puntuación media de Belbin para grupos formados entre 3 y 5 personas cuando este tamaño de clases aumenta (quizás podría ocurrir por el tipo de aulas creadas, aunque vemos que esto no ocurre cuando forzamos el tamaño de grupos según se ve en el primer gráfico de la figura 24, donde por ejemplo las medianas de las puntuaciones promedio se mantienen a medida que el aula crece). Esto se debe a la definición de esta heurística, donde para maximizar las puntuaciones obtenidas los algoritmos priorizan el formar muchos equipos pequeños que no necesariamente reportan la mayor puntuación alcanzable. Como se ve en la visualización de la derecha de a continuación, siempre se forman 12 equipos de 3 alumnos y uno de 4 para las instancias de tamaño 40:

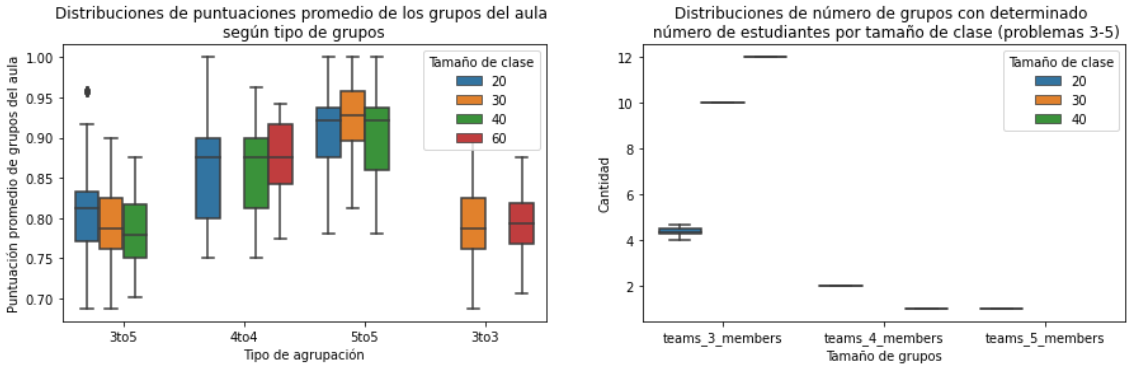


Figura 24. Distribuciones de puntuaciones promedio de los grupos del aula según tipo de grupos (izquierda) y distribuciones de número de grupos con determinado número de estudiantes por tamaño de clase en problemas de 3-5 (derecha).

Quizás este hallazgo no sea del todo favorable, dado que se busca gran diversidad de roles para el desempeño de los trabajos. Una primera alternativa a esto se basaría en optimizar la puntuación media de los equipos, esto es, la actual heurística dividida por el número de equipos formados, aunque el problema no sería lineal y por tanto mucho más complicado de resolver.

Finalmente se opta por añadir una nueva restricción al modelo que limite el número de equipos formados de determinado tamaño de la siguiente forma, la cual queda implementada para futuras ocasiones. Siendo M_j el número máximo de equipos de tamaño *j* a formar, la restricción se podría definir así:

[limit groups of size *j*]:

$$\sum_{t_i \in \mathcal{T}, |t_i| = j} t_i \times \delta_i \leq M_j \quad ; l \leq j \leq u$$

Hablamos ahora del motivo principal de este apartado, respondiendo a la pregunta ¿qué método exacto es el mejor para el problema? A priori, SCIP ha conseguido computar la solución óptima a más instancias de problemas que el resto, algo también importante es ver en cuánto tiempo.

Para contestar a la pregunta anterior, podríamos considerar que un subproblema es un escenario con un conjunto de alumnos concreto (i.e., un aula concreta), una función objetivo concreta (e.g., Belbin o Myers-Briggs) y una configuración de tamaños de equipo (e.g., tamaños mínimos y máximos). Del problema de formación de equipos en el aula hay definidos 22 subproblemas, pues se ha probado con clases de 20, 30, 40 o 60 alumnos, con heurísticas de Belbin o Myers-Briggs y con tamaños de grupo entre 3 y 5 o forzando dicho tamaño para solo obtener grupos de 3, 4 o 5 (subproblemas de 60 alumnos donde entre algún equipo de 5 no entran al no ser viable su cómputo)**.** La metodología seguida para obtener el *solver* más rápido se muestra en los siguientes apartados. En concreto, se sigue una metodología de análisis multiproblema, bastante común en el ámbito de la optimización con metaheurísticos [56].

En primer lugar, se obtiene el tiempo de ejecución medio que ha tardado cada *solver* en resolver cada instancia. Este valor no es más que el promedio de los tiempos obtenidos en las 5 repeticiones del experimento para dicha instancia, y es en lo que se basará el análisis. Una primera aproximación se puede centrar en observar visualmente las distribuciones de dichos valores, aunque para comprobar si existen realmente diferencias estadísticamente significativas entre ellas podemos usar métodos como el ANOVA de medidas repetidas [57], [58].

El análisis de la varianza (ANOVA) se puede entender como una extensión del test *t* de comparación de medias, y su variante para medidas repetidas es una prueba mucho más potente en los casos en los que los mismos factores o tratamientos se midan sobre los mismos sujetos (i.e., en este caso las instancias). La hipótesis nula implicaría que el uso de cualquier factor tiene un efecto indistinto en la media de la variable respuesta (i.e., el tiempo de ejecución), y rechazarla implicaría una diferencia entre al menos un par de ellos. De forma similar al ANOVA para datos independientes se deben cumplir las asunciones de:

- Independencia: nuestros análisis la asumen directamente al ser los individuos muestras aleatorias del conjunto total de alumnos.
- Normalidad: la distribución para cada tratamiento debe ser normal, pudiendo comprobarse visualmente mediante histogramas o gráficos Q-Q o estadísticamente mediante pruebas como la de Shapiro-Wilk.
- Esfericidad: se debe asumir que la varianza de la diferencia entre cada par de tratamientos es la misma, pudiendo utilizar para ello la prueba de Mauchly.

En caso de no cumplirse alguna de las anteriores se puede observar los resultados obtenidos mediante la corrección de los p-valores de Greenhouse-Geisser o de Huynd-Feldt o utilizar una prueba no paramétrica como el test de Friedman, y en caso de encontrar algún tipo de diferencia utilizar test post-hoc para determinar qué pares son diferentes entre sí [59]. La librería *pingouin* presenta funciones para computar las pruebas de normalidad y esfericidad según los datos de entrada, además de disponer de las pruebas de ANOVA y de Friedman y de test post-hoc para la comparación de muestras pareadas.

Mostraremos el proceso seguido para el estudio del promedio de tiempos de ejecución para las instancias de 30 alumnos en las que se forman equipos con la heurística de Belbin y variando su tamaño entre 3 y 5. Recordar, que en este caso se han resuelto 30 instancias de aulas con 30 alumnos empleando el criterio de Belbin y variando el tamaño de los equipos entre 3 y 5. La tabla 2 muestra el tiempo promedio obtenido por cada *solver* a la hora de resolver cada una de las 30 instancias de este tipo y a priori de las distribuciones de los tiempos medios de ejecución (figura 25) se extrae una clara diferencia en dichos valores para cada *solver*, donde si bien CBC no parece presentar una media de más de 9 segundos en promedio, SCIP presenta unos valores mucho más altos, estando la media algo superior a 190. Estructuras similares se han visto para el resto de los problemas, lo que parecería indicar que CBC es un buen candidato. Para verificar la existencia de las mencionadas diferencias mediante el ANOVA de medidas repetidas antes debemos comprobar el cumplimiento de las hipótesis de normalidad y esfericidad.

Solver	Promedio de tiempo de ejecución (s)
SCIP	192,62
CBC	8,66
CP-SAT	52,89

Tabla 2. Promedio de tiempo de ejecución por *solver* (problema de ejemplo).

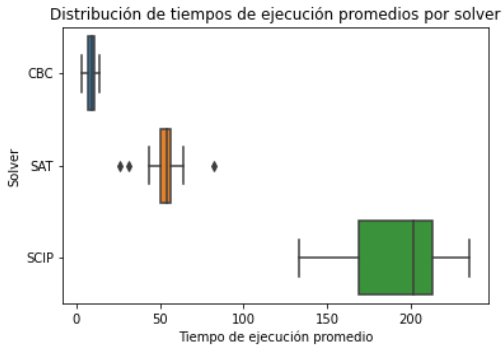


Figura 25. Distribución de tiempos promedio de ejecución por *solver* (problema de ejemplo).

Los gráficos Q-Q indican lo alejada que estaría una distribución de datos de entrada de una teórica (en este caso una normal) teniendo en cuenta los cuantiles de la anterior, y a efectos prácticos si ambas son similares entonces los puntos dibujados formarán una línea recta [60]. En este caso, y como se ve en la figura 26, los valores obtenidos con SCIP y CBC parecen seguir una distribución aproximadamente normal al estar cercanos a esa línea recta, lo cual corresponde con que no se rechace la hipótesis nula de normalidad de la prueba de Shapiro-Wilk (el p-valor es mayor que 0.05). Esto no ocurre así con CP-SAT, donde se observa la presencia de 3 valores extremos bastante alejados de la línea de referencia indicando la posible

no normalidad, algo que se constata con el p-valor dado por la prueba de Shapiro-Wilk, siendo este menor al riesgo de primera especie (alfa=0.05) y por tanto rechazando la hipótesis nula de normalidad. Además, el test de Mauchly arroja un p-valor igual a 3.94e28, así que no se puede afirmar la existencia de esfericidad. Esto da lugar a tener que utilizar la alternativa no paramétrica.

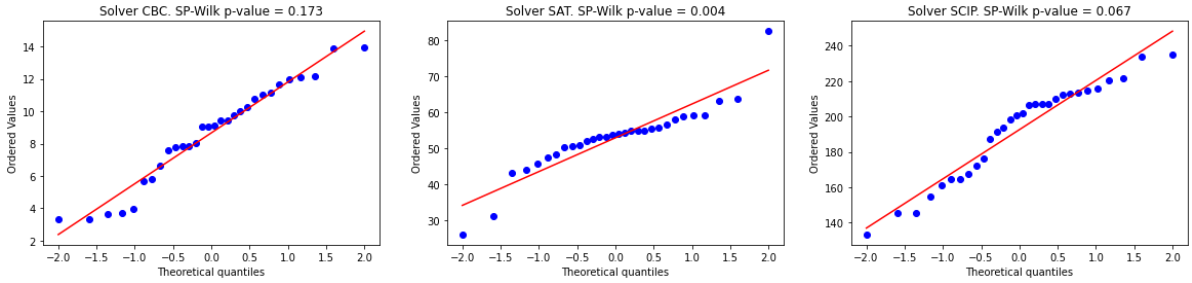


Figura 26. Gráficos de normalidad para los distintos *solvers* (problema de ejemplo).

El test de Friedman es la extensión para más de dos poblaciones de la prueba de los rangos con signo de Wilcoxon, y o bien considera que las distribuciones son idénticas (hipótesis nula) o al menos dos distribuciones son diferentes entre sí (dominancia estadística, hipótesis alternativa) y tras aplicarlo para este caso concreto se ha podido observar que el p-valor=9.36e-14, indicando que hay diferencias estadísticamente significativas en los tiempos medios de ejecución entre al menos un par de *solvers*. Dado que se está considerando pruebas no paramétricas, en este caso se utilizará la prueba de los rangos con signo de Wilcoxon (alternativa no paramétrica de *pingouin* en los test post-hoc) para ver entre qué pares de muestras existen diferencias, y si consideramos la hipótesis alternativa que dice que ”la media de A es menor que la de B” entonces concluimos mediante los p-valores de la tabla 3 que la media de los valores de CBC es menor que la del resto de *solvers* y a su vez la de CP-SAT es menor que la de SCIP.

Solver 1	Solver 2	Evidencia	p-valor
CBC	SAT	Menor que	4.146e-21
CBC	SCIP	Menor que	1.047e-25
SAT	SCIP	Menor que	1.06e-22

Tabla 3. Prueba de Wilcoxon para cada par de *solvers* (problema de ejemplo)

Todo el anterior análisis estadístico se ha replicado para cada familia de problemas definida, estudiando si los datos de entrada cumplen las asunciones del ANOVA y aplicando las pruebas correspondientes. Dada la gran variedad de problemas de los que se dispone, así como la propia repetitividad en el proceso, en su lugar se detallará en la tabla siguiente las conclusiones extraídas en cada escenario, donde los valores de la columna “Problema” siguen el formato “Heurística-TamañoClase-TamañoGrupos”:

Problema	Ranking de tiempo de ejecución	Promedio de tiempo por solver (s)		
		CBC	CP-SAT	SCIP
Belbin-20-3to5	CBC < CP-SAT < SCIP	2,17	5,82	19,84
Belbin-20-4to4	CBC < SCIP < CP-SAT	0,11	2,82	0,78

Belbin-20-5to5	CBC < SCIP < CP-SAT	0,5	4,84	3,35
Belbin-30-3to5	CBC < CP-SAT < SCIP	8,66	52,89	192,62
Belbin-30-3to3	CBC < SCIP < CP-SAT	0,09	1,71	0,78
Belbin-30-5to5	CBC < CP-SAT < SCIP	15,93	65,52	185
Belbin-40-3to5	SCIP	-	-	5045,93
Belbin-40-4to4	CBC < CP-SAT < SCIP	6,64	46,89	78,81
Belbin-40-5to5	CBC < CP-SAT < SCIP	284,59	555,17	5691,04
Belbin-60-3to3	CBC < SCIP < CP-SAT	1,07	15,11	11,31
Belbin-60-4to4	CBC < CP-SAT < SCIP	92,8	443,17	2081,97
MBTI-20-3to5	CBC < CP-SAT < SCIP	1,01	11,83	19,41
MBTI-20-4to4	CBC < SCIP < CP-SAT	0,13	2,93	0,59
MBTI-20-5to5	CBC < SCIP < CP-SAT	0,53	5,02	3,3
MBTI-30-3to5	CBC < CP-SAT < SCIP	17,32	160,33	330,04
MBTI-30-3to3	CBC < SCIP < CP-SAT	0,08	1,79	0,74
MBTI-30-5to5	CBC < CP-SAT < SCIP	17,55	63,11	194,02
MBTI-40-3to5	SCIP	-	-	10733,45
MBTI-40-4to4	SCIP	-	-	69,66
MBTI-40-5to5	SCIP	-	-	5878,13
MBTI-60-3to3	CBC < (CP-SAT & SCIP) (CP-SAT y SCIP no presentan diferencias significativas)	1,05	15,38	14,2
MBTI-60-4to4	CBC < CP-SAT < SCIP	103,22	491,35	1987,6

Tabla 4. Resultados finales de los *solvers*.

De la tabla 4 se podría extraer finalmente que CBC es el mejor *solver* al menos para este tipo de problemas, presentando los mejores resultados para la mayor parte de ellos. No obstante, puede presentar varios inconvenientes durante su ejecución o en la obtención de soluciones (por ejemplo, se probó a incluir pares de alumnos que deben ir juntos para probar la restricción asociada y este no daba respuestas, lo cual está relacionado con lo dicho en este hilo [61]) por lo que debería utilizarse en ausencia de nuevas restricciones. En general, podría tratarse de un *solver* apto para problemas bastante más simples, ya que además a partir de equipos de 40 resuelve menos instancias (ocurre con los de grupos entre 3 y 5 o también solo los 4 o 5 integrantes). Alternativas más "fiables" serían CP-SAT o SCIP (según el caso, uno es mejor que el otro), aunque el tiempo de ejecución podría ser bastante superior al alcanzado por CBC. Además, algunos problemas solo han podido resolverse en su totalidad con SCIP, por lo que sería el único *solver* a escoger para estas situaciones, si bien podría repetirse los experimentos en máquinas con más recursos para probar el rendimiento de CP-SAT, el cual solo ha sido útil en algunas instancias.

Otro tipo de conclusiones extraídas implican que por lo general los tiempos de ejecución son menores en el caso de considerar un tamaño único de los equipos, lo que puede deberse al hecho de que en el problema se tiene un menor número de equipos factibles entre los que elegir. Es decir, existe menor combinatoria.

Por último, se debe mencionar que todos los experimentos no son tarea de un día. De hecho, podrían considerarse como el cuello de botella del trabajo. Contando todos los segundos en ejecución de los algoritmos tenemos un total de 27.76 días, tiempo que realmente ha estado dominado por los problemas de mayor calibre (clases de 40 incluyendo al menos equipos de 5

personas) pudiendo durar hasta 5 horas o más en casos particulares (normalmente estarían alrededor de la hora o dos horas). Este último hecho es el que ha motivado el uso de metaheurísticas, ya que integrar un servicio en la web que dé respuestas tras numerosas horas no sería del todo factible.

6.3. Pruebas con el algoritmo genético

En esta sección se describe los resultados obtenidos con el algoritmo genético, primero haciendo una descripción de los parámetros utilizados y del diseño de los experimentos (6.3.1) y posteriormente relatando dichos resultados mediante los análisis llevados a cabo (6.3.2)

6.3.1. Puesta a punto

El objetivo de este apartado es doble: determinar la configuración del algoritmo genético que rinda mejor en cada una de las heurísticas seleccionadas y/o tamaños de aula, y verificar que no solamente la calidad de las soluciones obtenidas es satisfactoria, sino también que realmente el uso de esta metaheurística puede suponer una mejora en tiempo de ejecución frente a los *solvers* utilizados en la sección anterior.

El procedimiento seguido comienza por dividir cada conjunto de 30 instancias en una parte de entrenamiento (70% de las instancias), que servirá para seleccionar los mejores operadores para el subproblema considerado, y una parte de test (30% restante) para comprobar los resultados obtenidos con esa mejor configuración. Estas instancias serán las mismas que las empleadas en los métodos exactos para poder establecer una comparación justa. Una "configuración" se entiende como una 9-tupla donde cada elemento indica la siguiente información:

- Método de cruce de permutación.
- Probabilidad de cruce.
- Método de cruce de máscara de bits.
- Método de mutación de permutación.
- Probabilidad de mutación.
- Método de mutación de máscara de bits.
- Método de selección.
- Método de reemplazamiento.
- Tamaño de la población

En la etapa de entrenamiento, las configuraciones utilizadas serán todas las combinaciones posibles de los elementos que se resumen en la siguiente tabla (tabla 5):

Parámetro		Valores
Selección		Ruleta con rango con ránking lineal (RWS-sp-1.5)
Cruce	Permutación	Cycle Crossover (CX), Non-Wrapping Order Crossover (NWOX)
	Máscaras	Herencia (InheritMask)

Probabilidad de cruce		0.7, 0.9
Mutación	Permutación	Simple Swap (SS) Reverse Sequence Mutation (RSM)
	Máscaras	Mutación selectiva de divisiones (DivisionSelect), Randomised Mask Mutation (RandomisedMasking)
Probabilidad de mutación		0.2, 0.4
Reemplazamiento		Elitista (EL)
Tamaño de la población		50, 100

Tabla 5. Elementos de la experimentación con el algoritmo genético.

Vemos que por simplicidad se referirá a los operadores con nombres acortados. El método de selección dispone del hiperparámetro SP, pero por no aumentar la complejidad de los experimentos se ha establecido a SP=1.5, que daría un buen balance entre una selección uniforme y una selección en la que se apremia más las mejores soluciones. Además, en los algoritmos genéticos suele funcionar bien el uso de probabilidades de cruce elevadas (por encima de 0.7, por ejemplo) y de mutación bajas, pero en este caso las probabilidades de mutación se han definido un poco más altas (hasta 0.4) por el hecho de que la propuesta de algoritmo depende completamente de la mutación de máscaras de bits para realizar cambios en las divisiones que se deben hacer de los alumnos (el cruce de máscaras no provoca cambios en las mismas, sino que los hijos heredan la de los padres).

La resolución de cada instancia (tanto en el conjunto de entrenamiento como en el de pruebas) se repite 10 veces a modo de hacer frente a la aleatoriedad con la que actúan estos algoritmos. Cada repetición inicializa una población aleatoria distinta del tamaño especificado, pero las mismas poblaciones son utilizadas para la comparación de las configuraciones definidas para evitar que las diferencias entre un par de configuraciones no se deban a comenzar con poblaciones "mejores". Otros aspectos por destacar son los relativos al criterio de parada: el número máximo de iteraciones se fija en 500, aunque el algoritmo terminará si la mejor solución alcanzada hasta el momento no mejora tras 100 iteraciones seguidas o si una solución tiene el valor óptimo conocido.

6.3.2. Análisis

Para determinar la calidad del funcionamiento de los algoritmos genéticos se tendrán en cuenta fundamentalmente dos medidas: el valor de la solución obtenida y el tiempo de ejecución. Dado que se dispone de la solución óptima de las instancias consideradas, al haber sido resueltas con los métodos exactos, se decide normalizar los valores de fitness v obtenidos con el algoritmo genético de la forma $\frac{v}{v*}$, lo que indicaría qué proporción de la mejor solución se ha llegado a alcanzar y permite establecer una visión global de cómo se comportan las configuraciones de los genéticos sobre todos los problemas.

En primer lugar, se realiza un análisis cualitativo de los resultados obtenidos en el entrenamiento a fin de comprender el rendimiento de los algoritmos. De todos los experimentos realizados, en 151.237 ejecuciones de las 188.160 ejecuciones realizadas (el 80%

aproximadamente) se ha obtenido la solución óptima. La proporción de soluciones óptimas encontradas es alrededor del 85% para el conjunto de problemas de tamaño de clase 20, y es coherente que descienda ligeramente al aumentar este (pasando a poco menos del 80% en clases de 40). Desglosando finalmente las comparaciones a nivel de problema, puede verse que por lo general se consigue llegar al óptimo más fácilmente con la heurística de MBTI, además de ocurrir esto mismo para los problemas en los que únicamente se ha de crear los equipos de un tamaño fijo, como se verá en la tabla 7 del final. A pesar de que a veces el óptimo se alcanza en el 50% o 60% de los casos, las puntuaciones obtenidas son bastante buenas por lo general, superando el 80% para todos los problemas, como se ve en la figura 27 (gráfico izquierdo).

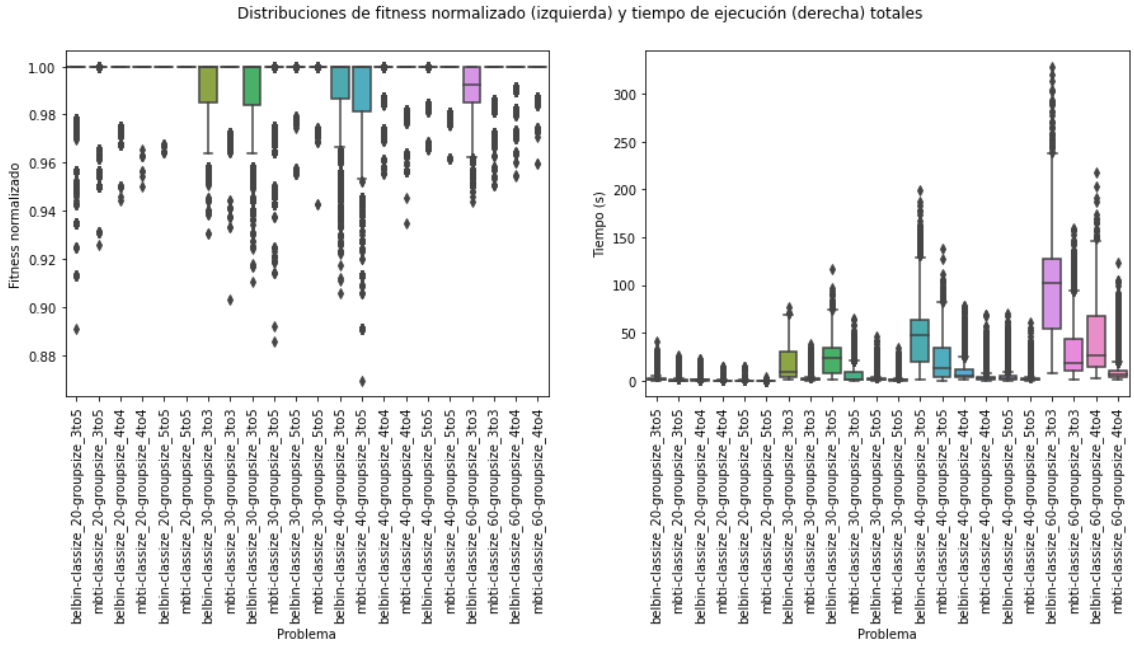


Figura 27. Distribuciones de fitness normalizado y tiempo de ejecución totales.

Como antes, el cómputo de los resultados ha llevado mucho tiempo, aunque ahora se debe más al gran volumen de experimentos más que a la duración individual de cada uno. Estaríamos hablando de 33.38 días en total, a los que se los podría sumar al menos una semana debido a la repetición de algunas pruebas como consecuencia de algún error en el código. Algo parecido a lo visto en los métodos exactos es que el tiempo aumenta con el tamaño de la clase y con la variedad en la creación del tamaño de los grupos, aunque en muchos casos los problemas se resuelven en cuestión de segundos o minutos. En la figura 27 (gráfico derecho) vemos cómo crece la distribución de estos tiempos cuanto mayor es el tamaño de la clase, por ejemplo, y los promedios globales se verán de nuevo en la tabla 7.

Pasamos ahora a la determinación de las mejores configuraciones para cada subproblema. La metodología es similar a la vista en las pruebas con los métodos exactos, por lo que en primer lugar se calcula el promedio del valor que se quiere comparar para todas las repeticiones de una misma instancia y configuración del algoritmo genético y se aplica un ANOVA de medidas repetidas (o la alternativa no paramétrica en caso de no cumplirse las asunciones necesarias) para verificar si existen diferencias entre alguna configuración, las cuales pueden

constatarse con un test post-hoc. No obstante, esto no resulta tan sencillo, ya que se ha de comparar dos métricas (media del valor normalizado de la solución y promedio de tiempo de ejecución) y existen hasta 64 configuraciones diferentes (32 en el caso donde se fuerza el tamaño de los grupos). Se ha puesto el foco en obtener primeramente la solución más cercana a la óptima y de esta idea surge el siguiente esquema, que es el que se ha seguido para obtener las mejores configuraciones:

1) Comprobar las asunciones de normalidad y esfericidad de los valores normalizados de fitness, y realizar la prueba de comparación de medias correspondiente (ANOVA de medidas repetidas o test de Friedman) con objeto de conocer si hay evidencias que indican una diferencia estadísticamente
2) En caso de haber diferencias, recuperar la configuración con mejor promedio de fitness normalizado. Con un test post-hoc escoger aquellas configuraciones que serían "iguales" a la de referencia, y limitar el análisis solo a estas. Si no las hay, proceder con todas las configuraciones actuales.
3) Repetir los 2 primeros pasos comparando las configuraciones resultantes según el tiempo de ejecución.
4) Las configuraciones restantes tendrán los mejores valores de fitness para el subproblema a considerar y, según estos, los tiempos más rápidos. Cualquiera valdría, aunque la decisión final se puede hacer según el promedio del fitness normalizado promedio, el tiempo promedio u otro criterio como examinar los gráficos de convergencia hacia los valores óptimos de solución a lo largo de las iteraciones.

El hecho de tomar las decisiones basándose en una configuración de referencia se debe a que no importa tanto ver entre qué pares de configuraciones existen diferencias y el tipo de diferencias encontradas, sino que bastaría con señalar cuáles obtienen resultados similares respecto del mejor promedio obtenido. Con esto claro, se procede a mostrar un ejemplo con el problema de obtener la mejor configuración en aulas de 30 alumnos en las que se forman equipos con la heurística de Belbin y variando su tamaño entre 3 y 5.

Primero se examina las distribuciones de los valores normalizados promedio de fitness según la configuración utilizada. El siguiente gráfico de caja y bigotes (Figura 28) muestra eso mismo, donde el eje de ordenadas está ordenado ascendentemente según el promedio de dichos valores. Este recuerda que las soluciones devueltas por los algoritmos son bastante buenas (por encima de 0.95) algo ya mencionado antes. No obstante, se observa que en algunos casos las distribuciones son más bien asimétricas, ya que en estos se consigue más frecuentemente los valores óptimos de la solución (lo que impone un límite a la distribución por la derecha) o que dada una configuración se encuentran valores bastante alejados del resto. Al comparar los p-valores de la prueba de Shapiro-Wilk se concluye que 36 de las 64 opciones no siguen una distribución normal, y el p-valor de la prueba de Mauchly es 0, rechazándose la hipótesis nula de esfericidad.

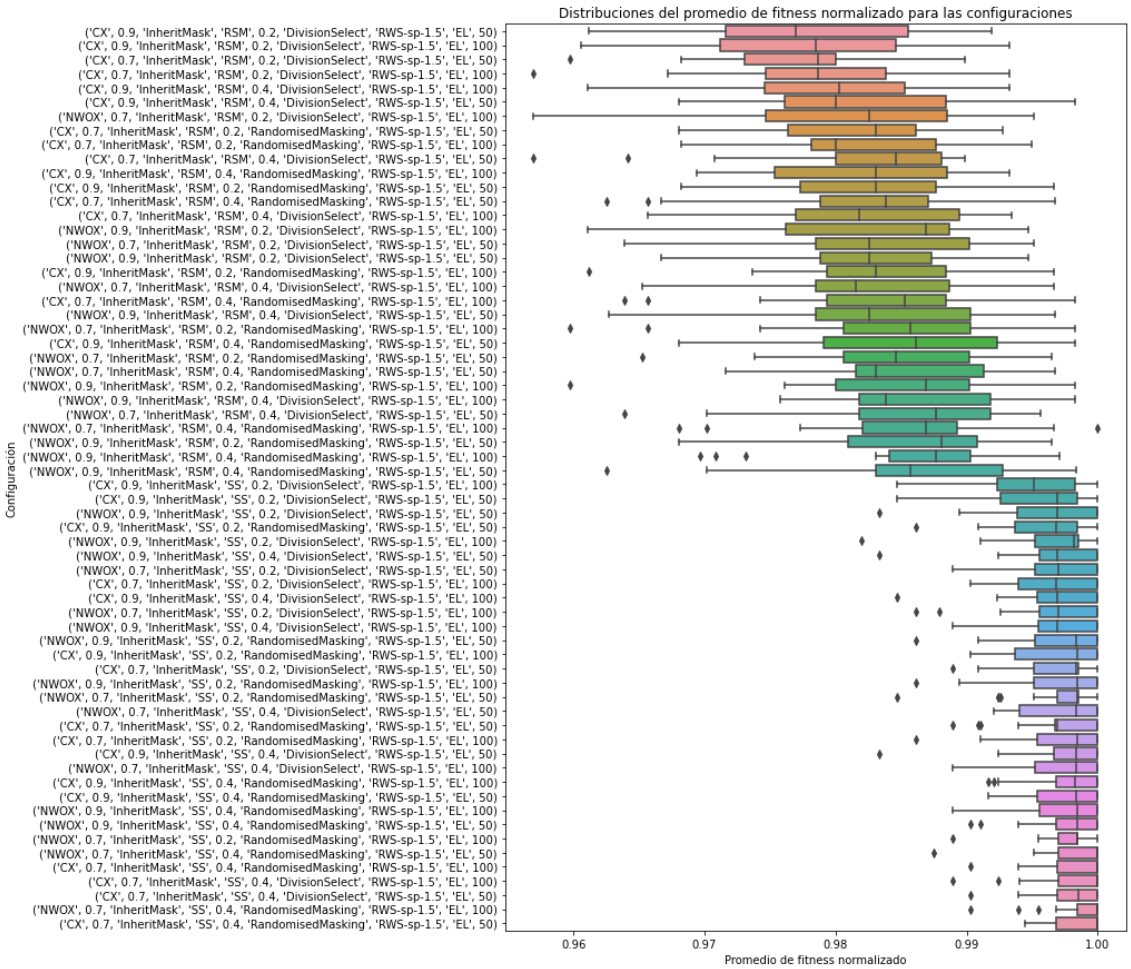


Figura 28. Distribuciones de fitness normalizados promedio por configuración (problema de ejemplo)

Para verificar si existen diferencias significativas en el promedio de los valores considerados se utilizará pues el test de Friedman, con resultado de p-valor prácticamente 0, indicando que no se acepta la idea de que todas las distribuciones sean iguales. La configuración de referencia aquí será ('CX', 0.7, 'InheritMask', 'SS', 0.4, 'RandomisedMasking', 'RWS-sp-1.5', 'EL', 50) con una media del promedio de fitness normalizado de 0.999, siendo esta la configuración con un mejor promedio de fitness normalizado de las estudiadas. Se ha realizado una comparación por pares con el resto de las configuraciones con una prueba post-hoc, con la que se concluye que otras 5 configuraciones obtienen resultados estadísticamente similares a la configuración de referencia.

Configuración 1	Configuración 2	Alternativa	p-valor
('CX', 0.7, 'InheritMask', 'SS', 0.4, 'DivisionSelect', 'RWS-sp-1.5', 'EL', 50)	('CX', 0.7, 'InheritMask', 'SS', 0.4, 'RandomisedMasking', 'RWS-sp-1.5', 'EL', 50)	Distinto	0.205
('CX', 0.7, 'InheritMask', 'SS', 0.4, 'DivisionSelect', 'RWS-sp-1.5', 'EL', 100)	('CX', 0.7, 'InheritMask', 'SS', 0.4, 'RandomisedMasking', 'RWS-sp-1.5', 'EL', 50)	Distinto	0.286
('CX', 0.7, 'InheritMask', 'SS', 0.4,	('CX', 0.7, 'InheritMask', 'SS', 0.4, 'RandomisedMasking', 'RWS-sp-1.5',	Distinto	0.475

'RandomisedMasking', 'RWS-sp-1.5', 'EL', 100)	'EL', 50)		
('NWOX', 0.7, 'InheritMask', 'SS', 0.4, 'RandomisedMasking', 'RWS-sp-1.5', 'EL', 50)	('CX', 0.7, 'InheritMask', 'SS', 0.4, 'RandomisedMasking', 'RWS-sp-1.5', 'EL', 50)	Distinto	0.833
('NWOX', 0.7, 'InheritMask', 'SS', 0.4, 'RandomisedMasking', 'RWS-sp-1.5', 'EL', 100)	('CX', 0.7, 'InheritMask', 'SS', 0.4, 'RandomisedMasking', 'RWS-sp-1.5', 'EL', 50)	Distinto	0.076

Tabla 6. Comparaciones post-hoc para diversas configuraciones del genético (problema de ejemplo).

Dado lo anterior, el siguiente análisis de tiempos se limita a las 6 configuraciones resultantes, de forma que pueda verse si una configuración obtiene soluciones con calidad similar, pero de forma más rápida. De nuevo se repite el proceso anterior, rechazándose ambas asunciones de normalidad (3 de 6 distribuciones no la cumplen) y esfericidad. Sin embargo, la prueba de Friedman ahora devuelve un p-valor=0.303, por lo que no se encuentran diferencias significativas en el promedio de los tiempos medios de ejecución según la configuración utilizada. La configuración de referencia ha resultado ser la misma. Se puede pasar a ver un gráfico de convergencia con la evolución de cada configuración a lo largo de las iteraciones, coloreadas de forma distinta. Concretamente en el gráfico de la figura 29 cada línea muestra el promedio del fitness normalizado medio de la mejor solución en cada iteración, y las bandas son los intervalos de confianza al 95% para dicho promedio. Una tendencia general en este es que los valores crecen a medida que aumenta el número de iteraciones, algo que resulta comprensible ya que en cada generación se encuentra una solución mejor o igual a la de la generación anterior, aunque luego se llegue a una cierta estabilización en dicho crecimiento. Hablando de nuevo sobre las configuraciones, la línea roja asociada a la de referencia es ligeramente superior al resto para un número de iteraciones bajo, mientras que otras configuraciones como la asociada a la línea azul o naranja han tenido casos con un número algo más elevado de iteraciones a la hora de calcular la solución del subproblema. Es decir, la configuración de referencia parece tener una convergencia más rápida, aunque no existen evidencias estadísticas para afirmarlo con las pruebas realizadas.

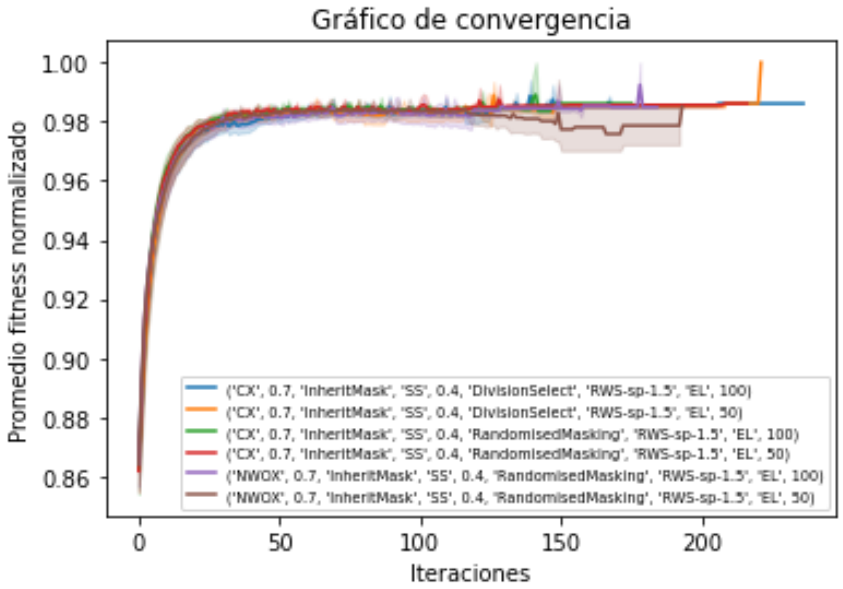


Figura 29. Gráfico de convergencia para las 6 configuraciones escogidas en entrenamiento (problema de ejemplo).

En realidad, se podría realizar la ejecución del conjunto de instancias de test con cualquiera de las anteriores, aunque se escogerá la de referencia (que coincide en todos los casos) para ello. Algo interesante de las configuraciones resultantes es que, para esta situación en concreto, todas ellas comparten que la probabilidad de cruce debe ser 0.7, la de mutación 0.4 y la mutación de permutación el Simple Swap, y algunas son prácticamente similares entre sí a excepción del tamaño de la población, lo cual ocurre también para otro tipo de subproblemas. Como antes, se muestra los resultados obtenidos para el resto de los problemas en una tabla, con la configuración escogida y el número de configuraciones "similares":

Problema	Configuración escogida	Nº de configuraciones similares	Proporción soluciones óptimas	Fitness global promedio	Tiempo global promedio (s)
Belbin-20-3to5	('NWOX', 0.9, 'InheritMask', 'SS', 0.4, 'RandomisedMasking', 'RWS-sp-1.5', 'EL', 100)	8	0,89	0,99	3,15
Belbin-20-4to4	('NWOX', 0.9, 'InheritMask', 'SS', 0.4, 'DivisionSelect', 'RWS-sp-1.5', 'EL', 50)	9	0,97	0,99	0,81
Belbin-20-5to5	('NWOX', 0.9, 'InheritMask', 'SS', 0.4, 'DivisionSelect', 'RWS-sp-1.5', 'EL', 50)	13	0,998	0,99	0,14
Belbin-30-3to5	('CX', 0.7, 'InheritMask', 'SS', 0.4, 'RandomisedMasking', 'RWS-sp-1.5', 'EL', 50)	6	0,56	0,99	22,89
Belbin-	('CX', 0.9, 'InheritMask',	5	0,66	0,99	15,81

30-3to3	'SS', 0.4, 'DivisionSelect', 'RWS-sp-1.5', 'EL', 50)				
Belbin-30-5to5	('NWOX', 0.9, 'InheritMask', 'SS', 0.4, 'DivisionSelect', 'RWS-sp-1.5', 'EL', 50)	5	0,91	0,99	3,1
Belbin-40-3to5	('NWOX', 0.9, 'InheritMask', 'SS', 0.4, 'RandomisedMasking', 'RWS-sp-1.5', 'EL', 100)	3	0,50	0,99	46,63
Belbin-40-4to4	('NWOX', 0.9, 'InheritMask', 'SS', 0.4, 'DivisionSelect', 'RWS-sp-1.5', 'EL', 100)	5	0,85	0,99	11,72
Belbin-40-5to5	('NWOX', 0.9, 'InheritMask', 'SS', 0.4, 'DivisionSelect', 'RWS-sp-1.5', 'EL',100)	2	0,93	0,99	5,82
Belbin-60-3to3	('CX', 0.9, 'InheritMask', 'SS', 0.4, 'DivisionSelect', 'RWS-sp-1.5', 'EL', 50)	3	0,44	0,99	96,46
Belbin-60-4to4	('NWOX', 0.9, 'InheritMask', 'SS', 0.4, 'DivisionSelect', 'RWS-sp-1.5', 'EL', 50)	4	0,75	0,99	39,52
MBTI-20-3to5	('NWOX', 0.9, 'InheritMask', 'SS', 0.4, 'RandomisedMasking', 'RWS-sp-1.5', 'EL',50)	20	0,84	0,99	2,42
MBTI-20-4to4	('NWOX', 0.9, 'InheritMask', 'RSM', 0.4, 'DivisionSelect', 'RWS-sp-1.5', 'EL',50)	12	0,99	0,99	0,12
MBTI-20-5to5	('NWOX', 0.9, 'InheritMask', 'RSM', 0.2, 'DivisionSelect', 'RWS-sp-1.5', 'EL',50)	27	1	1	0,03
MBTI-30-3to5	('NWOX', 0.9, 'InheritMask', 'SS', 0.2, 'DivisionSelect', 'RWS-sp-1.5', 'EL',100)	12	0,82	0,99	6,61
MBTI-30-3to3	('NWOX', 0.9, 'InheritMask', 'SS', 0.4, 'DivisionSelect', 'RWS-sp-1.5', 'EL',50)	5	0,95	0,99	2,81
MBTI-30-5to5	('NWOX', 0.9, 'InheritMask', 'SS', 0.4, 'DivisionSelect', 'RWS-sp-1.5', 'EL',50)	11	0,85	0,99	3,81
MBTI-40-3to5	('NWOX', 0.9, 'InheritMask', 'SS', 0.4, 'RandomisedMasking',	5	0,62	0,99	20,08

	'RWS-sp-1.5', 'EL',50)				
MBTI-40-4to4	('NWOX', 0.9, 'InheritMask', 'SS', 0.4, 'DivisionSelect', 'RWS-sp-1.5', 'EL', 50)	5	0,95	0,99	4,36
MBTI-40-5to5	('NWOX', 0.9, 'InheritMask', 'SS', 0.4, 'DivisionSelect', 'RWS-sp-1.5', 'EL', 50)	7	0,98	0,99	2,76
MBTI-60-3to3	('CX', 0.9, 'InheritMask', 'SS', 0.4, 'DivisionSelect', 'RWS-sp-1.5', 'EL', 50)	3	0,82	0,99	30,33
MBTI-60-4to4	('CX', 0.9, 'InheritMask', 'SS', 0.4, 'DivisionSelect', 'RWS-sp-1.5', 'EL', 50)	8	0,97	0,99	10,11

Tabla 7. Resultados del análisis en los experimentos de entrenamiento (AG).

De la tabla 7 se puede observar la coincidencia de varios parámetros, como son en general las probabilidades de cruce o de mutación. Se ha probado a hacer un análisis global, esto es, considerando las instancias de todos los subproblemas para así disponer de una configuración de referencia. Previamente al análisis se han normalizado los tiempos de ejecución mediante una transformación MinMax en cada subproblema, dado que tiempos muy grandes para las instancias de tamaño de clase grandes podrían ocultar los valores obtenidos para clases pequeñas, y el resultado han sido 4 configuraciones que resultan similares, entre las que se encuentra una recurrente en la tabla anterior, ('NWOX', 0.9, 'InheritMask', 'SS', 0.4, 'DivisionSelect', 'RWS-sp-1.5', 'EL', 50).

Tras lo descrito, se ha evaluado el rendimiento de las configuraciones referencia seleccionadas sobre el conjunto de instancias de pruebas o test, y los resultados han sido bastante satisfactorios. La proporción global de soluciones óptimas alcanzadas es del 94% tanto en entrenamiento como en test. La semejanza de proporciones entre resultados de entrenamiento y test se ha mantenido al desglosar estos por tamaño de clase y también por la heurística utilizada y el tamaño de grupo, conclusiones que también se derivan del estudio de los tiempos de ejecución. De la combinación de los datos de entrenamiento y test también se ha querido estudiar el carácter *anytime* de estos algoritmos, buscando cómo de buena sería la calidad de la mejor solución encontrada si paráramos la ejecución tras un cierto número de iteraciones. Esto se puede ver en gráficos de convergencia, donde en este caso las líneas muestran el promedio de fitness normalizado en cada iteración y las bandas los intervalos de confianza al 95% para dichos promedios. Los gráficos de la figura 30 muestran en su mayoría un comportamiento parecido para cada problema, en el que ya de por sí se parte de una mejor solución en promedio bastante buena (fitness normalizado por encima del 80% en todos los casos) y existe un crecimiento elevado en el fitness obtenido en unas pocas iteraciones. Tras ello, se estabiliza el mejor fitness promedio obtenido en iteraciones futuras, pudiendo llegarse a un gran número de iteraciones hasta que el algoritmo finalmente se detiene. El número de iteraciones de rápido crecimiento dependen del problema, y podría cortarse la ejecución según comienza la etapa de estabilización para acelerar la ejecución.

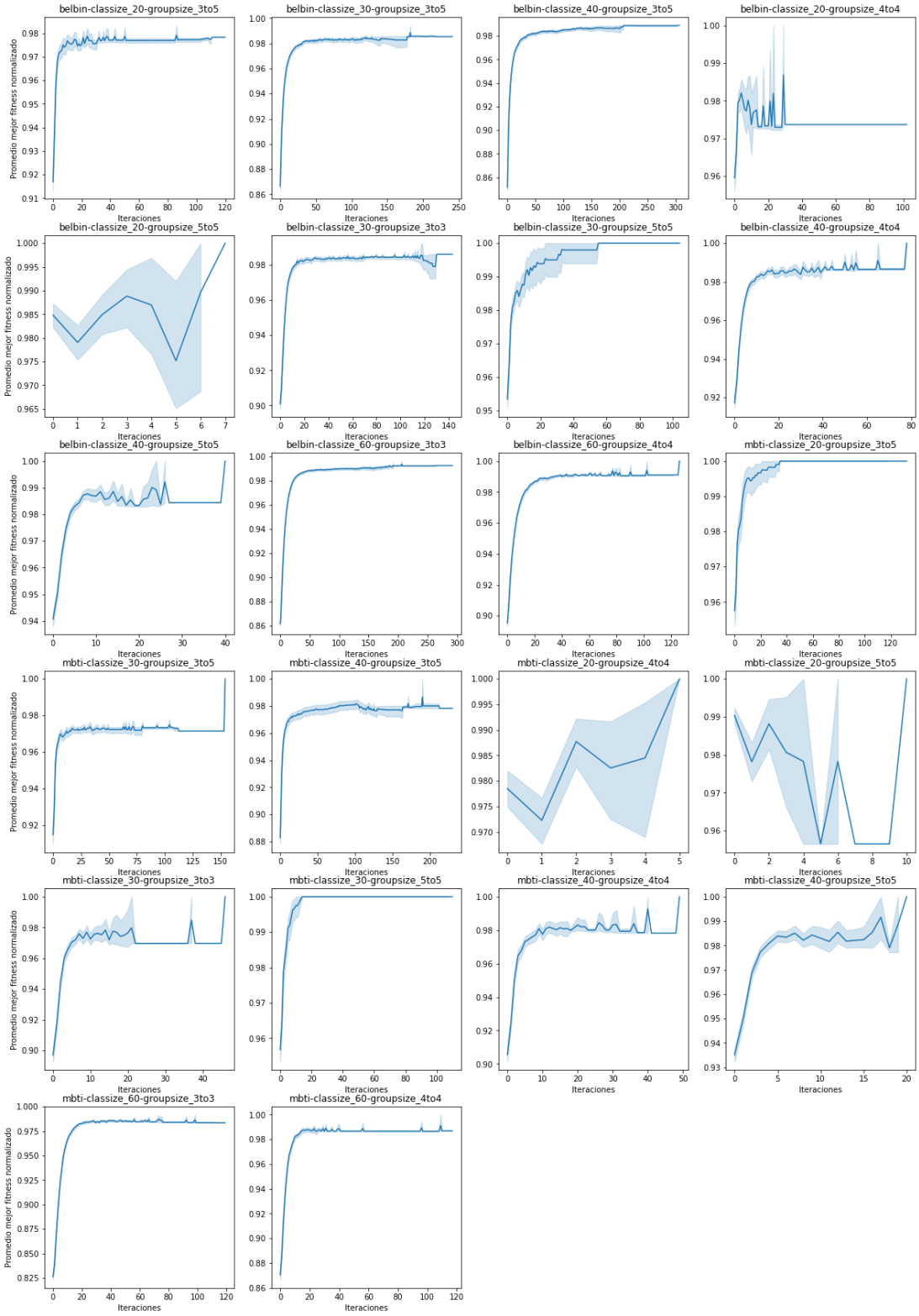


Figura 30. Gráficos de fitness frente a número de iteraciones por cada problema.

Finalmente se ha hecho una comparación de los tiempos de resolución de los métodos exactos y los algoritmos genéticos seleccionados (aunando para ello los datos de entrenamiento y test). Por ejemplo, la tabla 8 muestra el promedio global devuelto por cada método en los

diferentes problemas, donde se puede ver que en bastantes casos el algoritmo genético consigue los mejores valores, siendo superado en otros por el *solver* CBC. Esta diferencia se hace evidente cuando el tamaño de introduce un mayor tamaño de grupos, así como del aula de alumnos. De todas maneras, dada la inestabilidad de CBC en el cálculo de soluciones, el algoritmo genético se presenta como una gran alternativa al resto de *solvers*, teniendo en cuenta también la ratio de soluciones óptimas encontradas.

Problema	Proporción de soluciones óptimas AG	Promedio de tiempo del AG (s)	Promedio de tiempo mejor *solver* (s)
Belbin-20-3to5	0,96	1,55	2,17 (CBC)
Belbin-20-4to4	0,99	0,4	0,11 (CBC)
Belbin-20-5to5	1	0,09	0,5 (CBC)
Belbin-30-3to5	0,90	13,26	8,66 (CBC)
Belbin-30-3to3	0,94	8,05	0,09 (CBC)
Belbin-30-5to5	0,97	1,45	15,93 (CBC)
Belbin-40-3to5	0,797	26,84	5045,93 (SCIP)
Belbin-40-4to4	1	4,45	6,64 (CBC)
Belbin-40-5to5	1	1,92	284,59 (CBC)
Belbin-60-3to3	0,75	63,84	1,07 (CBC)
Belbin-60-4to4	0,997	16,82	92,8 (CBC)
MBTI-20-3to5	0,83	2,24	1,01 (CBC)
MBTI-20-4to4	1	0,08	0,13 (CBC)
MBTI-20-5to5	1	0,04	0,53 (CBC)
MBTI-30-3to5	0,92	3,66	17,32 (CBC)
MBTI-30-3to3	1	1,23	0,08 (CBC)
MBTI-30-5to5	0,90	2,37	17,55 (CBC)
MBTI-40-3to5	0,78	12,58	10733,45 (SCIP)
MBTI-40-4to4	1	1,93	69,66 (SCIP)
MBTI-40-5to5	1	1,34	5878,13 (SCIP)
MBTI-60-3to3	0,98	14,52	1,05 (CBC)
MBTI-60-4to4	0,99	7,51	103,22 (CBC)

Tabla 8. Comparación mejores AG y *solver* por problema.

7. Conclusiones

Hasta ahora se han comentado numerosos resultados con relación al rendimiento de las soluciones implementadas. Todo ello responde a los objetivos que se planteaban al comienzo de la memoria, los cuales han logrado cumplirse.

El primero hablaba sobre la aplicación y comparación de *solvers* de código libre en el problema de formación de equipos en aulas de estudiantes. Para ello se ha proporcionado una implementación en Python que, dados unos datos que contienen la puntuación de alumnos en los roles de Belbin o las personalidades de Myers-Briggs, construye un modelo y lo resuelve, devolviendo la estructura de equipos óptima y la puntuación global. Además de la realización de un análisis cualitativo que ha permitido la introducción de mejoras en la modelización matemática, se ha visto mediante métodos estadísticos que existen diferencias notables en cuanto al rendimiento (basado en el tiempo de ejecución) de los *solvers* para los problemas definidos, además de encontrar fallos inherentes en el funcionamiento de algunos para problemas algo más complejos. Una conclusión extraída de los experimentos es la necesidad de contar con métodos heurísticos o metaheurísticos para la resolución de instancias grandes de problemas, dada la incapacidad de resolución de dicho una gran mayoría de dicho tipo de instancias por parte de los *solvers* probados.

El segundo y tercero referían al uso de metaheurísticas. El uso de estas técnicas ha venido motivado precisamente por la conclusión anterior, ya que se requiere de algoritmos *anytime* y eficientes para el cálculo de soluciones con una notable calidad. Este requisito viene dado debido a que los algoritmos desean integrarse en una herramienta web. En los resultados obtenidos en este caso, Aquí se ha empleado algoritmos genéticos, y en la experimentación con varias configuraciones de estos se ha obtenido bastante buenos resultados, consiguiendo para las mejores configuraciones alcanzar la solución óptima al menos en el 80% de los casos para cada tipo de problema, y con tiempos de ejecución de segundos o minutos. Una comparación final con los métodos exactos demuestra que los algoritmos genéticos diseñados consiguen tiempos de ejecución muy competitivos e incluso mejores en bastantes ocasiones.

Durante la realización de este trabajo se cometieron errores en algunas partes de la programación de las soluciones, como en la implementación de algún operador genético, y al verse estos tras la ejecución de algunos experimentos hubo que detener todo el proceso y repetir estos de nuevo, lo que deriva en un problema de uso innecesario de recursos. Otro problema tuvo que ver con las versiones del lenguaje de programación y sus librerías entre el entorno local de trabajo y la máquina virtual, aunque su resolución fue sencilla. Finalmente habría que destacar que, al ser un proyecto de gran magnitud, se ha aprendido a distribuir el código de forma que las soluciones presentadas tengan en cuenta dependencias comunes, siendo al principio una dificultad al tener poca experiencia en la distribución en módulos de los scripts asociados a dichas soluciones.

7.1. Legado

Como se ha ido describiendo a lo largo del trabajo se busca proporcionar soluciones que puedan ser integradas en una herramienta actual de formación de equipos. Para ello, todo el código desarrollado se ha puesto a disposición en un repositorio público de GitHub [62], donde se incluyen además los *scripts* empleados con los que se lleva a cabo la experimentación, y que pueden utilizarse para probar por ejemplo configuraciones con nuevos operadores en los algoritmos genéticos. Adicionalmente se añaden los *notebooks* en los que se ha hecho los análisis estadísticos, de forma que se pueda reproducir en cualquier momento el proceso seguido en la comparación del rendimiento de los *solvers*/algoritmos empleados.

7.2. Relación del trabajo con los estudios cursados

El trabajo realizado puede vincularse con los contenidos teóricos y prácticos vistos en diversas asignaturas del Grado en Ciencia de Datos.

La mayor relación se encuentra con la asignatura *Optimización*, donde se explican conceptos de optimización, modelización matemática de problemas de optimización o metaheurísticas. Además, se ha trabajado en proyectos para la implementación y resolución de problemas con *solvers* exactos o el uso de algoritmos genéticos para abordar el problema del viajante de comercio con Python. La definición y resolución de determinados problemas de optimización, no obstante, ya venía introducido en asignaturas anteriores como *Algorítmica*.

Todas las soluciones se han desarrollado utilizando el lenguaje de programación Python, sirviendo como base lo visto en *Fundamentos de Programación* y *Programación*, donde se estudia desde lo básico en estructuras de datos, lectura de ficheros o llamadas a librerías hasta la implementación de clases propias, entre otros conocimientos.

Hay que destacar lo aprendido en *Análisis Exploratorio de Datos*, que ha influido en cómo se han realizado los gráficos descriptivos de los resultados, y donde se estudian conceptos básicos de estadística y visualización. También indicar lo visto en *Modelos estadísticos para la toma de decisiones (I y II)*, donde en conjunto se explica lo fundamental en inferencia estadística, métodos de regresión, análisis de la varianza (ANOVA) u otros métodos de comparación de poblaciones (tanto paramétricos como no paramétricos) y que han ayudado en la aplicación de todas las pruebas estadísticas vistas en el trabajo.

Por último, cabría dar una breve mención a algunas de las competencias transversales que han sido necesarias para el desarrollo del proyecto. *Diseño y proyecto* es una que podría aplicarse directamente a TFG de cualquier tipo, donde es necesaria la integración de varios conocimientos y habilidades; *Análisis y resolución de problemas* sería una segunda, pues se ha requerido entender cómo programar las distintas soluciones, así como estructurar el código necesario para mayor legibilidad. Finalmente, otra relativamente importante sería *Planificación y gestión del tiempo*. Esto es debido a que los experimentos diseñados tienen un elevado coste temporal, por lo que resulta necesario definir qué tareas priorizar y cuáles se pueden relegar a otros momentos (por ejemplo, cerciorarse del buen funcionamiento del código es lo primero, y mientras los experimentos tienen lugar se puede investigar soluciones alternativas o redactar la memoria del trabajo).

7.3. Trabajos futuros

Con esta sección finalmente se hablará de qué aspectos del trabajo realizado podrían ampliarse o qué otras alternativas se podrían introducir para la resolución del problema de formación de equipos dentro de aulas de trabajo.

Ya durante la redacción del diseño e implementación de las soluciones, concretamente para el algoritmo genético, se han ido indicando algunas funcionalidades que serían adecuadas añadir, como podrían ser el muestreo de las máscaras válidas que se han de recorrer en el método de mutación *Randomised Mask Mutation* o el precálculo de estas para que puedan ser reutilizadas en problemas del mismo tipo. Un aspecto que vale la pena introducir es un método de cruce personalizado que opere sobre las representaciones de bits del problema. Como se había visto, solo se incluyen métodos de mutación adaptados a la naturaleza de este tipo de representaciones en el problema, y la falta de un operador que recombine la información de dos máscaras de bits posiblemente sea el causante de que la probabilidad de mutación idónea para cada problema sea generalmente la más alta de las usadas en la experimentación. Y hablando de probabilidades también sería interesante, aunque también más complejo, utilizar probabilidades de cruce y mutación independientes tanto para la representación de permutación como la de bits, y obtener aquellas que mejor se ajustan de forma global o particular a los problemas definidos.

Se podría implementar otro tipo de metaheurísticas y adaptarlas según las restricciones del problema. Se ha visto la existencia de recocido simulado, búsqueda tabú [35] o GRASP [36] aplicados al problema de generación de estructuras de coaliciones, por lo que podría decidirse su implementación y/o definición de nuevos operadores de búsqueda en el vecindario de una solución considerando el funcionamiento de estos algoritmos. Con ellos se podría hacer una comparación de las distintas metaheurísticas empleadas y decidir la mejor. Aunque ya se haya visto este tipo de solución, podría hacerse un estudio del rendimiento de un algoritmo genético con operadores clásicos con respecto al aportado durante este trabajo.

Finalmente podría utilizarse librerías de multiprocesamiento para integrar la ejecución en varios núcleos de la máquina con la implementación del algoritmo genético, lo que podría ofrecer varias ejecuciones de este a la vez y poder obtener aquella que devuelva una mejor solución, por ejemplo.

8. Bibliografía

[1] «Competencias Transversales: UPV». http://www.upv.es/contenidos/COMPTRAN/info/957657normalc.html(accedido 12 de junio de 2022).

[2] «PROYECTO COMPETENCIAS TRANSVERSALES UPV.» upv.es. 2022. Accedido: 22 de junio de 2022. Disponible en: http://www.upv.es/entidades/ICE/info/Proyecto_Institucional_CT.pdf

[3] R. M. Belbin, *Management teams: why they succeed or fail*, 3rd ed. Oxford, U.K: Butterworth-Heinemann, 2010.

[4] «16 Personality Types: Myers-Briggs Type Indicator (MBTI)». https://www.simplypsychology.org/the-myers-briggs-type-indicator.html (accedido 13 de mayo de 2022).

[5] T. Rahwan, T. P. Michalak, M. Wooldridge, y N. R. Jennings, «Coalition structure generation: A survey», *Artificial Intelligence*, vol. 229, pp. 139-174, dic. 2015, doi: 10.1016/j.artint.2015.08.004.

[6] Sánchez-Anguix, V. «Contenido teórico de la asignatura Optimización (grado en Ciencia de Datos)»

[7] W. L. Winston y J. B. Goldberg, *Operations research: applications and algorithms*, 4th ed. Belmont, CA: Thomson/Brooks/Cole, 2004.

[8] «Optimization Taxonomy | NEOS». https://neos-guide.org/content/optimization-taxonomy (accedido 20 de mayo de 2022).

[9] «Branch and cut», *Wikipedia*. 4 de agosto de 2021. Accedido: 2 de junio de 2022. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=Branch_and_cut&oldid=1037121170

[10] M. W. P. Savelsbergh, «Branch and Price: Integer Programming with Column Generation», en *Encyclopedia of Optimization*, C. A. Floudas y P. M. Pardalos, Eds. Boston, MA: Springer US, 2001, pp. 218-221. doi: 10.1007/0-306-48332-7_47.

[11] R.- ASALE y RAE, «heurístico, heurística | Diccionario de la lengua española», *«Diccionario de la lengua española» - Edición del Tricentenario*. https://dle.rae.es/heurístico (accedido 22 de mayo de 2022).

[12] P. S. Game, Dr. V. Vaze, y Dr. E. M, «Bio-inspired Optimization: metaheuristic algorithms for optimization», 2020, doi: 10.48550/ARXIV.2003.11637.

[13] J. Dréo, J.-P. Aumasson, W. Tfaili, y P. Siarry, «ADAPTIVE LEARNING SEARCH, A NEW TOOL TO HELP COMPREHENDING METAHEURISTICS», *Int. J. Artif. Intell. Tools*, vol. 16, n.º 03, pp. 483-505, jun. 2007, doi: 10.1142/S0218213007003370.

[14] «Darwin, evolución y selección natural (artículo) | Khan Academy». https://es.khanacademy.org/science/ap-biology/natural-selection/natural-selection-ap/a/darwin-evolution-natural-selection (accedido 30 de abril de 2022).

[15] D. Simon, «Evolutionary optimization algorithms : biologically-Inspired and population-based approaches to computer intelligence», *undefined*, 2013, Accedido: 30 de abril de 2022. [En línea]. Disponible en: https://www.semanticscholar.org/paper/Evolutionary-optimization-algorithms-%3A-and-to-Simon/9abf1effb2092a91c7581f0cb2a75ef2010da39a

[16] «Alteraciones del material genético», 17:29:10 UTC. Accedido: 1 de mayo de 2022. [En línea]. Disponible en: https://es.slideshare.net/AlgenisRodriguez/alteraciones-del-material-gentico

[17] «Descubren una mutación que hace a sus portadores más inteligentes». https://www.20minutos.es/salud/actualidad/descubren-una-mutacion-que-hace-a-sus-portadores-mas-inteligentes-4998706/ (accedido 1 de mayo de 2022).

[18] A. E. Eiben y J. E. Smith, *Introduction to Evolutionary Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015. doi: 10.1007/978-3-662-44874-8.

[19] «DEAP documentation — DEAP 1.3.1 documentation». https://deap.readthedocs.io/en/master/index.html (accedido 4 de mayo de 2022).

[20] D. Thompson, I. Anitsal, y H. Barrett, «ATTITUDES TOWARD TEAMWORK IN HIGHER EDUCATION: A COMPARATIVE STUDY OF RELIGIOUSLY AFFILIATED UNIVERSITIES AND SECULAR-BASED UNIVERSITIES», vol. 15, n.º 2, p. 12, 2008.

[21] E. De Prada, M. Mareque, y M. Pino-Juste, «Teamwork skills in higher education: is university training contributing to their mastery?», *Psicol. Refl. Crít.*, vol. 35, n.º 1, p. 5, dic. 2022, doi: 10.1186/s41155-022-00207-1.

[22] C. M. University, «What are the benefits of group work? - Eberly Center - Carnegie Mellon University». https://www.cmu.edu/teaching/designteach/design/instructionalstrategies/groupprojects/benefits.html (accedido 18 de mayo de 2022).

[23] «Teamwork Skills: Being an Effective Group Member», *Centre for Teaching Excellence*, 8 de noviembre de 2012. https://uwaterloo.ca/centre-for-teaching-excellence/teaching-resources/teaching-tips/tips-students/being-part-team/teamwork-skills-being-effective-group-member (accedido 11 de mayo de 2022).

[24] «Tuckman's stages of group development», *Wikipedia*. 24 de mayo de 2022. Accedido: 12 de mayo de 2022. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=Tuckman%27s_stages_of_group_development&oldid=1089583995

[25] E. Mannix y M. A. Neale, «What Differences Make a Difference?: The Promise and Reality of Diverse Teams in Organizations», *Psychol Sci Public Interest*, vol. 6, n.º 2, pp. 31-55, oct. 2005, doi: 10.1111/j.1529-1006.2005.00022.x.

[26] V. M, S. Salimath, A. S. Shettar, y G. Bhadri, «A Study of Team Formation Strategies and Their Impact on Individual Student Learning Using Educational Data Mining (EDM)», en *2018 IEEE Tenth International Conference on Technology for Education (T4E)*, Chennai, dic. 2018, pp. 182-185. doi: 10.1109/T4E.2018.00047.

[27] «La guía definitiva para conocer los roles de equipo de Belbin», *Belbin*. Accedido: 12 de mayo de 2022. Disponible en: https://www.belbin.es/wp-content/uploads/2019/07/Ebook-Belbin.pdf

[28] «Tipos de personalidad | 16Personalities». https://www.16personalities.com/es/descripcion-de-los-tipos (accedido 13 de mayo de 2022).

[29] «Big Five personality traits», *Wikipedia*. 19 de junio de 2022. Accedido: 20 de mayo de 2022. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=Big_Five_personality_traits&oldid=1093894414

[30] 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, ene. 2022, doi: 10.1016/j.ece.2021.09.001.

[31] V. Rodríguez Montequín, J. M. Mesa Fernández, J. V. Balsera, y A. García Nieto, «Using MBTI for the success assessment of engineering teams in project-based learning», *Int J Technol Des Educ*, vol. 23, n.º 4, pp. 1127-1146, nov. 2013, doi: 10.1007/s10798-012-9229-1.

[32] J. P. Contreras, P. Bosch, M. Varas, y F. Basso, «A New Genetic Algorithm Encoding for Coalition Structure Generation Problems», *Mathematical Problems in Engineering*, vol. 2020, pp. 1-13, abr. 2020, doi: 10.1155/2020/1203248.

[33] T. Sandholm, K. Larson, M. Andersson, O. Shehory, y F. Tohme, «Anytime Coalition Structure Generation with Worst Case Guarantees», 1998, doi: 10.48550/ARXIV.CS/9810005.

[34] S. Sen y P. S. Dutta, «Searching for optimal coalition structures», en *Proceedings Fourth International Conference on MultiAgent Systems*, Boston, MA, USA, 2000, pp. 287-292. doi: 10.1109/ICMAS.2000.858465.

[35] A. Hussin y S. Fatima, «Heuristic Methods for Optimal Coalition Structure Generation», en *Multi-Agent Systems and Agreement Technologies*, vol. 10207, N. Criado Pacheco, C. Carrascosa, N. Osman, y V. Julián Inglada, Eds. Cham: Springer International Publishing, 2017, pp. 124-139. doi: 10.1007/978-3-319-59294-7_11.

[36] N. Di Mauro, T. M. A. Basile, S. Ferilli, y F. Esposito, «Coalition Structure Generation with GRASP», en *Artificial Intelligence: Methodology, Systems, and Applications*, vol. 6304, D. Dicheva y D. Dochev, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 111-120. doi: 10.1007/978-3-642-15431-7_12.

[37] «Team-Maker», *CATME*. https://info.catme.org/features/team-maker/ (accedido 22 de mayo de 2022).

[38] R. A. Layton, M. L. Loughry, M. W. Ohland, y G. D. Ricco, «Design and Validation of a Web-Based System for Assigning Members to Teams Using Instructor-Specified Criteria», *Advances in Engineering Education*, vol. 2, n.º 1, 2010, Accedido: 22 de mayo de 2022. [En línea]. Disponible en: https://eric.ed.gov/?id=EJ1076132

[39] M. Caserta y S. Voss, «Workgroups Diversity Maximization: A Metaheuristic Approach», may 2013, vol. 7919, pp. 118-129. doi: 10.1007/978-3-642-38516-2_10.

[40] J. M. Alberola, E. del Val, V. Sanchez-Anguix, y V. Julián, «A General Framework for Testing Different Student Team Formation Strategies», en *Methodologies and Intelligent Systems for Technology Enhanced Learning, 6th International Conference*, vol. 478, M. Caporuscio, F. De la Prieta, T. Di Mascio, R. Gennari, J. Gutiérrez Rodríguez, y P. Vittorini, Eds. Cham: Springer International Publishing, 2016, pp. 23-31. doi: 10.1007/978-3-319-40165-2_3.

[41] L. Zhang y X. Zhang, «Multi-objective team formation optimization for new product development», *Computers & Industrial Engineering*, vol. 64, n.º 3, pp. 804-811, mar. 2013, doi: 10.1016/j.cie.2012.12.015.

[42] V. Sanchez-Anguix, J. M. Alberola, E. D. Val, A. Palomares, y M.-D. Teruel, «Assessing the use of intelligent tools to group tourism students into teams: a comparison between team formation strategies», Enviado a *ACM Transactions on Intelligent Systems and Technology*, p. 14.

[43] «Self Perception Inventory» *belbin.es*. Accedido: 22 de junio de 2022. Disponible en: http://www.belbin.ie/wp-content/uploads/2014/08/BELBINUK-Self-PerceptionInventory+CompletionGrid.pdf

[44] «Python Software Foundation License», *Los diccionarios y las enciclopedias sobre el Académico*. https://es-academic.com/dic.nsf/eswiki/1351984 (accedido 16 de junio de 2022).

[45] «OR-Tools», *Google Developers*. https://developers.google.com/optimization (accedido 2 de mayo de 2022).

[46] K. Bestuzheva *et al.*, «The SCIP Optimization Suite 8.0», p. 114.

[47] «CBC User Guide». https://www.coin-or.org/Cbc/cbcuserguide.html#assClasses (accedido 9 de mayo de 2022).

[48] L. Perron, «Answer to "Which solver do Googles OR-Tools Modules for CSP and VRP use?"», *Stack Overflow*, 20 de julio de 2019. https://stackoverflow.com/a/57125734 (accedido 9 de mayo de 2022).

[49] A. Rauniyar y P. K. Muhuri, «Multi-robot coalition formation problem: Task allocation with adaptive immigrants based genetic algorithms», en *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Budapest, Hungary, oct. 2016, pp. 000137-000142. doi: 10.1109/SMC.2016.7844232.

[50] J. M. Alberola, E. del Val, A. Costa, P. Novais, y V. Julian, «A genetic algorithm for group formation in elderly communities», *AI Communications*, vol. 31, n.º 5, pp. 409-425, sep. 2018, doi: 10.3233/AIC-180771.

[51] S. Mousavi, F. Afghah, J. D. Ashdown, y K. Turck, «Use of a quantum genetic algorithm for coalition formation in large-scale UAV networks», *Ad Hoc Networks*, vol. 87, pp. 26-36, may 2019, doi: 10.1016/j.adhoc.2018.11.008.

[52] I. M. Oliver, D. J. Smith, y J. R. C. Holland, «A study of permutation crossover operators on the traveling salesman problem», en *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, USA, oct. 1987, pp. 224-230.

[53] V. A. Cicirello, «Non-wrapping order crossover: an order preserving crossover operator that respects absolute position», en *Proceedings of the 8th annual conference on Genetic and evolutionary computation - GECCO '06*, Seattle, Washington, USA, 2006, p. 1125. doi: 10.1145/1143997.1144177.

[54] A. Otman y J. Abouchabaka, «A Comparative Study of Adaptive Crossover Operators for Genetic Algorithms to Resolve the Traveling Salesman Problem», *International Journal of Computer Applications*, vol. 31, p. 9, mar 2012

[55] A. Otman, J. Abouchabaka, y C. Tajani, «Analyzing the Performance of Mutation Operators to Solve the Travelling Salesman Problem», *Int. J. Emerg. Sci.*, vol. 2, mar. 2012. doi: 10.48550/ARXIV.1203.3099.

[56] F. Peres y M. Castelli, «Combinatorial Optimization Problems and Metaheuristics: Review, Challenges, Design, and Development», *Applied Sciences*, vol. 11, n.º 14, p. 6449, jul. 2021, doi: 10.3390/app11146449.

[57] «Repeated Measures ANOVA - Understanding a Repeated Measures ANOVA | Laerd Statistics». https://statistics.laerd.com/statistical-guides/repeated-measures-anova-statistical-guide.php (accedido 15 de mayo de 2022).

[58] J.M. Marín, «Análisis de varianza con medidas repetidas. El procedimiento Modelo lineal general: Medidas repetidas», *Guía de SPSS*, p. 32. Disponible en http://halweb.uc3m.es/esp/Personal/personas/jmmarin/esp/GuiaSPSS/16anovar.pdf

[59] «Análisis de varianza (ANOVA) con Python». https://www.cienciadedatos.net/documentos/pystats09-analisis-de-varianza-anova-python.html (accedido 15 de mayo de 2022).

[60] «Gráfico Q-Q», *Wikipedia, la enciclopedia libre*. 28 de octubre de 2020. Accedido: 15 de mayo de 2022. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Gr%C3%A1fico_Q-Q&oldid=130441335

[61] «CBC solver cannot be stopped by setTimeLimit nor interrupt · Issue #3256 · google/or-tools», *GitHub*. https://github.com/google/or-tools/issues/3256 (accedido 22 de junio de 2022).

[62] G. CP, *Exact-models-and-metaheuristics-for-team-formation-TFG*. 2022. Accedido: 28 de junio de 2022. [En línea]. Disponible en: https://github.com/GonxaTroll/Exact-models-and-metaheuristics-for-team-formation-TFG_Code

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.

Este TFG tiene relación con varios ODS, señalados más arriba. Principalmente se podría asociar al cuarto, educación de calidad, pues parte de la motivación está en encontrar los mejores equipos de trabajo para asignaturas que requieran la realización de proyectos conjuntos dentro del ámbito de la educación superior. El trabajo en grupo en el aula implica la aplicación del contenido de una asignatura en concreto desde una perspectiva diferente, y permite el desarrollo de competencias tanto técnicas (como la resolución de problemas o el uso de ciertas herramientas) como personales (habilidad para comunicar de forma efectiva, liderazgo...). Una estructuración de los alumnos de esta forma atendiendo a teorías psicológicas o criterios bien conocidos no solamente facilita la existencia de un buen clima de trabajo (que influye en la experiencia que tiene el individuo sobre este tipo de metodologías) sino también la potenciación en el aprendizaje de las anteriores habilidades. Las metas del ODS que cumplen en gran parte con lo descrito son la 4.3 y 4.4, en las que se describe el acceso universal a formación de calidad (incluida la universitaria) y el aumento de las competencias necesarias para el emprendimiento o el trabajo decente respectivamente, ya que se favorece el aprendizaje en muchos sentidos al aplicar técnicas distintas. Al hilo de lo anterior, se podría establecer una relación con el octavo ODS, trabajo decente y crecimiento económico. Como se ha comentado en el propio trabajo, las industrias buscan cada vez más gente que sepa desenvolverse en un ámbito con diversas personas con las que trabajar, ya que se pueden tratar problemas complejos que requieran de comunicación y distintos puntos de vista o capacidades técnicas. Con las soluciones creadas se podrían generar equipos óptimos en clase cuyos integrantes aprenderían la realidad del trabajo con otras personas, competencia que puede influir en la asignación de nuevos puestos de trabajo.
Otro ODS relacionado relativamente importante a destacar es el noveno: industria, innovación e infraestructuras. Aunque la utilización de modelos exactos o metaheurísticos no sea algo nuevo dentro de problemas de generación de estructuras de coaliciones, de igual forma que no lo es involucrar teorías psicológicas para formación de equipos en el aula, se trata de un trabajo principalmente de investigación en el uso de estas tecnologías en el mencionado ámbito, algo que está relacionado con las metas 9.5 o 9.b. Además, la innovación se introduce en la posibilidad de incluir alguna de las soluciones propuestas en la herramienta actual de formación de equipos de la universidad y en la definición de operadores del algoritmo genético desde cero y a medida para el tipo de problemas a tratar.
Por último, cabría destacar que el TFG tiene una asociación, aunque de manera más indirecta, con el undécimo objetivo, ciudades y comunidades sostenibles. Las metas de este trabajo consisten en estudiar los tiempos de ejecución de métodos exactos y ver cómo escalan estos en problemas grandes, así como los reportados por las

metaheurísticas. En un caso real, un método exacto como el que usaba en la herramienta original podría tardar horas en devolver una solución a la estructuración de una clase en equipos de trabajo, lo que se traduce en un gran uso de recursos energéticos. Con las soluciones proporcionadas, a pesar de que no se logre obtener resultados óptimos (caso de metaheurísticas) sí que se puede lograr una reducción de este tiempo a segundos o minutos. Esto es, las soluciones proporcionadas tienen el mismo objetivo, pero pueden resultar mucho más eficientes energéticamente en la práctica, lo que se traduce en una mejora en los costes del uso de la electricidad.

ETS Enginyeria Informàtica
Camí de Vera, s/n. 46022. València
T +34 963 877 210
F +34 963 877 219
etsinf@upvnet.upv.es - www.inf.upv.es