UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Departamento de Sistemas Informáticos y Computación

Universitat Politècnica de València

Algoritmo genético para la generación automática de equipos de trabajo

---

Proyecto Final de Máster

Máster Universitario en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

Autor:
Pau Aguilar González

Directores:
Vicente Julián Inglada
Elena Del Val Noguera
Juan M. Alberola Oltra

València, 2017

Resumen

El trabajo en equipo es ahora una competencia crítica en el área de la enseñanza superior, y se ha convertido en una tarea crítica en los entornos educativos y de gestión. Desafortunadamente, buscar equipos óptimos o cercanos a lo óptimo es una tarea costosa para los humanos debido al número exponencial de resultados. Por este motivo, en el presente proyecto se sigue una línea de investigación que abarca la generación automática de equipos basados en la generación de estructuras de coaliciones mediante el diseño, desarrollo, implementación y evaluación de un algoritmo genético. Se han llevado a cabo simulaciones en escenarios reales de clases que muestran que la política es capaz de converger hacia una solución óptima mejorando el rendimiento de la herramienta estudiada.

Palabras clave: formación de equipos, coaliciones, educación, algoritmos genéticos, inteligencia artificial, roles de Belbin.

Abstract

Nowadays, teamwork is a critical competence in the area of higher education, and it has become a critical task in educational and management environments. Unfortunately, finding optimal or near-optimal teams is a costly task for humans because of the exponential number of results. Therefore, in the present paper, it is followed a line of research that guides the automatic generation of teams based on the generation of coalition structures through the design, development, implementation and evaluation of a genetic algorithm. Simulations have been carried out in real classrooms scenarios that show that the policy is able to converge towards an optimal solution by improving the performance of the studied tool.

Keywords: team formation, coalitions, education, genetic algorithm, artificial intelligence, Belbin roles.

Tabla de contenidos

Índice de figuras

1. Introducción

1.1. Presentación

En los últimos años, las organizaciones educativas han mostrado un creciente interés hacia paradigmas de enseñanza que promuevan el trabajo en equipo. De hecho, el trabajo en equipo es ahora considerado una competencia general prominente en el Espacio Europeo de Enseñanza Superior. La inclusión del trabajo en equipo como una competencia general responde a dos razones principales: está fuertemente relacionado al aprendizaje cooperativo, una metodología que ayuda a mejorar el aprendizaje en el aula; en la actualidad, los negocios más exitosos y los proyectos de ingeniería son llevados a cabo por pequeños equipos multidisciplinarios. Sin embargo, la tarea de la formación de equipos es compleja. Varios factores, como la personalidad, la experiencia, la competitividad, y el comportamiento humano pueden interferir en el rendimiento del equipo.

El trabajo en equipo es ahora una competencia crítica en el área de la enseñanza superior, y se ha convertido en una tarea crítica en los entornos educativos y de gestión. Desafortunadamente, buscar equipos óptimos o cercanos a lo óptimo es una tarea costosa para los humanos debido al número exponencial de resultados. Por esta razón, en este documento presentamos una política asistida por ordenador que facilita la generación automática de equipos cercanos a lo óptimo basados en inteligencia colectiva, generación de estructuras de coaliciones y aprendizaje bayesiano. Se han llevado a cabo simulaciones en hipotéticos escenarios de clases que muestran que la política es capaz de converger hacia una solución óptima siempre y cuando los estudiantes no tengan grandes dificultades evaluando a otros.

Por lo tanto, es de crucial importancia identificar equipos que puedan rendir correctamente. Varios eruditos han estudiado bajo qué circunstancias el trabajo en equipo positivo puede emerger. Uno de estos estudios es el conocido inventario de roles de Belbin. Se identifican nueve patrones de comportamiento que son útiles para equipos: generadores, investigadores de recursos, coordinadores, moldeadores, monitores, corporativos, implementadores, finalizadores y especialistas. Como sugieren algunos estudios, los equipos suelen beneficiarse de tener una mezcla variada de roles.

La formación de equipos también puede ser una tarea cognitiva compleja. Por ejemplo, una clase formada por 30 estudiantes puede revelar hasta 767746 equipos diferentes de tamaños 3, 4, 5 y 6, el número de diferentes particiones de la clase en equipos disjuntos crece exponencialmente con el número de equipos posibles. El problema es especialmente complicado si los profesores quieren encontrar una asignación de equipo óptima o casi óptima. Los profesionales de la educación ciertamente se beneficiarían de políticas y herramientas que guiasen el proceso de formación de equipos. Por esta razón, se considera interesante proporcionar un mecanismo que facilite la realización de esta tarea de una manera automática y que proporcione una solución apropiada según los requisitos del contexto donde se vaya a aplicar.

En el presente trabajo final de máster, se presenta un algoritmo genético que permite obtener configuraciones de equipos de trabajo cercanas a las óptimas de una manera automática.

1.2. Objetivos del proyecto

El objetivo principal de este trabajo final de máster consiste en realizar una solución a un problema real mediante una herramienta web para la formación de grupos que haga uso de algoritmos genéticos. Tomando de referencia [1], se estudiará este escenario para la implementación de un algoritmo que siga el criterio de roles de Belbin para la formación de grupos de alumnos en el aula mejorando los tiempos de computación de la herramienta actual. Como objetivos secundarios se establecen:

- Analizar la herramienta implementada para comprender el funcionamiento y las pautas del método implementado.
- Estudiar y diseñar un algoritmo genético capaz de seguir los criterios de Belbin cumpliendo los requisitos necesarios para el uso de la herramienta.
- Implementar un algoritmo genético capaz de parametrizarse atendiendo a los distintos métodos de operadores genéticos.
- Obtención y evaluación de los resultados del algoritmo en diferentes escenarios recreando casos reales obtenidos de la herramienta actual.
- Valoración de la viabilidad del algoritmo genético y comparación de resultados con la herramienta actual.

1.3. Metodología

El tipo de metodología elegido para el desarrollo del proyecto es el modelo en espiral. Las actividades de este modelo, comenzando por el bucle interior, se conforman en una espiral en la que cada bucle representa un conjunto de acciones. Este tipo de modelo permite reducir riesgos e introducir mejoras y nuevos requerimientos durante el proyecto. El proyecto dividido en las siguientes actividades:

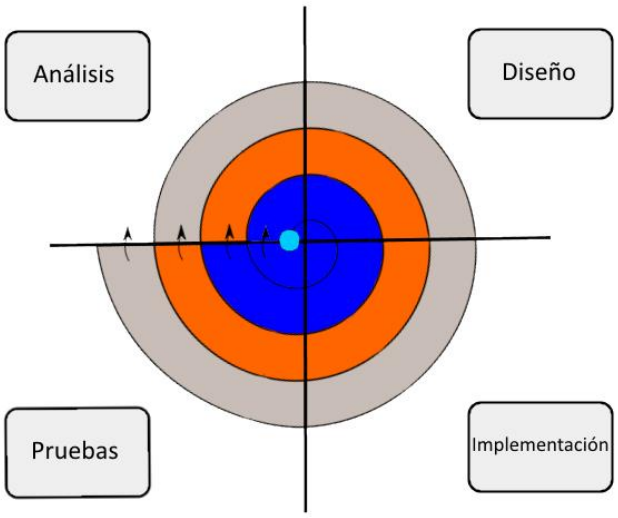

Figura 1: Desarrollo en espiral

1. Análisis: En esta etapa se estudian detalladamente los objetivos requeridos, se identifican los riesgos y se determinan las posibles soluciones y alternativas. Durante esta etapa se establecen los detalles funcionales deseados.

2. Diseño: Con los datos de la etapa anterior, se diseña el sistema. Los diseños realizados son por ejemplo, las funciones o las herramientas para hacer funcionar la aplicación.

3. Implementación: En este tercer paso se realiza la programación del diseño. Se elige un buen modelo de programación para que el código esté bien estructurado, en este caso se siguen las fases convencionales de los algoritmos genéticos.

4. Pruebas: En este último paso es donde se toma la decisión si se continúa con el siguiente ciclo del modelo. Se realizan distintas pruebas analizando si se han cumplido los objetivos. Con cada iteración de la espiral, se crean sucesivas versiones de la aplicación, cada vez más completas. Al final de todas las iteraciones, el sistema es uno completamente funcional.

1.4. Partes del proyecto

En este apartado se describe brevemente el contenido de los diferentes apartados de la memoria.

Capítulo I: Introducción

En este apartado se explican de forma introductoria los objetivos y características principales del proyecto, así como la metodología empleada y la distribución de los capítulos.

Capítulo II: Antecedentes

En este capítulo se dan a conocer las bases teóricas sobre las que se fundamenta el proyecto tanto a nivel de algoritmos genéticos como modo de uso y política implementada en la herramienta en la que se basa el proyecto.

Capítulo III: Diseño del Algoritmo Genético

En el tercer capítulo se plantea la estructura general del algoritmo, se realiza un estudio de la arquitectura de los algoritmos genéticos y las técnicas utilizadas para el desarrollo de la aplicación.

Capítulo IV: Implementación

En este apartado se documenta la implementación de las técnicas expuestas en el apartado anterior para el desarrollo del algoritmo.

Capítulo V: Resultados

A lo largo de este capítulo, se evalúan los resultados obtenidos con las distintas configuraciones implementadas del algoritmo.

Capítulo VI: Evaluación del sistema

En este apartado se realiza una evaluación final de la evolución del algoritmo y se comparan los resultados obtenidos frente a la solución ya existente.

Capítulo VII: Conclusiones

En este capítulo se justifican los objetivos cumplidos en el proyecto.

Capítulo VIII: Bibliografía

En este último apartado se presentan las referencias que consultadas para el desarrollo del documento del proyecto.

2. Antecedentes

A continuación se expondrán dos apartados. En el primero se presenta el estado actual de las principales aplicaciones de formación de coaliciones y en qué posición se encuentra el proyecto dentro de este campo; y en el segundo, se exponen las distintos conocimientos en los que se basa el proyecto para su desarrollo.

2.1. Estado del arte

En [1], se presenta una política asistida por ordenador para formar equipos disjuntos de estudiantes basada en inteligencia colectiva, generación de estructuras de coaliciones y aprendizaje bayesiano. Específicamente, los estudiantes deben puntuar al resto de compañeros de equipo basándose en el inventario de Belbin al finalizar cada actividad en grupo. La información relativa a los roles predominantes de cada estudiante es actualizada a través del aprendizaje bayesiano. Los equipos son formados siguiendo un mecanismo de formación de coalición que emplea la información inferida a través del proceso de aprendizaje bayesiano. El principal problema de este método es el tiempo de computación para la formación de grupos en medida que el tamaño de la clase aumenta. La solución que se plantea en el presente trabajo consiste en implementar un algoritmo genético que pueda resolver la agrupación de alumnos siguiendo el criterio de roles de Belbin en menor tiempo de cómputo.

[2] En este trabajo se presenta un marco general para la creación de equipos en entornos educativos. Se mencionan diferentes criterios de agrupación como pueden ser la personalidad basada en el modelo MBTI o en el modelo Big Five, en roles de Belbin o en el rendimiento de los estudiantes o en sus preocupaciones y expectativas entre otros. Sin embargo, en este trabajo no se profundiza en la estrategia para la búsqueda de la mejor configuración de equipos en base a los criterios de agrupación presentados.

[3,4] En este trabajo se plantea el uso de una herramienta web para la asistencia a la hora de generar grupos de trabajo de manera automática cuando el número de estudiantes es elevado. En este trabajo los autores proponen el uso de un algoritmo de programación lineal (CPLEX) que encuentra la solución óptima. Este tipo de algoritmos implican un alto coste temporal en casos donde el número de individuos aumenta y hay que encontrar la mejor combinación en equipos. Los autores se basan en esta idea y la amplían en [1]. En este

trabajo evalúan la propuesta de la herramienta automática en el caso de la formación de grupos en un contexto real con un número limitado de estudiantes.

Otros trabajos consideran el uso de algoritmos genéticos para la generación de grupos. En [5] los autores proponen la aplicación de un algoritmo genético para la planificación de actividades en la tercera edad. La aproximación es parecida a la propuesta de este trabajo con la diferencia de las estrategias utilizadas en el genético, el dominio de aplicación y los criterios y restricciones para la creación de los equipos.

2.2. Algoritmos genéticos

Los algoritmos genéticos fueron desarrollados para resolver problemas en los que el espacio de solución es tan grande que un algoritmo de fuerza bruta demoraría demasiado tiempo. Por ejemplo, si se elige un número entre uno y mil millones. ¿Cuánto tiempo tomaría adivinarlo? Resolver un problema con fuerza bruta se refiere al proceso de comprobar todas las soluciones. Sin embargo, ¿y si podemos saber si la respuesta es buena o mala? Si se pudiera evaluar cómo encajar una suposición, se podría elegir otros números más cercanos a esa conjetura y llegar a la respuesta de manera más rápida, la respuesta podría evolucionar.

Por tanto, un algoritmo genético es un método de búsqueda que imita la teoría de la evolución biológica de Darwin para la resolución de problemas. Es un procedimiento computacional heurístico dedicado a la búsqueda estocástica, que no aleatoria, normalmente utilizado para resolver problemas de optimización con un alto grado de éxito [17][18].

Aunque las simulaciones por ordenador de los procesos evolutivos se remontan a los años cincuenta, mucho de lo que conocemos como algoritmos genéticos fue desarrollado por John Holland, profesor de la Universidad de Michigan, cuyo libro *Adaptation in Natural and Artificial Systems* [23] fue pionero en la investigación de algoritmos genéticos. Hoy en día, los algoritmos genéticos son parte de un campo de investigación más amplio, a menudo denominado "Computación Evolutiva".

2.2.1 Principios básicos

Antes de empezar a comprender el funcionamiento de un algoritmo genético, es importante tener en cuenta tres principios básicos de la evolución darwiniana los cuales serán requeridos para la implementación del algoritmo. Para que la selección natural ocurra como lo hace en la naturaleza, estos tres elementos deben estar presentes:

- **Herencia.** Proceso por el cual los hijos reciben las propiedades de sus padres. Si los individuos viven el tiempo suficiente para reproducirse, entonces sus rasgos se transmiten a sus hijos en la siguiente generación.

- **Variación.** Variedad de rasgos presentes en la población o un medio con el cual introducir la variación. Si la población no varía, los hijos serán siempre idénticos a sus progenitores. Por tanto, no se darán nuevas combinaciones de rasgos y la población no podrá evolucionar.

- **Selección.** Mecanismo por el cual algunos miembros de una población tienen mayor oportunidad de ser padres y transmitir su información genética y otros no. La selección natural opera sobre el principio de supervivencia del más apto, algunos rasgos están mejor adaptados al entorno de la población y por lo tanto tienen una mayor probabilidad de sobrevivir y reproducirse generando individuos más aptos.

2.2.2 Modo de funcionamiento

Un algoritmo genético realiza una serie de pasos de manera iterativa hasta cumplir un criterio de parada encontrando una solución real o una aproximación óptima. Las operaciones son las siguientes:

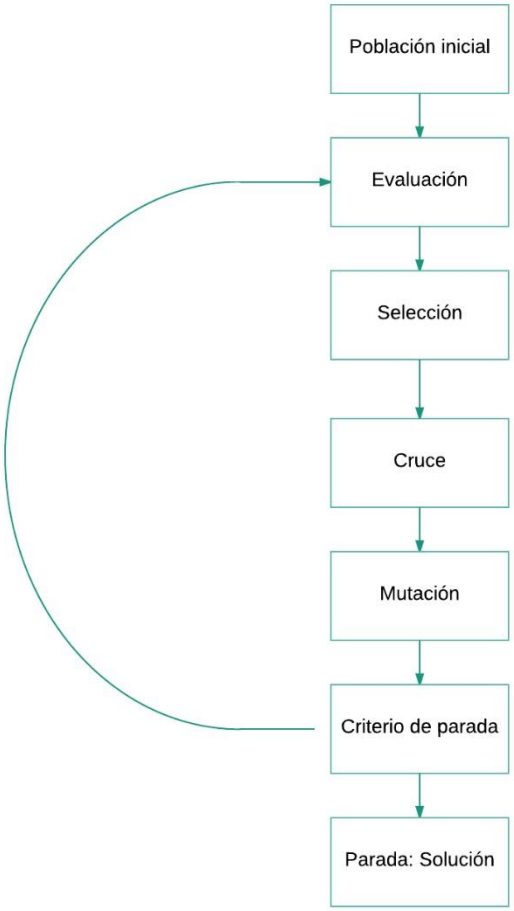

Figura 2: Diagrama de un algoritmo genético

- Población inicial: se genera un conjunto inicial de soluciones aleatorias al problema dado el cual se denominará población inicial. Se debe procurar que sea lo suficientemente grande para garantizar la diversidad de soluciones.

- Evaluación: se evaluará cada individuo (solución) con la función de aptitud la cual determinará su adaptación al medio, es decir, el valor que esta solución presenta al problema.

- Selección: se seleccionan n individuos para la generar la siguiente población dependiendo del método de selección. Los individuos más aptos se les da una mayor probabilidad de aparearse.

- Cruce: se cruzan los individuos seleccionados y se obtienen nuevas soluciones al problema.

- Mutación: sobre las nuevas generaciones existe cierta probabilidad de alguno de sus genes sufra una mutación como en la genética natural. En este momento, se reemplaza la población actual con los nuevos individuos obtenidos y se repite este proceso tantas generaciones hasta cumplir el criterio de parada.

- Criterio de parada: consiste en una serie de normas para detener la generación de individuos y devolver una solución al problema. Se puede fijar un número máximo de iteraciones o incluso detener la generación de soluciones cuando no se produzcan más mejoras en las soluciones generadas (convergencia del algoritmo).

2.2.3 Ventajas y desventajas

Los algoritmos genéticos tienen varias ventajas que los han hecho muy populares:

- No requieren ninguna información derivada (que puede no estar disponible para muchos problemas del mundo real).
- Son más rápidos y más eficientes en comparación con los métodos tradicionales.
- Tienen muy buenas capacidades paralelas.
- Optimizan las dos funciones continuas y discretas y también problemas multi-objetivo.
- Proporcionan una lista de soluciones "buenas" y no sólo una única solución.
- Siempre obtienen una respuesta al problema, que mejora con el tiempo.
- Útiles cuando el espacio de búsqueda es muy grande y hay un gran número de parámetros implicados.

Al igual que cualquier técnica, los algoritmos genéticos también sufren de algunas limitaciones:

- Los algoritmos genéticos no son adecuados para todos los problemas, especialmente los problemas que son simples y para los cuales la información derivada está disponible.
- El función de aptitud de las soluciones se calcula en repetidas ocasiones, donde se podría disparar el tiempo de coste de computación para algunos problemas.

- Siendo un proceso estocástico, no hay garantías sobre la calidad o lo óptima que puede ser la solución.
- Si no se ha aplicado correctamente, el algoritmo genético no puede converger a la solución óptima.
- La generación de las codificaciones para algunos problemas puede ser costosa.

2.3. Taxonomía de Belbin

Belbin [7] hace un profundo estudio sobre la influencia de los roles o tipos de comportamiento de las personas que forman un equipo de trabajo. En este trabajo se remarca la importancia de la composición de un equipo de trabajo para que el resultado sea satisfactorio. Entre las conclusiones se puede remarcar la de que equipos compuestos por roles poco heterogéneos tienden a producir resultados insatisfactorios.

Belbin admite que posiblemente hay infinitos patrones de comportamientos asociados a las personas, pero que el rango de comportamientos útiles, que realmente tienen una influencia efectiva en el grado de rendimiento del equipo de trabajo, es limitado. Más concretamente propone nueve patrones útiles (roles) en un equipo de trabajo:

- Generadores: Solucionan problemas mediante pensamiento creativo y enfoques poco ortodoxos.
- Investigadores de recursos: Comunicativos, extrovertidos e investigadores.
- Coordinadores: Pensamiento maduro, seguro y generalmente ayudan a promover objetivos y toman parte en el proceso de toma de decisiones.
- Moldeadores: Trabajan bien con presión y tienen la energía para superar obstáculos.
- Monitores: Tienden a evaluar todas las opciones con precisión.
- Corporativos: Tienden a ser cooperativos y diplomáticos dentro de un equipo.
- Implementadores: Transforman ideas en acciones. Generalmente son disciplinados, eficientes y conservadores.
- Finalizadores: Constantemente están buscando errores y descuidos.
- Especialistas: Proporcionan habilidades específicas y conocimiento técnico.

Para que un equipo sea eficaz, es importante que haya una distribución heterogénea de ellos. Basándose en este modelo de patrones de comportamiento en un equipo de trabajo, a continuación se presentará la herramienta desarrollada en [1] para la formación de equipos de trabajo heterogéneos.

2.4. Funcionamiento de la aplicación

La aplicación actual sigue una política basada en la inteligencia colectiva, la formación de coaliciones, y el aprendizaje bayesiano para formar distribuciones adecuadas de equipos de estudiantes. Esta política se ayuda de una aplicación de software que previene a los profesores de la costosa tarea de dividir alumnos en equipos óptimos o casi óptimos. El flujo de trabajo de las tareas seguidas por la política puede observarse en la figura 3. A continuación se describe este proceso con más detalle.

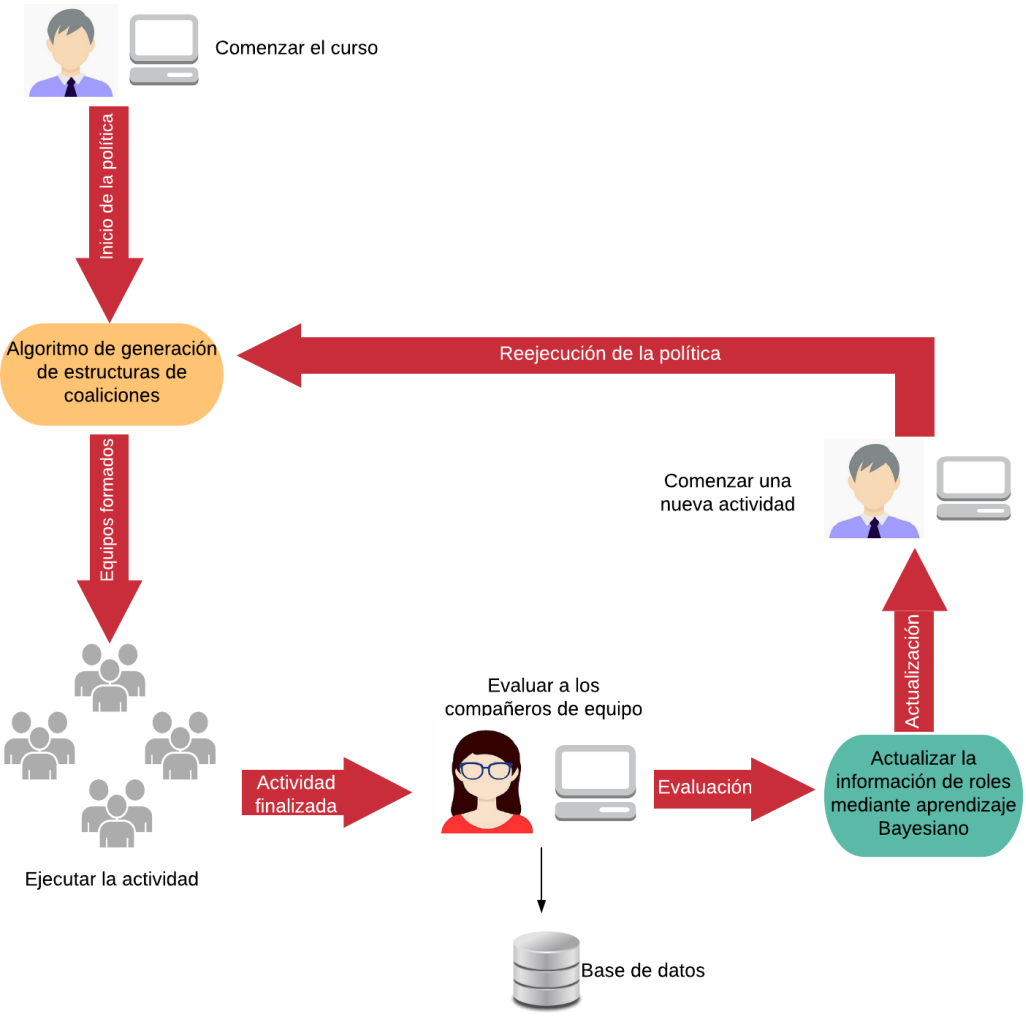

Figura 3: Diagrama de la aplicación

1. Al comienzo del curso, no hay información previa sobre las preferencias naturales de cada estudiante por uno de los roles de Belbin. La política empieza con la primera actividad en grupo del curso.

2. El profesor indica a la herramienta del ordenador que hay una nueva actividad en grupo. Entonces, los estudiantes son divididos en grupos separados para la tarea a mano. Como se observará en el siguiente apartado, el problema de dividir alumnos en equipos óptimos y disjuntos se funde con un algoritmo de generación de estructuras de coalición.
3. A continuación, la actividad se lleva a cabo.
4. Una vez finalizada la actividad, cada estudiante debe iniciar sesión en la aplicación de software. Entonces, a él/ella se le muestra una descripción completa de los roles de Belbin. En este punto, el estudiante debe clasificar cada compañero de equipo en un rol. Esta información es recolectada por la aplicación y almacenada en una base de datos.
5. Después de la evaluación por pares, la aplicación actualiza la información para cada estudiante. Entonces, se aplica el aprendizaje bayesiano para determinar qué roles son más predominantes para cada estudiante.
6. El proceso continúa hasta que el curso ha finalizado.

2.5. Política de formación de equipos empleada

En esta sección, se formaliza nuestro plan de acción para dividir estudiantes en grupos disjuntos. Primero se describe cómo dividir estudiantes en equipos óptimos es equivalente a un problema de generación de estructuras de coaliciones. Después, cómo el aprendizaje Bayesiano es empleado con el fin de actualizar la información de la clase.

2.4.1 Formación de grupos

Sea $A = \{a_i, \ldots, a_n\}$, un conjunto de estudiantes, y $R = \{r_i, \ldots, r_n\}$ sea un conjunto de roles que el estudiante puede interpretar (en nuestro caso el conjunto de roles de Belbin), y sea rol_i el verdadero papel predominante de a_i. El subconjunto $T \epsilon A$ se llama equipo, y la estructura de equipo $S = \{T_1, T_2 \ldots, T_n\}$ es una partición de equipos disjuntos tal como $\cup_{\forall Tj \in S} T_j = A$ y $S \epsilon 2^A$.

El objetivo de la aplicación es determinar una estructura de equipo óptima para la clase $\underset{S \in 2^A}{argmax}\, v(S)$, donde $v(S)$ es una función de evaluación para la estructura de equipo.

En este estudio, se considera que la calidad de cada equipo es independiente de otros equipos. Por tanto, se calcula el valor de la estructura de equipo como $v(S) = \sum_{Tj \in S} v(T_j)$. El valor de un equipo $v(T_j)$ puede calcularse atendiendo al rol predominante que cada estudiante $a_i \epsilon T_j$ tiene ($role(a_i)$). $|T_j| = k$ denota el tamaño del equipo $\pi_j = \{r'_1, \ldots, r'_k\}$ con $\forall r'_i \epsilon R$, un vector con el verdadero rol predominante de cada miembro del equipo. En este caso, $v(T_j) = v(\pi_j)$.

De acuerdo a diferentes estudios, el equipo debe beneficiarse de tener una distribución de roles balanceada (por ejemplo: una persona por rol). Esta puntuación puede ser proporcionada por un experto.

Desafortunadamente, no es posible saber exactamente el rol predominante de cada miembro del equipo π_j y por lo tanto $v(\pi_j)$ no puede calcularse con precisión. Sin embargo, es posible calcular una estimación del valor de la coalición dada la historia de evaluaciones H que es recolectada de los estudiantes durante el curso.

Sea $\pi_j = \{role_1 = r'_1, \ldots, role_k = r'_k\}$ un vector que contiene un conjunto de hipótesis para los roles predominantes de cada miembro del equipo, y Π sea el conjunto de todos los posibles vectores de hipótesis para los roles predominantes de T_j. En ese caso, podemos calcular el valor esperado de un equipo dado el historial de evaluaciones como:

$$\check{v}(T_j|H) = \sum_{\pi' \in \Pi} p(\pi'|H) \times v(\pi') = \sum_{\pi' \in \Pi} v(\pi') \times \prod_{a_i \in T_j} p(rol_i = r'_i|H) \qquad (1)$$

Donde $p(\pi'|H)$ representa la probabilidad para π' de ser la distribución de roles real en T_j dado el historial de evaluaciones H. Cada $p(\pi'|H)$ puede dividirse en su $p(rol_i = r'_i|H)$ dado que asumimos que el rol de cada estudiante es condicionalmente independiente dado el historial de evaluaciones.

Por lo tanto, nuestro problema de formación de equipo en cada iteración se funde en un problema que sigue la siguiente expresión:

$$\underset{S\in 2^A}{argmax} \sum_{T\in S} \check{v}(T|H) \tag{2}$$

Resulta que particionar estudiantes en grupos disjuntos mientras se optimiza una función de bienestar social se corresponde a la formalización de problemas de generación de estructuras de coalición. Para los experimentos de simulación, se formaliza el problema de generación de estructuras de coalición como un problema lineal de programación y se soluciona con el software comercial ILOG CPLEX 12.5[1].

2.4.2 Aprendizaje Bayesiano

Después de cada actividad, los estudiantes evalúan sus pares estableciendo el rol predominante de cada uno de sus compañeros de equipo. Entonces, la nueva información respecto al rol predominante de cada estudiante se vuelve disponible y el historial de evaluaciones H crece. Por lo tanto, en cada iteración podemos actualizar información respecto a la probabilidad de un agente a_i a tener r_i' como su rol predominante dado el historial de evaluaciones $p(role_i = r_i'|H)$.

Se emplea aprendizaje Bayesiano para este asunto:

$$p(role_i = r_i'|H) = \frac{p(H|role_i = r_i') \times p(role_i = r_i')}{\sum_{r\in R} p(H|role_i = r_i') \times p(role_i = r)} \tag{3}$$

Donde $p(H|role_i = r_i')$es la función de probabilidad y $p(role_i = r_i)$ es la probabilidad previa para la hipótesis. Para la función de probabilidad, se puede calcular como $p(H|role_i = r_i') = \frac{\#\{r_i'\in H_i\}}{|H_i|}$, donde H_i denota las evaluaciones de pares del agente a_i, y $\#\{r_i' \in H_i\}$ indica el número de veces que r_i' aparece como evaluación en H_i. En cuanto a la anterior probabilidad, se calcula como $p(role_i = r_i) = \frac{\#\{r_i'\in H_i\}}{|H|}$. El suavizado de Laplace se emplea para asegurar que la probabilidad para cada hipótesis de rol pueda ser calculada en las primeras iteraciones.

[1] http://www.ibm.com/software/commerce/optimization/cplex-optimizer/

2.4.3 Problemática

Una vez expuesto el modo de empleo de la aplicación y la política empleada para la formación de estructuras de coaliciones queda destacar cuales son las bondades y las desventajas de este método.

La herramienta es funcional y ha sido implementada en el curso 2013-2014 del grado de Turismo de la Universitat Politècnica de València, donde en [3] y [4] se muestra como la formación de grupos heterogéneos influye finalmente en el rendimiento de los alumnos en clase, como bien exponía Belbin.

El problema de esta aplicación, reside en el tiempo de cómputo de la formación de los grupos. Uno de los peores casos es el formado por una clase de 24 alumnos y 6 alumnos por grupo donde el algoritmo demora aproximadamente 5h para la distribución de los equipos. Por este motivo, se planteó mejorar este punto de aplicación con algoritmos genéticos, el cual es el objetivo principal de este proyecto.

Por estos motivos, se propone el uso de los algoritmos genéticos en base a las bondades expuestas anteriormente como la rapidez de cómputo y la amplia capacidad de exploración del espacio de búsqueda. Aunque los algoritmos genéticos no presentan una única solución, puede que esta ventaja no sea útil para la resolución del problema ya que se considerará una solución real aquella que contenga todos los grupos con alumnos con roles diferentes.

3. Diseño del Algoritmo Genético

En este apartado se describen las diferentes técnicas empleadas en cada una de las fases de los algoritmos genéticos que les permite ser flexibles y aptos para obtener soluciones óptimas en multitud de problemas.

3.1. Codificación

Un aspecto importante para el éxito de un algoritmo genético suele ser su codificación. La codificación depende del problema a resolver, no es un tema trivial, se debe conocer qué estructura de datos se necesita y cómo se va a visualizar y entender la representación de las posibles soluciones del problema.

Cualquier solución potencial a un problema debe ser representada a través de una serie de parámetros, los genes, codificados en una cadena de valores denominada cromosoma. El gen hace referencia a la posición de un elemento de un cromosoma y su valor es denominado alelo como se observa en la figura 4.

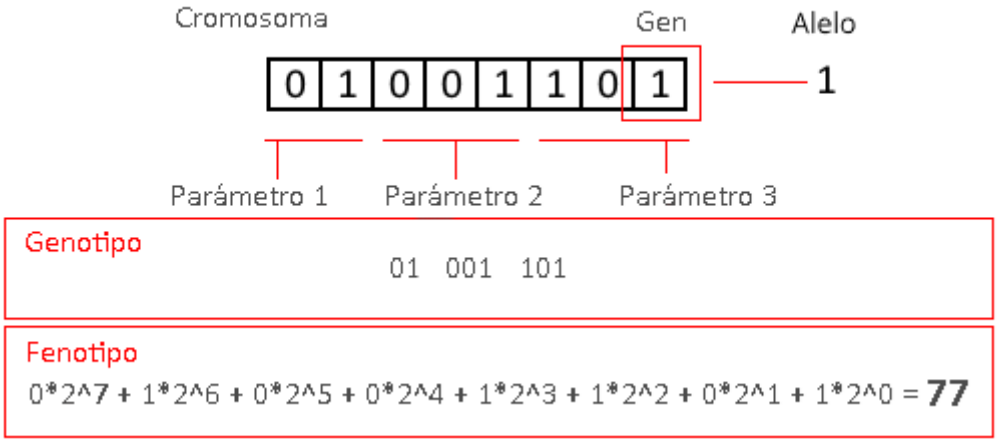

Figura 4: Ejemplo de un cromosoma

El conjunto de los parámetros representado por un cromosoma particular recibe el nombre de genotipo. Este genotipo construye la solución al problema en el espacio del cálculo de manera que un computador pueda comprender los datos y pueda manipularlos. Por otra parte, el fenotipo representa la solución en el mundo real.

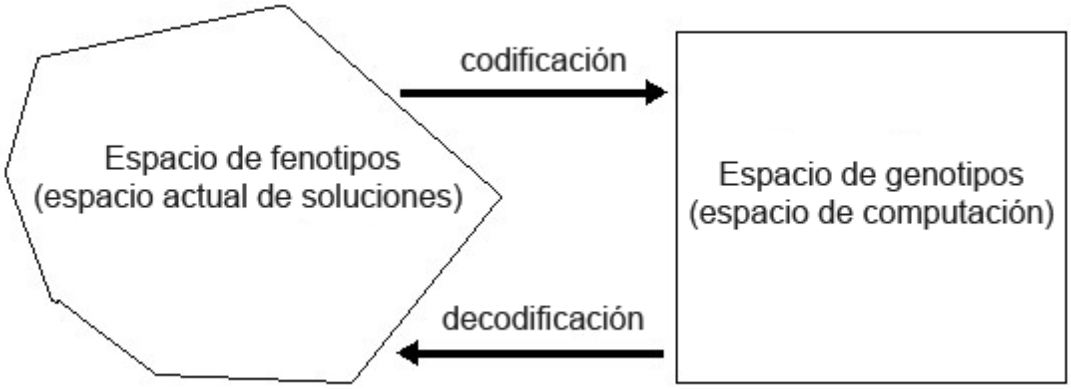

Figura 5: *Relación entre genotipo y fenotipo*

Existen problemas sencillos en los que el genotipo y el fenotipo son representados del mismo modo. No obstante, existen otros casos en los que se requiere de una codificación y decodificación rápida para pasar de un espacio de soluciones al otro. La decodificación es el proceso que transforma el genotipo en fenotipo y la codificación el caso contrario.

Desde los primeros trabajos de John Holland [23] la codificación binaria es una de las representaciones más simple y más ampliamente utilizada. Se asigna un determinado número de bits a cada parámetro y se realiza una discretización de la variable representada por cada gen. Evidentemente no todos los parámetros tienen por qué estar codificados con el mismo número de bits.

También existen representaciones que codifican directamente cada parámetro con un valor entero, real, punto flotante, etc. Estas codificaciones permiten el desarrollo de operadores genéticos más específicos al campo de aplicación del Algoritmo Genético.

En algunos problemas, la solución está representada por un orden de los elementos como es el caso del problema del agente viajero (TSP, travelling salesman problem). Este tipo de representación es comúnmente denominada permutación. En este problema el viajero debe recorrer todas las ciudades una única vez y volver a la ciudad inicial. La representación de la solución a este problema es la permutación de las ciudades a visitar.

Para el caso de estudio en este documento, se precisa generar una solución que contenga una agrupación de todos los alumnos de una clase en x grupos donde el número de los alumnos por grupo venga predefinido por el usuario y que cada uno de los integrantes del grupo contenga un rol distinto siguiendo el criterio de Belbin.

Tras conocer la naturaleza del problema, la representación de las soluciones se realizará por permutación. El tamaño del cromosoma representará el tamaño de la clase, donde cada gen representará a un único alumno, es decir, para una clase de 24 alumnos tendremos un cromosoma de 24 genes donde cada alelo represente el grupo al que pertenece cada alumno. El orden de los genes dentro del cromosoma representa el orden de los alumnos dentro del aula, por ejemplo, el alumno 2 corresponde al gen 2 y así hasta llegar al tamaño del aula que se requiera evaluar.

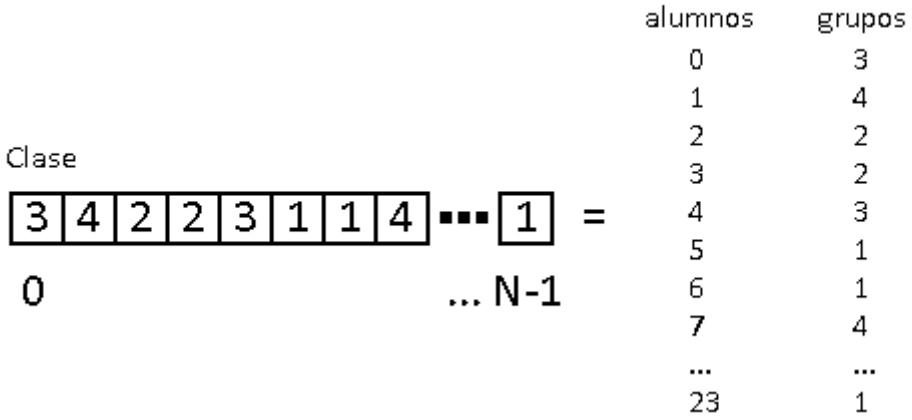

Figura 6: Codificación de una clase

Uno de los criterios de la evaluación de las soluciones generadas será comparar el número de roles distintos que existen dentro de cada grupo formado. Al inicio del programa se generará una lista con los roles correspondientes a cada alumno de la clase del mismo modo que se codifican los grupos de clase. Se evalúan ocho tipos de roles distintos y su orden dentro de la lista pertenecerá al número de alumno correspondiente.

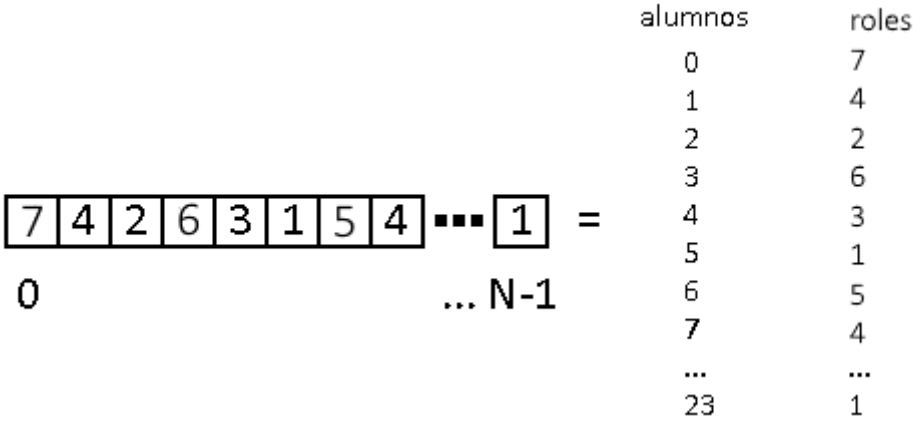

Figura 7: Codificación de los roles de la clase

3.2. Población

La población es un subconjunto de soluciones (cromosomas) en la generación actual, se define generalmente como una matriz bidimensional de N soluciones por el tamaño de éstas.

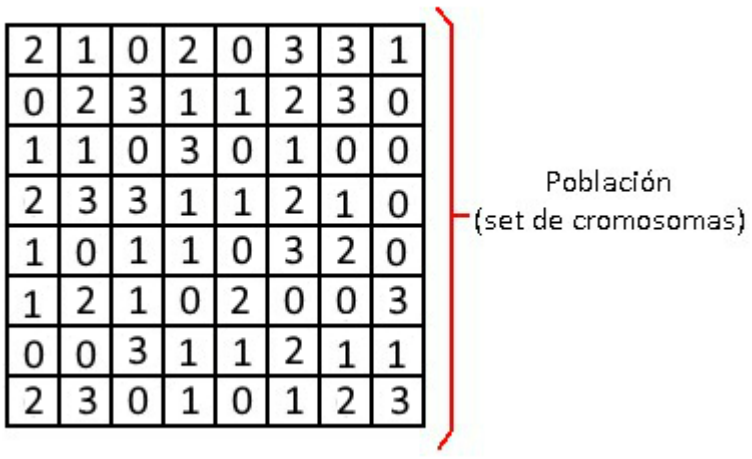

Figura 8: Ejemplo de población

En el caso de que el tamaño de la población sea insuficiente, el algoritmo genético tiene pocas posibilidades de realizar reproducciones con lo que se realizaría una búsqueda de soluciones escasa y poco óptima. Por otro lado si la población es excesiva, el algoritmo genético será excesivamente lento. Por lo tanto, un tamaño óptimo de la población tiene que ser decidido por ensayo y error.

Habitualmente la población inicial se escoge generando ristras al azar, pudiendo obtener cada gen uno de los posibles valores del alfabeto con la misma probabilidad. Por el contrario, la inicialización no aleatoria de la población, puede acelerar la convergencia del algoritmo genético obteniendo máximos locales.

Para este problema se evaluará la relación del tamaño de la población, su tiempo de cómputo y el resultado obtenido. Las población inicial se generará de forma aleatoria, pudiendo elegir entre un reparto equitativo de los grupos o aleatorio, siendo constante siempre el número de roles de la clase ya que es la condición indispensable para la resolución de nuestro problema.

3.3. Función de aptitud y función de evaluación

Para evaluar las soluciones generadas se hace uso de una función de evaluación. Ésta refleja el valor real de la solución. También se utiliza la función de aptitud o fitness, la cual suele ser más simple y evalúa el grado de adaptación de los individuos. Ambas soluciones suelen ser iguales, pero puede que la función de evaluación sea más compleja, tome valores negativos o no proporcione un valor numérico y por lo tanto sea necesario definir una función de aptitud diferente. Para ello es necesario decodificar la solución presente (genotipo) en el cromosoma para representarla como se ha explicado anteriormente, ver figura 5.

Dependiendo de la naturaleza del problema se pueden proponer distintos tipos de funciones objetivo. La dificultad reside en la existencia de la gran cantidad de óptimos locales así como el hecho de que el óptimo global se encuentre muy aislado. Para ello se puede elaborar la función de evaluación tanto como sea necesario.

Para este problema los principales requisitos para construir una solución son: cumplir el número de grupos deseado, el número de alumnos por grupo y obtener el máximo número de roles distintos por grupo. Se partirá de una función básica que evalúe estos tres requisitos y se irá elaborando en función de los resultados obtenidos.

$$Función\ de\ Evaluación = \alpha\frac{g_{gruposFormados}}{G_{gruposEsperados}} + \beta\frac{a_{alumnosPorGrupo}}{A_{alumnosPorGrupoEsperados}} + \gamma\frac{r_{rolesDistintos}}{R_{rolesEsperados}} \quad (4)$$

Si un individuo cumple el 100% de estos requisitos se considerará una solución real, pues el objetivo de este algoritmo es crear grupos de alumnos con el tamaño deseado y que todos los componentes tengan roles distintos.

3.4. Selección

El proceso de selección es el encargado de escoger qué individuos van a disponer de oportunidades de reproducirse y cuáles no. Siguiendo las leyes de la naturaleza, se concede mayor probabilidad de reproducción a los individuos más aptos. Sin embargo, no se debe eliminar por completo los individuos menos aptos, ya que en pocas generaciones la población se volvería homogénea. Por lo tanto, la selección de un individuo estará relacionada con su valor de aptitud.

El método utilizado en este proyecto es la selección por torneo. Se implementará y evaluará las dos versiones del mismo: la opción determinista y la probabilística. En la versión determinística se selecciona al azar un número n de individuos (generalmente se escoge n=2) y de entre los seleccionados se escoge el más apto para generar descendencia.

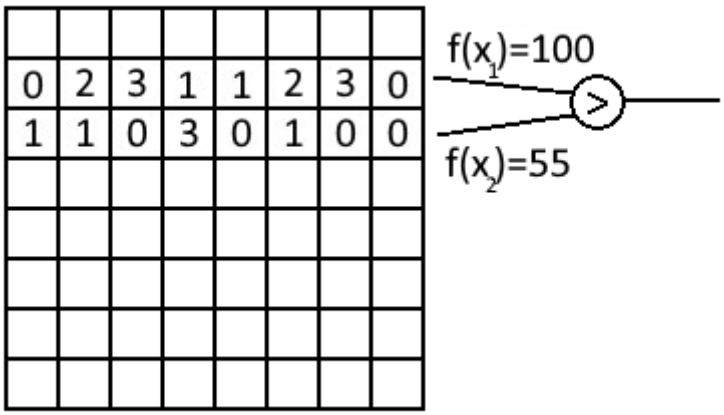

Figura 9: Selección de dos individuos

La versión probabilística únicamente se diferencia en el paso de selección del ganador del torneo. En vez de escoger siempre el mejor se genera un número aleatorio del intervalo [0,1], si es menor que un parámetro p (fijado para todo el proceso evolutivo) se escoge el individuo más alto y en caso contrario el menos apto.

3.5. Cruce

Una vez seleccionados los individuos, éstos son recombinados para producir la descendencia que se insertará en la siguiente generación. Si en el procedimiento anterior se han seleccionado los individuos más aptos, su descendencia al compartir las características buenas de ambos, es probable que obtenga una bondad igual o mayor que cada uno de los padres por separado.

Existen diversas técnicas para realizar el cruce o crossover. La más sencilla es el cruce por un punto. Se dividen los individuos seleccionados por un punto elegido de manera aleatoria, generando dos segmentos en cada uno de ellos: la cabeza y la cola. Se intercambian las colas entre los dos individuos para generar los nuevos descendientes.

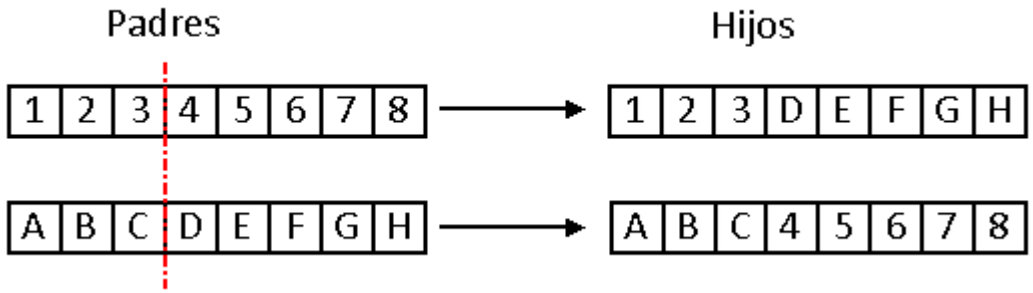

Figura 10: Cruce por un punto

Otra técnica es el cruce por dos puntos, en vez de cortar por un único punto los cromosomas de los padres, se realizan dos cortes. Para generar la descendencia se escoge el segmento central de uno de los padres y los segmentos laterales del otro padre.

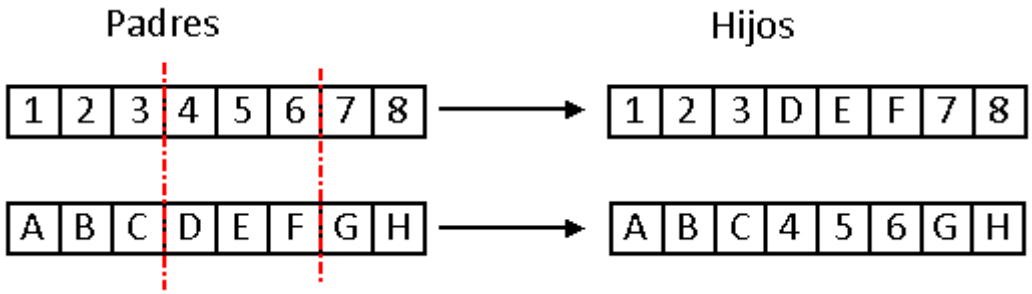

Figura 11: Cruce por dos puntos

Se pueden añadir más puntos de cruce dando lugar a algoritmos de cruce multipunto. Sin embargo, añadir nuevos puntos de cruce radica en que es más fácil que los segmentos originados pierdan las características de bondad que poseían conjuntamente. Este problema se analizará en el proyecto, ya que, cada gen representa a un alumno con su grupo y su rol, la dependencia entre estos factores puede que sea delimitante.

Otra técnica presente es la denominada cruce uniforme, donde los genes de la descendencia tienen la misma probabilidad de pertenecen a uno u otro padre. Una de las implementaciones más comunes consiste en generar una máscara binaria, variable o no durante todo el proceso evolutivo. Si en una de las posiciones de la máscara hay un 1, el gen situado en esa posición en uno de los descendientes se copia del primer padre. Si por el contrario hay un 0 el gen se copia del segundo padre. Para el segundo descendiente se intercambian los papeles de los padres, o bien se intercambia la interpretación de los unos y los ceros de la máscara de cruce.

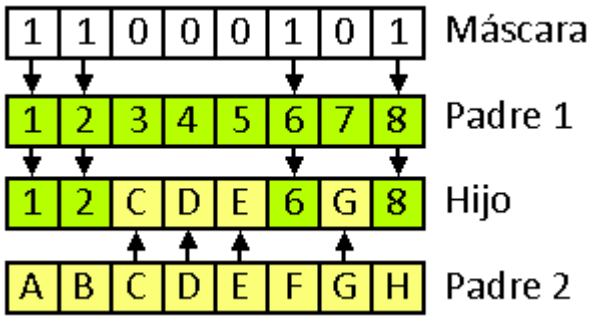

Figura 12: Cruce uniforme

3.6. Mutación

La mutación de un individuo consiste en el cambio de alguno de sus genes, generalmente uno, en un valor aleatorio. Suele aplicarse una vez realizado el cruce, en este momento, los descendientes tienen una probabilidad de mutar. De este modo se introduce y mantiene la diversidad genética en la población evitando la rápida convergencia del algoritmo. La probabilidad de mutación suele ser muy baja, por debajo del 1%, pero esta operación puede permitir que ningún punto del espacio de búsqueda quede sin inspeccionar.

Existen varios métodos para aplicar la mutación a los individuos de una población, pero los más comunes son: reemplazo aleatorio e intercambio de valores. El reemplazo aleatorio consiste en variar de manera aleatoria el valor de un alelo del cromosoma, figura 13. Por otro lado, el intercambio de valores, como su propio nombre indica, intercambia los valores de dos alelos del mismo cromosoma figura 14.

Figura 13: Mutación aleatoria

Figura 14: Mutación por intercambio de valores

3.7. Reemplazo

Una vez realizado el cruce, se obtiene un número de nuevos individuos a integrar dentro de la población para formar la siguiente generación. Se puede optar por reemplazar a los padres aunque estos tengan mejor adaptación. Por otra parte, se puede insertar aquellos descendientes que superen la bondad de sus progenitores o incluso permitir algún porcentaje de que los peores pasen para que los valores no sean tan homogéneos.

En algunos casos se emplea el elitismo, este método consiste en copiar uno o varios de los mejores individuos de la generación actual en la siguiente. De este modo se garantiza que en la siguiente iteración no se empeora la calidad de la población.

4. Implementación

Para resolver el problema y poner en práctica las distintas ideas expuestas en el apartado anterior, se ha desarrollado un script desde cero que permite parametrizar el Algoritmo Genético y evaluar los resultados obtenidos probando las distintas técnicas implementadas en los operadores genéticos. Se ha utilizado el lenguaje de programación Python para el desarrollo del algoritmo debido a su sencillez y versatilidad.

4.1. Inicialización

Para evaluar el algoritmo en distintos escenarios, se parametrizan las siguientes variables:

- numPoblacion: tamaño de la población inicial.
- tamClase: tamaño de la clase, número de 24 y 48 alumnos.
- alumGrupo: número de alumnos por grupo, se evaluarán casos de 4 y 6 alumnos por grupo.
- numRoles: número de roles distintos
- repartoRol: reparto de roles, si existe el mismo número de roles de cada tipo o bien se generan aleatoriamente.
- repartoGrupos: reparto de grupos, si existe el mismo número de grupos de cada tipo o bien se generan aleatoriamente.
- Fitness: la función de evaluación puede realizar la media lineal o ponderada.
- umbralSeleccion: umbral de selección para la función probabilística de la selección por torneo.
- reemplazoPadres: para la generación de la nueva población, reemplazando a los progenitores o conservarlos.
- nPadres: número de padres que participan en el torneo.
- divisiones: tipo de divisiones en el crossover, por un punto, dos o cruce uniforme.
- probMutacion: probabilidad de la mutación.
- tipoMutacion: tipo de mutación, aleatoriedad del valor de un gen o intercambio de los valores.
- maxIterIguales: número máximo de iteraciones en las que la utilidad máxima es la misma.
- numMaxIter: número máximo de generaciones (iteraciones) del bucle principal.

Una vez descritos los principales parámetros de evaluación del algoritmo genético, a continuación se presenta en pseudocódigo el funcionamiento global del algoritmo diseñado:

```
main():
    laPoblacion  = crear_Poblacion
    while(true):
        listaPuntuados = evaluar_Poblacion
        ordenar (listaPuntuados)
        mejorPuntuacion = listaPuntuados[0]
        SI cumple criterio de parada:
            break
        SI NO cumple criterio de parada
            SI reempladoPadres = True
                repetirSeleccion = nPadres
            SI NO
                repetirSeleccion = nPadres / 2
            for i in range (repetirSeleccion):
                padresSeleccionados = seleccionar_Padres(nPadres)
            nuevaPoblacion, viejaPoblacion = cruzarPadres(padresSeleccionados)
            nuevaPoblacion += nuevaPoblacion
            viejaPoblacion += viejaPoblacion
            SI reempladoPadres = True
                laPoblacion = nuevaPoblacion
            SI NO
                laPobacion = nuevaPoblacion + viejaPoblacion
            iteraciones ++
```

Figura 15: Pseudocódigo del algoritmo principal

4.2. Codificación de las soluciones

Una vez realizada la inicialización del algoritmo se genera la población inicial mediante la función *crearPoblacion()*. Este método genera N listas de soluciones del tamaño de la población inicial llamando a la función auxiliar *crearGrupos()*. Además, se generará una lista numérica única, mediante la función *crearRoles()*, la cual representará los roles de los alumnos de la clase.

4.2.1 Crear grupos

Para la codificación de las soluciones (cromosomas) se utiliza la función *crearGrupos()* la cual genera una lista numérica del tamaño de la clase con valores entre 0 y N-1 grupos. Por ejemplo, para una clase de 24 alumnos y 4 grupos generados, se obtiene:

[0, 1, 2, 1, 2, 0, 3, 1, 3, 3, 2, 0, 0, 1, 2, 1, 0, 1, 2, 1, 2, 0, 3, 1]

Dependiendo de la variable *repartoGrupos* el reparto de grupos puede ser:

- No aleatorio: mismo número de grupos y alumnos por grupo.
  [0, 0, 1, 1, 2, 2]
- Aleatorio: siempre el mismo número de grupos de grupos pero no alumnos por grupo.
  [2, 0, 1, 0, 0, 1]
- Aleatorio Total: ni número de grupos de grupos ni alumnos por grupo.
  [2, 0, 1, 3, 4, 5]

El número de grupos se calcula a partir del tamaño de la clase y el número de alumnos por grupo deseados. Para una generación robusta a distintas configuraciones se sigue la siguiente lógica en el reparto de grupos en el caso de reparto equitativo:

1. Si el número de alumnos en la clase es divisible por el número de alumnos por grupo, se realiza un reparto equitativo:

   8 Alumnos y 4 Alumnos por grupo → [0, 0, 0, 0, 1, 1, 1, 1]

2. Si el número de alumnos en la clase o el número de alumnos por grupo es impar y genera un alumno sin agrupar, este se une a uno de los grupos formados de manera aleatoria:

   9 Alumnos y 4 Alumnos por grupo → [0, 0, 0, 0, 1, 1, 1, 1, **X**]

3. Si el número de alumnos sin agrupar es par, se reparten de modo equitativo entre los grupos formados:

   10 Alumnos y 4 Alumnos por grupo → [0, 0, 0, 0, 1, 1, 1, 1, **0, 1**]

4. Si el número de alumnos sin grupo es mayor a la mitad de alumnos por grupo, se genera un grupo nuevo:

   11 Alumnos y 4 Alumnos por grupo → [0, 0, 0, 0, 1, 1, 1, 1, **2, 2, 2**]

5. En los casos en los que sólo existe un grupo y queda algún alumno sin agrupar, por conveniencia, se entiende que estos datos proporcionados no son erróneos por lo que se crearían dos grupos:

   5 Alumnos y 4 Alumnos por grupo → [0, 0, 0, 0, **1**]

4.2.2 Crear roles

Como se ha expuesto anteriormente, la función encargada de generar la lista de roles de los grupo es crearRoles(). Se creará una lista del tamaño de la clase con 8 tipos de roles como indica el criterio de Belbin, los valores están comprendidos entre 0 y 7. El reparto de estos valores será aleatorio o equitativo dependiendo de la variable repartoRol.

En los primeros experimentos se generará un reparto equitativo para comprobar que existe una solución real, es decir, que exista el caso en el que todos los grupos obtengan alumnos con roles distintos. Se puede dar el caso en el que el número de roles distintos no permita formar grupos sin repetir algún rol, en ese caso no podemos conseguir el 100% de la utilidad y por tanto evaluar el rendimiento del algoritmo diseñado.

Por otra parte, se evaluará el caso más complejo para la formación de los grupos para estudiar las capacidades del algoritmo. El peor de los casos con posibilidad de obtener una utilidad del 100% es aquel que contiene el mínimo número de roles distintos necesario para que todos los grupos no repitan roles. Para este caso se generará el mismo número de roles que de alumnos por grupos, es decir, si los grupos están formados por 4 alumnos, sólo existirán 4 tipos de roles, así pues, la única solución posible será formar tantos grupos de 4 alumnos como el tamaño de la clase lo permita y que cada uno de ellos contenga los 4 tipo de roles distintos.

4.3. Función de evaluación

Como se ha expuesto en el capítulo anterior, en este problema la función de evaluación y de aptitud (fitness) será la misma. Para evaluar la población se hace uso de la función *evaluarPoblación()*, ésta devuelve la lista con la utilidad de todas las soluciones de la generación actual. Para que esto sea posible, se hace uso de las siguientes funciones auxiliares:

- getInfoRolesGrupo
    - getNumAlumnosFormadosPorGrupo
    - getNumRolesDistintos
- calcularFitness
    - evaluarClasePorNumGrupos
    - evaluarClasePorAlumnosGrupo
    - evaluarClasePorRolesDiferentes

4.3.1 Get información roles grupo

La función getInfoRolesGrupo extrae información necesaria para el cálculo de la función fitness llamando a las siguientes funciones auxiliares:

- getNumAlumnosFormadosPorGrupo: devuelve una lista con el número de alumnos pertenecientes a cada grupo de la clase. Se observa como en el ejemplo, el número de alumnos pertenecientes al grupo 2 son 3 y para el resto de grupos sólo existe 1 alumno.
- getNumRolesDistintos: devuelve una lista con el número de roles distintos de cada grupo de la clase. En el ejemplo se observa como en el grupo 2 donde hay 3 alumnos sólo existen 2 roles distintos.

Ejemplo:

gruposClase = [0, 1, 1, 1, 2, 3]
rolesClase = [0, 1, 1, 2, 2, 3]
getInfoRolesGrupo(gruposClase, rolesClase)
resultado:
alumnos por grupo [1, 3, 1, 1] roles distintos por grupo [1, 2, 1, 1]

4.3.2 Calcular Fitness

Los principales requisitos para constituir una solución en este caso práctico son: cumplir el número de grupos deseado, el número de alumnos por grupo y obtener el máximo número de roles distintos por grupo. Para ello, se utiliza la función c*alcularFitness()* para realizar el cálculo de la función de evaluación (4).

Con los datos obtenidos en el apartado anterior (alumnos por grupo y roles distintos por grupo), la función *calcularFitness()* hace uso de las siguientes funciones auxiliares para el cálculo de la aptitud de las soluciones:

- evaluarClasePorNumGrupos(): calcula la relación entre grupos formados y el número de grupos esperados.
- evaluarClasePorAlumnosGrupo(): calcula la relación entre números de alumnos obtenidos por grupo y el número de alumnos por grupo esperados.
- evaluarClasePorRolesDiferentes(): calcula la relación entre los roles distintos obtenidos y los esperados.

Una vez obtenidos estos resultados, comprendidos entre 0 y 1, se realizará el cálculo final de la función de evaluación. Si el valor de la variable tipoFitness es igual a "lineal" se realizará la media aritmética de los parámetros evaluados, si por otro lado, el valor de la variable es igual a "ponderada", se realizará la media ponderada como se indica en la ecuación 4.

En el caso de *evaluarClasePorNumGrupos()* el cálculo se realiza siguiendo (5), siendo nGruposFormados el número de grupos en la clase y nGrupos el número de grupos esperado. La diferencia máxima consiste en la resta entre el tamaño de la clase y el número de grupos deseado, de este modo, en el caso de que el número de grupos formados sea igual al número de alumnos de la clase la puntuación de esta utilidad será 0.

$$1 - \left| \frac{n\ GruposFormados - n\ Grupos}{diferencia\ Max} \right| \tag{5}$$

En la presente tabla se puede observar la distribución de puntuaciones para una clase de 10 alumnos y la formación de 4 grupos.

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0.5	0.67	0.83	1	0.83	0.67	0.5	0.33	0.17	0

Tabla 1: Distribución de puntuaciones

El caso de *evaluarClasePorAlumnosGrupo()* se realiza un cálculo muy similar al realizado anteriormente. Siguiendo la fórmula (6), siendo nAlumnosPorGrupoFormados el número de alumnos por grupo formados en la clase y nAlumnosPorGrupo el número de alumnos por grupo deseado. En este caso, la diferencia máxima consiste en la resta entre el tamaño de la clase y el número de alumnos por grupo deseado, de este modo, en el caso de que el número de alumnos por grupo formados sea igual al número de alumnos de la clase la puntuación de esta utilidad será 0.

$$1 - \left| \frac{n\ AlumnosPorGrupoFormados - n\ AlumnosPorGrupo}{diferencia\ Max} \right| \tag{6}$$

En este caso como se evalúa la formación de alumnos en cada grupo, el resultado obtenido tras llamar esta función será la utilidad media de cada grupo. Aunque el número de grupos totales no sea el deseado, en este apartado simplemente se calcula el número de alumnos para cada grupo.

Por último, en el caso de la función *evaluarClasePorRolesDiferentes(),* se evaluará la cantidad de roles diferentes dentro de cada grupo, es decir, se calculará el cociente entre número de roles distintos y el número de alumnos por grupo. Finalmente se devuelve la utilidad media de todos los grupos, al igual que en el caso anterior, no importa el número de grupos formados o el número de alumnos dentro de estos, ya que solamente interesa el número de roles distintos.

Si en la función de evaluación se cumplen los tres factores a evaluar, se obtendrá la utilidad máxima de la solución. Puede ocurrir que alguno de estos factores sea más fácil de cumplir que otros, para ello, siguiendo la función (4) se ajustarán los pesos finales en la ecuación para dar mayor importancia a los factores más complicados de obtener. Por ejemplo, si se obtienen todos los grupos deseados y todos ellos tienen roles distintos pero en cada grupo hay un número dispar de alumnos la solución no es válida. Estos casos serán estudiados en el siguiente capítulo.

4.4. Método de selección

Para la selección de los padres en el algoritmo genético se empleará, como se ha presentado en el capítulo anterior, la selección por torneo. La función *seleccionTorneo()* tendrá como parámetros de entrada una lista con las utilidades de la población actual, el número de padres que intervendrán en el torneo y un umbral de selección para la elección del padre ganador.

Si la variable *umbralSeleccion* es igual a 1, el método de selección se vuelve determinista eligiendo siempre al mejor individuo. Se evaluará este parámetro, si se consigue una convergencia rápida del algoritmo, se reducirá el valor de *umbralSeleccion* para obtener una selección de los padres probabilística.

La implementación de la selección de los padres por torneo se puede resumir en el siguiente pseudocódigo:

```
seleccionTorneo(puntuados, nPadres, umbral):
    //creamos una lista de índices del tamaño de los puntuados
    indices = listar(puntuados)
    //se desordena la lista
    random(indices)
    //se calcula el número de repeticiones para la seleccion de padres
    nVeces = len(puntuados) / nPadres
    //evaluamos los nPadres primeros y seguimos con los nPadres siguientes, nVeces
    Desde 0 hasta nVeces:
        //de los nPadres seleccionados se extraen sus utilidades
puntuados_indices = puntuados(nPadres)
```

```
puntuados_utilidades = puntuados(puntuados_indices)
        //selección el padre ganandor dependiendo del umbral
SI random(1,100)/100 es menor al umbral
            ganador = max(puntuados_utilidades)
        SI NO:
            ganador = min(puntuados_utilidades)
        //se añade el ganador a la lista
        ganadores.append(ganador)
return ganadores
```

Figura 16: Pseudocódigo de la selección por torneo

4.5. Método de cruce

Para el método de cruce se implementa la sección por un punto, dos puntos y uniforme con el fin de evaluar qué opción es más favorable. Las funciones implicadas en este apartado son:

- *cruzarPadres()*: es la función principal, la cual realiza el crossover de toda la población y en el caso de que la probabilidad de mutación sea mayor que 0, llama a la función de mutación.
- *crossover()*: esta función dependiendo del método elegido dividirá el cromosoma por un punto, dos o realizará el cruce uniforme.

El funcionamiento principal de la sección de cruce se puede resumir en los siguientes puntos:

1. Se hace uso de la lista de los padres seleccionados, se eligen en pares de manera aleatoria para generar la descendencia.
2. La función auxiliar *crossover()* se encarga de devolver los dos cromosomas generados a partir de dos soluciones padre.
3. La función *dividirClase()* se encarga sólo de dividir dos cromosomas y unirlos dependiendo de la variable divisiones, la cual indicará qué método de cruce emplear de los mencionados anteriormente.
4. Una vez realizado el crossover, se aplicará la mutación sobre los hijos generados en el caso de que la probabilidad de mutación sea mayor a 0.

4.6. Método de Mutación

Respecto al apartado de la mutación existe una única función llamada *mutarClase()* la cual se encarga de todo el proceso. Como se ha mencionado en el apartado anterior, ésta es llamada por la función *cruzarPadres()* para facilitar su uso en el mismo cruce de las soluciones. La implementación de esta función se resume en el siguiente pseudocódigo:

```
mutarClase(clase, probMutacion, aleatoria, numGrupos):
    SI random(1,100)/100 es menor o igual a la probabildadMutación
        SI (aleatoria == 1)
            indice_random = de 0 a tamaño de la clase - 1
            posibles_valores = lista de valores de 0 a numGrupos
            //para no mutar al valor actual
            posibles_valores = elimina de los posibles el valor actual del índice elegido
            clase[indice_random] =  posibles_valores[random]
        SI NO:
            //intercambio de valor entre 2 genes
            índices_disponibles = lista de 0 a tamaño de la clase - 1
            #sacamos el valor del primer gen
            ind_gen1 = índices_disponibles[random]
            gen1 = clase[ind_gen1]
            //quitamos el indice elegido en el primero para no repetirlo
            índices_disponibles.remove(ind_gen1)

            //sacamos el valor del segundo gen
            ind_gen2 = índices_disponibles[random]
            gen2 = clase[ind_gen2]

            #intercambiamos los valores
            clase[ind_gen1] = gen2
            clase[ind_gen2] = gen1
    return clase
```

Figura 17: Pseudocódigo de la mutación

4.7. Condición de parada

Para finalizar, una vez expuestas las diferentes implementaciones de los operadores genéticos queda presentar cual ha sido la condición de parada para la finalización de la ejecución del algoritmo. Las condiciones son:

1. Si se sobrepasa un valor de utilidad deseado, aunque en este caso, se desea encontrar el 100% de utilidad de la solución. Para conseguirlo se debe cumplir el número de grupos deseado, el número de alumnos por grupo y que todos los roles de los grupos sean distintos.
2. Si la primera condición no se cumple y el algoritmo converge obteniendo siempre el mismo grado de utilidad, se finalizará la ejecución del algoritmo tras obtener el mismo resultado N iteraciones seguidas.
3. Por otro lado, en los casos que se aplique mutación es muy probable que la mejor solución fluctúe y el algoritmo no llegue a converger, en ese caso, el algoritmo finalizaría tras llegar a un máximo de iteraciones N.

5. Resultados

En este apartado se evaluará el algoritmo genético con las distintas parametrizaciones posibles expuestas anteriormente. El objetivo del algoritmo es conseguir una combinación de alumnos por grupo que cumpla el criterio de Belbin, donde se espera que un grupo formado por personas con perfiles heterogéneos obtendrá mejores resultados.

Se van a evaluar 4 escenarios posibles para posteriormente compararlos con los resultados obtenidos mediante CPLEX. Los escenarios son:

- Escenario 1: 24 alumnos con grupos formados por 4 alumnos.
- Escenario 2: 24 alumnos con grupos formados por 6 alumnos.
- Escenario 3: 48 alumnos con grupos formados por 4 alumnos.
- Escenario 4: 48 alumnos con grupos formados por 6 alumnos.

A continuación se seguirá el siguiente listado de experimentos para la evaluación del algoritmo:

1. Evaluación de la población inicial
2. Evaluación de la función de evaluación
3. Evaluación del umbral en la selección de padres
4. Evaluación del número de padres en la selección
5. Evaluación del reemplazo de padres en la siguiente generación
6. Evaluación del tipo de crossover
7. Evaluación del tipo de método de mutación variando la probabilidad de mutación

Los primeros experimentos el reparto de los roles será equitativo para comprobar que la combinación de alumnos por grupo y roles distintos es posible. 8

- Fitness: media aritmética.
- umbralSeleccion: igual a 1, se elige el mejor candidato.
- reemplazoPadres: se reemplaza la población actual por sus hijos.
- nPadres: dos padres elegidos para el torneo.
- divisiones: en el crossover, la sección de las soluciones es por 1 punto.
- probMutacion: inicialmente la probabilidad de mutación será 0 para estudiar la evolución de las soluciones.
- maxIterIguales: 10 para considerar que converge el algoritmo.
- numMaxIter: se marca un máximo de 200 iteraciones.

5.1. Evaluación de la población inicial

Para la primera evaluación del algoritmo se evalúa el número de población inicial para estudiar la evolución de las soluciones en cada iteración, cual es el grado de utilidad máxima que se ofrece y qué rápido converge el algoritmo. Como el algoritmo tiene un componente de aleatoriedad, se evaluará cada opción 10 veces para obtener una media de las utilidades, tiempo de computación y número de iteraciones.

Población inicial	Iteraciones	Tiempo (ms)	Utilidad %	Varianza Utilidad	Soluciones reales
10	51	40	99.3	3e-05	2/10
20	11	40	99.7	7e-06	6/10
50	0	3	100	0	10/10
100	0	6	100	0	10/10

Tabla 2: Incremento de la población inicial, escenario 1.

Población inicial	Iteraciones	Tiempo (ms)	Utilidad %	Varianza Utilidad	Soluciones reales
10	35	30	98.6	4.2e-05	1/10
20	50	66	99.4	5.1e-05	6/10
50	15	65	100	0	10/10
100	14	90	100	0	10/10

Tabla 3: Incremento de la población inicial, escenario 2.

Población inicial	Iteraciones	Tiempo (s)	Utilidad %	Varianza Utilidad	Soluciones reales
10	126	0.22	98.4	4.9e-05	0/10
20	44	0.13	99.2	7e-06	0/10
50	58	0.43	99.7	2e-06	0/10
100	38	0.68	99.8	1e-06	3/10

Tabla 4: Incremento de la población inicial, escenario 3.

Población inicial	Iteraciones	Tiempo (s)	Utilidad %	Varianza Utilidad	Soluciones reales
10	42	0.07	96.3	2.4e-05	0/10
20	115	0.27	97.8	8e-05	0/10
50	39	0.24	99.3	3e-05	0/10
100	45	0.57	99.8	4e-06	3/10

Tabla 5: Incremento de la población inicial, escenario 4.

Para ser una primera prueba los resultados obtenidos de los distintos escenarios son buenos. La media de utilidad de las soluciones es superior al 90% en todos los casos y a medida que el tamaño de la población aumenta el número de soluciones reales crece. Debido al bajo coste computacional se incrementa la población inicial para evaluar los resultados obtenidos.

Población inicial	Iteraciones	Tiempo (s)	Utilidad %	Varianza Utilidad	Soluciones reales
500	0	0.03	100	0	10/10
1000	0	0.05	100	0	10/10
2000	0	0.12	100	0	10/10
4000	0	0.21	100	0	10/10

Tabla 6: Incremento de la población inicial, escenario 1.

Población inicial	Iteraciones	Tiempo (s)	Utilidad %	Varianza Utilidad	Soluciones reales
500	2	0.08	100	0	10/10
1000	0	0.05	100	0	10/10
2000	0	0.10	100	0	10/10
4000	0	0.17	100	0	10/10

Tabla 7: Incremento de la población inicial, escenario 2.

Población inicial	Iteraciones	Tiempo (s)	Utilidad %	Varianza Utilidad	Soluciones reales
500	65	4.3	99.9	0	4/10
1000	63	8.3	99.9	0	5/10
2000	68	20	99.9	0	6/10
4000	95	50	99.9	1e-06	7/10

Tabla 8: Incremento de la población inicial, escenario 3.

Población inicial	Iteraciones	Tiempo (s)	Utilidad %	Varianza Utilidad	Soluciones reales
500	65	3	99.9	1e-06	8/10
1000	55	8.3	99.9	1e-06	8/10
2000	71	23	99.9	0	8/10
4000	108	52	99.8	1e-06	4/10

Tabla 9: Incremento de la población inicial, escenario 4.

En este segundo incremento de la población, se observa como en los escenarios 1 y 2 se obtienen en todos los casos el 100% de las soluciones reales. Además, a excepción del escenario 2, para el caso de una población de 500 individuos, las soluciones reales se obtienen en la generación de la población inicial sin llegar a aplicarse los operadores genéticos.

Para los escenarios 3 y 4 los mejores resultados han sido obtenidos para los casos de una población inicial de 2000 y 4000 individuos, no obstante, el tiempo de computación para un tamaño de la población de 2000 soluciones es notoriamente inferior al de 4000. Así pues, a partir de este momento, en la evaluación del algoritmo genético se elegirá una población inicial de 2000.

Cabe destacar que el criterio de parada en este primer experimento ha sido la convergencia del algoritmo, en ningún momento se ha llegado al número máximo de iteraciones si no que, en un momento dado el algoritmo ha obtenido 10 veces consecutivas la misma utilidad máxima.

5.2. Evaluación de la función de evaluación

En la mayoría de las soluciones generadas anteriormente, se observaba como el problema principal para la obtención de una utilidad del 100% era la repetición de algún rol dentro de los grupos. Para dar solución a este problema, se han modificado los pesos de la función de evaluación para dar mayor importancia a las soluciones con roles diferentes en todos los grupos, quedando finalmente del siguiente modo:

$$\text{Utilidad} = 0.10 \times \text{NúmeroGruposFormados} + 0.55 \times \text{RolesDiferentesPorGrupo} + 0.35 \times \text{NúmeroAlumnosPorGrupo}$$

En la tabla 10 se presentan los resultados obtenidos para una población de 2000 individuos para los 4 escenarios diferentes.

Escenario	Alumnos por grupo	Iteraciones	Tiempo (s)	Utilidad %	Varianza Utilidad	Soluciones reales
24	4	0	0.10	100	0	10/10
24	6	0	0.12	100	0	10/10
48	4	52	18	99.9	0	8/10
48	6	83	20	99.9	0	9/10

Tabla 10: Evaluación de la función de evaluación para los 4 escenarios.

En la siguiente tabla, se pueden comparar los resultados obtenidos en la función de evaluación aritmética y la ponderada. Se han mejorados los resultados, aunque no parezcan muy significativos, se ha mejorado el número de soluciones reales y el número de iteraciones por ejecución del algoritmo, reduciendo así el tiempo de computación.

Iteraciones Aritmética	Iteraciones Ponderada	Tiempo (s) Aritmética	Tiempo (s) Ponderada	Soluciones reales Aritmética	Soluciones reales Ponderada
0	0	0.10	0.10	10/10	10/10
0	0	0.12	0.12	10/10	10/10
68	52	20	18	6/10	8/10
71	83	23	20	8/10	9/10

Tabla 11: Comparativa de la función de evaluación aritmética y ponderada.

5.3. Evaluación del umbral en la selección de padres

En este apartado empieza la evaluación del operador genético de selección. En capítulos anteriores se ha expuesto el funcionamiento y la implementación del método de torneo para la selección de los padres. A continuación, se evaluará la variación del umbral de selección de este proceso, que hasta el momento era igual a 1 donde se obtenía una versión determinista de la selección escogiendo siempre el mejor padre (solución) de los evaluados.

Tamaño de la Clase	Alumnos por grupo	Iteraciones	Tiempo (s)	Utilidad %	Varianza Utilidad	Soluciones reales
24	4	0	0.14	100	0	10/10
24	6	0	0.12	100	0	10/10
48	4	140	37	99.2	2.6e-05	3/10
48	6	200	45	96.1	3e-05	0/10

Tabla 12: Umbral de selección 0.25.

Tamaño de la Clase	Alumnos por grupo	Iteraciones	Tiempo (s)	Utilidad %	Varianza Utilidad	Soluciones reales
24	4	0	0.12	100	0	10/10
24	6	7	1	100	0	10/10
48	4	180	57	99.4	8e-06	1/10
48	6	200	55	96.6	3.7e-05	0/10

Tabla 13: Umbral de selección 0.50.

Tamaño de la Clase	Alumnos por grupo	Iteraciones	Tiempo (s)	Utilidad %	Varianza Utilidad	Soluciones reales
24	4	0	0.12	100	0	10/10
24	6	1	0.10	100	0	10/10
48	4	175	50	99.9	1e-06	5/10
48	6	174	42	99.9	1e-06	7/10

Tabla 14: Umbral de selección 0.75.

Tamaño de la Clase	Alumnos por grupo	Iteraciones	Tiempo (s)	Utilidad %	Varianza Utilidad	Soluciones reales
24	4	0	0.10	100	0	10/10
24	6	1	0.10	100	0	10/10
48	4	72	20	99.9	1e-06	7/10
48	6	113	28	99.9	1e-06	7/10

Tabla 15: Umbral de selección 0.90.

A medida que el umbral de selección sea más bajo, se escogerá de entre los padres seleccionados para el torneo el peor de los casos, originando así peores resultados y con ello mayor número de iteraciones y por consecuencia de tiempo de computación.

Cabe resaltar que, efectivamente, al variar el umbral de selección se obtiene una versión probabilística del torneo. Ésta cualidad es notoria en el incremento de iteraciones medio de cada ejecución del algoritmo, pues se observa, como el criterio de parada de algunos casos es la obtención del número máximo de iteraciones y no por la convergencia del algoritmo.

Todo y que en algunos casos puede ser beneficiosa la opción probabilística de la selección de los padres, para la naturaleza de este caso de estudio, la versión determinista del torneo ha obtenido mejores resultados frente a la probabilística como se observa en la tabla 16.

Tamaño de la Clase	Alumnos por grupo	Soluciones reales 0.9	Soluciones reales 1
24	4	10/10	10/10
24	6	10/10	10/10
48	4	7/10	8/10
48	6	7/10	9/10

Tabla 16: Comparativa de umbrales de selección.

5.4. Evaluación del número de padres en la selección

En este apartado, se evalúa el número de padres seleccionados en el torneo. Hasta ahora se elegían dos padres y dependiendo del umbral de selección se elegía el mejor o el peor. Realizando una prueba rápida incrementando el número de padres a 4 para el torneo, se obtienen muy buenos resultados, como se observa en la tabla 17.

Tamaño de la Clase	Alumnos por grupo	Iteraciones	Tiempo (s)	Utilidad %	Varianza Utilidad	Soluciones reales
24	4	0	0.10	100	0	10/10
24	6	0	0.11	100	0	10/10
48	4	26	7	100	0	10/10
48	6	24	5	100	0	10/10

Tabla 17: Selección por torneo con 4 padres.

Para evaluar de manera más rigurosa este apartado, se incrementa el número de pruebas por escenario de 10 a 100 para los escenarios 3 y 4, los cuales son los más complejos de satisfacer.

N Padres	Iteraciones	Tiempo (s)	Utilidad %	Varianza Utilidad	Soluciones reales
2	73	20	99.95	1e-06	70
4	20	5	100	0	100
6	12	4	99.9	0	97
8	10	3	99.9	0	90

Tabla 18: Selección de N padres, escenario 3.

N Padres	Iteraciones	Tiempo (s)	Utilidad %	Varianza Utilidad	Soluciones reales
2	85	21	99.9	0	92
4	25	6	99.9	0	99
6	17	5	99.7	2e-06	92
8	19	5	99.9	0	78

Tabla 19: Selección de N padres, escenario 4.

Observando los distintos escenarios, se observa como aumentando el número de padres seleccionados para el torneo el número de iteraciones disminuye y se obtienen mejores resultados hasta el caso de los 8 padres donde el número de soluciones reales disminuye.

5.5. Evaluación del reemplazo de padres

En este apartado, se repite el mismo experimento que el anterior pero esta vez sin sustituir los padres en la siguiente generación de soluciones. En este caso la población está formada por una mitad de los padres seleccionados y por la otra, los hijos generados.

N Padres	Iteraciones	Tiempo (s)	Utilidad %	Varianza Utilidad	Soluciones reales
2	17	5	99.8	2e-06	24
4	12	4	99.1	0	46
6	10	4	99.9	0	45
8	10	4	99.8	3e-06	43

Tabla 20: Selección de N padres, escenario 3 sin reemplazo.

N Padres	Iteraciones	Tiempo (s)	Utilidad %	Varianza Utilidad	Soluciones reales
2	36	9	99.1	0	36
4	25	7	99.2	0	15
6	20	6	99.7	7.3e-05	11
8	18	5	98.7	7.2e-05	5

Tabla 21: Selección de N padres, escenario 4 sin reemplazo.

Como puede observarse, los resultados están muy por debajo de los obtenidos con el reemplazo de los padres. Por el número de iteraciones se puede deducir que este cambio introduce redundancia en las soluciones y algoritmo acaba convergiendo rápidamente.

5.6. Evaluación del tipo de cruce

Una vez seleccionados los padres ganadores, se realiza el cruce de estos para la generación de la nueva población. Hasta este punto, el tipo de crossover empleado en el algoritmo era la sección por un punto.

A continuación se analizarán los resultados obtenidos con las opciones: cruce por 2 puntos y cruce uniforme. Una vez más, se utilizarán los escenarios 3 y 4 debidos a su complejidad, además se estudiarán los casos para los N Padres evaluados anteriormente.

N Padres	Iteraciones	Tiempo (s)	Utilidad %	Soluciones reales
2	87	25	99.9	73
4	35	11	99.9	97
6	18	5	100	100
8	11	3	100	100

Tabla 22: Cruce por dos puntos, escenario 3.

N Padres	Iteraciones	Tiempo (s)	Utilidad %	Soluciones reales
2	110	28	99.9	83
4	40	10	100	100
6	21	5	100	100
8	15	4	100	100

Tabla 23: Cruce por dos puntos, escenario 4.

N Padres	Iteraciones	Tiempo (s)	Utilidad %	Soluciones reales
2	152	52	99.1	24
4	144	50	99.1	28
6	148	68	99.1	26
8	160	81	99	20

Tabla 24: Cruce uniforme, escenario 3.

N Padres	Iteraciones	Tiempo (s)	Utilidad %	Soluciones reales
2	200	60	95.8	0
4	200	65	95.9	0
6	200	72	95.7	0
8	200	86	95.7	0

Tabla 25: Cruce uniforme, escenario 4.

A primera instancia, se observa que el cruce uniforme es el peor de los métodos de cruce implementados, de hecho, viendo el número de iteraciones medio en el caso del escenario 4 es del máximo de iteraciones. Este método genera tanta variabilidad en las soluciones que los resultados obtenidos en las nuevas generaciones no conservan las bondades de sus antecesores.

En las siguientes tablas se presentan los resultados obtenidos con los cruces por 1 punto y por 2, donde se observa cómo el cruce por 2 puntos es la mejor opción para la resolución del problema, obteniendo la solución óptima al problema el 100% de las veces para los dos escenarios a partir de los 6 padres.

N Padres	Soluciones reales Cruce 1 punto	Soluciones reales Cruce 2 puntos
2	70	73
4	100	97
6	97	100
8	90	100

Tabla 26: Comparativa métodos de cruce, escenario 3.

N Padres	Soluciones reales Cruce 1 punto	Soluciones reales Cruce 2 puntos
2	92	83
4	99	100
6	92	100
8	78	100

Tabla 27: Comparativa métodos de cruce, escenario 4.

5.7. Evaluación del tipo de método de mutación

En capítulos anteriores se han expuesto los dos tipos de mutación implementados en el algoritmo genético que son: mutación aleatoria e intercambio de valores. En este apartado se estudiarán las soluciones reales obtenidas tras 100 ejecuciones del algoritmo para los escenarios 3 y 4 con N padres. Estos ejercicios se realizarán para las probabilidades de mutación del 1%, 10%, 25%, 50%, 75% y 100%.

5.7.1 Mutación aleatoria

Para obtener un visionado global de los resultados obtenidos con la implementación de la mutación aleatoria, se recogen los datos en dos gráficos, figura 18 y figura 19.

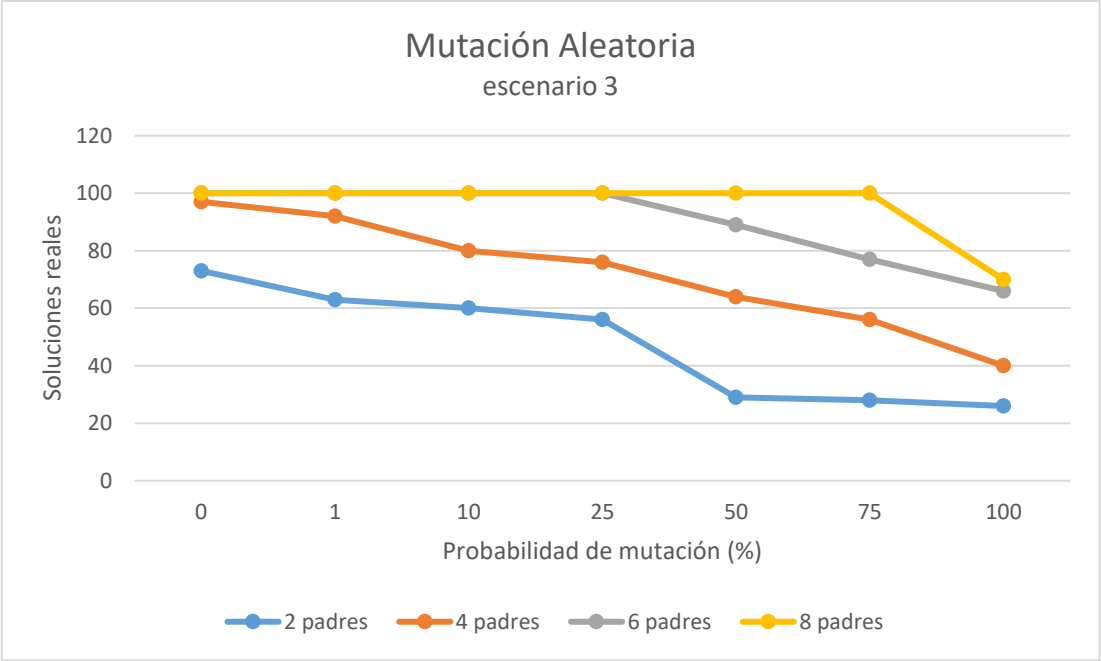

Figura 18: Resultados de la mutación aleatoria, escenario 3.

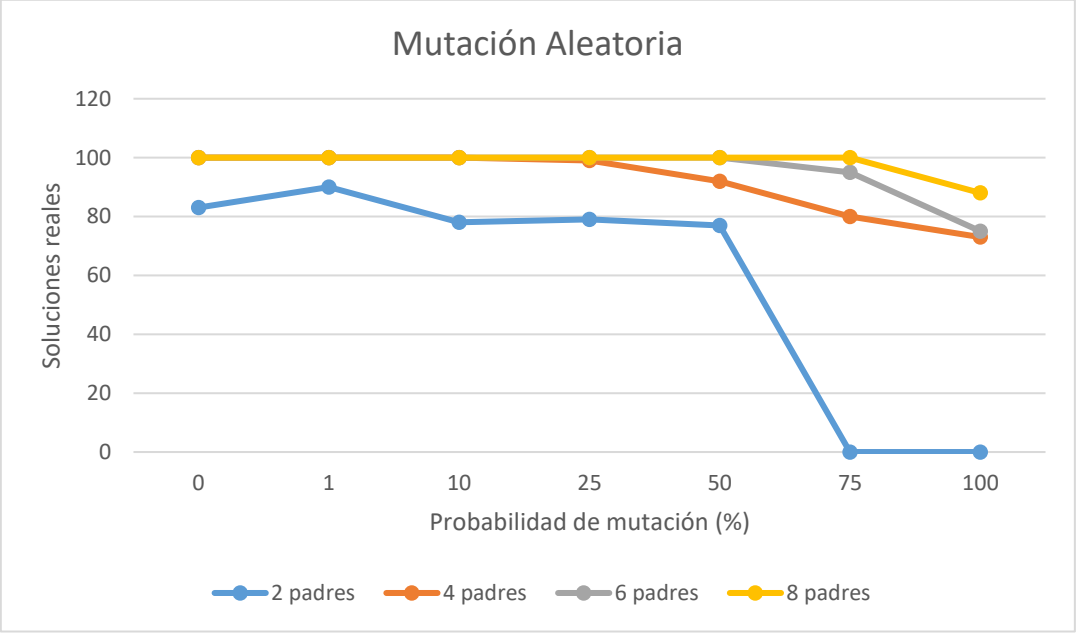

Figura 19: Resultados de la mutación aleatoria, escenario 4.

Para ambos escenarios, como era de esperar, a medida que la probabilidad de mutación aumenta, el número de soluciones reales disminuye. La mutación permite ampliar el espacio de búsqueda de las soluciones, pero cuantas más soluciones sean alteradas, más dispares pueden ser las utilidades que ofrezcan.

Para este caso de estudio, el hecho de cambiar un gen de la solución generada, implicaría que dos grupos varíen su tamaño o que un grupo formado por un único alumno apareciera o despareciera. Este tipo de mutación sería útil en aquel caso donde algunos de los grupos con un tamaño no deseado, pero a su vez, para que resulte una solución 100% válida, el gen mutado debe tener un rol distinto a los que ya existen en su nuevo grupo.

5.6.2 Mutación por intercambio de valores

Para obtener un visionado global de los resultados obtenidos con la implementación de la mutación por intercambio de valores, se recogen los datos en dos gráficos, figura 20 y figura 21.

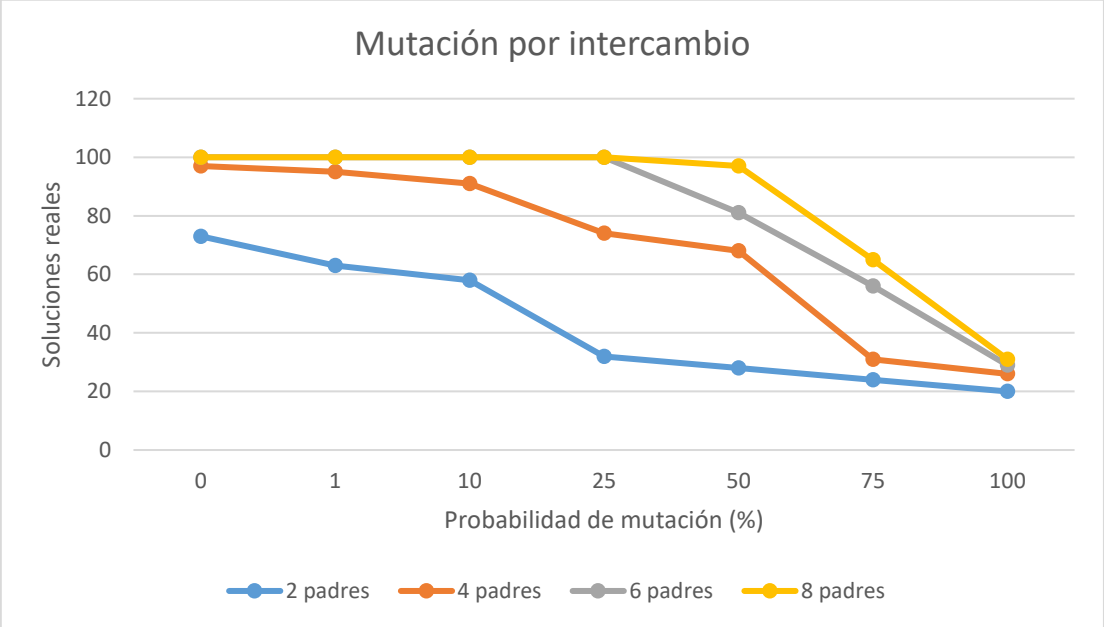

Figura 20: Resultados por intercambio de valores, escenario 3.

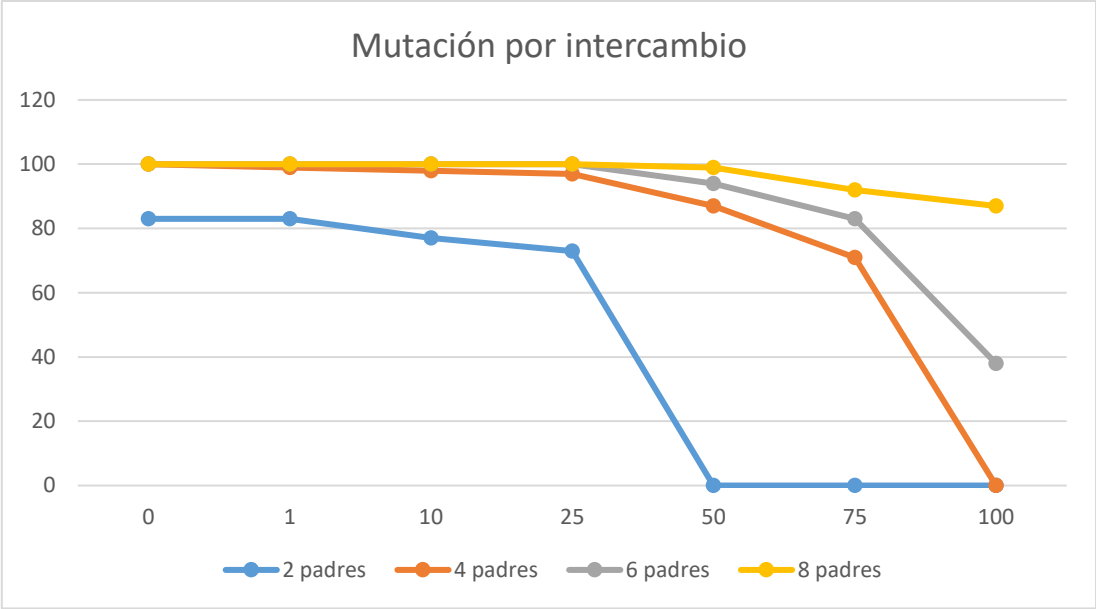

Figura 21: Resultados por intercambio de valores, escenario 4.

Una vez más, la mutación no ha mejorado la cantidad de soluciones reales al problema. Este tipo de mutación puede ser útil en aquel caso donde, se obtienen todos los grupos con los tamaños deseados en la solución pero que al menos, un alumno obtenga un rol repetido dentro de su grupo, se intercambie por otro alumno con un rol distinto a los ya existentes, de este modo la utilidad por número roles distintos por grupos aumentaría.

6. Evaluación del sistema

En este apartado se presenta la evaluación final del sistema implementada y se comparan los resultados obtenidos con el algoritmo genético frente los resultados de la solución implementada con CPLEX.

6.1. Configuración del sistema

A continuación se presenta la configuración final del algoritmo con la cual se han obtenido los mejores resultados:

- Población inicial: 2000
- Función de evaluación: media ponderada
- Selección padres:
  - Umbral: 1
  - N Padres: 8
  - Reemplazo padres en la nueva generación
- Crossover: sección por dos puntos
- Mutación: probabilidad 0%

Respecto a la parametrización del algoritmo cabe comentar algunos puntos que han surgido a lo largo del proyecto.

En el caso de que el tamaño de la población a medida que se ha incrementado, el algoritmo genético ha realizado una búsqueda de soluciones más amplia y óptima hasta llegar a un punto máximo donde el coste computacional era aceptado. Como se exponía en capítulos anteriores, el tamaño óptimo de la población tenía que ser decidido por ensayo y error.

Para la evaluación de la utilidad de las soluciones, finalmente se ha optado por la realización de una media ponderada de los distintos parámetros a evaluar. Este método ha permitido proporcionar mayor valor a la formación de grupos con roles distintos, el cual era el primordial para la resolución del problema y a priori el más complicado.

Respecto al método de selección, cabe destacar que la mejoría conseguida con el incremento de los padres ha sido en parte inesperada. Tras la documentación de distintas implementaciones de algoritmos genéticos, la norma anunciaba que la selección de 2 padres como candidatos en el torneo era la óptima pero tras las pruebas realizadas en el proyecto, para la naturaleza de este problema, el resultado del algoritmo mejoraba al incrementar el número de padres en el torneo.

Por lo que respecta a los otros parámetros en la selección, el reemplazo de los padres por la nueva generación ha sido vital para la resolución del problema. En algunos casos suele mantenerse algunos de los padres o individuos élite que sobreviven en la siguiente generación, pero los resultados obtenidos sin reemplazo de los padres, indicaban la convergencia prematura del algoritmo.

A cerca del umbral de selección, sólo destacar que en algunos casos, los umbrales entre 0.75 y 1, generan cierta variación entre la población ayudando al algoritmo a explorar ampliamente el espacio de soluciones. Para la resolución de este proyecto, la versión probabilística del torneo (umbral igual a 1) ha sido la mejor opción, dónde mejores soluciones se obtenían y menos iteraciones se realizaban.

Respecto al operador genético de cruce, la mejor opción ha sido el cruce por 2 puntos. Se intuye que genera mayor variabilidad en las soluciones obtenidas frente el cruce por un punto y con ello obtener mejores resultados y así ha sido. El método cruce uniforme generaba demasiada variabilidad en las soluciones, dado que cada gen representaba un alumno, la variación de estos de manera individual era muy probable que la solución empeorara.

Finalmente, la mutación no ha generado ninguna mejora en el algoritmo. Variando mínimamente la probabilidad de mutación, los resultados obtenidos en todos los experimentos emporaban las soluciones generadas. De base, es notorio que una probabilidad de mutación baja, genera un cierto grado de variabilidad en el algoritmo que permite explorar ampliamente el espacio de soluciones. Una vez más, por la naturaleza del problema, los cambios individuales de los genes son propensos a generar peores resultados debido a las dependencias entre grupos, números de grupos y los roles representados por los genes modificados.

6.2. Comparativa de resultados

Para comparar los resultados obtenidos por la política de formación de grupos expuesta en [1] y el algoritmo genético implementado en este proyecto, se presentan a continuación los mejores resultados obtenidos en ambos sistemas.

Los resultados obtenidos con la parametrización más óptima del algoritmo genético expuesta en el apartado anterior son:

Tamaño de la clase	Alumnos por grupo	Iteraciones	Tiempo (s)	Utilidad %
24	4	0	0.10	100
24	6	0	0.10	100
48	4	11	3	100
48	6	15	4	100

Tabla 28: Mejores resultados obtenidos por el algoritmo genético.

Los mejores resultados obtenidos con la política de formación de equipos implementada con CPLEX se recogen en la siguiente tabla:

Tamaño de la clase	Alumnos por grupo	Iteraciones	Tiempo (HH:MM:SS)	Utilidad %
24	4	0	00:00:19	100
24	6	0	05:42:03	100
48	4	11	00:05:37	100
48	6	--	--	--

Tabla 29: Resultados obtenidos por CPLEX.

Observando los resultados obtenidos en ambos sistemas se puede afirmar que se ha cumplido el objetivo principal del algoritmo genético. Se ha demostrado que el algoritmo ha sido adecuado para la resolución del problema, donde los tiempos de computación del algoritmo superan con creces los obtenidos por la política empleada en la herramienta actual. Incluso, para el caso de una clase de 48 alumnos y grupos formados por 6 alumnos, el algoritmo ha sido capaz de resolver este escenario en 4 segundos, frente a la solución implementada con CPLEX la cual saturaba la memoria RAM siendo incapaz de obtener una solución.

7. Conclusiones

Durante la realización del presente proyecto se han alcanzado con éxito los objetivos propuestos, entre los que destaca la implementación de un algoritmo genético capaz de formar grupos siguiendo el criterio de roles de Belbin mejorando el tiempo de computación de la solución actual.

Se ha analizado la política implementada en [1] para comprender las necesidades de la herramienta que, junto al estudio sobre los algoritmos genéticos, ha permitido diseñar un algoritmo capaz de parametrizarse dependiendo de los distintos métodos implementados en cada uno de los operadores genéticos.

Como era de esperar, la capacidad de explorar el espacio de búsqueda de los algoritmos genéticos frente la fuerza bruta, ofrecen buenos resultados de manera rápida y eficiente. En los primeros experimentos, el algoritmo genético generaba una lista de soluciones buenas (utilidad superior al 90%), pero no una solución óptima que cumpliera con los tres requisitos necesarios para la formación de grupos, como son el número de grupos, número de alumnos por grupo y roles distintos por grupo de toda la clase.

Finalmente, se ha presentado una configuración óptima del algoritmo para la resolución de los diferentes escenarios reales expuestos obteniendo en el 100% de los casos una solución óptima. Además, se ha realizado una comparativa de los resultados obtenidos por ambos sistemas donde se ha comprobado la eficiencia del algoritmo genético en cuanto a tiempo y fiabilidad en la obtención de soluciones reales cumpliendo los requerimientos del sistema.

8. Bibliografía

[1] Alberola, J. M., Del Val, E., Sanchez-Anguix, V., Palomares, A., y Teruel, M. D. (2016). "An artificial intelligence tool for heterogeneous team formation in the classroom" en *Knowledge-Based Systems.* Vol.101, p. 1-14.

[2] Alberola J.M., del Val E., Sanchez-Anguix V., y Julián V. (2016) "A General Framework for Testing Different Student Team Formation Strategies." en Caporuscio M., De la Prieta F., Di Mascio T., Gennari R., Gutiérrez Rodríguez J., Vittorini P. (eds) *Methodologies and Intelligent Systems for Technology Enhanced Learning. Advances in Intelligent Systems and Computing.* Vol 478. Cham: Springer. 23-31.

[3] del Val, E., Alberola, J. M., Sanchez-Anguix, V., Palomares, A., y Teruel, M. D. (2014). "A team formation tool for educational environments" en *Trends in Practical Applications of Heterogeneous Multi-agent Systems. The PAAMS Collection. Advances in Intelligent Systems and Computing.* Vol 293. Cham: Springer. 173-181.

[4] Alberola, J. M., del Val, E., Sanchez-Anguix, V., y Julián, V. (2013). "Simulating a collective intelligence approach to student team formation" en Pan JS., Polycarpou M.M., Woźniak M., de Carvalho A.C.P.L.F., Quintián H., Corchado E. (eds) *Hybrid Artificial Intelligent Systems. HAIS 2013. Lecture Notes in Computer Science*, Vol 8073. Heidelberg: Springer. 161-170.

[5] HERNÁNDEZ, J. J., COSTA, A., DEL VAL, E., ALBEROLA, J. M., NOVAIS, P., y JULIAN, V. (2017). "Using Genetic Algorithms for Group Activities in Elderly Communities" en Criado Pacheco N., Carrascosa C., Osman N., Julián Inglada V. (eds) Multi-Agent Systems and Agreement Technologies. EUMAS 2016, AT 2016. Lecture Notes in Computer Science, Vol 10207. Cham: Springer. 524-537.

[6] Alberola, J. M., del Val, E., Sanchez-Anguix, V., y Julián, V. (2013). "Una herramienta de aprendizaje colaborativo orientada a la formación de grupos" en *Simposio Internacional de Informática Educativa* (SIE 2013). 56-61.

[7] Belbin, R.M. (2010). *Team roles at work.* Abingdon: Routledge.

[8] Goldberg, D. E. (1989). *Genetic Algorithms en Search, Optimization and Machine Learning*. New York: Addison – Wesley.

[9] Holland, J. H. (1992). "Genetic Algorithms" en *Scientific American Journal.* July 1992, Volume 267, Issue 1.

[10] Kalyanmoy, D. (1999). "An Introduction To Genetic Algorithms" en Sadhana. Agosto y Octubre 1999, Vol. 24 Parts 4 and 5, p.293-315.

[11] Mitchell, M. (1999). *An Introduction To Genetic Algorithms*, Cambridge, Massachusets-London, England: The MIT Press.

[12] Michalewicz, Z. (1994) *Genetic Algorithms + Data Structures = Evolution Programs*, Second, Extended Edtion. Berlin-Heidelberg-New York: Springer-Verlag. p. 340.

[13] Caballero, R., Molina, J., Luque, M., Torrico, A., y Gómez, T. (2002). "Algoritmos genéticos para la resolución de problemas de Programación por Metas Entera. Aplicación a la Economía de la Educación." en *Asociación Española de Profesores Universitarios de Matemáticas aplicadas a la Economía y la Empresa (ASEPUMA).*

[14] Gestal, M., Rivero, D., Rabuñal, J. R., Dorado, J., y Pazos, A. (2010*). Introducción a los Algoritmos Genéticos y la Programación Genética*. A Coruña: Universidad da Coruña, Servizo de Publicacións.

[15] Kramer, O. (2017). *Genetic Algorithm Essential, Studies en Computational Intelligence,* Vol 679. Cham: Springer.

[16]Palacios, F., Colomer, V., Martínez, A. V. (2012) "Optimización de la Estrategia en Carreras de Bajo Consumo Mediante Algoritmos Genéticos" en *XVI Congreso Internacional de Ingeniería de Proyectos.* (2012.Valencia). 1316-1328

[16] Jiménez, G. (2009) *Optimización.*, Manizales, Colombia: Universidad Nacional de Colombia, Sede Manizales.

[17] Coello Coello, C. A (2017) *Introducción a la Computación Evolutiva*. Méjico: Departamento de Computación del Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional (CINVESTAV).

[18] Moujahid, A., Inza, I. y Larrañaga, P. (2008) *Algoritmos genéticos*. Departamento de Ciencias de la Computación e Inteligencia Artificial Universidad del País Vasco–Euskal Herriko Unibertsitatea (UPV/EHU)

[19] Sánchez Rodríguez, I., (2016). Aplicación de Algoritmos Genéticos para la optimización del corte de material. Tutor: Federico Barber Sanchís. Trabajo Fin de Grado. Valencia: Universidad Politécnica de Valencia.

[20] Daniel Shiffman. *The Nature Of Code, Chapter* 9.1 *Genetic Algorithms: Inspired by Actual Events* <http://natureofcode.com/book/chapter-9-the-evolution-of-code/> [Consulta: septiembre 2017]

[21] W3ii. *W3ii*. <http://www.w3ii.com/es/index.html> [Consulta: septiembre 2017]

[22] Holland, J. H. (1992). *Adaptation natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. Cambridge (Massachusetts): The MIT Press.

[23] Holland, J. H. (1975). *Adaptation natural and artificial systems.* Ann Arbor (Michigan): University of Michigan Press.