

OPTIMIZACIÓN MEDIANTE MÉTODOS BIOINSPIRADOS

TRABAJO DE FIN DE GRADO
Curso: 2021-2022



UNIVERSIDAD COMPLUTENSE
DE MADRID

FACULTAD DE CIENCIAS MATEMÁTICAS
GRADO EN MATEMÁTICAS

ALUMNA: Siria Catherine Íñiguez Brito
TUTOR: Fernando Rubio Diez

Madrid, 5 de Julio de 2022

Agradecimientos

A mi familia, sobre todo a mi madre y hermano pequeño, gracias por el apoyo incondicional que siempre me habéis brindado, por ayudarme en los momentos más difíciles, acompañarme en las noches de desvelo y no soltarme nunca de la mano.

A la gran familia que he encontrado a lo largo de esta maravillosa experiencia universitaria. Gracias por formar parte de la mejor etapa de mi vida, llenar mis días de risas, nuevas experiencias, y a veces, incluso lloros, pero sobre todo por regalarme todo vuestro tiempo a lo largo de estos cuatro fascinantes años.

A mi tutor, Fernando Rubio Diez por ayudarme y guiarme en el desarrollo de este trabajo. Así como a todos los profesores que he tenido, tanto en la carrera como fuera de ella, por haber impulsado mi curiosidad por las Matemáticas.

Resumen

La planificación de un calendario de exámenes es una de las gestiones administrativas más importantes que realizan las instituciones educativas. Debido a la complejidad del problema es imposible tratar de resolverlo de forma manual. En este trabajo, nos basaremos en el método heurístico conocido como “Algoritmos Genéticos” para desarrollar un programa informático capaz de automatizar el proceso de cronometrización de horarios, donde la distribución de las pruebas en un número limitado de franjas horarias logre espaciar y evitar los exámenes consecutivos para los alumnos, teniendo como objetivo encontrar soluciones aceptables en calidad y tiempo computacional.

Palabras Clave: programación de horarios, algoritmos genéticos, exámenes, optimización.

Abstract

Planning of an examination schedule is one of the most important administrative tasks performed by educational institutions. Due to the complexity of the problem it is impossible to try to solve it manually. This project is based on the heuristic method known as “Genetic Algorithms” in order to develop a computer program capable of automating the process of scheduling timetables, where the distribution of tests in a limited number of time zones gets space and avoids consecutive exams for students. We want to find good solutions in a reasonable computational time.

Key Words: timetabling, genetic algorithms, exams, optimization.

Índice

Índice de figuras	VI
Índice de tablas	VII
1 INTRODUCCIÓN	1
1.1 Objetivos	2
1.2 Metodología	2
1.3 Estructura de la memoria	2
1.4 Antecedentes	3
2 ALGORITMOS GENÉTICOS	4
2.1 Introducción	4
2.2 Esquema algorítmico	4
2.3 Codificación	5
2.4 Generar población inicial	6
2.5 Selección de padres	6
2.6 Operador de cruce	8
2.7 Operador de mutación	9
2.8 Remplazamiento generacional	9
2.9 Función objetivo	10
2.10 Condición de parada	10
2.11 Ventajas e inconvenientes	10
2.12 Aplicaciones	11
3 PROGRAMACIÓN DE HORARIOS	12
3.1 Planteamiento y modelización	12
3.2 Implementación	14
3.3 Datos	19
3.4 Resultados y valoraciones	21
3.5 Trabajos futuros	30
4 CONCLUSIONES	32
5 BIBLIOGRAFÍA	33

Índice de figuras

1	Combinaciones de horarios para $m = 2$ y $n = 4$	1
2	Esquema general de los Algoritmos Genéticos.	5
3	Ejemplo de Selección por Torneo.	7
4	Ejemplo de Método por Ruleta.	7
5	Ejemplo Cruce 1 punto.	8
6	Ejemplo Cruce N puntos.	8
7	Ejemplo Cruce Uniforme.	8
8	Ejemplo de Mutación para $pm = 0.02$	9
9	Datos para un ejemplo ficticio.	14
10	Codificación de una solución.	14
11	Generar una población de horarios.	15
12	Generar una población de forma aleatoria.	15
13	Generar una población con una técnica diferente.	15
14	Construir un horario con la función <code>def vectorS2(..)</code>	15
15	Seleccionar padres.	16
16	Cruce en N puntos.	16
17	Cruce Uniforme.	17
18	Elitismo.	17
19	Mutación.	17
20	Criterio para elegir el mejor horario.	18
21	Algoritmos Genéticos.	18
22	Archivo Carleton91.exm.	20
23	Archivo YorkMills83.stu.	20
24	Archivo Carleton91.sol.	21
25	Leyenda.	22
26	Estudio con número fijo de vueltas.	22
27	Estudio con tamaño de la población fijo.	23
28	Estudio relacionado con el tiempo de espera.	24
29	Estudio Aleatoriedad vs Heurística.	25
30	Estudio evolución de la población.	26
31	Estudio tamaño del torneo.	27
32	Estudio comparando las mejores soluciones.	29

Índice de tablas

1	Conjunto de datos de entrada.	19
2	Conjunto de archivos de las soluciones “Results”, UAX.	21
3	Datos para el estudio por número de vueltas fijo.	22
4	Datos para el estudio por tamaño de la población fijo.	23
5	Datos para el estudio por tiempo de espera.	24
6	Datos para el estudio de la evolución de la población.	25
7	Datos para el estudio del tamaño del torneo.	27
8	Datos para la comparación de soluciones con “Results”, UAX. . . .	29
9	Datos para la comparación de soluciones con “BestSolution”. . . .	30

1 INTRODUCCIÓN

En este mundo globalizado que nos rodea, se pretende resolver problemas con una cantidad de datos desmedida, al lado de numerosas restricciones, donde la búsqueda de una solución exacta es una tarea prácticamente imposible. Por lo que para dar respuesta a este tipo de asuntos se han desarrollado diversas técnicas heurísticas.

En el desarrollo del presente trabajo trataremos la problemática relacionada con la elaboración de horarios donde procuraremos minimizar el solapamiento de exámenes y aumentar la distancia entre los mismos. Esta no es una tarea realmente sencilla, pues para encontrar la solución óptima tendríamos que explorar multitud de posibilidades. En concreto, si generamos un horario asignando a cada examen una de las horas disponibles, las combinaciones a examinar, siendo **n = número de exámenes** y **m = franjas horarias**, deberían ser m^n horarios distintos.

En un caso particular donde hay **2 horas disponibles** y **4 exámenes** a realizar, tenemos 2^4 configuraciones posibles, de las cuales debemos elegir la que mejor se adapte al alumnado. Este problema se vuelve totalmente intratable con métodos tradicionales a medida que aumentemos el conjunto de datos, tanto de alumnos, de exámenes como de franjas horarias, por lo que resulta prácticamente imposible encontrar la solución óptima, por lo menos de manera manual. Con tan solo aumentar el número de franjas horarias a **8**, tenemos 8^4 posibles horarios a explorar.

	E ₁	E ₂	E ₃	E ₄
HORARIO	2	2	2	2
1	H1	H1	H1	H1
2	H1	H1	H1	H2
3	H1	H1	H2	H2
4	H1	H1	H2	H1
5	H1	H2	H1	H1
6	H1	H2	H1	H2
7	H1	H2	H2	H1
8	H1	H2	H2	H2
9	H2	H2	H2	H2
10	H2	H2	H2	H1
11	H2	H2	H1	H1
12	H2	H2	H2	H2
13	H2	H1	H3	H1
14	H2	H1	H4	H2
15	H2	H1	H2	H1
16	H2	H1	H2	H2

Figura 1: Combinaciones de horarios para $m = 2$ y $n = 4$.

En este trabajo, apostaremos por estudiar el problema de “**programación de horarios**” con la ayuda de algoritmos genéticos, para aproximarnos, así, de manera aceptable a la solución óptima, dentro de un tiempo razonable.

1.1 Objetivos

El presente trabajo tiene como objetivos principales investigar, analizar, plantear e implementar una solución al problema relacionado con la distribución adecuada de exámenes.

Trataremos de proponer un modelo matemático para el problema de planificación de horarios, a la vez que implementamos en la computadora una posible solución apoyándonos en los algoritmos genéticos.

Una vez implementado correctamente el algoritmo, se valorará la eficiencia y comparará la calidad de las soluciones con otros resultados ya testeados.

1.2 Metodología

Presentaremos a continuación la metodología a seguir para llevar a cabo la realización de este proyecto:

-Información: empezaremos con una pequeña indagación informativa sobre el tema general del que estamos tratando, como es en este caso, el estudio de algoritmos genéticos. Una vez realizada esta tarea, intentaremos plasmar de manera informal los conocimientos adquiridos al caso particular que estamos tratando.

-Análisis: valoraremos las diferentes variantes de diseño, tanto las relacionadas intrínsecamente con el funcionamiento de los algoritmos genéticos, como el de nuestro problema concreto.

-Diseño: para llevar a cabo posteriormente un análisis más profundo, abriremos la posibilidad a diferentes diseños, buscando conseguir diversas vertientes con las que después examinar nuestras soluciones.

-Implementación: se implementarán las diferentes variantes y describiremos los objetivos que queremos lograr con ellas.

-Valoración: para poder realizar una valoración correcta de los algoritmos, consideraremos varios conjuntos de datos y compararemos los resultados con trabajos realizados previamente. En caso de un juicio totalmente negativo, nos planteamos la posibilidad de retroceder en alguna etapa de la metodología. En el caso de resultados aceptables, encauzamos el camino para la realización de trabajos futuros.

1.3 Estructura de la memoria

En este apartado se describe brevemente el contenido de las diferentes secciones de la memoria:

Capítulo I: INTRODUCCIÓN. Se describen los objetivos y características principales del proyecto, así como la distribución de los capítulos, además de una pequeña redacción sobre los antecedentes teóricos de este proyecto.

Capítulo II: ALGORITMOS GENÉTICOS. Presentamos las características generales sobre los Algoritmos Genéticos.

Capítulo III: PROGRAMACIÓN DE HORARIOS. Expondremos el diseño e implementación de un algoritmo genético que resuelva el problema de planificación de horarios de forma automatizada. Además, valoraremos los resultados obtenidos y evaluaremos el rendimiento de la aplicación final junto con la calidad de sus soluciones.

Capítulo IV: CONCLUSIONES. Mostrarémos las conclusiones y reflexiones generales obtenidas tras la realización de este proyecto.

Capítulo V: BIBLIOGRAFÍA. Se mencionan las diversas referencias usadas para el desarrollo del trabajo.

1.4 Antecedentes

Uno de los problemas más clásicos dentro de las ciencias de la computación, concretamente de la Inteligencia Artificial, es la generación de horarios debido a la multitud de campos donde aparece, como pueden ser: horarios escolares, laborales, de transportes, programación de exámenes, eventos deportivos y un sin fin de etcéteras.

Para poder abordar la solución de este asunto podríamos recurrir a las técnicas tradicionales si se tratase de un conjunto de datos pequeño, sin embargo en este caso buscamos resolverlo para conjuntos de datos grandes, por lo que el problema se vuelve intratable pues está clasificado como un problema NP-difícil, luego la siguiente opción es buscar una solución con métodos no tradicionales como son los algoritmos evolutivos, búsqueda tabú, colonia de hormigas, algoritmos voraces, redes neuronales entre otros.

En concreto, nos enfocaremos en los algoritmos genéticos, pieza fundamental del presente trabajo, cuyos inicios se remontan al año 1960 con John Holland quien planteó la posibilidad de asemejar los sistemas de selección y supervivencia del más adaptado dentro de la naturaleza a una estrategia para la resolución de problemas de optimización. El nacimiento de ordenadores de mejor rendimiento a mediados de la década de 1980 permitió emplear los algoritmos evolutivos a la resolución de diversos problemas de ingeniería que antes eran inasequibles, y a partir de entonces los avances de estas técnicas han sido continuos.

Debido a la dificultad del problema en cuestión, las investigaciones siguen intentando mejorar los resultados experimentales encontrados en lo que se refiere tanto a calidad como a tiempo requerido. Como no se ha hallado una solución general, sigue siendo un problema abierto sin haberse resuelto todavía por completo.

2 ALGORITMOS GENÉTICOS

Dado un problema cualquiera podríamos intentar buscar la solución a base de probar todas las combinaciones posibles, esta técnica podría resultarnos útil si intentamos resolver problemas pequeños, sin embargo, en numerosas ocasiones tratamos de resolver problemas complejos con una gran cantidad de datos, por lo que resulta imposible realizar todo este bosquejo combinatorio, tanto manual como computacionalmente. En este caso nos apoyaremos en técnicas más sofisticadas que solucionen este tipo de problemas de forma automática y eficiente como son los AG's. Para esta sección nos basaremos en las fuentes [3–10] de la bibliografía.

2.1 Introducción

Los AG's son métodos heurísticos empleados para tratar de solucionar problemas de búsqueda y optimización, los cuales se inspiran como su nombre indica en el proceso de evolución biológico. De acuerdo con Charles Darwin, las poblaciones progresan según los principios de selección natural y supervivencia del mejor adaptado. Los individuos compiten entre sí por los recursos, aquellos que son capaces de sobrevivir y atraer más parejas tendrán más probabilidades de dejar descendencia, por lo que su material genético se extenderá de generación en generación. Gracias a la combinación de buenos rasgos las especies van evolucionando y adquiriendo características cada vez más adaptadas al medio que les rodea.

“Así como la selección natural trabaja exclusivamente para y por el bien de cada ser viviente, todos los dotes mentales y corporales tienden a progresar en dirección a la perfección.” Charles Darwin, El Origen de las Especies.

Holland estableció los principios básicos de este tipo de esquema algorítmico que trata de establecer una correspondencia con el método evolutivo de la naturaleza. Se trabaja con un grupo de individuos, donde cada uno simboliza una solución del problema planteado. A todos ellos se les asigna una puntuación asociada con la calidad de dicha solución, esto se corresponde con el grado en el que un organismo se adapta al medio, cuanto mayor sea su aptitud, más probabilidades tendrá para reproducirse, cruzando su material genético con otro individuo seleccionado de la misma manera, de modo que sus genes se extenderán a lo largo de las siguientes generaciones. Este cruce producirá nuevos descendientes que engendrarán una nueva población de posibles soluciones, probablemente con mejores características que la anterior.

2.2 Esquema algorítmico

En la construcción de un algoritmo genético se deberán tener en cuenta los siguientes aspectos, los cuales se irán desglosando en el tránscurso de este capítulo:

- Determinar una representación (CODIFICACIÓN)
- Elegir cómo inicializar una población (GENERAR POBLACIÓN INICIAL)
- Estipular la forma de elegir a los padres (SELECCIÓN DE PADRES)
- Diseñar un operador de cruce (OPERADOR DE CRUCE)

- Diseñar el operador de mutación (OPERADOR DE MUTACIÓN)
- Decidir el remplazo poblacional (REMPLAZAMIENTO GENERACIONAL)
- Determinar la forma de evaluar a los individuos (FUNCIÓN OBJETIVO)
- Dictaminar la condición de parada (CONDICIÓN DE PARADA)

El funcionamiento de un algoritmo genético sigue el esquema de la Figura 2:

PROCEDIMIENTO ALGORITMO GENÉTICO

```
poblacion = generar()
repeat:
    parents = seleccionar ()
    children = cruzar ( parents )
    mutados = mutar ( individuos )
    poblacion = recombinar ( individuos )
until Stop ( poblacion )
return poblacion
```

Según la implementación concreta de cada algoritmo las funciones: **seleccionar, cruzar, mutar, recombinar, stop** variarán según el diseño elegido.

***Individuos** hace referencia a hijos, padres o hijos+padres.

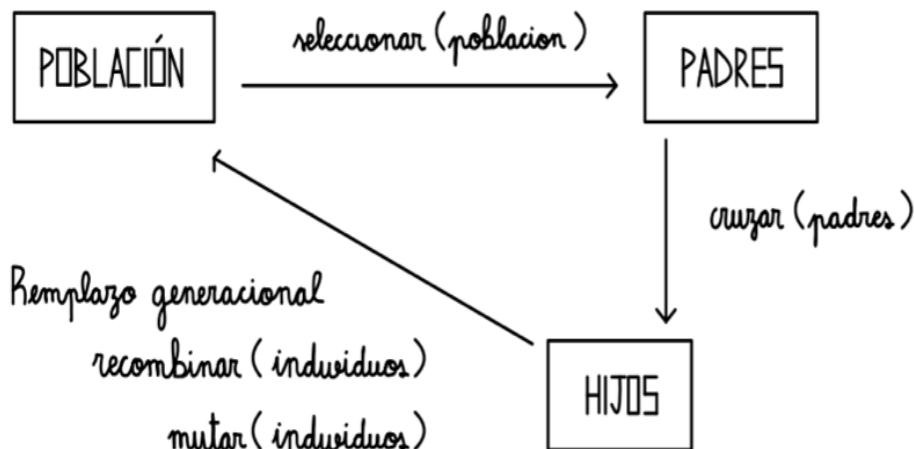


Figura 2: Esquema general de los Algoritmos Genéticos.

2.3 Codificación

Uno de los primeros pasos en la construcción de los AG's es la elección de una representación adecuada para codificar los individuos, esta tarea está intrínsecamente ligada a cada problema en particular. Las representaciones más usuales para llevar a cabo la codificación son las siguientes:

- **Binaria:** los cromosomas se representan por cadenas de 0's y 1's, sirve para representar tanto variables de decisión binarias, enteras o incluso reales.
- **Entera:** codificamos las variables con números enteros. Resulta útil en problemas donde hay que ordenar algo, por ejemplo, el **problema del viajante de comercio**.
- **En árbol:** representamos al individuo por medio de un árbol con ciertos objetos.

- **Codificación por valor directo:** aplicaremos este tipo de codificación para aquellos problemas en los que se necesite el empleo de valores de cifrado complicados. Cada cromosoma está formado por números decimales, cadenas de caracteres o una combinación de ambos.

Estableciendo una analogía con la terminología genética tenemos: las cadenas que representan a los individuos juegan el papel de *cromosoma*, cada posición de la cadena es conocida como *gen*, los valores que toma un gen se les llama *alelo*, y por último el *genotipo* es la codificación elegida (binaria, entera...).

2.4 Generar población inicial

Para inicializar el algoritmo debemos ofrecer una primera población, pero la cuestión es cómo ofrecer esta población inicial y cuál debe de ser el tamaño apropiado de esta.

Respondiendo a la primera cuestión tenemos dos posibilidades:

- **Aleatoria:** la primera opción y la más sencilla es engendrar una población de forma totalmente aleatoria, generando los individuos al azar.
- **Heurística:** otra opción, podría ser llevar a cabo previamente un bosquejo poblacional con otra técnica heurística independiente. Si bien es cierto que podríamos aligerar su convergencia, sería conveniente supervisar una posible convergencia precipitada hacia óptimos locales.

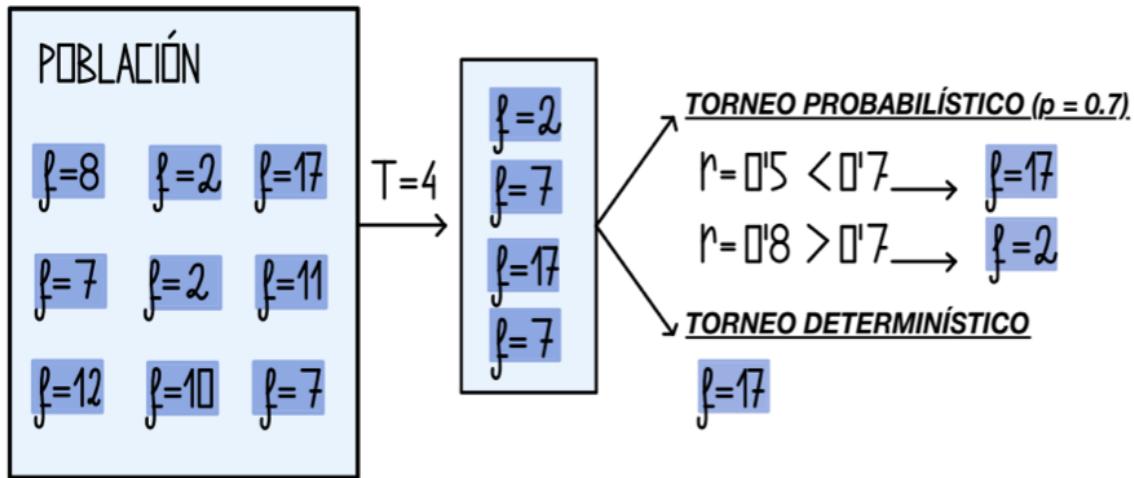
Por otro lado, nos planteamos la discusión sobre el tamaño idóneo de la población, pues las poblaciones reducidas son propensas a no cubrir correctamente el espacio de búsqueda, en cambio, trabajar con poblaciones muy extensas puede generar problemas asociados al excesivo costo computacional.

Una vez establecida nuestra población inicial, el algoritmo comienza a ejecutarse y operar iterativamente, renovando la población de diversas maneras según el algoritmo planteado, como se explicará posteriormente.

2.5 Selección de padres

En la naturaleza, los individuos más capacitados tendrán mayor posibilidad de reproducirse, aunque no serán los únicos. A imitación de esto, los AG's tratarán de seleccionar a los individuos más aptos. Para llevar a cabo esta tarea podremos elegir entre varios procedimientos como pueden ser los siguientes:

- **Selección por torneo:** formamos un subgrupo de tamaño K (tamaño del torneo) de individuos tomados de forma aleatoria con o sin remplazamiento y seleccionamos uno de ellos, según la forma de elegir a este, distinguimos dos posibles tipos de torneos. **Torneo determinístico:** entre todos los sujetos del subconjunto nos quedamos con el más apto. **Torneo probabilístico:** generamos un número aleatorio r siguiendo una distribución uniforme $\mathcal{U}(0,1)$ y de probabilidad $p > 0,5$ que determinará cuándo se elige el mejor, en caso contrario tomaremos al peor.



f representa la función de adaptación.

Figura 3: Ejemplo de Selección por Torneo.

- **Método de la ruleta:** a cada individuo le asignamos su aptitud A_i , el cual tendrá una porción de la ruleta, mayor o menor en función de A_i . Se hace girar la ruleta quedándonos con el individuo en el cual se detenga esta. Aquellos individuos más aptos saldrán con mayor probabilidad. En caso de que las probabilidades se diferencien mucho, este método no favorecerá la diversidad genética.

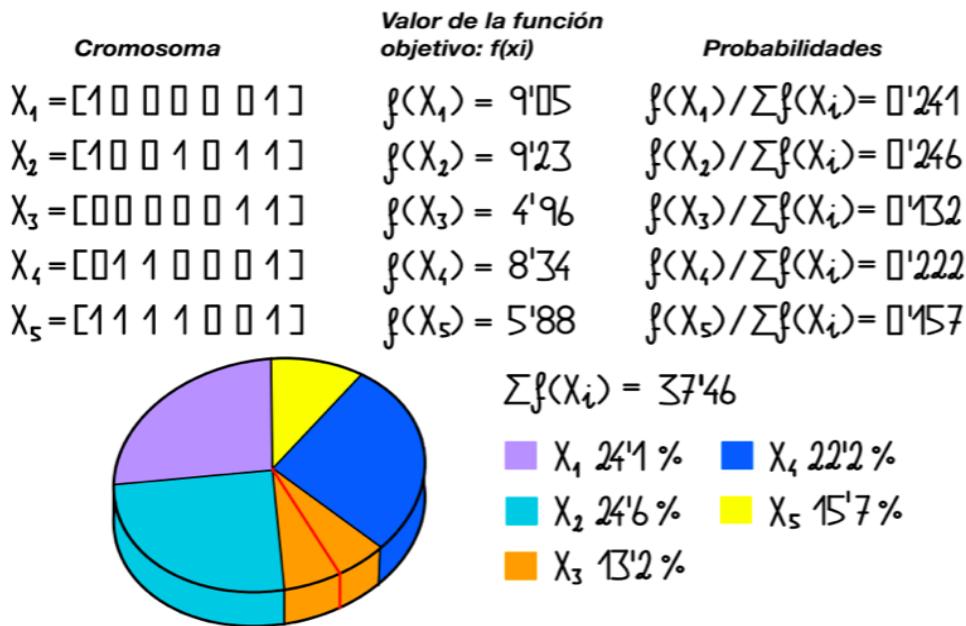


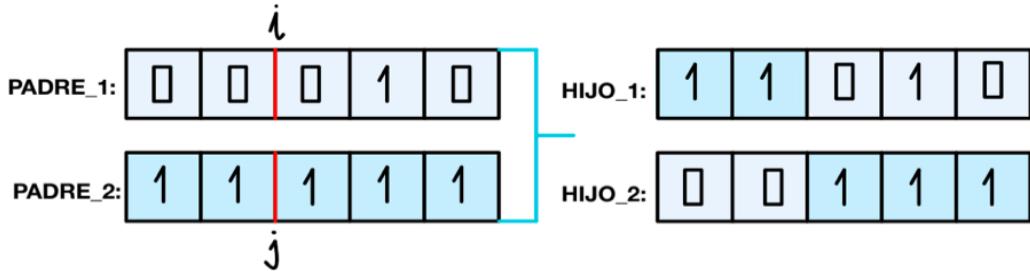
Figura 4: Ejemplo de Método por Ruleta.

Hay otros muchos métodos para realizar esta selección como pueden ser **selección elitista**, **selección por rango**, **selección por estado estacionario**, **selección escalada**... en todos ellos tratamos de garantizar que aquellos individuos mejor adaptados tengan mayores posibilidades de reproducirse, y según sea el método se alentará en mayor o menor medida a la biodiversidad.

2.6 Operador de cruce

Una vez realizada la selección de progenitores procedemos a la reproducción o cruce de estos individuos. Para ello haremos uso de nuestro operador de cruce donde su principal objetivo es mezclar el material genético con el anel de conseguir descendientes mejor adaptados. Hay diferentes formas de realizar el cruce como bien explicaremos a continuación:

- **Cruce por 1 punto:** sean i, j los puntos elegidos en la cadena de cada padre, formamos 4 subcadenas y las concatenamos según se muestra en la Figura 5:



Los índices i, j no tienen necesariamente que coincidir.

Figura 5: Ejemplo Cruce 1 punto.

- **Cruce por N puntos:** se determinan aleatoriamente los puntos de cruce de los dos progenitores y formamos a los hijos concatenando de forma alterna los genes del PADRE_1 con los del PADRE_2.

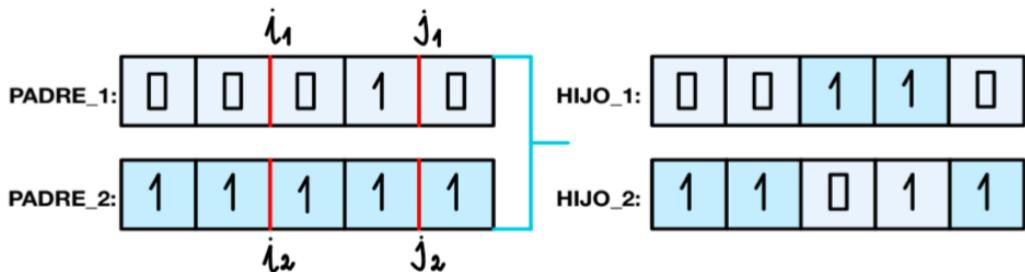


Figura 6: Ejemplo Cruce N puntos.

- **Cruce uniforme:** sea pm la probabilidad de cruce, generamos un número aleatorio r , si $r < pm$ el elemento del primer y segundo parente pertenecerá al primer y segundo hijo respectivamente; en caso contrario se intercambiarán.

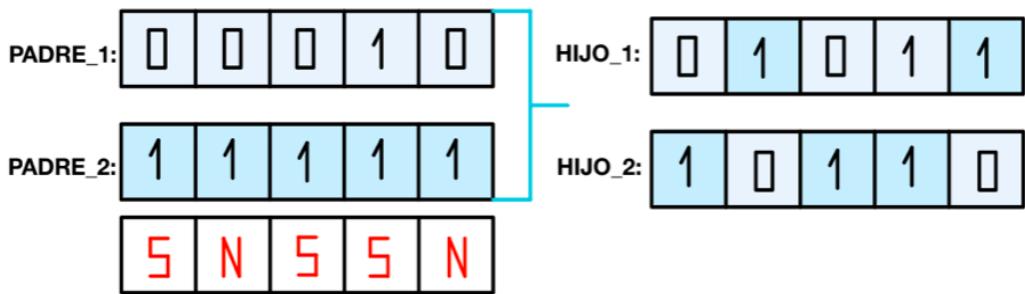


Figura 7: Ejemplo Cruce Uniforme.

2.7 Operador de mutación

Tras el cruce tienen lugar las mutaciones las cuales suelen aportar variedad genética a la especie. Además, en el funcionamiento del algoritmo ayudan a evitar el estancamiento de las poblaciones en óptimos locales.

El operador de mutación depende de la codificación del problema, sin embargo, una forma común de proceder puede ser la siguiente tanto para codificaciones binarias, enteras o reales:

1. Determinamos de forma aleatoria si una posición será mutada por medio de una probabilidad de mutación pm . Para evitar excesivas mutaciones se suele elegir valores bajos de pm , aunque otros algoritmos optan por utilizar valores elevados al comienzo y poco a poco ir reduciéndolo.

2. Asignamos los nuevos valores asociados a las posiciones llamadas a ser mutadas. En el caso de codificación binaria adjudicaríamos el opuesto, en codificación entera sustituimos un valor por otro o los intercambiamos de posición en la cadena, en el caso real se aplican técnicas más sofisticadas.

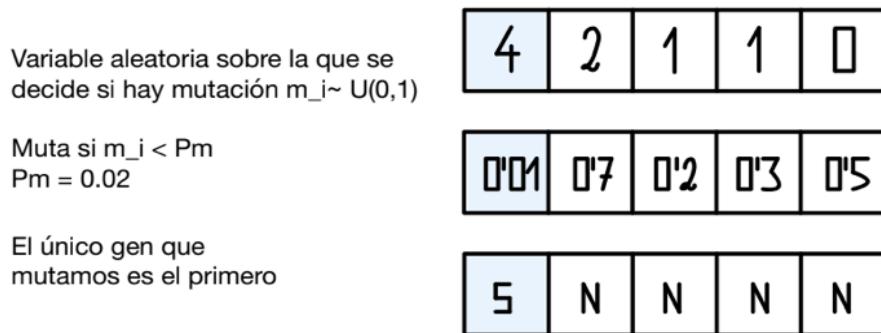


Figura 8: Ejemplo de Mutación para $pm = 0.02$.

Tal y como se dice en [10], “Si bien el operador de cruce se encarga de hacer una búsqueda en el espacio de posibles soluciones, es el operador de mutación el encargado de aumentar o reducir el espacio de búsqueda en un algoritmo genético y de proporcionar cierta variabilidad genética de los individuos.” Ahora bien, abusar de este operador puede conducirnos a una simple búsqueda aleatoria.

2.8 Remplazamiento generacional

Una vez finalizado el proceso de reproducción y mutación debemos seleccionar a los individuos, en general, a los más aptos que pasarán a la siguiente generación. Aquí exponemos dos formas posibles de proceder:

- **Sustitución generacional:** en la fase de reproducción debemos garantizar que se creen tantos hijos como individuos hay en la población, pues todos estos pasarán a formar parte de la nueva generación y todos los padres desaparecerán. Al sustituir completamente la población por una nueva podríamos perder la mejor solución encontrada hasta el momento, luego será necesario guardar de alguna manera la mejor solución.

- **Elitismo:** en cada población forzamos a contener al mejor individuo, el resto se eligen entre los nuevos o entre los nuevos y los antiguos, pero favoreciendo a los mejor adaptados.

2.9 Función objetivo

Uno de los aspectos más importantes dentro del buen funcionamiento de los AG's es la determinación de una apropiada función fitness, donde a cada individuo se le puntuá, a través de esta función, la bondad de dicha solución. Tal y como se indica en [3], “La regla general para construir una buena función objetivo es que esta debe reflejar el valor del individuo de una manera “real”, pero en muchos problemas de optimización combinatoria, donde existen gran cantidad de restricciones, buena parte de los puntos del espacio de búsqueda representan individuos no válidos.” En aquellos problemas donde los individuos están sometidos a demasiadas restricciones de factibilidad se han diseñado varias soluciones. En primer lugar, una posible solución para aquellos individuos infactibles es no considerarlos hasta que, tras efectuar cruces y mutaciones, obtengamos individuos aptos, o bien, otra forma es evaluar a estos con asignación nula en la función objetivo. Otro enfoque consiste en la penalización del individuo dentro de la función objetivo.

2.10 Condición de parada

Tras la generación de una nueva población evaluamos si debemos parar, en caso de hacerlo escogeremos al individuo mejor adaptado y lo ofreceremos como solución, en caso contrario volveremos a generar una nueva población por medio de cruces y mutaciones. Los criterios de parada más habituales son los siguientes: número de iteraciones, tiempo, aptitud suficiente de los mejores individuos...

2.11 Ventajas e inconvenientes

Dentro de las ventajas que podemos observar en los algoritmos genéticos es la posibilidad de poder trabajar con varias soluciones simultáneamente, a diferencia de los métodos tradicionales que exploran el espacio de soluciones secuencialmente.

Además, son capaces de resolver una amplia gama de problemas a base de realizar cambios aleatorios y estudiar si producen una mejora a través de su función objetivo donde podemos manipular muchos parámetros a la vez.

En contraposición, podremos encontrar dificultades a la hora de determinar una adecuada representación del problema, pues debe estar diseñado para tolerar cambios sin producir resultados sin sentido.

Uno de los problemas más importantes de los algoritmos genéticos está relacionado con la convergencia: confluencia prematura hacia óptimos locales o incluso la no convergencia, por lo que posteriormente los resultados obtenidos deberán ser analizados. Entre las posibles causas se encuentran: el temprano nacimiento de un superindividuo que disminuya la variedad de la población, pocas iteraciones o el pequeño tamaño de la población.

Si bien es cierto, se espera del algoritmo que converja al óptimo del problema, no hay garantía de que este encuentre la solución óptima, aunque existen evidencias empíricas de que se puede hallar un nivel aceptable de solución en un tiempo razonable.

2.12 Aplicaciones

Una de las aplicaciones más importantes de los AG's es la resolución de problemas de optimización. Para poder aplicar fácilmente esta técnica debemos tener en cuenta ciertas características: debemos ser capaces de poder definir una función objetivo apropiada, elegir una codificación fácil de implementar en el ordenador, espacio de búsqueda limitado...

3 PROGRAMACIÓN DE HORARIOS

En esta sección, nos centraremos en analizar completamente el problema relacionado con la “**Planificación de Horarios de Exámenes**”, una de las tareas más frecuentes y a la vez más importantes en entidades académicas debido a la gran cantidad de alumnos que se ven afectados por los calendarios de exámenes. Para poner en marcha nuestro estudio realizaremos las siguientes tareas: planteamos un modelo matemático, detallamos su implementación en un programa informático, extraemos un conjunto de datos ya testeado, comparamos los resultados obtenidos con otras soluciones ya planteadas, valoramos la eficiencia del algoritmo propuesto, y finalmente exponemos diversas ideas para la realización de futuros proyectos. Para poder abarcar con todo ello, nos apoyaremos en las fuentes [11–17] de la bibliografía.

3.1 Planteamiento y modelización

A continuación, detallamos el modelo matemático propuesto para la planificación de un calendario de exámenes basándose en los artículos [13] y [17]. Para ello, debemos tener en cuenta las diversas restricciones y políticas que se establecen en las instituciones académicas, y así, poder satisfacerlas en la medida de lo posible, intentando proponer un horario ideal.

En concreto, la organización de un horario de exámenes se identifica, en líneas generales, con la asignación de un conjunto de exámenes dentro de un número limitado de franjas horarias ligadas a un conjunto de restricciones.

Las restricciones las podemos clasificar en restricciones fuertes y débiles. Las restricciones duras deben ser cumplidas completamente, aquellas soluciones donde se satisfacen todas las restricciones de este tipo se denominan soluciones factibles. Entre las restricciones estrictas más comunes se encuentran: (i) ningún estudiante puede tener dos exámenes al mismo tiempo, (ii) el número de estudiantes que realizan un examen no puede sobrepasar la capacidad del aula donde se realiza. Sin embargo, hay que tener en cuenta que dentro de cada institución se pueden exigir restricciones particulares de obligado cumplimiento. Por otro lado, existen requisitos que son prescindibles pero deben cumplirse en la medida de lo posible, siendo estas las restricciones blandas, las cuales consisten en distribuir los exámenes de la manera más uniforme posible, evitando concatenar varias pruebas seguidas; en este grupo también se encuentran las preferencias específicas de cada centro. Debido a la complejidad del problema, en general, no es posible satisfacer todas estas condiciones. En efecto, la calidad de un horario será evaluada en base a alguna función de coste que mida el incumplimiento en las restricciones débiles, donde se asocia un valor de penalización ponderado cada vez que se viole una de estas restricciones, teniendo como objetivo minimizar el valor de penalización total.

En particular, el problema se plantea de la siguiente manera: *Se acerca la época de exámenes y la secretaría académica de cierta universidad tiene la tarea de organizar un horario de exámenes donde se dispone de un número limitado de franjas horarias para la realización de estos (H_1, H_2, \dots, H_P). Además, deben de ser conscientes de no solapar ninguna prueba para ningún alumno, es decir, si el alumno*

A_1 se presenta a los controles E_1 , E_2 no se permite un horario cuyos exámenes E_1 , E_2 se dispongan a la misma hora. Para optimizar las calificaciones de los alumnos y evitar su saturación procurarán distribuir los exámenes lo más espaciados posibles.

En particular, el diseño de nuestro modelo matemático presenta las siguientes variables:

- \mathbf{N} = número de exámenes (E_1, E_2, \dots, E_N)
- \mathbf{M} = número de estudiantes (A_1, A_2, \dots, A_M)
- \mathbf{P} = número de horas disponibles (H_1, H_2, \dots, H_P)
- \mathbf{C} = matriz $(C_{ij})_{NXN}$ donde cada elemento dado por C_{ij} es el número de alumnos que realiza ambos exámenes E_i y E_j . Se trata de una matriz simétrica donde los elementos diagonales C_{ii} indican el número de estudiantes que se presentan al examen E_i .

Presentamos la solución al problema de la siguiente manera:

- $\mathbf{T} = (t_k)_N$ donde cada t_k especifica la franja horaria en la que se realiza E_k , luego $1 \leq t_k \leq P$.

La factibilidad del problema, siendo la restricción fuerte a cumplir: no solapar ningún examen para ningún alumno, viene dada a través de la siguiente fórmula:

$$\sum_{i=1}^{N-1} \sum_{j=i+1}^N C_{ij} * igualH(t_i, t_j) = 0 \text{ donde } igualH(t_i, t_j) = \begin{cases} 1 & \text{si } t_i = t_j \\ 0 & \text{otro caso} \end{cases} \quad (1)$$

Además, para poder medir la optimalidad de una solución, e intentar satisfacer la condición débil de distribución uniforme de exámenes, utilizamos el siguiente criterio: si un alumno tiene dos exámenes consecutivos le asignamos una penalización igual a 16. A dos exámenes con una hora de descanso entre ellos se le asignará un valor de penalización de 8 y así sucesivamente. Para tener una medida relativa, esta suma se fracciona por el número total de alumnos. En este caso, debemos minimizar la expresión:

$$\frac{\sum_{i=1}^{N-1} \sum_{j=i+1}^N C_{ij} * prox(t_i, t_j)}{M} \text{ donde } prox(t_i, t_j) = \begin{cases} 2^{5-|t_i-t_j|} & \text{si } 1 \leq |t_i - t_j| \leq 5 \\ 0 & \text{otro caso} \end{cases} \quad (2)$$

Por lo tanto, la formulación del problema queda de la siguiente manera:

$$\begin{aligned} \min & \quad \frac{\sum_{i=1}^{N-1} \sum_{j=i+1}^N C_{ij} * prox(t_i, t_j)}{M} \\ \text{S.A.} & \quad \sum_{i=1}^{N-1} \sum_{j=i+1}^N C_{ij} * igualH(t_i, t_j) = 0 \end{aligned} \quad (3)$$

3.2 Implementación

Para poder resolver el problema de cronometrización de horario, buscamos la solución a partir de la aplicación de la técnica heurística conocida como Algoritmos Genéticos. Para ello, desarrollamos un pequeño programa en Python donde implementamos diferentes funciones para aplicar los conceptos vistos en la Sección 2.

Para detallar las estrategias utilizadas durante el diseño del programa, proponemos un pequeño problema a modo de ejemplo ficticio que nos ayudará a entender mejor el funcionamiento e implementación del programa expuesto.

EJEMPLO. Tratamos de encontrar un calendario de exámenes con los siguientes datos: **N = EXAMENES = 5**; **M = ALUMNOS = 10**; **P = HORAS = 7**.

EXÁMENES = 5
HORAS = 7

VECTOR_C = diagonal de la matriz C

$E_0 \quad E_1 \quad E_2 \quad E_3 \quad E_4$

2	3	1	5	4
---	---	---	---	---

EL VECTOR C ES:
[2, 3, 1, 5, 4]

LA MATRIZ MATRIZ_C ES:
[[2, 5, 3, 7, 6], [3, 4, 8, 7], [1, 6, 5], [5, 9], [4]]]

MATRIZ_C = matriz C

	E_0	E_1	E_2	E_3	E_4
E_0	2	5	3	7	6
E_1	—	3	4	8	7
E_2	—	—	1	6	5
E_3	—	—	—	5	9
E_4	—	—	—	—	4

Figura 9: Datos para un ejemplo ficticio.

La **MATRIZ_C** aporta los datos relacionados con los exámenes y n^o de alumnos que realizan los mismos, correspondiente con la matriz C de la Subsección 3.1; el **VECTOR_C** se corresponde con la diagonal de **MATRIZ_C**. Estas variables son construidas a partir de los archivos que contienen los datos de entrada a través de **def lectura (dataset, forma, examenes, horas, alumnos)** y **def matrizC (vectorC)**.

A continuación explicamos las diferentes funciones, variables, constantes y diversas cuestiones utilizadas para el funcionamiento del algoritmo genético diseñado:

- **CODIFICACIÓN:** para la representación del problema utilizaremos una codificación entera siguiendo el esquema de la Figura 10:

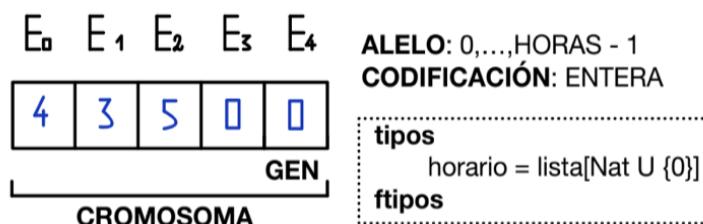


Figura 10: Codificación de una solución.

- **GENERAR POBLACIÓN:** de acuerdo con lo visto en la Sección 2 podemos generar una población inicial de forma aleatoria o con alguna otra técnica heurística independiente.

P: { tamano: ≥ 2 , nat \wedge vector = VECTOR_C \wedge tipo $\in \{1,2\} \wedge$
exámenes = EXAMENES \wedge horas = HORAS }
def population (tamano, vector, tipo, exámenes, horas):
dev población
Q: { población = P[0, ..., tamano - 1] de horarios }

tipo = {1, 2}
tipo = 1 función vectorS1
tipo = 2 función vectorS2

GENERAMOS UNA POBLACIÓN:

$\underbrace{[3, 4, 5, 2, 0]}_{I_1}, \underbrace{[4, 0, 2, 3, 0]}_{I_2}, \underbrace{[0, 0, 3, 1, 6]}_{I_3}]$

I = individuo
TAMANO = 3
tamano = **TAMANO**

Figura 11: Generar una población de horarios.

P: { examenes = EXAMENES \wedge horas = HORAS }
 def vectorS1 (examenes, horas): (tipo = 1)
 dev horasexam
 Q: { horasexam = H[0 ,..., examenes - 1]
 tq para todo i $0 \leq i \leq$ examenes - 1] $0 \leq H[i] \leq$ (horas-1) }

GENERACIÓN HORARIO TIPO 1: FACTIBILIDAD: False
 [3, 0, 1, 1, 4] **APITUD:** 62.29

Figura 12: Generar una población de forma aleatoria.

P: { vector = VECTOR_C \wedge examenes = EXAMENES \wedge horas = HORAS }
 def vectorS2 (vector, examenes, horas):
 dev horasexam
 (tipo = 2)
 Q: { horasexam = H[0 ,..., examenes - 1]
 tq para todo i $0 \leq i \leq$ examenes - 1 $0 \leq H[i] \leq$ (horas-1) }

GENERACIÓN HORARIO TIPO 2: FACTIBILIDAD: False
[3, 4, 5, 3, 2] **APTITUD:** 62.86

Figura 13: Generar una población con una técnica diferente.

La estrategia utilizada en la función `def vectorS2(..)` para generar un horario consiste en seleccionar primero los dos exámenes con mayor número de inscripciones y asignarle dos horas separadas en 6 unidades de tiempo, y continuar así sucesivamente para los siguientes exámenes y horas hasta concretar una hora a todos los exámenes.

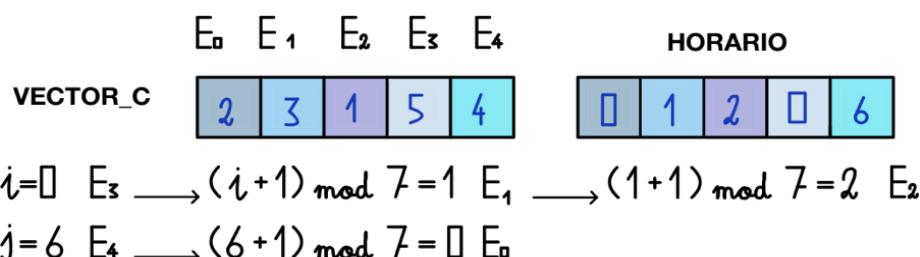


Figura 14: Construir un horario con la función `def vectorS2(..)`.

- **SELECCIÓN PADRES:** para la selección de padres nos hemos decantado por la opción de *torneo determinístico* utilizando la función `def seleccionPadres (poblacion, matriz, k, alumnos)`.

```
P: { poblacion = P[0 ,..., TAMANO - 1] de horarios ∧
    matriz = MATRIZ_C ∧ k >= 2, nat ∧ alumnos = ALUMNOS }
    def seleccionPadres (poblacion, matriz, k, alumnos):
        dev padres
Q: { padres = P[0 ,..., TAMANO - 1] de horarios }
```

k = tamaño del torneo

LOS PADRES SON:

`[[3, 4, 5, 2, 0], [4, 0, 2, 3, 0], [3, 4, 5, 2, 0]]`

Figura 15: Seleccionar padres.

- **OPERADOR DE CRUCE:** en este caso utilizamos tanto el *cruce por N puntos* como el *cruce uniforme*.

```
P: { 0 <= N < EXAMENES ∧ examenes = EXAMENES }
    def creaLista (N, examenes):
        dev lista
Q: { lista = L[0 ,..., N + 1] de ent ∧ L[0] = -1 ∧ L[N + 1] = examenes ∧
      para todo i tq. 0 <= i <= N | L[i] <= L[i + 1] }
```

N = número de puntos para el cruce

```
P: { padre1, padre2: horario ∧ listaN = [0 ,..., N + 1] de ent ∧
    listaN[0] = -1 ∧ listaN[N + 1] = EXAMENES ∧
    para todo i | 0 <= i < N + 1 listaN[i] <= listaN[i+1] }
    def cruceN (padre1, padre2, listaN):
        dev hijo1, hijo2
Q: { hijos1, hijo2: horario }
```

PADRE 1: `[3, 4, 5, 2, 0]` **PADRE 2:** `[4, 0, 2, 3, 0]`

LISTA_N: `[-1, 0, 1, 2, 5]`

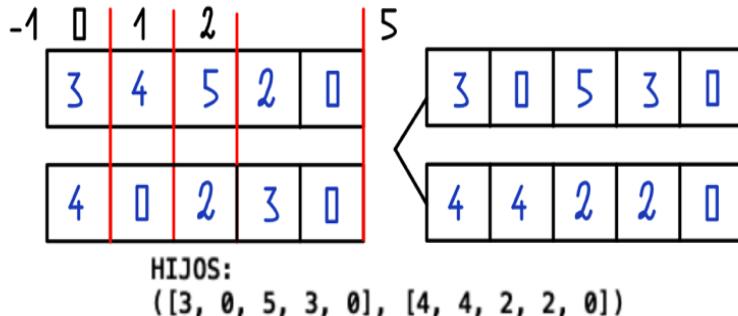


Figura 16: Cruce en N puntos.

La variable *listaN* contendrá la lista de *N* puntos que indican dónde se realizará el cruce entre los padres, el primer y último elemento de la lista se corresponden con los valores -1 y la constante *EXAMENES*, los cuales se utilizan para facilitar la implementación del cruce.

```

P: { padre1, padre2: horario ∧ 0 <= PM <= 1 }
    def cruceU (padre1, padre2, PM):
        dev hijo1, hijo2
Q: { hijos1, hijo2: horario }

PM: probabilidad de cruce
    CRUCE UNIFORME PM = 0.2:
    HIJOS: ([4, 4, 2, 3, 0], [3, 0, 5, 2, 0])

    CRUCE UNIFORME PM = 0.8:
    HIJOS: ([3, 4, 5, 2, 0], [4, 0, 2, 3, 0])

```

Figura 17: Cruce Uniforme.

- **REEMPLAZAMIENTO:** para el remplazamiento poblacional nos hemos decantado por la estrategia conocida como *elitismo* apoyándonos en `def recombinar_Elitismo (poblacion, hijos, matriz, alumnos)`.

```

P: { poblacion = P[0 ,..., TAMANO - 1] de horarios ∧
    hijos = H[0 ,..., TAMANO - 1] de horarios ∧ matriz = MATRIZ_C ∧
    alumnos = ALUMNOS }
    def recombinar_Elitismo (poblacion, hijos, matriz, alumnos):
        dev nueva_generación
Q: { nueva_generacion = N[0 ,..., TAMANO - 1] de horarios }

```

Figura 18: Elitismo.

- **MUTACIÓN:** la mutación se realiza como se indicó en la Subsección 2.7 con la función `def mutacion (calen, PM, horas)`.

```

P: { calen: horario ∧ 0 <= PM <= 1 ∧ horas = HORAS }
    def mutacion (calen , PM, horas):
        dev calen
Q: { calen: horario }

```

PM: probabilidad de mutación

VECTOR QUE SERÁ MUTADO: <code>[3, 6, 4, 0, 1]</code>	MUTACIÓN PARA PM = 0.2: <code>[2, 6, 5, 0, 1]</code>
	MUTACIÓN PARA PM = 0.8: <code>[4, 1, 4, 0, 5]</code>

Figura 19: Mutación.

- **FUNCIÓN OBJETIVO:** la función objetivo a minimizar es la propuesta por la fórmula (2) de la Subsección 3.1.
`def funcionObj (horario, matriz, alumnos)` según fórmula (2)
`def factibilidad (horario, matriz)` según fórmula (1). Por cuestiones de eficiencia para diseñar esta función sabemos que una vez la fórmula (1) tome un valor mayor que cero, ese horario ya no va a ser factible, luego esta visión mejorará considerablemente la eficiencia del programa.

`def mejorF (horario1, horario2, matriz, alumnos)` definimos la comparación entre dos horarios, para quedarnos con aquel que además de ser factible, el coste asociado a la función objetivo definida sea el mínimo.

`def el_mejorP (poblacion, matriz, alumnos)` buscamos el mejor horario de una población concreta.

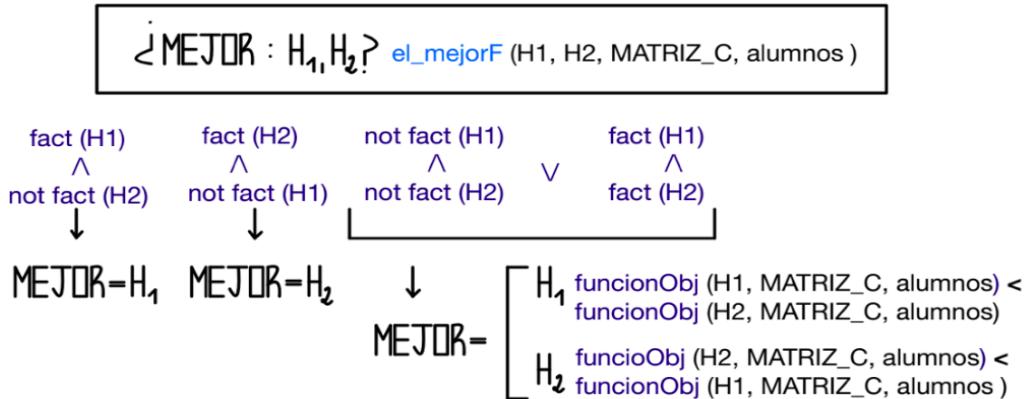


Figura 20: Criterio para elegir el mejor horario.

- **ALGORITMO GENÉTICO:** implementamos dos algoritmo `def algoritmoGeneticoV (matriz, vector, tipo, tamano, vueltas, k, examenes, horas, alumnos, necesito)` y `def algoritmoGeneticoT (matriz, vector, tipo, tamano, tiempo, k, examenes, horas, alumnos, necesito)` basadas en el número de iteraciones - tamaño y tiempo de espera respectivamente.

```
P: { matriz = MATRIZ_C ∧ vector = VECTOR_C ∧ tipo ∈ {1, 2} ∧ tamano >= 2, nat+ ∧
    vueltas : nat+ ∧ k >= 2, nat ∧ examenes = EXAMENES ∧ horas = HORAS ∧
    necesito : bool ∧ alumnos = ALUMNOS }
    def algoritmoGeneticoV (matriz, vector, tipo, tamano, vueltas, k, examenes , horas,
    alumnos, necesito):
        dev mejor, mejores
Q: { mejor: horario ∧ mejores = lista con el mejor horario de cada iteración }
```

(a) Algoritmo Genético (Vueltas)

```
P: { matriz = MATRIZ_C ∧ vector = VECTOR_C ∧ tipo ∈ {1, 2} ∧ tamano >= 2, nat ∧
    tiempo > 0 ∧ k >= 2, nat ∧ examenes = EXAMENES ∧ horas = HORAS ∧
    necesito: bool b ∧ alumnos = ALUMNOS }
    def algoritmoGeneticoT (matriz, vector, tipo, tamano, tiempo, k, examenes, horas,
    alumnos, necesito):
        dev mejor, listaT, mejores
Q: { mejor: horario ∧ listaT = lista de tiempo ∧ mejores = lista con el mejor horario de
    cada iteración }
```

Tiempo: en segundos

(b) Algoritmo Genético (Tiempo)

Figura 21: Algoritmos Genéticos.

Encontramos las funciones utilizadas en los archivos `prueba.txt`, `TFG_AG.py` y `TFG_EjemploFicticio.py` del siguiente enlace https://github.com/Siria-Iniguez/HORARIOS_AG_TFG.git.

3.3 Datos

Para la elaboración y completitud de este artículo, necesitamos contrastar la validez y eficiencia de los algoritmos propuestos, por lo que para ello, nuestros experimentos tomarán como conjuntos de datos aquellos publicados por la comunidad de investigadores en la siguiente página web: <http://www.cs.nott.ac.uk/~pszrq/data.htm>. En concreto, nos enfocaremos en los datos relacionados con las entidades académicas que mostramos en la Tabla 1 :

DATOS	CENTRO	EXÁMENES	ALUMNOS INSCRIPCIONES	HORAS
CAR91 Carleton91.exm	Carleton University, Ottawa Toronto	682	16925	35
			56877	
CAR92 Carleton92.exm	Carleton University, Ottawa	543	18419	32
			55522	
EAR83 EarlHaig83.exm	Earl Haig Collegiate Institute, Toronto	190	1125	24
			8109	
HEC92 EdHEC92.exm	Ecole des Hautes Etudes Commerciales, Montereal	81	2823	18
			10632	
KFU93 KingFahd93.exm	King Fahd University, Dharan	461	5349	20
			25113	
LSE91 LSE91.exm	London School of Economics	381	2726	18
			10918	
STA83 St.Andrews83.exm	St Andrew's Junior High School, Toronto	139	611	13
			5751	
TRE92 Trent92.exm	Trent University, Peterborough, Ontario	261	4360	23
			14901	
UTA92 TorontoAS92.exm	Faculty of Arts and Sciences, University of Toronto	622	21266	35
			58979	
UTE92 TorontoE92.exm	Faculty of Engineering, University of Toronto	184	2750	10
			11793	
YOR83 YorkMills83.stu	York Mills Collegiate Institute, Toronto	181	941	21
			6034	

Tabla 1: Conjunto de datos de entrada.

En referencia a los datos de entrada, damos una pequeña explicación de la configuración de estos: los datos se encuentran en formato **.exm** ó **.stu**, estructurados de la siguiente manera:

- **.exm:**

COLUMNA_1 = Examen

COLUMNA_2 = Número de alumnos que realizan el examen

Número de líneas / filas = Número de exámenes.

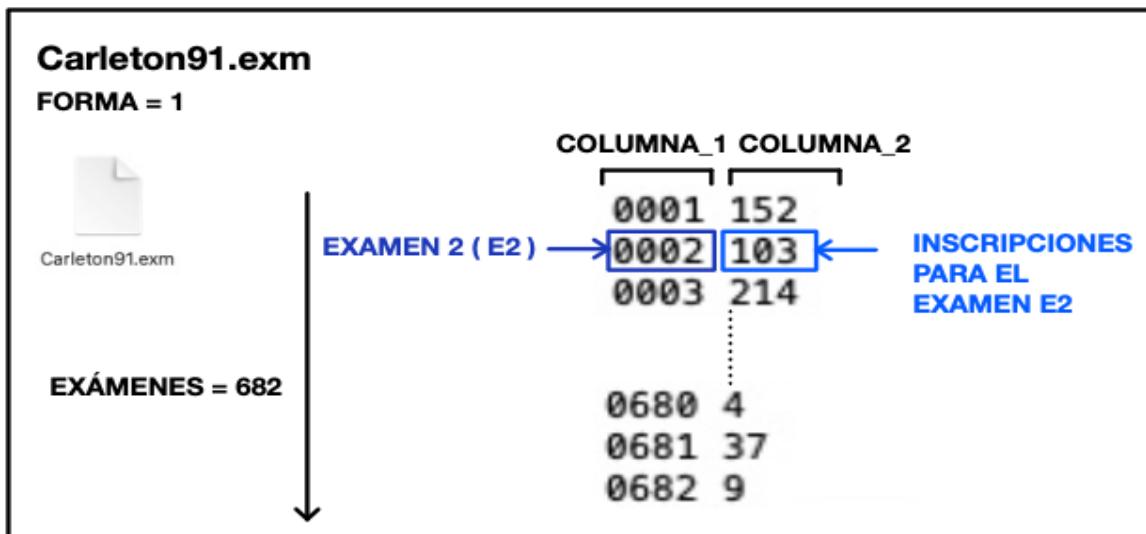


Figura 22: Archivo Carleton91.exm.

- **.stu:**

Se representa a los alumnos en las diferentes filas de la primera columna, y a su derecha se muestran los distintos exámenes a los que está inscrito; en el banco de datos también se ofrecen otras formas de distribuir los datos. En este caso, adjuntamos el formato seleccionado en nuestra recopilación:

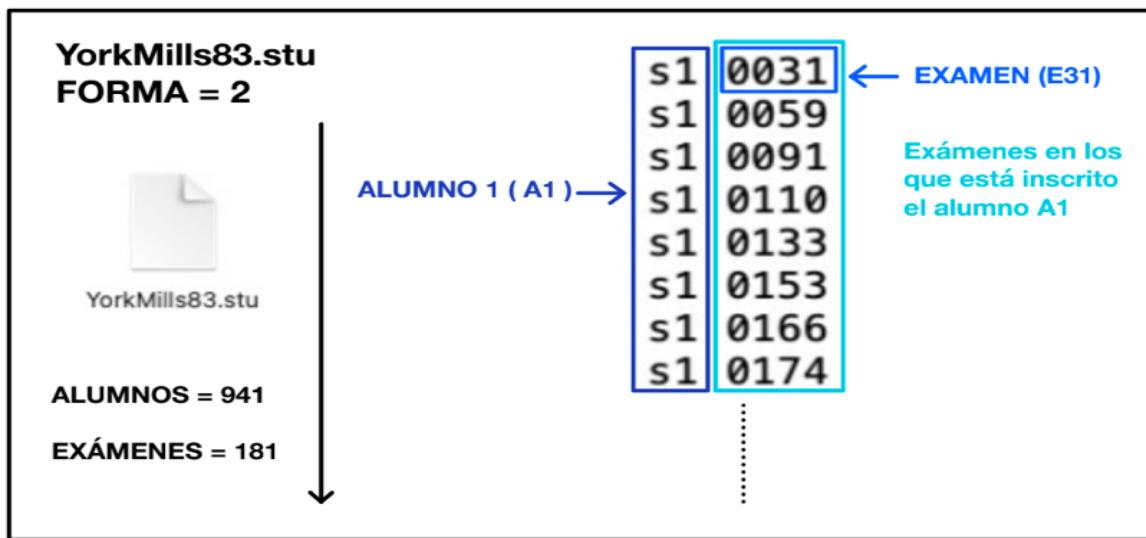


Figura 23: Archivo YorkMills83.stu.

Por otro lado, en lo que se refiere a los datos de salida, las soluciones ofrecidas por Pilar Moreno Díaz en [17] se presentan como un archivo de extensión .sol estructurado como se indica en la Figura 24, cuya lectura se realiza con la función **def lecturaS (datasal, examenes)**.

Carleton91.sol		COLUMNA_1	COLUMNA_2
	Carleton91.sol	EXAMEN (E261) → 0261	32 ← HORA 32 (H32)
	EXÁMENES = 682	0262 0660	3 22
		⋮	⋮
		0146 0657 0349	1 4 16

Figura 24: Archivo Carleton91.sol.

En este caso, cabe destacar que el n^o de horas empieza indexado en cero, es decir, si para el problema X hay 21 franjas horarias, en las soluciones propuestas las horas recorren el siguiente conjunto $\{0, \dots, 20\}$. En nuestro programa, se trata de una lista que representa el vector que se describió en la Subsección 3.1 de modelización, indexando el número de exámenes y franjas horarias desde 0.

DATASET	ARCHIVO
CAR91	Carleton91.sol
CAR91	Carleton92.sol
EAR83	EarlHaig83.sol
HEC92	EdHEC92.sol
KFU93	KingFahd93.sol
LSE91	LSE91.sol

DATASET	ARCHIVO
STA83	St.Andrews83.sol
TRE92	Trent92.sol
UTA92	TorontoAS92.sol
UTE92	TorontoE92.sol
YOR83	YorkMills83.sol

Tabla 2: Conjunto de archivos de las soluciones “Results”, UAX.

3.4 Resultados y valoraciones

Con el fin de comprobar la validez de los algoritmos propuestos a la hora de resolver el problema de cronometrización de exámenes, estos serán sometidos a diferentes análisis donde compararemos los resultados obtenidos con soluciones ya propuestas, para así poder derivar conclusiones acertadas.

Para analizar los dos algoritmos diseñados: `def algoritmoGeneticoV (matriz, vector, tipo, tamano, vueltas, k, examenes, horas, alumnos, necesito)` y `def algoritmoGeneticoT (matriz, vector, tipo, tamano, tiempo, k, examenes, horas, alumnos, necesito)` necesitamos conocer las variables que influyen en la búsqueda de un horario ideal, por lo que llevamos a cabo los siguientes estudios:

(X -Y, Z)	ALGORITMO	TIPO
X = comienzo (tamaño, vueltas, tiempo)	A.1 = algoritmoGeneticoV(..)	T.1 = vectorS1(..)
Y = final (tamaño, vueltas, tiempo)	A.2 = algoritmoGneticoT(..)	T.2 = vectorS2(..)
Z = paso		

Figura 25: Leyenda.

- **ESTUDIO POR Nº DE VUELTAS FIJO:** buscamos estudiar la influencia al aumentar el tamaño de la población fijando el número de vueltas en `def algoritmoGeneticoV(..)`.

DATOS	TAMAÑO	VUELTAS	TIPO
HEC92	(40 - 165, 5)	100	vectorS1(..)
HEC92	(40 - 165, 5)	150	vectorS1(..)

Tabla 3: Datos para el estudio por número de vueltas fijo.

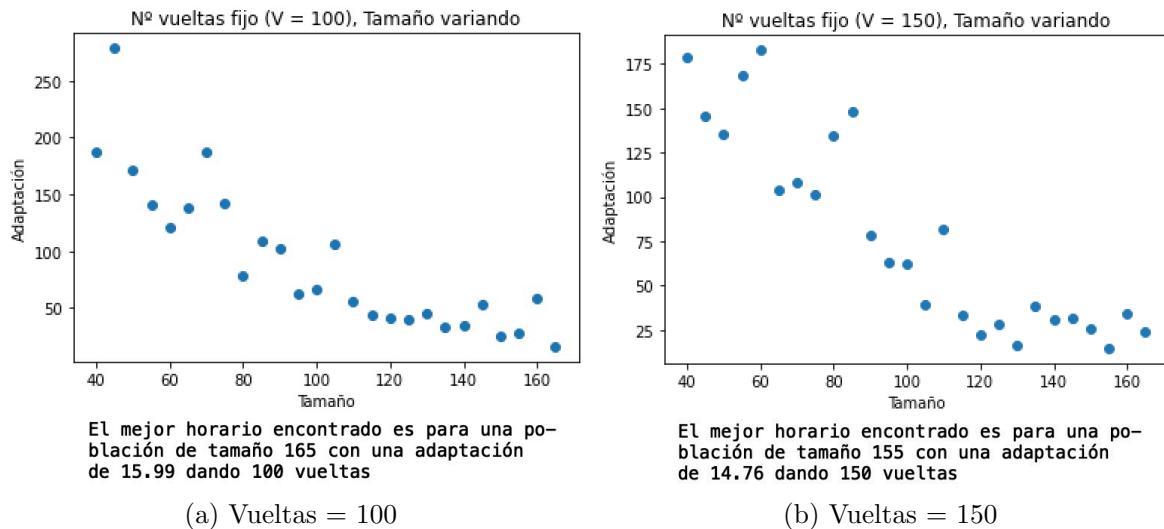


Figura 26: Estudio con número fijo de vueltas.

Si nos fijamos en los resultados para 150 vueltas, el mejor horario encontrado es aquel con una función objetivo igual a 14,76, en cambio si tomamos 100 vueltas, obtenemos 15,99 como mejor resultado. En comparación con los resultados obtenidos en [17], cuyo valor de la función objetivo es 528,41, podemos destacar la lejanía entre las soluciones. En un principio, podríamos pensar en la infactibilidad de las soluciones encontradas por nuestro algoritmo, pero tras comprobar que la solución facilitada resulta ser infactible, podríamos estar ante un problema sin solución factible, luego, en ese caso optaríamos por aquel horario que nos ofrece menor penalización en la función objetivo, pues estamos trabajando con un problema de minimización.

En concreto, atendiendo a ambas gráficas, podemos notar una tendencia decreciente a medida que aumentamos el tamaño de la población, luego aumentar la variable *tamaño* parece conducir a soluciones cada vez mejores, aunque también es de esperar que el aumento excesivo de esta no conduzca a ninguna mejoría y solo empeore el tiempo computacional.

Además, aumentando el número de vueltas, no hemos conseguido mejorar excesivamente los resultados, pues en las gráficas 26a y 26b el eje y, ‘Adaptación’, recorre en ambas un rango de (200 – 15) aproximadamente. De igual forma, las mejores soluciones arrojadas en ambas situaciones muestran resultados con una diferencia ínfima.

- **ESTUDIO POR TAMAÑO FIJO:** queremos conocer si existe alguna relevancia en la búsqueda de soluciones fijando el tamaño de la población y aumentando el n^o de iteraciones en `def algoritmogeneticoV(..)`, atendiendo a cómo se genera la población inicial.

DATOS	TAMAÑO	VUELTAS	TIPO
STA83	100	(100 - 700, 50)	vectorS1(..)
STA83	150	(100 - 700, 50)	vectorS1(..)
STA83	100	(100 - 700, 50)	vectorS2(..)
STA83	150	(100 - 700, 50)	vectorS2(..)

Tabla 4: Datos para el estudio por tamaño de la población fijo.

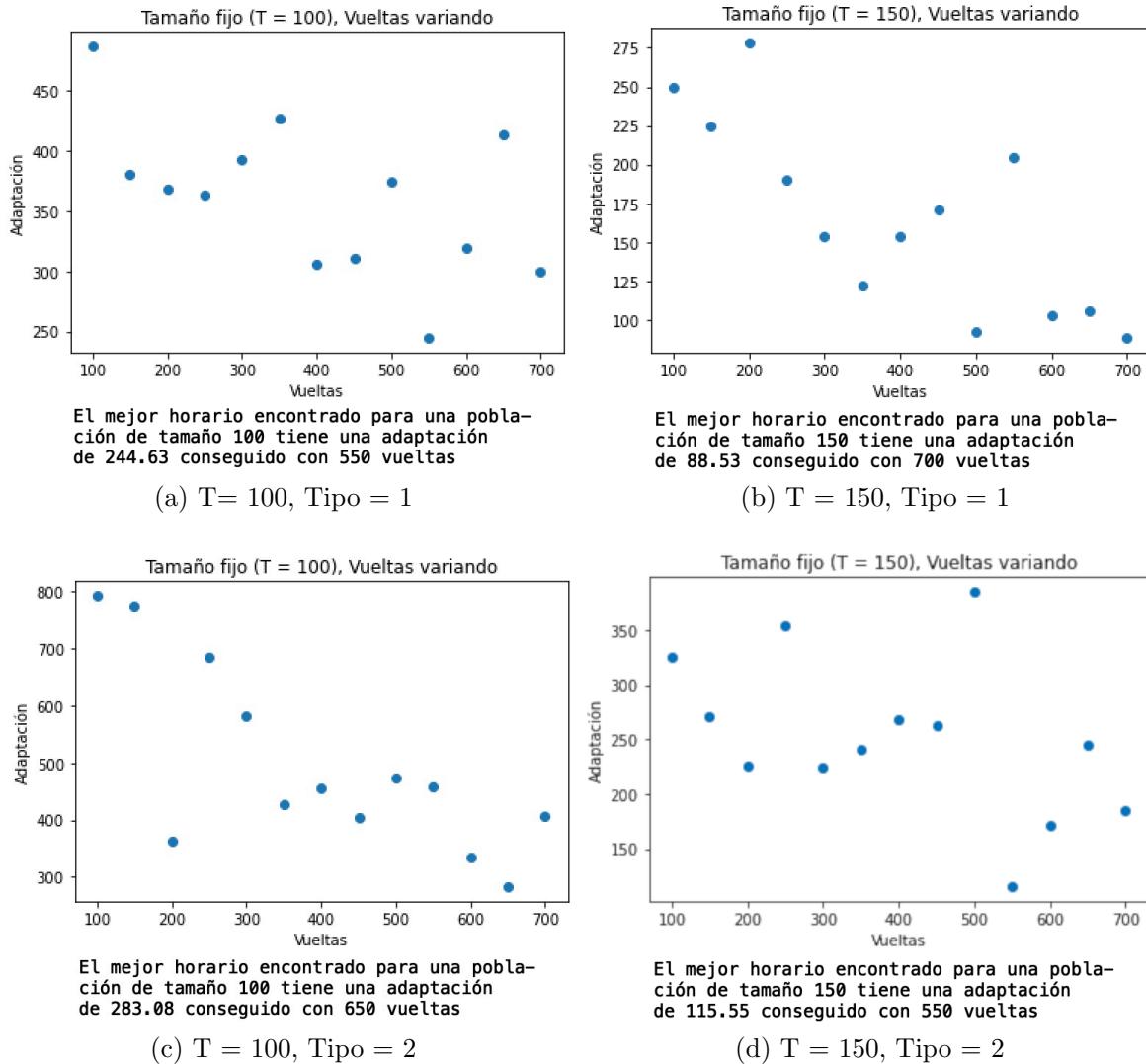


Figura 27: Estudio con tamaño de la población fijo.

Entre las cuatro pruebas realizadas conseguimos la mejor solución para una población de tamaño 150, dando 700 vueltas, con valor de la función objetivo igual a 88,53. Centrándonos en la gráfica 27b podemos observar cómo en ella se concentran los mejores resultados, pues el intervalo de su imagen es (275 - 85) aproximadamente, considerablemente mejor que (500 - 240), (800 - 280) ó (400 - 115) para las gráficas 27a, 27c, 27d respectivamente.

En comparación con la solución obtenida en [17], cuyo valor de la función objetivo es 1.968,38 nuestras soluciones encontradas son notablemente mejores. Para conseguir la mejor solución en los cuatro estudios realizados se han dado 550, 650 o 700 vueltas, cerca de las 700 vueltas propuestas como límite, véase las gráficas de la Figura 27. También, hemos de destacar como en la gráfica 27c se encuentran los peores resultados, aún así, conseguimos un horario con una aptitud de 283,08. A la vista de los gráficos, tiene más importancia el tamaño de la población que el número de vueltas, compare los resultados para los dos tamaños propuestos, observará notables diferencias en cuanto a la aptitud de los horarios obtenidos.

En este análisis, ninguna de las gráficas parecen mostrar alguna mejoría al aumentar el número de vueltas en el algoritmo, pues no se observa una clara tendencia decreciente en ninguna de ella, sin embargo se anima a utilizar un número de vueltas razonable que dependerá del conjunto de datos concreto.

- **ESTUDIO POR TIEMPO:** estamos interesados en adaptarnos a la paciencia del usuario por lo que necesitamos conocer si el tiempo de espera influye en la calidad de las soluciones, para ello nos centramos en estudiar la función `def algoritmogeneticoT(..)`.

DATOS	TAMAÑO	TIEMPO	TIPO
CAR91	150	(60 - 900, 30)	vectorS1(..)
CAR91	175	(60 - 900, 30)	vectorS1(..)

Tabla 5: Datos para el estudio por tiempo de espera.

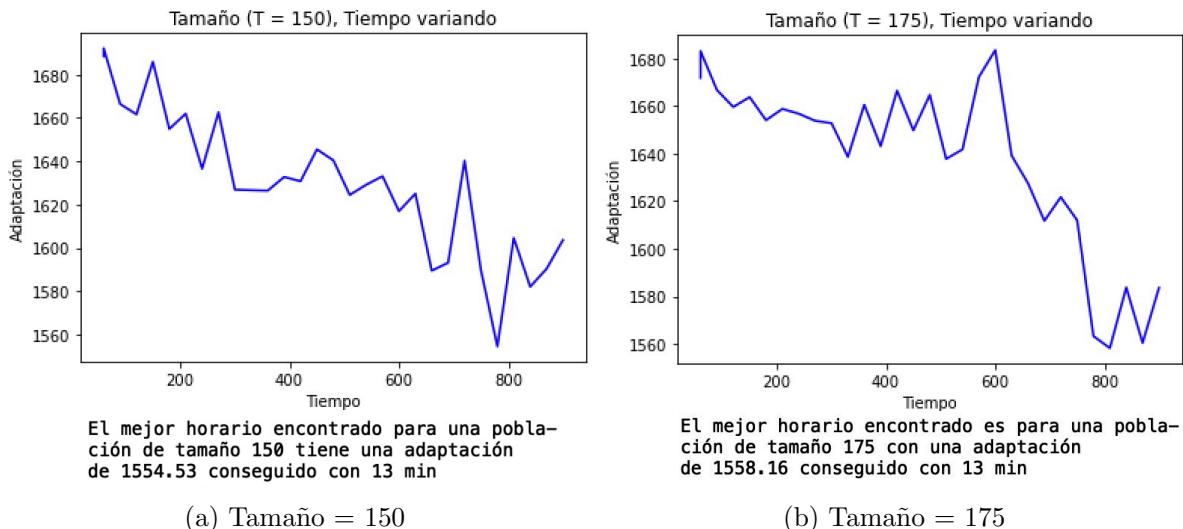


Figura 28: Estudio relacionado con el tiempo de espera.

En ambas gráficas podemos observar una tendencia decreciente de forma claramente irregular con ciertos altibajos, destacando los picos pronunciados de la Figura 28b. Encontramos los mejores resultados para ambas gráficas, 28a y 28b, en un tiempo de 13 minutos con valores de la función objetivo igual a 1.554,53 y 1.558,16 respectivamente. Estas soluciones se encuentran a la par de la solución encontrada en **Carleton91.sol** con una aptitud de 1.644,52, aunque cabe mencionar como la solución proporcionada se acerca más a las soluciones encontradas para los primeros tiempos. También sería interesante conocer lo que pasa más allá de los 15 minutos límite estipulados.

En esta prueba, el conjunto de datos elegido es el más grande en lo que se refiere a número de exámenes y el segundo en número de inscripciones (Véase Tabla 1), luego es de esperar que tanto el valor de la función objetivo como el tiempo de espera sea elevado.

Por otro lado, para decantarnos definitivamente por una opción de diseño, también resulta interesante comparar los resultados aplicando las diversas técnicas utilizadas en la construcción de los algoritmos:

- **GENERAR POBLACIÓN INICIAL:** en este apartado tratamos de examinar cómo evoluciona la mejor solución dentro del algoritmo genético atendiendo a la forma de generar la población inicial.

DATOS	A.	TAMAÑO	VUELTAS	TIEMPO	TIPO
UTE92	A.1	100	80	-	vectorS1(..)
UTE92	A.1	100	80	-	vectorS2(..)
UTE92	A.2	100	-	7 min	vectorS1(..)
UTE92	A.2	100	-	7 min	vectorS2(..)

Tabla 6: Datos para el estudio de la evolución de la población.

Primero de todo obtendremos cómo se comporta la aleatoriedad y heurística a la hora de proponer individuos como miembros de la población inicial.

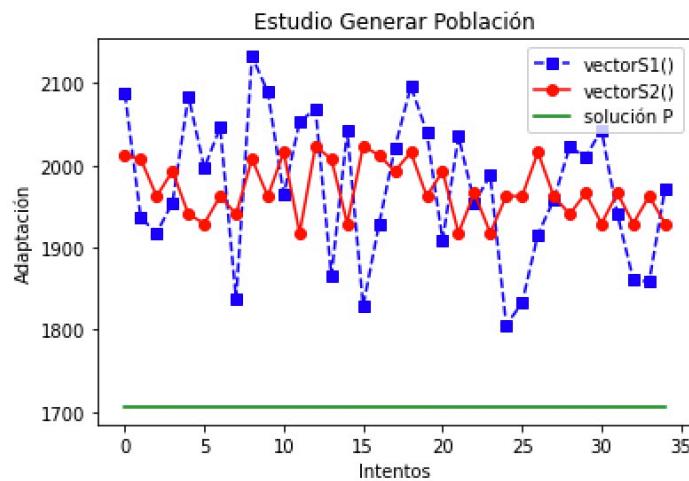


Figura 29: Estudio Aleatoriedad vs Heurística.

Contemplando la Figura 29 apreciamos cómo la función `def vectorS2(..)` ofrece resultados más regulares, a diferencia de `def vectorS1(..)` en la cual se obtienen ciertos altibajos, esto podría resultar útil en la búsqueda del horario ideal pues parece ofrecer variedad a la población inicial que se forme. Sin embargo, se consiguen mejores individuos en la primera generación si utilizamos la segunda opción de diseño, aunque, como hemos comentado en ocasiones anteriores, esto puede derivar en la aparición de superindividuos que lleven al algoritmo a converger de forma prematura hacia óptimos locales, luego es importante conocer cómo evoluciona la mejor solución en cada iteración del algoritmo tras la selección de padres, cruce, mutación y remplazamiento generacional.

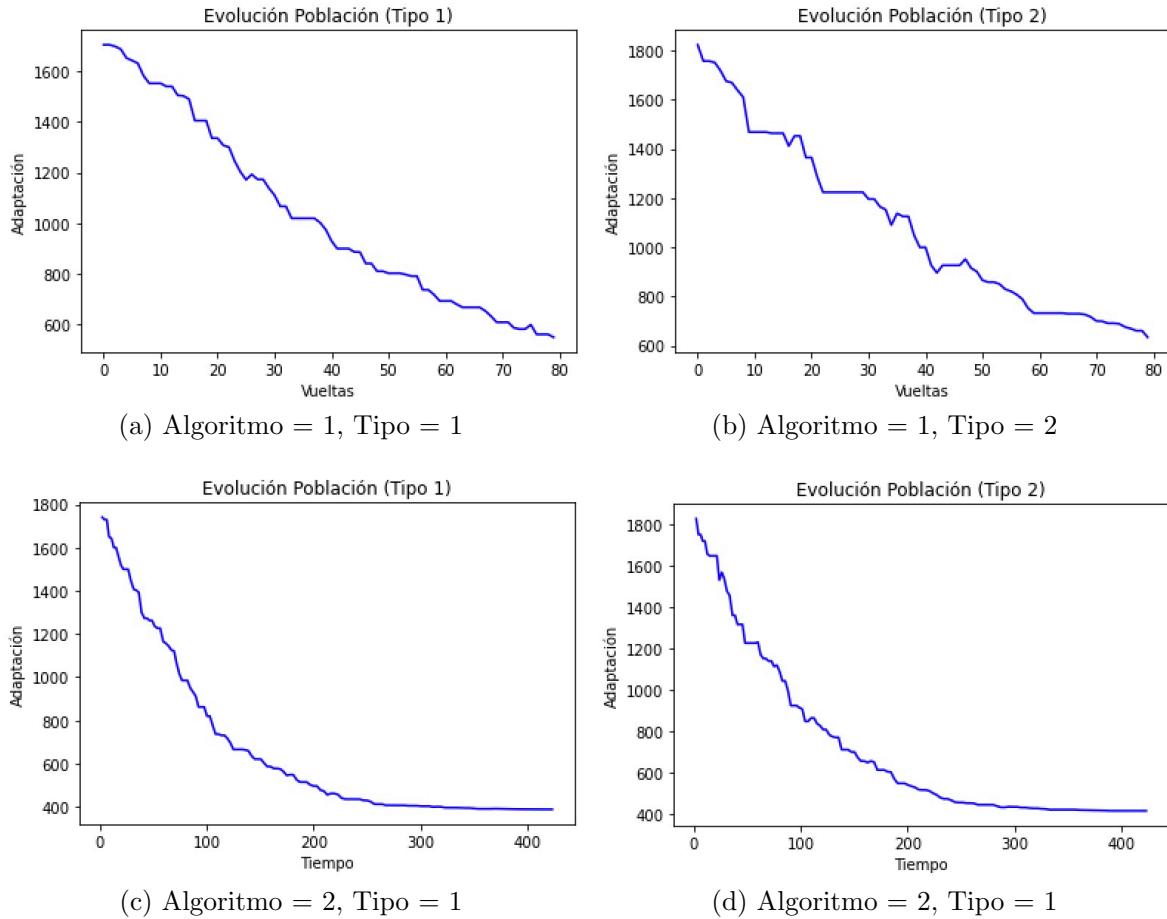


Figura 30: Estudio evolución de la población.

A la vista de las gráficas cabe subrayar la tendencia decreciente que se observa en todos los casos a medida que se avanza en la ejecución del algoritmo, tanto en vueltas como en tiempo.

Para las gráficas relacionadas con el segundo algoritmo genético, se observa que una vez llegado a los 3 minutos la mejor solución comienza a estabilizarse.

En atención a los resultados producidos no se puede garantizar conseguir mejores calendarios utilizando `def vectorS2(..)`, pues comparando las gráficas dos a dos (30a - 30b) y (30c - 30d) encontramos resultados demasiado parecidos. Si bien es cierto, hemos obtenido menor penalización en la función objetivo utilizando el algoritmo relacionado con el tiempo, donde para las últimas iteraciones rondamos la cifra 400 en adaptación.

En particular, podemos destacar como en la Figura 30a a pesar de comenzar con una población inicial donde el mejor individuo tiene una *adaptación* menor que en el resto de poblaciones (1.700 vs 1.800 aproximadamente), esto no conlleva a conseguir el mejor de todos los resultados, pues para las gráficas relacionadas con el tiempo, una vez llegan a los 2 minutos mejoran lo encontrado en 30a tras ejecutarse las 80 vueltas.

En todos los casos se han encontrado soluciones mejores a la facilitada en el archivo **TorontoE92.sol** donde el horario propuesto tiene una aptitud de 1.706,41, acercándose a las primeras iteraciones de las cuatro pruebas.

- **SELECCIÓN DE PADRES:** buscamos conocer si en nuestro diseño el tamaño del torneo tiene relevancia a la hora de encontrar una solución óptima, para ello realizamos pruebas con los dos algoritmos genéticos junto con las dos formas de construir una población inicial.

DATOS	ALGORITMO	TAMAÑO	VUELTAS	TIEMPO	TIPO
TRE92	A.1	(75 - 325, 50)	150	-	vectorS1(..)
TRE92	A.1	(75 - 325, 50)	150	-	vectorS2(..)
TRE92	A.2	90	-	(60 - 601, 60)	vectorS1(..)
TRE92	A.2	90	-	(60 - 601, 60)	vectorS2(..)

Tabla 7: Datos para el estudio del tamaño del torneo.

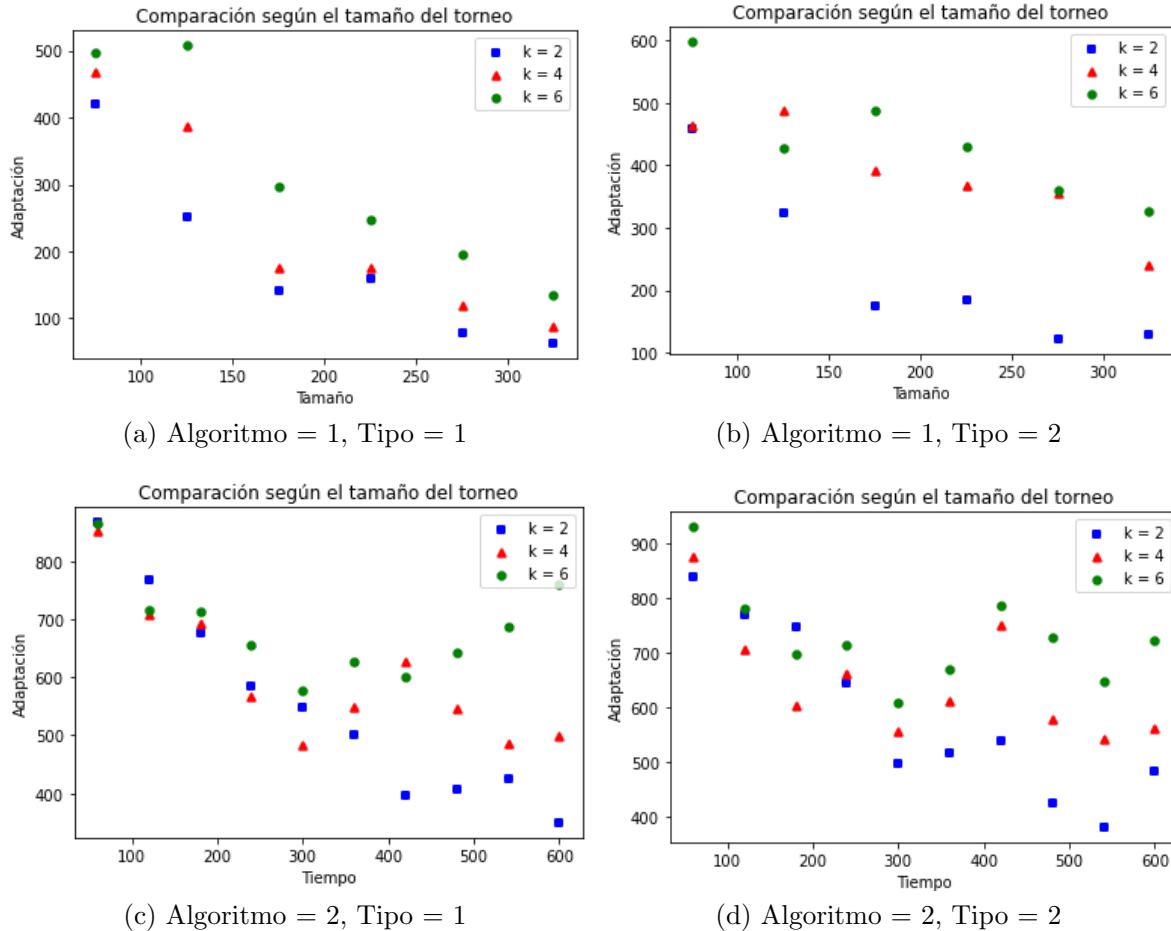


Figura 31: Estudio tamaño del torneo.

En vista a los cuatro gráficos expuestos, estos parecen indicar como tamaño ideal del torneo $K = 2$, pues en todas las gráficas los puntos azules, marcando el valor de la función objetivo, en general, ofrecen los mejores resultados, en cambio si tomamos $K = 6$ no conseguimos, en líneas generales, mejorar las soluciones. Luego, aumentar el tamaño del torneo no conduce a ninguna mejora de los algoritmos, y tan solo empeora la eficacia de estos, por lo tanto para la utilización tanto de `def algoritmoGeneticoV(..)` como de `def algoritmoGeneticoT(..)` recomendamos tomar $K = 2$.

En base a los resultados obtenidos con los diferentes análisis efectuados podemos afirmar que existe una relación directa entre el tamaño de la población y la calidad de las soluciones obtenidas, es decir, a mayor tamaño cabe esperar conseguir mejores horarios, como se demuestra en las gráficas 26a y 26b. Asimismo, en el resto de estudios donde esta variable permanece constante también podemos observar su importancia, pues hay una clara diferencia en los resultados para los diferentes tamaños que se fijan ($T = 100$, $T = 150$, $T = 175$). Sin embargo, como hemos comentado en ocasiones anteriores, una excesiva dimensión de la población está asociada a grandes tiempos computacionales como hemos podido corroborar empíricamente en los distintos experimentos.

Por otro lado, al incrementar el tiempo de ejecución probablemente conseguiremos horarios de mejor calidad, aunque podríamos caer en un estancamiento donde solo estemos alargando la espera sin obtener mejores resultados, como podemos confirmar en las gráficas de la Figura 28.

En contraste, el n^o de vueltas no parece jugar un papel demasiado importante como se comprobó en “**Estudio por tamaño fijo**”, aunque siempre se recomienda un número de iteraciones en concordancia a la proporción del problema.

En vista a las soluciones obtenidas en los diferentes estudios es significativa la diferencia, en la mayoría de los casos, entre las mejores soluciones encontradas por nuestro programa y las proporcionadas en los diferentes archivos facilitados en [17]. Para examinar más en detalle estos resultados, utilizamos 4 conjuntos de datos distintos para los cuales ejecutamos los algoritmos varias veces modificando diferentes parámetros.

DATOS	ALGORITMO	TAMAÑO	VUELTAS	TIEMPO	TIPO
EAR83	A.1	100	80	-	T.1
EAR83	A.1	100	80	-	T.2
EAR83	A.1	125	80	-	T.1
EAR83	A.1	125	80	-	T.2
EAR83	A.2	150	-	(180 - 720, 60)	T.1
KFU93	A.1	300	145	-	T.1
KFU93	A.1	300	145	-	T.2
KFU93	A.1	325	145	-	T.1
KFU93	A.1	325	145	-	T.2
KFU93	A.2	350	-	(180 - 720, 60)	T.1

DATOS	ALGORITMO	TAMAÑO	VUELTAS	TIEMPO	TIPO
HEC92	A.1	125	95	-	T.1
HEC92	A.1	125	95	-	T.2
HEC92	A.1	150	95	-	T.1
HEC92	A.1	150	95	-	T.2
HEC92	A.2	175	-	(180 - 720, 60)	T.1
LSE91	A.1	125	90	-	T.1
LSE91	A.1	125	90	-	T.2
LSE91	A.1	190	90	-	T.1
LSE91	A.1	190	90	-	T.2
LSE91	A.2	200	-	(180 - 720, 60)	T.1

Tabla 8: Datos para la comparación de soluciones con “Results”, UAX.

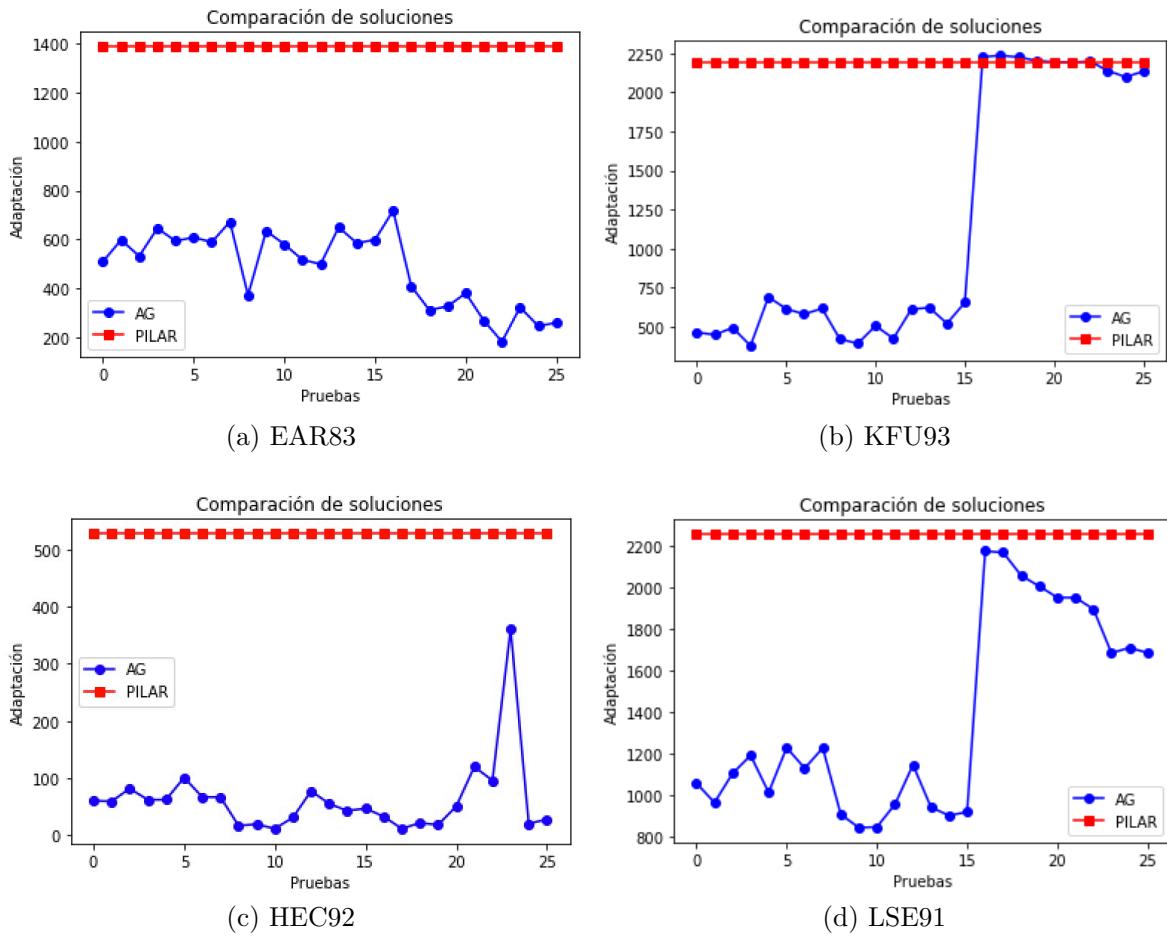


Figura 32: Estudio comparando las mejores soluciones.

Gracias a las cuatro imágenes podemos revalidar de forma sencilla cómo nuestros algoritmos parecen ofrecer mejores soluciones en todos los casos, sumándole además los estudios previos que nos arrojaban una hipótesis parecida. La clara diferencia que encontramos entre los resultados nos lleva a pensar en lo siguiente: en nuestra versión no penalizamos la función objetivo cuando un horario resulta ser infactible, en cambio cuando se llevan a cabo diversos procesos de selección de los mejores horarios (`def el_Mejor(..)`, `def recombinar_Elitismo(..)`, `def algoritmoGeneticoV(..)`, `def algoritmoGeneticoT(..)`) otorgamos menor probabilidad de ser escogidos

a aquellos calendarios que resultan no ser válidos. Como no conocemos el criterio exacto elegido en las soluciones propuestas, no podemos sacar conclusiones lo suficientemente esclarecedoras, por lo que deberíamos realizar más estudios variando estas situaciones, como por ejemplo, penalizar la infactibilidad en la función objetivo, para poder emitir conclusiones más clarificadoras.

Con el fin de afianzar y contrastar nuestros buenos resultados decidimos compararlos también con los archivos provistos en <http://www.cs.nott.ac.uk/~pszrq/data.htm>, University of Toronto Benchmark Data, “BestSolution”:

DATOS	M.S._A.1	M.S._A.2	“BestSolution”	1ªVUELTA
CAR91	733,23	1.595,3	1.684,73	1.669,03
CAR92	391,59	1.166,03	1.302,65	1.327,23
TRE92	298,45	405,91	1.006,58	991,54
UTA92	496,78	1.352,12	1.447,95	1.500,15
YOR83	400,34	294,8	1.394,18	1.358,05

M.S._A. = Mejor Solución del Algoritmo

Tabla 9: Datos para la comparación de soluciones con “BestSolution”.

En este caso nos volvemos a encontrar con resultados semejantes a los extraídos en los estudios anteriores, donde somos capaces de mejorar las soluciones ofrecidas. En particular, hemos notado como cuando se trata de conjuntos de datos voluminoso las cuatro soluciones se mueven en los mismos rangos, en cambio si se trata de conjuntos de datos pequeños como son *TRE92* y *YOR83*, las diferencias entre las mejores soluciones se hacen notar.

Las soluciones encontradas son satisfactorias en cuanto a calidad se refiere, sin embargo, esto es acosta de invertir en tiempo computacional, donde es importante recalcar como las soluciones facilitadas se acercan a los resultados lanzados por las primeras iteraciones de nuestros algoritmos. En nuestro trabajo, hemos priorizado la calidad antes que el tiempo de ejecución, para ello hemos diseñado la función `def algoritmoGeneticoV(..)` donde la búsqueda de buenas soluciones está vinculada con el tamaño de la población y el número de iteraciones. Para subsanar la impaciencia del usuario en la elaboración de un horario hemos diseñado la función `def algoritmoGeneticoT(..)` en la cual controlaremos el tiempo que se está dispuesto a esperar, sabiendo que a medida que dilatemos la duración probablemente hallaremos soluciones más óptimas.

Para el desarrollo de los estudios realizados, hemos diseñado una serie de funciones, las cuales se pueden encontrar en los archivos *AG_MejoresSoluciones.py* y *AG_Estudios.py* de https://github.com/Siria-Iniguez/HORARIOS_AG_TFG.git

3.5 Trabajos futuros

Por último, en este apartado, abriremos camino a futuras líneas de investigación, tanto que extiendan los horizontes propuestos en este proyecto como diseñen planteamientos nuevos para el problema de programación de horarios.

En primer lugar, para ampliar el problema de cronología de horarios que hemos expuesto, podemos considerar más restricciones, preferencias o prohibiciones, añadiendo así una visión más realista al problema, como pueden ser las siguientes:

- **Aulas:** capacidad limitada en las aulas, se consideran distancias entre las salas, nos interesa realizar el examen E_i en el aula S_j .
- **Profesores:** se deben asignar una serie de profesores para supervisar los diferentes exámenes, luego un profesor no puede cuidar dos exámenes a la misma hora, el profesor P_i debe cuidar el examen E_j .
- **Horaria:** preferencias a la hora de realizar algún examen, por ejemplo, si el examen E_i está en precedencia del examen E_k entonces las soluciones deben garantizar que $t_i < t_k$.

Con todo ello, estaríamos añadiendo tanto restricciones fuertes como débiles, las cuales enriquecerían nuestro problema, dotándolo a su vez de una mayor complejidad.

Por otra parte, en lo que se refiere a la implementación resultaría atractivo estudiar otras variantes: realizar un criterio de parada relacionado con la convergencia, generar un horario con alguna técnica heurística, emplear la sustitución generacional para el remplazamiento poblacional. Además podríamos ampliar las diferentes pruebas que hemos realizado en la Subsección 3.4, cambiando cualquiera de las siguientes variables: *datos, tamaño, vueltas, algoritmo, tipo*.

Además, puede ser interesante analizar nuestro planteamiento del problema con otras técnicas heurísticas como pueden ser: búsqueda tabú, redes neuronales, enjambres y comparar los resultados de los diferentes métodos con los obtenidos en el presente proyecto.

Finalmente, podemos plantearnos el problema desde una perspectiva distinta. En este trabajo, hemos expuesto el problema asignando a cada examen una hora concreta, sin embargo, también sería interesante estudiar el problema asignando a cada hora una serie de exámenes o calcular la optimalidad de un horario de formas diferentes. Estas perspectivas fueron unas de las primeras líneas de indagación por lo que contamos con un pequeño desarrollo de la implementación en Python, la cual se puede encontrar en https://github.com/Siria-Iniguez/HORARIOS_AG_TFG.git en los archivos *TFG_aplitudA.py* y *TFG_sinMatriz_C*.

4 CONCLUSIONES

Para finalizar este trabajo, nos gustaría recalcar la importancia que ha tenido la utilización de los Algoritmos Genéticos. Estos se amoldan perfectamente al problema relacionado con la programación de horarios, donde hemos sido capaces de cumplir uno de los grandes objetivos del proyecto, consiguiendo calendarios excelentes en cuanto a calidad se refiere. Sin embargo, no hemos sido capaces de satisfacer completamente la búsqueda de soluciones en tiempos razonables, pues para problemas de extensas dimensiones con un gran número de exámenes y alumnos inscritos, el tiempo computacional es elevado.

Por otro lado, encontrar una función objetivo que muestre una adecuada adaptación de los individuos, junto con la evaluación de su factibilidad, y la selección de una apropiada codificación requiere de tiempo y la realización de diversas pruebas para que puedan ser programadas fácilmente en la computadora en concordancia con los algoritmos evolutivos.

Este trabajo, nos ha permitido conocer sobre la complicada tarea de elaboración de calendarios de exámenes que favorezcan al alumnado, además nos ha servido para experimentar cómo las diferentes variables y los operadores genéticos influyen en las soluciones finales.

En conclusión, consideramos los algoritmos genéticos una técnica muy útil para la resolución de una amplia gama de problemas relacionados con la elaboración de calendarios de diversos ámbitos, donde además podemos controlar una numerosa cantidad de parámetros.

5 BIBLIOGRAFÍA

- [1] Miguel Ángel Chacón Martínez. Algoritmo genético para la generación automática de equipos de trabajo en entornos educativos. Escola Tècnica Superior d'Enginyeria Informàtica Universitat Politècnica de València. Universitat Politècnica de València (UPV). 2017 - 2018.
- [2] Víctor Yamil Neira González. El problema de timetabling para colegios chilenos. Solución mediante Algoritmos Genéticos. Magíster en Ingeniería Industrial, Dirección de Postgrado – Universidad de Concepción, Noviembre de 2014. http://repositorio.udc.cl/bitstream/11594/1743/1/Tesis_El_Problema_de_Timetabling.Image.Marked.pdf
- [3] Abdelmalik Moujahid, Inaki Inza y Pedro Larrañaga. Algoritmos Genéticos Departamento de Ciencias de la Computación e Inteligencia Artificial. Universidad del País Vasco–Euskal Herriko Unibertsitatea. <http://www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/temageneticos.pdf>
- [4] Jorge Arranz de la Peña and Antonio Parra Truyol. Algoritmos Genéticos. Universidad Carlos III. <https://www.it.uc3m.es/~jvillena/irc/practicas/06-07/05.pdf>
- [5] Soft Computing and Intelligent Information Systems Group Bioinformática. Capítulo 6. Universidad de Córdoba, 2013. <https://sci2s.ugr.es/sites/default/files/files/Teaching/GraduatesCourses/Bioinformatica/Tema%2006%20-%20AGs%20I.pdf>
- [6] Marcos Gestal Pose. Introducción a los Algoritmos Genéticos. Depto. Tecnologías de la Información y las Comunicaciones, Universidade da Coruña. <https://cursa.ihmc.us/rid=1KNKN1H44-V8CC04-1M4K/Algoritmos%20de%20Terminos.pdf>
- [7] Yadira Solano Sabatier, Miguel Calvo Martín and Leandro Trejos Picado. Implementación de un Algoritmo Genético para la Asignación de Aulas en un centro de estudios. Escuela de Ciencias de la Computación e Informática, Facultad de Ingeniería, Universidad de Costa Rica, 2008. <https://www.revistas.una.ac.cr/index.php/uniencia/article/view/3915/3759>
- [8] Eva Besada, A. Herrán, Óscar Garnica, Rafael del Vado. Optimización heurística basada en técnicas evolutivas. Material docente de la asignatura Optimización Heurística y Aplicaciones del Máster Universitario en Ingeniería de Sistemas y de Control (UCM-UNED), Campus Virtual de la UNED curso 2021-2022.
- [9] Mario Alberto Villalobos Arias. Algoritmos Généticos: Algunos resultados de convergencia. Departamento de Matemáticas. Centro de Investigación y de Estudios Avanzados de I.P.N. Diciembre, 2003. <https://semana.mat.uson.mx/Memorias/villalobos.pdf>
- [10] Hernán Sosa, Silvia Myriam Villagra, Norma Andrea Villagra. Operadores de Mutación en Algoritmos Genéticos Celulares Aplicados a Problemas Continuos Universidad Nacional de la Patagonia Austral, 2014. Dialnet.

- [11] Mirani Oktavia, Amril Aman and Toni Bakhtiar. Courses timetabling problem by minimizing the number of less preferable time slots. Department of Mathematics, Faculty of Mathematics and Natural Sciences, Bogor Agricultural University, Indonesia Sciences. IOP Conference Series, 2017. <https://iopscience.iop.org/article/10.1088/1757-899X/166/1/012025/pdf>.
- [12] Masri Ayob1, Ariff Md Ab Malik, Salwani Abdullah, Abdul Razak Hamdan, Graham Kendall, and Rong Qu. Solving a Practical Examination Timetabling Problem. Faculty of Information Science and Technology, Universiti Kebangsaan Malaysia. https://www.researchgate.net/publication/221432805_Solving_a_Practical_Examination_Timetabling_Problem_A_Case_Study.
- [13] Edmund Burke, Yuri Bykov, James Newall and Sanja Petrovic. A time-predefined local search approach to exam timetabling problem. https://www.academia.edu/6858522/A_time_predefined_local_search_approach_to_exam_timetabling_problems.
- [14] Manuel David Lozano Amézquita. Diseño de un algoritmo para realizar la programación de horarios de la carrera de ingeniería industrial de la Pontificia Universidad Javeriana. <https://repository.javeriana.edu.co/bitstream/handle/10554/20537/LozanoAmezquitaManuelDavid2016.pdf?sequence=1>.
- [15] Karel Rodríguez Varona. Aplicación de algoritmos genéticos en la generación automática de horarios docentes en la Facultad Regional de Granma. Revista Cubana de Ciencias Informáticas. <https://www.redalyc.org/pdf/3783/378343677005.pdf>
- [16] Moustapha Abdellahi 1and Hussein Eledum. The University Timetabling Problem: Modeling and SolutionUsing Binary Integer Programming with Penalty Functions. International Journal of Applied Mathematics and Statistics, Int. J. Appl. Math. Stat.; Vol. 56; Issue No. 6; Year 2017. https://www.researchgate.net/publication/319979600_The_University_Timetabling_Problem_Modeling_and_Solution_Using_Binary_Integer_Programming_with_Penalty_Functions
- [17] Rong Qu, Edmund K. Burke, Barry McCollum, L.T.G. Merlot, and S.Y. Lee A Survey of Search Methodologies and Automated System Development for Examination Timetabling. Journal of SchedulingInternational, 2009. <http://www.cs.nott.ac.uk/~pszrq/data.htm>
- [18] Miguel Vázquez Fernández de Lezeta. Algoritmos Genéticos para la selección de variables en la predicción de partidos de baseball. Universidad Autónoma de Madrid, Septiembre 2015. https://repositorio.uam.es/bitstream/handle/10486/669354/Vazquez_Fernandez_de_Lezeta_Miguel_tfm.pdf?sequence=1
- [19] Algoritmos Genéticos Inteligencia Artificial. Universidad del Magdalena Ingeniería De Sistemas. <https://slideplayer.es/amp/13626012/>