Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Máster Universitario en Inteligencia Artificial

Trabajo Fin de Máster

Ajuste del Modelo de Probabilidad y de Preferencias de un Diagrama de Influencia basado en Reglas

Autor(a): Daniel Henríquez Quesada
Tutor(a): Juan Antonio Fernández del Pozo Salamanca

Madrid, Junio/2024

Este Trabajo Fin de Máster se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Máster
Máster Universitario en Inteligencia Artificial

Título: Ajuste del Modelo de Probabilidad y de Preferencias de un Diagrama de Influencia basado en Reglas

Junio/2024

Autor(a): Daniel Henríquez Quesada
Tutor(a): Juan Antonio Fernández del Pozo Salamanca
Departamento de Inteligencia Artificial
ETSI Informáticos
Universidad Politécnica de Madrid

Resumen

Los diagramas de influencia se destacan como una herramienta gráfica de gran utilidad para representar de manera concisa y visual problemas de ayuda a la toma de decisiones. Su propósito fundamental radica en asistir a los tomadores de decisiones en el análisis de datos y modelos para abordar problemas complejos y no estructurados.

El proceso de creación y ponderación de estos diagramas demanda un esfuerzo colaborativo entre expertos en el campo específico tratado. En este sentido, el presente Trabajo de Fin de Máster se enfoca en la utilización de metaheurísticas como los algoritmos genéticos para la asignación óptima de parámetros en el diagrama, tanto en el modelo de probabilidad condicionada como en el modelo de preferencias. Esta aproximación permite reducir significativamente el tiempo necesario que los expertos dedican a estimaciones numéricas para la educción de los modelos probabilísticos y de preferencias que representan el problema, lo cual facilita la implementación del método en diversos contextos. Cabe destacar que esta propuesta es para modelos de variables discretas, dejando como línea futura el extenderlo para permitir variables continuas.

La evaluación de la calidad de los resultados se llevará a cabo mediante la aplicación de estas técnicas en un diagrama de influencia preexistente. Para ello, se generará una población de posibles configuraciones de parámetros para los modelos mencionados, las cuales serán comparadas con las decisiones previamente tomadas por expertos. Este proceso iterativo permitirá mejorar la estimación a lo largo de un número determinado de generaciones, disminuyendo así la necesidad de un conocimiento experto extenso. En lugar de requerir modelos completos de probabilidad y preferencia, bastará con un conjunto reducido de decisiones relevantes y la estructura de dependencias o influencias del modelo para obtener resultados satisfactorios.

Al final, se realizará una comparación exhaustiva de los resultados obtenidos utilizando un algoritmo genético, mediante el ajuste de sus hiperparámetros para alcanzar el modelo óptimo. Este análisis detallado permitirá evaluar la eficacia relativa de la metaheurística seleccionada y determinar la estrategia más efectiva para abordar el problema en cuestión.

Abstract

Influence diagrams stand out as a useful graphical tool for concisely and visually representing decision support problems. Their fundamental purpose is to assist decision makers in analyzing data and models to address complex and unstructured problems.

The process of creating and weighting these diagrams demands a collaborative effort among experts in the specific field being addressed. In this sense, this Master's Thesis focuses on the use of metaheuristics such as genetic algorithms for the optimal assignment of parameters in the diagram, both in the conditional probability model and in the preference model. This approach significantly reduces the time required by the experts to spend on numerical estimations for the education of the probabilistic and preference models that represent the problem, which facilitates the implementation of the method in various contexts. It should be noted that this proposal is for discrete variable models, leaving as a future line to extend it to allow for continuous variables.

The evaluation of the quality of the results will be carried out by applying these techniques on a pre-existing influence diagram. To this end, a population of possible parameter configurations for the aforementioned models will be generated, which will be compared with the decisions previously made by experts. This iterative process will allow improving the estimation over a given number of generations, thus decreasing the need for extensive expert knowledge. Instead of requiring complete probability and preference models, a reduced set of relevant decisions and the structure of dependencies or influences of the model will suffice to obtain satisfactory results.

At the end, a comprehensive comparison of the results obtained using a genetic algorithm will be performed by adjusting its hyperparameters to reach the optimal model. This detailed analysis will make it possible to evaluate the relative efficiency of the selected metaheuristic and to determine the most effective strategy to address the problem in question.

Tabla de contenidos

Capítulo 1

Introducción

1.1. Definición del problema

Los sistemas de ayuda a la decisión son sistemas informáticos interactivos que ayudan a los decisores en la utilización de datos y modelos que resuelvan problemas no estructurados [4, 5, 6]. Los diagramas de influencia es un método de representación más compacta que otros procedimientos, y además permiten evaluarlo directamente y obtener los resultados directamente [1, 2]. En los diagramas de influencia la utilidad de un conjunto de decisiones viene dado por un modelo de probabilidad condicionada y un modelo de preferencias.

1.2. Alcance

El campo de los sistemas de ayuda a la decisión es bastante amplio y con multitud de representaciones diferentes. Por tanto, este trabajo se centrará únicamente en diagramas de influencia con variables discretas, un único nodo de valor y con el grafo de dependencias ya definido.

1.3. Motivación

Con el propósito de facilitar la participación del experto, en este proyecto se estimarán las probabilidades condicionadas y la utilidad de las preferencias usando metaheurísticas. Al depender en menor medida del experto, se conseguirá acelerar el proceso de los proyectos que emplean esta herramienta, donde la figura del experto siga siendo necesaria en el proyecto. La participación del experto debe ser más eficiente, y tratamos de evitar que el experto dedique mucho tiempo, con los analistas, a la estimación de todos los parámetros que subyacen a la estructura del modelo de dependencias. Además, los métodos empleados actualmente (OR-noisy gate [14],...) requieren condiciones e hipótesis que no son ciertas, o métodos combinatorios que no escalan (test de independencia en utilidad,...)

1.4. Objetivos

Por tanto, el objetivo de este proyecto es el de realizar una prueba de concepto con un problema real conocido, y simular la interacción con un experto para estimar los modelos. Además, se probará una metaheurística capaz de encontrar los parámetros y analizar el rendimiento y calidad de las soluciones.

1.5. Descripción de la memoria

Para poder alcanzar esta meta se dividirá el proyecto en cinco capítulos, siendo esta introducción el primero de ellos, que asegurará que el proyecto siga un desarrollo adecuado.

En el segundo capítulo se hará una explicación teórica tanto de los diagramas de influencias como de los algoritmos genéticos. Esto permitirá al lector conocer los elementos principales de este trabajo, facilitando la comprensión del proyecto.

En el siguiente capítulo se comentará la implementación realizada de la representación [1, 2] de un diagrama de influencia en el lenguaje de programación Python, reflejando correctamente tanto las probabilidades condicionadas de cada factor del diagrama como la utilidad del conjunto de condicionantes.

Tras esto, se hablará sobre la programación del algoritmo de evaluación de diagrama de influencia [3], que muestra unas tablas con las decisiones óptimas que varía según los valores de la probabilidad condicionada y de utilidad. Con esto se podrá comparar con las reglas proporcionadas por el experto. Dichas reglas servirán como medida y ejemplo para los individuos del algoritmo genético, que irán evolucionando los valores que se escojan como base hasta alcanzar los resultados que proporcionaría un experto.

Para asegurar la mejoría del algoritmo, en el siguiente capítulo se ajustarán una serie de hiperparámetros a través de la realización de diversas pruebas, con las que se medirá la cercanía a las reglas del experto.

Gracias a esto, se podrá conseguir un diagrama de influencia que de reglas similares a un experto reduciendo la participación de este. Además, permite que las interacciones con el experto se hagan a más alto nivel, en lugar de estar centradas en los parámetros de las distribuciones y la función de utilidad. Esto es debido a que para el entrenamiento del modelo no se le pedirá valores exactos y numéricos, sino que se le requerirá que decida cuales son las mejores decisiones en conjunto de combinaciones posibles de factores.

Por último, hablaré sobre mis conclusiones de este proyecto, dando mi opinión sobre el trabajo realizado y como he podido ahondar en un tema tan interesante como son los sistemas de ayuda a la decisión. Además, se comentarán las posibles mejoras y líneas futuras que se podrían implementar. El campo de los sistemas de ayuda a la decisión es bastante amplio, y además existen una gran variedad de metaheurísticas diferentes, por lo que hay muchas vías de investigación posibles.

Capítulo 2

Marco teórico

En este capítulo se otorgará una base teórica de vital importancia para comprender la naturaleza de este proyecto. Para ello, primero se explicará con mayor detalle que son los sistemas de ayuda a la decisión (SAD) [4, 5, 6] y como se pueden mostrar de manera gráfica empleando un diagrama de influencia. Tras esto, se detallará el funcionamiento del algoritmo genético (AG) [7, 8], que será el encargado de obtener unos parámetros funcionales en el diagrama de influencia.

2.1. Sistema de ayuda a la decisión

El propósito fundamental de este estudio es explorar hasta qué punto es factible mejorar y racionalizar la participación directa de un experto en la configuración y diseño de un sistema de ayuda a la decisión. Estos sistemas, conocidos también como SAD [4, 5, 6], son piezas de software diseñadas para proporcionar asistencia a los tomadores de decisiones al facilitar el acceso y la interpretación de datos relevantes, así como la aplicación de modelos y técnicas analíticas para abordar problemas complejos.

El concepto de SAD es sumamente amplio y versátil, ya que puede ser aplicado en una variedad de contextos y situaciones, desde la gestión empresarial hasta la planificación urbana y la medicina. Cada uno de estos ámbitos presenta desafíos y características particulares que requieren enfoques específicos en el diseño y la implementación de estos sistemas.

Es importante destacar que los SAD no tienen la intención de reemplazar por completo la toma de decisiones humana, sino más bien de complementarla y mejorarla. Actúan como herramientas de apoyo que permiten a los tomadores de decisiones acceder a información relevante, analizar datos complejos y evaluar diferentes escenarios antes de tomar una decisión final. En este sentido, los SAD se convierten en aliados estratégicos para los expertos, ayudándoles a tomar decisiones más informadas, eficientes y efectivas en sus respectivos campos de acción.

2.1.1. Diagrama de influencia

Los Diagramas de Influencia (DI) [1, 2] han emergido como una herramienta indispensable en el ámbito de los SAD, proporcionando una representación visual y compacta

de la complejidad inherente a los procesos de toma de decisiones en contextos caracterizados por la incertidumbre. Su utilidad radica en su capacidad para modelar y visualizar las relaciones entre variables, eventos y decisores, lo que resulta esencial para comprender las implicaciones de diversas decisiones y eventos en un sistema dado.

Los DI pueden conceptualizarse como grafos dirigidos, donde los nodos representan diferentes elementos del sistema de decisión. Estos elementos se dividen en tres categorías principales: nodos de decisión, nodos de azar y nodos de valor. Los nodos de decisión representan las elecciones que deben realizarse en el proceso de toma de decisiones, mientras que los nodos de azar reflejan eventos o condiciones cuyos resultados son inciertos. Por otro lado, los nodos de valor representan las consecuencias o resultados asociados con las diferentes elecciones y eventos.

La estructura de un DI se define mediante las aristas que conectan los nodos, representando las relaciones condicionales e informativas entre ellos. Estas aristas capturan cómo la ocurrencia de un evento o la elección de una decisión afecta a otros elementos del sistema, permitiendo una representación visual clara de la interdependencia de las diferentes variables y eventos. Hay tres tipos de arcos dirigidos, los que van hacia nodos de azar son llamados probabilísticos, hacia nodos de decisión son informativos, mientras que los que al nodo de utilidad son funcionales. Además, cabe destacar que los arcos entre variables de azar se pueden invertir usando el teorema de Bayes, siempre y cuando no se introduzcan ciclos provocados por la inversión.

La figura 2.1 muestra un ejemplo de diagrama de influencia, donde cada tipo de variables se representan con un nodo con una forma propia. Los nodos de azar están representados con un círculo, mientras que los de decisión se representan con un cuadrado y los de utilidad tienen forma de rombo. Además, las aristas apuntan a los nodos de los que son predecesores, de tal forma que se puede ver gráficamente las relaciones entre variables.

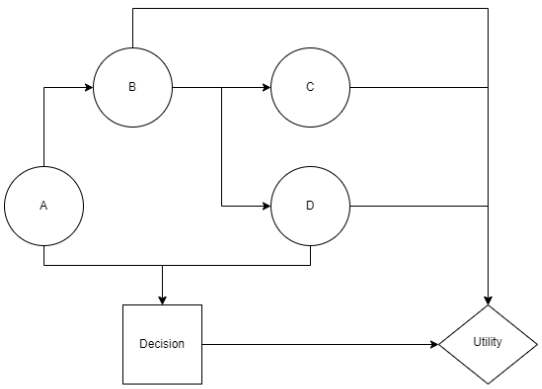

Figura 2.1: DI de ejemplo

2.1.1.1. Estructura del problema

Para poder evaluar un DI, es necesario que este sea un DI regular. Para que un diagrama de influencia sea considerado regular, debe cumplir con tres condiciones. Primero, el gráfico dirigido no debe contener ciclos, asegurando así que es un gráfico

dirigido acíclico. Segundo, el nodo de valor, si está presente, no debe tener sucesores, lo que garantiza que es un nodo terminal en el diagrama. Tercero, debe existir un camino dirigido que incluya todos los nodos de decisión, estableciendo una secuencia lógica y coherente para las decisiones a tomar.

Estas condiciones garantizan que el diagrama de influencia sea una herramienta eficaz y coherente para modelar y analizar problemas de toma de decisiones, permitiendo una representación clara y ordenada de las dependencias entre decisiones, incertidumbres y valores.

2.1.1.2. Nodos de azar

Los nodos de azar, una categoría en los Diagramas de Influencia, desempeñan un papel importante al representar las variables aleatorias relevantes para el problema en consideración. Cada nodo de azar presenta una tabla de probabilidad condicionada, una estructura de datos discreta que permite especificar cómo varían las probabilidades en función de los valores de sus nodos predecesores en el grafo dirigido. Además, la complejidad de la tabla crece exponencialmente con el número de predecesores condicionales.

Esta tabla de probabilidad condicionada encapsula la relación entre el nodo de azar y sus predecesores, ofreciendo una representación probabilística de las interdependencias dentro del sistema. Al ajustarla función de probabilidad condicionada en función de las distintas combinaciones de valores de los nodos precedentes, se logra una descripción de cómo las diferentes variables influyen en el resultado del nodo de azar en cuestión.

2.1.1.3. Nodos de decisión

Los nodos de decisión constituyen el segundo tipo de nodo disponible para representar los diagramas de influencia. Mientras que los nodos de azar representan las variables aleatorias que influyen en las decisiones, los nodos de decisión, como su nombre indica, son los encargados de representar las diferentes opciones disponibles al tomar una decisión. Este tipo de nodo no posee una tabla propia, ya que esta se generará una vez que el nodo de decisión sea eliminado, fijando la opción mejor valorada según el valor que adopten las variables precedentes que sean atributos directos del nodo de valor en la fase del problema asociada al nodo de decisión

2.1.1.4. Nodos de utilidad

El tercer y último tipo de nodo es el nodo de utilidad. En todo diagrama de influencia debe haber uno y solo un nodo de utilidad, ya que a través de este nodo se determina cuál sería la mejor decisión posible según la información proporcionada por los expertos. Esto viene directamente de la teoría de la utilidad [5], donde se otorga una función de utilidad en la toma de decisiones con incertidumbre.

Este nodo, al igual que los nodos de azar, presenta una matriz de información. Sin embargo, en lugar de medir la probabilidad condicionada, en este caso se mide la utilidad que proporciona la combinación de los valores de sus nodos predecesores. De esta forma, se puede comparar qué conjunto de decisiones y valores de las variables de azar beneficiaría más al usuario.

Como se mencionó en el apartado anterior, en este nodo se generarán tablas auxiliares una vez que se elimine un nodo de decisión que sea predecesor del nodo de utilidad. En estas tablas, se elegirá la opción que tenga una mayor utilidad para formar el conjunto de reglas.

2.1.2. Modelo del problema de decisión

Para poder medir la utilidad de usar algoritmos genéticos para ponderar los valores de utilidad y probabilidad de un diagrama de influencia, es necesario el disponer de un ejemplo real y donde ya se haya hecho un estudio de la forma tradicional. En este proyecto se empleará un DI para el diagnóstico y tratamiento del linfoma gástrico NHL [10, 11].

El linfoma gástrico primario no Hodgkin (conocido como linfoma gástrico NHL) es un trastorno relativamente raro, representando aproximadamente el 5% de los tumores gástricos. Para los pacientes con un estado general de salud deficiente y con la enfermedad en estado avanzado, es difícil seleccionar un tratamiento adecuado. Por lo tanto, se construyó un modelo que sea un apoyo en la toma de decisiones para ayudar en el tratamiento de estos pacientes.

Este DI tiene tres nodos de decisión y diecisiete variables aleatorias con una cantidad variable de valores posibles, como se puede ver en 2.2. En 2.3 se pueden ver los posibles valores que pueden tomar cada nodo.

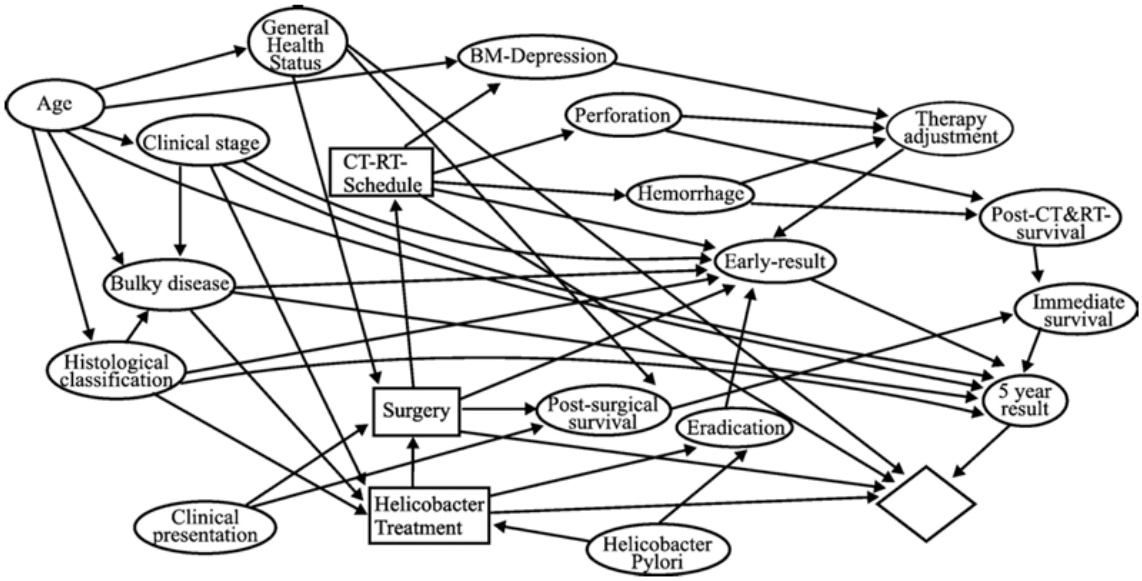

Figura 2.2: DI para el tratamiento del linfoma gástrico NHL

Variable	Possible values	Variable	Possible values
helicobacter-treatment	No, Yes	age	v10.19, v20.29, ...
surgery	None, Curative,		..., v80.89, GE90
	Palliative	eradication	No, Yes
ct-rt-schedule	None, Radio,	bone.marrow.depression	No, Yes
	Chemo, Ch.Next.Rad	perforation	No, Yes
		hemorrhage	No, Yes
general-health-status	Poor, Average, Good	therapy.adjustment	No, Yes
clinical-stage	I, II1, II2, III, IV	post.ct-rt.survival	No, Yes
bulky-disease	Yes, No	post.surgical.survival	No, Yes
histological-classification	Low.Grade,	immediate.survival	No, Yes
	High.Grade	EARLY.RESULT	CR (complete remission),
helicobacter-pylori	Absent, Present		PR (partial remission),
clinical-presentation	None, Hemorrhage,		NC (no change),
	Perforation,		PD (progressive disease)
	Obstruction	FIVE.YEAR.RESULT	alive, dead

Figura 2.3: Variables del linfoma gástrico NHL y sus valores posibles [11]

2.1.3. Proceso de adquisición de conocimiento en SAD

El proceso de adquisición de conocimiento en SAD dentro del ciclo de análisis de decisiones se divide en los siguientes apartados [5]. Para empezar, se invierte tiempo en identificar el problema y la formación del objetivo a conseguir. Tras esto, se hace una generalización de alternativas y se realiza una modelización de la estructura que represente las relaciones probabilísticas y preferencias. Por último, se evalúa para inferir la alternativa óptima y se lleva a cabo un proceso de análisis de sensibilidad y validación, derivando finalmente en la implementación.

Cabe destacar que el enfoque de este trabajo es aprender la parte cuantitativa del DI, pero a diferencia de otros modelos como las redes Bayesianas, no será con una base de datos, ya que en general o no hay datos o son pocos, sino que será con información y conocimiento experto.

2.2. Algoritmo genético

Para poder obtener un resultado deseable, se usará un algoritmo genético [7, 8]. Un algoritmo genético es una metaheurística de optimización inspirada en la evolución natural, e implementando conceptos como las recombinaciones genéticas, la mutación o la selección natural. Una metaheurística es un algoritmo de propósito general, ya que no depende del problema para producir buenos resultados, por lo que se suelen usar en problemas que no tienen un algoritmo propio que los resuelva de forma óptima.

John Holland en los años setenta fue el que diseñó lo que se conoce por algoritmos genéticos. Estos generan una población inicial de posibles soluciones al problema que se quiere resolver, a la que se le aplican diferentes técnicas con el objetivo de generar una nueva población más preparada. Este proceso se repite sucesivamente hasta que se considere oportuno. Se suele dividir este algoritmo en cuatro etapas principales: evaluación, selección, cruce y mutación.

Además, cabe destacar que para que un algoritmo genético sea eficaz se necesita ajustar sus parámetros y operadores, eligiendo estos valores según el problema que se quiera resolver. Por lo que se abordara el problema de encontrar parámetros como un problema de optimización combinatoria [15].

En la figura 2.4 se puede ver un diagrama de flujo de como funciona el algoritmo genético. Para empezar, se establece una población inicial que servirá como base del algoritmo genético. Estos individuos será evaluados en el módulo de evaluación para poder seleccionar a los mejores de ellos en el siguiente bloque. Los individuos seleccionados pasarán por un proceso de cruce y mutación, lo que permite obtener mejores individuos. Este proceso se repite hasta que se alcance el número máximo de generaciones, aunque se pueden aplicar muchas condiciones de paradas diferentes.

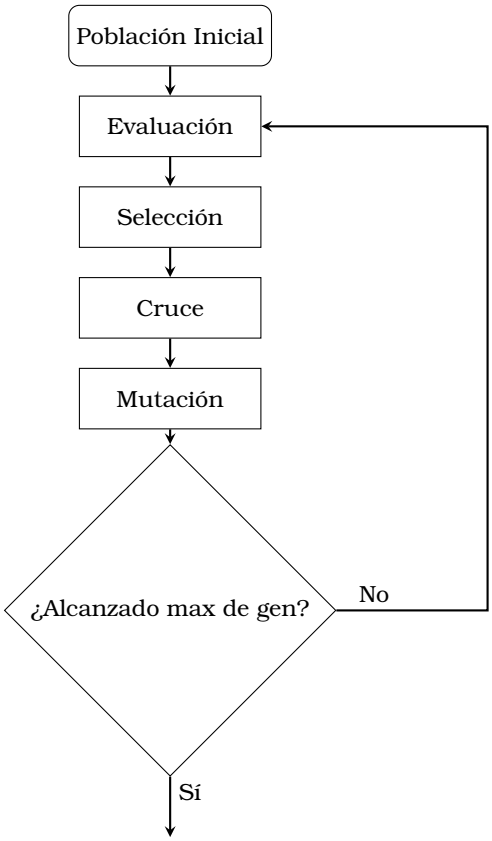

Figura 2.4: Diagrama de flujo del algoritmo genético

2.2.1. Módulo de Evaluación

El primer módulo a estudiar de este algoritmo es el módulo de evaluación. Este proceso sirve para asignar a cada individuo una puntuación, que se usará en la siguiente generación en el módulo de selección. Esta puntuación es determinada por el problema a resolver, siendo esta parte muy importante, ya que es la encargada de permitir usar el algoritmo genético en el problema que se está resolviendo.

Usualmente esta puntuación puede ser o la similitud con la solución que se quiere alcanzar, o el grado de error que tiene esta, o incluso una combinación de las dos. La forma de implementarlo depende del problema, si se quiere maximizar una medida probablemente se empleará el resultado de dicha medida, mientras que si se quiere disminuir el error como se haría en una red neuronal, se emplearán técnicas de inteligencia artificial para encontrar una manera de calcular la tasa de error que sea justa.

2.2.2. Módulo de Selección

Tras esto, será el turno del módulo de selección. Esta parte se encarga de seleccionar los individuos que actuarán como padres de la siguiente generación. Lo importante en este módulo es no solo seleccionar a los mejores individuos, sino dar opciones a otros individuos peores pero que pueden abrir caminos a soluciones mejores. En todo el conjunto del algoritmo lo que se busca es el equilibrio entre profundizar y diversificar, cambiando los valores de los hiperparámetros para encontrar dicho equilibrio.

Este módulo es el que presenta más opciones diferentes de implementación, permitiendo escoger la que mejor se ajuste al problema a resolver. En este trabajo se ha preferido implementarlo con un torneo. Se define como torneo el escoger entre la población un grupo al azar de un tamaño específico, y emparejarlos para que formen las llaves de un torneo. Tras esto, se irá eligiendo como ganador al individuo con mayor puntuación hasta que se obtenga un ganador global del torneo, y donde dicho individuo será seleccionado como padre de la siguiente generación. Con esto, se irán realizando torneos hasta seleccionar el número de individuos que se requiera.

Otra de las formas más comunes de implementar este módulo es usando una ruleta. En este método se le asigna a cada individuo un porcentaje de ser escogido en función de su puntuación. Una vez asignadas las probabilidades, se tirará de la ruleta el número de individuos a seleccionar, eligiendo en cada ronda al individuo asignado al porcentaje sacado.

2.2.3. Módulo de Cruce

Una vez seleccionado los padres en el anterior módulo, se procederá a mezclarlos por parejas para obtener a los individuos de la siguiente generación. Este módulo intenta imitar a la evolución de las especies permitiendo a los individuos con mejores genes combinarse para obtener una solución que tiene posibilidades de obtener los buenos genes de cada uno de sus padres.

Existen diferentes formar de elegir que gen de cada padre obtiene el individuo, aunque depende de la codificación de la información, ya que hay que modificar los métodos si está codificado en binario o si es una ordenación.

En cuanto a los métodos específicos, por lo general lo que varía es el número de puntos de cortes. Los puntos de corte es la posición del gen donde el individuo hijo cambia de heredar los genes de un ascendiente al otro. El cruce más sencillo es poner el punto de cruce en el 50 % de los genes, pero también es usual usar dos o tres cortes en diferentes posiciones. Además, es posible generar dos hijos del mismo par de padres a la vez, obteniendo el segundo las partes de los genes no empleadas en el primer individuo generado. Sin embargo, existe una última opción diferente, y es la de elegir al azar en cada gen de que ascendiente se obtendrá dicho gen. Esto le añade un componente de aleatoriedad que puede ser positivo para aumentar la diversidad del algoritmo, ya que aumenta el número de posibles individuos que se pueden generar de cada par.

2.2.4. Módulo de Mutación

El módulo principal para añadir la diversidad es el de la mutación. Este se encarga de aumentar las diferencias de un individuo con sus ascendientes para evitar tener alguna generación posterior con individuos muy similares. La forma usual de implementar este módulo es el de dar a cada gen una probabilidad de mutar una cantidad específica. Tanto la probabilidad de mutación como la cantidad son dos hiperparámetros muy importantes para el algoritmo genético. Además, se puede emplear un proceso de mutación diferente que se adapte a cada proyecto, ya que se pueden usar funciones de distribución como la normal o la unitaria, que variarán el grado de diversidad que provoque este módulo.

2.2.5. Otras metaheurísticas

Aunque se vaya a implementar un algoritmo genético en este proyecto, existen muchas más metaheurísticas que se podrían aplicar y que podrían mejorar los resultados. Por ejemplo los Algoritmos de Estimación de Distribución (llamado EDA por sus siglas en inglés) [16] constituyen una familia de metaheurísticas derivadas de los algoritmos evolutivos. Estos se basan en que en lugar de manipular directamente las soluciones individuales, las EDAs se centran en modelar y actualizar una distribución de probabilidad que capture las características prometedoras de las soluciones encontradas hasta el momento.

Otras metaheurísticas que se podrían aplicar sería la búsqueda tabu o el recocido simulado. La búsqueda tabú introduce un mecanismo de memoria para evitar volver a estados previamente visitados, lo que ayuda a evitar ciclos y a explorar nuevas áreas del espacio de soluciones. Por otro lado, el recocido simulado es una técnica inspirada en el proceso físico de recocido de metales. Comienza con una solución inicial y, a través de iteraciones, modifica la solución explorando vecindarios según una temperatura que disminuye gradualmente.

Capítulo 3

Implementación del AG para la estimación del DI

Una vez se ha explicado la teoría sobre los DI y los algoritmos genéticos, a continuación, se describirá en detalle el progreso que se ha seguido en este proyecto.

3.1. Implementación de diagrama de influencia

En la implementación realizada en Python, el DI ha sido representado como un conjunto de nodos, donde los nodos pueden ser un nodo de azar, de decisión o de utilidad. En todos los nodos se guarda los predecesores de dicho nodo, ya que guardar esta información es vital para poder evaluar el diagrama completo, además de que es la forma estándar para describir el algoritmo.

Aunque tienen elementos en común, cada tipo de nodo tiene características específicas. En los nodos de decisión se almacenan los posibles valores a tomar en la decisión que represente. Estos valores serán los que se usen en el algoritmo de evaluación para representar la salida del algoritmo, ya que esta serán las decisiones que tomar dependiendo de los valores de los resultados anteriores.

La mayoría de los nodos en un DI suelen ser los nodos de azar, ya que estos son los que añaden los condicionantes en la toma de decisiones. Para representar estos nodos, se emplea una matriz de probabilidad de condicionada, donde el número de dimensiones es el número de predecesores más uno, ya que será en esta última dimensión en donde se almacenará la probabilidad de los valores, siendo dependientes del valor de sus nodos predecesores. Además, al igual que los nodos de decisión, también se creará un vector de valores donde se guarde los posibles valores que puede tener la variable aleatoria. Por otro lado, para saber en todo momento que dimensión en la matriz corresponde a cada variable, se ha creado un puntero con el nombre de los predecesores y el del propio nodo, donde su propio nombre refleja las probabilidades de sus valores.

Por último, está el nodo de utilidad, este es el único tipo donde solo puede haber un nodo de esa clase, además de que este no podrá tener sucesores. Este tipo de nodo tiene una estructura similar al de azar, ya que también tiene una matriz, pero en vez de almacenar probabilidades se almacena la utilidad de cada conjunto de valores de

sus predecesores. Sin embargo, este nodo no tiene vector de valores, ya que al ser el nodo final ni se toma una decisión ni se obtiene un valor con una probabilidad. Esto hace que el vector puntero sea igual al vector de predecesores, pero se mantendrán los dos para tener una mayor claridad de código.

3.2. Algoritmo de evaluación

Con esta implementación completa del DI el siguiente paso es la construcción de un algoritmo de evaluación [3], que permita obtener una serie de reglas usando los datos de la utilidad y la probabilidad condicionada del diagrama. Se ha subido el código en el anexo .2

Algorithm 1 Algoritmo de evaluación [3]

Inicio
Verificar para diagrama de influencia orientado y regular
añadir arcos de memoria
eliminar todos los nodos sumideros
while $C(v) \neq \emptyset$ **do**
    Inicio
    if existe $i \in C \cap C(v)$ tal que $S(i) = \{v\}$ **then**
        Entonces eliminar nodo de azar i
    else if existe $i \in D \cap C(v)$ tal que $C(v) \subseteq I(i) \cup \{i\}$ **then**
        Inicio
        eliminar nodo de decisión i
        eliminar nodos sumidero
        Fin
    else
        Inicio
        encontrar $i \in C \cap C(v)$ tal que $D \cap S(i) = \emptyset$
        while $C \cap S(i) \neq \emptyset$ **do**
            Inicio
            encontrar $j \in C \cap S(i)$ tal que no hay otro camino dirigido (i, j)
            invertir arco (i, j)
            Fin
        end while
        eliminar nodo de azar i
        Fin
    end if
    Fin
end while
Fin

3.2.1. Cumplimiento de las precondiciones

Lo primero que se realiza es el añadir arcos de memoria, que consiste en añadir tanto los predecesores de un nodo de decisión como el propio nodo, al conjunto de predecesores de un nodo de decisión posterior al anterior. El añadir estos arcos ayuda a crear un orden total en todos los nodos de decisión, lo que es necesario

para el cumplimiento de la tercera condición que tiene que cumplir un diagrama de influencia para que sea regular.

Tras esto, se comprobará que el nodo de utilidad no tenga sucesores y que se pueda alcanzar todos los nodos de decisión retrocediendo desde el propio nodo de utilidad

3.2.2. Eliminación de nodos de sumidero

Aunque se cumplan todas las condiciones, un diagrama de influencia puede tener nodos que no sean el de utilidad y que no tengan ningún sucesor. Estos nodos, al no influir en algún otro nodo diferente, no tienen influencia en el diagrama, por lo que se eliminarán del mismo, repitiendo las comprobaciones hasta que no exista ningún nodo sumidero. Esta función no solo se emplea al principio del algoritmo, ya que, como se verá posteriormente, la eliminación de nodos de decisión puede generar un nuevo nodo sin sucesores que habrá que eliminar.

3.2.3. Eliminación de nodo de decisión

Una vez se han revisado todas las precondiciones y se han eliminado los nodos sumideros, se procederá a la eliminación continua de nodos. La primera función para comentar es la que elimina un nodo de decisión cuando se cumplan sus condiciones.

Un nodo de decisión puede ser eliminado únicamente cuando dicho nodo es predecesor del nodo de utilidad y tiene como predecesores al menos todos los predecesores del nodo de utilidad menos sí mismo. En otras palabras, para que se pueda eliminar un nodo de decisión este tiene que depender como mínimo de todas las variables de las que depende la utilidad.

Para eliminar el nodo de decisión se traspondrá la matriz del nodo de utilidad para dejar la fila que corresponde con el nodo de decisión al final. Tras esto, se calculará el máximo de cada fila, dejándolo como valor y perdiendo una dimensión en la matriz. Lo que simboliza es que en el nodo de decisión se tomará siempre la mejor elección posible, de ahí que se guarde el valor máximo. Además, se guardará en una tabla aparte las decisiones que se tomarían en cada combinación de variables posibles. Esto representará la base de conocimiento asociada a su etapa de decisión, y de donde se sacan las reglas que se deberían seguir cuando se enfrente dicha decisión.

Una vez se ha eliminado el nodo de decisión existe la posibilidad de que uno de sus predecesores se quede sin sucesores, lo que crearía un nodo sumidero. Para solucionar esto, se aplicará la función que se comentó en el anterior apartado.

3.2.4. Eliminación de nodos de azar

La siguiente función que comentar es la que elimina un nodo de azar. Un nodo de azar se puede eliminar únicamente cuando su único sucesor es el nodo de utilidad, por lo tanto, lo que hará el algoritmo, será buscar entre los predecesores del nodo de utilidad viendo si alguno cumple las condiciones.

Para eliminar este nodo lo que se hará es multiplicar la matriz de utilidad por las probabilidades almacenadas en el nodo de azar, que simboliza las probabilidades que hay de que la variable tome cada uno de sus posibles valores. Sin embargo, hay que tener en cuenta que el nodo de utilidad y el nodo de azar que se quiere eliminar

pueden o no compartir sucesores, por lo que la multiplicación de matrices se separará por partes.

Se dividirán los predecesores tanto del nodo de azar como el de utilidad en tres grupos: A,B,C, y donde el grupo A estará formado por los predecesores únicos del nodo de utilidad, el grupo B serán los predecesores compartidos y el grupo C serán únicamente los predecesores únicos del nodo de azar.

Tras esto, se transpondrán las matrices de los dos nodos para conseguir un orden específico de las variables. En las dos matrices se coloca primero las dimensiones de los predecesores que comparten, es decir, las variables pertenecientes al grupo B. En cuanto a sus diferencias, en el nodo de utilidad se colocará la dimensión que corresponde al nodo de azar al final de la lista de dimensiones, mientras que el nodo de azar su propio valor será colocado en la penúltima dimensión. Esto se hace de esta manera para que se multipliquen correctamente las matrices, lo que permite multiplicar la utilidad con respecto al nodo de azar con la probabilidad de que sea cada valor.

Para empezar la multiplicación se recorren de manera recursiva las dimensiones que representan los predecesores comunes. Cuando se hayan recorrido todas las variables comunes, será el turno de las variables únicas del nodo de utilidad, pero al contrario que con las variables comunes, esta vez solo se recorrerán en la matriz de utilidad, ya que estas variables no se encuentran entre los predecesores del nodo de azar. Una vez hecho esto, en la matriz de utilidad solo quedarán vectores en función de la variable del nodo de azar, y que serán multiplicadas usando la librería numpy por la matriz del nodo de azar, que incluirá su propia variable en la penúltima posición y el resto de las posiciones serán ocupadas por los predecesores propios de este nodo. Cabe destacar que las dos matrices no tienen por qué ser de las mismas dimensiones, ya que usando la función de multiplicar de numpy se tratarán las matrices de más de dos dimensiones como “rodajas” de matrices bidimensionales que se multiplicarán de manera usual con el vector de la matriz del nodo de utilidad.

Por último, antes de eliminar el nodo de azar, se añadirán sus predecesores al nodo de utilidad, eliminando de este la referencia del nodo de azar que se está eliminando.

3.2.5. Inversión de arcos entre nodos de azar

En el caso de que no exista un nodo, ya sea de azar o de decisión, que se pueda eliminar, se realizará un proceso de inversión de arcos entre nodos de azar que permita que uno de esos nodos cumpla las condiciones descritas en el apartado anterior y que se pueda eliminar. Las precondiciones para que se pueda realizar esta inversión son sencillas, simplemente no puede haber otro arco que conecte estos dos nodos.

Para este problema definiremos los nodos como X, Y, y en donde el arco a invertir es el (X, Y). Esto significa que el nodo X es predecesor del nodo Y, y donde, tras realizar este proceso de inversión, será el nodo Y el que sea predecesor del nodo X.

Para realizar la inversión de arco entre el nodo X y el nodo Y se empezará siguiendo un proceso similar al del apartado anterior, aunque esta vez en vez de tener un nodo de utilidad y un nodo de azar serán los dos nodos de azar. El puesto del nodo de utilidad será ocupado por el nodo Y, por lo que el otro nodo será el nodo X. El cambio que se presenta con respecto al caso anterior es que el nodo Y sí que

tiene una dimensión en su matriz que representa las probabilidades de que salga un valor en concreto del nodo, mientras que esta dimensión no está presente en el nodo de utilidad. Sin embargo, esto se soluciona considerando el valor de Y como un predecesor único del grupo A, haciendo que el proceso de multiplicación sea igual. Con esto se obtendrá la nueva matriz de la variable Y, donde no depende el nodo X, y que además es necesaria para calcular la nueva matriz del nodo X. Para obtener la nueva matriz de X, en donde dependerá del valor de Y, se seguirá el teorema de Bayes, que dice que:

$$P(X|Y) = \frac{P(Y|X) \cdot P(X)}{P(Y)}$$

En la aplicación de esta fórmula, $P(Y)$ es la matriz calculada anteriormente, ya que tras la multiplicación de matrices realizadas ya no depende de la variable X. Por lo que usando esta matriz y las matrices que estaban originalmente en el nodo X e Y se tienen todos los elementos necesarios para aplicar el teorema.

La multiplicación y división de este teorema se ha llevado de manera simultánea y siguiendo un proceso similar al de eliminación de un nodo de azar. En este caso el grupo A estará formado por las variables que eran originalmente predecesoras de X, aunque tras realizar el paso anterior, esas variables son también predecesores del nuevo nodo Y. Por otro lado, el grupo B constituyen las variables que originalmente eran predecesores de X e Y, por lo que también son predecesores del nuevo nodo Y. Por último, el grupo C son los predecesores únicos de Y, y que influyen tanto en la nueva matriz como en la antigua.

Una vez se han definido las variables de cada grupo, a continuación, se realiza un proceso de trasposición de matrices dejándolas con el siguiente orden:

$$P(Y|X) \rightarrow B, C, Y, X$$

$$P(X) \rightarrow B, A, X$$

$$P(Y) \rightarrow B, C, A, Y$$

La importancia del orden radica en el proceso siguiente, ya que se seguirá un proceso iterativo donde se recorrerán las dimensiones de las matrices en el orden diseñado. Para empezar, el proceso recorrerá todos los posibles valores de cada variable del grupo B, ya que al estar presente en las tres matrices se pueden recorrer a la vez sin contratiempos. Tras esto, se recorrerá de la misma forma por las variables del grupo C, pero al no estar presente como predecesor de X, no se alterará dicho nodo en el proceso. Por último, se recorrerán las variables del grupo A de la misma forma que las del grupo C, ya que en este caso estas variables no son predecesores de el nodo original de Y. Una vez se han recorrido los tres grupos en las matrices solo se quedarán los valores de los nodos X e Y, y en donde se multiplicará y dividirá con las funciones de numpy de la forma mostrada en el Teorema de Bayes.

Tras haber acabado todo este largo proceso, se corregirán los predecesores para que no haya incongruencias en ninguno de los dos nodos. Con esto se habrá realizado la inversión de la arista (X, Y) y se habrá creado la arista (Y, X). Al invertir un arco se obtienen las distribuciones marginales y a posteriori de las correspondientes

variables que forman parte de la base de conocimiento. Tras invertir el arco, se repetirá la búsqueda de nodos que cumplan las condiciones de eliminación hasta dejar únicamente el nodo de utilidad.

3.3. Algoritmo genético

El siguiente punto por tratar será la implementación del AG [7, 8]. Con este algoritmo se pretende obtener un DI con todos sus parámetros inicializados que, una vez evaluado, consiga dar un conjunto de reglas coherente con el conocimiento del experto.

Como se ha explicado en el marco teórico, el AG consta de cuatro módulos principales, selección, cruce, mutación y evaluación. Con esta separación fomenta el poder cambiar cada uno individualmente sin que los otros sufran cambios, facilitando su implementación.

Para empezar, se han definido dos conceptos importantes en el desarrollo del algoritmo genético. El primero de ellos es un individuo. Un **individuo** es simplemente el diagrama de influencia con unos pesos específicos inicializado, y que además contiene una función que calcule su valor en el módulo de evaluación. La segunda clase es la **población**, en la que no solo se almacenan los individuos que consten en una generación en concreto, sino que además guardará información relevante como es el conjunto de reglas, que se usará para la evaluación del individuo. Además, también se han implementado unas funciones que calculan la mejor, la peor, y la media de puntuaciones de una generación, y que servirá para estudiar la mejoría del algoritmo a lo largo de las generaciones.

Ya pasando al algoritmo en sí, lo primero que se realiza es la inicialización de una población donde cada individuo está generado con pesos completamente al azar. Esta generación servirá como base y punto de partida del algoritmo, y del que se intentará ir mejorando el puntuaje hasta alcanzar resultados esperados. Una vez se tiene la generación inicial, esta se evaluará con el módulo de evaluación, y se guardará la puntuación de cada individuo. Esta puntuación será útil en el siguiente módulo que es de selección, y del que se elegirán los padres de la siguiente generación. Dichos individuos se irán cruzando y mutando para obtener los individuos de la siguiente generación. La generación creada se rellenará hasta completar el número fijado. Tras esto, se volverá a evaluar y se repetirá el ciclo descrito un número máximo de generaciones.

Una vez visto la implementación general del algoritmo genético, a continuación, se detallará como se ha implementado cada uno de los módulos. Esto permitirá entender el funcionamiento interno del algoritmo, y como se puede ajustar a través de sus hiperparámetros para obtener el mejor modelo. El código en Python de esta algoritmo esta en el anexo .3

3.3.1. Módulo de evaluación

La parte más compleja de esta implementación radica en el módulo de evaluación. Esto es debido a que hay que diseñar una puntuación que refleje la calidad de cada individuo. Para poder realizarlo satisfactoriamente se ha implementado un sistema

que compara las reglas obtenidas tras evaluar el diagrama de influencia de un individuo, con las reglas proporcionadas por un experto.

Las reglas otorgadas por un experto permiten implementar el conocimiento de expertos en el algoritmo genético, ya que su objetivo es conseguir tener un diagrama de influencia que *extienda* las reglas del experto. Con esto, disminuiría la necesidad de contar con un experto, ya que se reduce la precisión que se le requiere. La forma de implementar estas reglas es preguntándole al experto que valores de las variables son necesarias para dar una decisión en concreto. Con esto el experto se podrá centrar en las reglas más influyentes y necesarias, dejando que el resto de las reglas sean generadas tras el proceso del algoritmo genético. Además, el experto no tendrá que explicar que valores debería tomar cada variable en cada regla, ya que se ha implementado una opción para indicar que una variable no es importante en dicho razonamiento.

Por tanto, la forma de comparar las reglas generadas por un individuo con las generadas por el experto es contar las veces que, con los valores de las variables indicados, el individuo elige la misma opción que el experto. Cabe destacar que en las variables que el experto haya anotado como no importantes en una regla específica, se comprobará si el valor coincide en las opciones otorgadas por el individuo en cada opción posible de dicha variable.

Tras todo esto, se le podrá asignar a cada individuo una puntuación que corresponderá al número de reglas en las que ha dado el mismo resultado que el experto. Esta puntuación será la que se use en el resto del algoritmo y es la que se buscará mejorar tras cada generación.

3.3.2. Módulo de selección

Para la implementación del módulo de selección se ha optado por hacerlo mediante un torneo. Una ventaja del uso de torneos es su fácil implementación, ya que, al tener una puntuación fija y única para cada individuo, simplemente se elige al azar un grupo de individuos y se elige al que posea la mayor puntuación. Además, para el uso de selección por ruleta es necesario tener una puntuación clara para escoger y que sea fácilmente transformable a porcentaje sin perder precisión. Por tanto, aunque se haya escogido una forma de puntuación que se explicará en el módulo de evaluación, como se ha usado torneo se puede convertir a la medición del error en el diagrama sin modificar de forma considerable este módulo.

Para poder optimizar la selección se usarán tres hiperparámetros diferentes. El primero de ellos es simplemente saber el *porcentaje de población que será escogida*. Esta decisión influye tanto en el número de individuos generados en el cruce como en la parte de elección al final.

Los otros dos hiperparámetros miden la intensidad de diversificación que presenta el módulo. Uno de ellos es el *tamaño de los torneos*, pudiendo variar entre dos, cuatro u ocho individuos en cada torneo. Cuanto más numeroso sea un torneo, más probabilidades hay de que se elijan a los mejores individuos, lo que podría ocasionar que se perdiera la diversidad en una generación. El otro hiperparámetros es un booleano que indica *si puede ganar un individuo varios torneos*. Al igual que con el otro parámetro, si se permite la posibilidad de que gane más de un torneo se podría tener al individuo duplicado, lo que haría perder diversidad, pero quizás mejore la conver-

gencia del algoritmo. Mas adelante, se harán algunas pruebas para determinar que valores conviene en un ejemplo específico.

3.3.3. Módulo de cruce

El segundo módulo a estudiar es el módulo de cruce. Este módulo consiste en unir el material genético de sus padres seleccionados y que posteriormente se mutará antes de ser introducido en la nueva generación.

La implementación de este módulo ha consistido en elegir al azar entre los dos padres cada fila de cada matriz de probabilidad que es el conjunto de parámetros que define una distribución condicionada. Con esto, se elige en cada nodo de azar que vector de probabilidades empelar en cada conjunto de valores de entrada del nodo. Por tanto, se decidirá si empelar las probabilidades de un padre o del otro, permitiendo que su comportamiento sea parecido al de estos. En caso de la utilidad es similar, lo que la elección al azar no sea sobre el vector de probabilidades, sino que el de utilidad con respecto a su última variable de entrada en la matriz.

3.3.4. Módulo de mutación

El módulo restante es el módulo de mutación. Este módulo esta diseñado para evitar que en una generación todos sus individuos sean muy parecidos. Para conseguirlo, habrá una probabilidad de modificar tanto las probabilidades como la matriz de utilidad, con la que se generará un individuo con probabilidades y utilidades diferentes al resto.

La implementación de este módulo es muy parecida al de cruce, ya que la modificación se realiza con el último vector de cada matriz. Lo diferente con respecto en este módulo es que, si se supera una probabilidad mínima, se elegirá un valor del vector que estamos modificando. Tras esto, se modificará el valor por uno cercano, y se ajustarán el resto de las probabilidades del vector para que siga siendo una distribución de probabilidades. Además, este operador requiere un procedimiento de reparación de las distribuciones, que se asegure que la suma de las probabilidades sea igual a uno y que ningún elemento tenga una probabilidad menor que cero.

En este módulo influyen dos hiperparámetros diferentes, la probabilidad de mutación y la variación de mutación. El primero de ellos controla *la probabilidad de que un vector sea mutado*. Cuanto mayor sea esta probabilidad más se mutarán los genes de un individuo y más se alejará al modelo de sus padres.

El segundo hiperparámetro mide cuanto *puede mutar el elemento escogido de un vector*. La mutación de un valor sigue una distribución uniforme, por lo que, a mayor valor de variación, más probabilidad hay de que se aleje del valor original.

3.4. Entrada y salida del sistema

Para que la implementación sea fácil de utilizar es necesario introducir una manera de poder comunicarse con el programa.

3.4.1. Ficheros de entrada

La entrada del proyecto consistiría principalmente en introducir el diagrama que se va a estudiar y el conjunto de reglas especificadas por el experto que conduzcan el entrenamiento del algoritmo genético. En cuanto al diagrama, es necesario la creación de un archivo .py que cree el diagrama de influencia y que le añada todos sus nodos de forma ordenada. En cada nodo, será necesario dar los posibles valores que puede adoptar exceptuando el nodo de utilidad. Además, también se pide añadir los predecesores de dicho nodo, esto es necesario para poder construir el diagrama y poder evaluarlo correctamente.

La segunda entrada es el fichero de reglas aportado por el experto. En este caso se usará un fichero .txt, para que así no sea necesario que el experto manipule la implementación. En este archivo se deberá añadir las reglas de cada uno de los nodos de decisión que hubiera en el diagrama. Además, antes de añadir las reglas se tendrá que añadir una especie de índice, donde se ordenarán las variables que influían en el nodo en el momento de eliminación en el algoritmo de evaluación. Tras esto, las reglas se redactarán como una lista de valores de las variables que influyen separadas por comas, acabando con la decisión que el experto concluya que haya que tomar. Se le permitirá al experto poner un -1 en vez del valor de la variable, lo que significaría que esa variable no es relevante en dicha regla y por tanto debería cumplirse para cualquier valor de esa variable.

Esta parte es muy importante en el proyecto, porque implementa la representación del conocimiento experto como una base de conocimientos (BC) implícita, al experto no se le pide una lista de reglas enorme, porque las tablas de la BC son de tamaño exponencial respecto al numero de atributos).

También se ha implementado un código para poder fabricar fácilmente un conjunto de reglas funcionales y reales. Esto se realiza usando las reglas obtenidas de resolver un diagrama de influencia donde los pesos han sido creados por expertos. Por tanto, lo que hace este código es cambiar el formato de las reglas al empleado en este proyecto y reducirlo a un menor número de reglas. Esto servirá para comprobar más adelante la efectividad del algoritmo genético aplicado en diagramas de influencias.

3.4.2. Ficheros de salida

En cuanto al fichero de salida simplemente se escribirán los resultados en un libro de Excel. Para ello, se sacan tres medidas diferentes en cada generación, y además se hace la media de tres ejecuciones diferentes para reducir la aleatoriedad en los resultados. Con estos resultados se podrá estudiar la evolución que se realiza en cada generación y comprobar que cambios aparecen al modificar los hiperparámetros.

Las tres medidas que se obtienen en cada generación son las puntuaciones obtenidas, y donde se buscará acercarse a la puntuación máxima que coincide con el número de reglas. Son tres medidas diferentes ya que se mide una cualidad diferente de la generación. La primera medida es la puntuación del individuo que *mejor rendimiento* ha dado en esa generación, por lo que se sería el modelo que se ha aproximado mejor a las reglas dadas por el experto. Las otras dos medidas son la *puntuación media* y *la peor* puntuación. Con estas dos medidas se obtendrá el rendimiento general de la generación, y se podrá ver cuanta diversidad presenta.

Capítulo 4

Evaluación del rendimiento del AG aplicado a los DI

Después de haber detallado la implementación llevada a cabo en este trabajo, este capítulo se centrará en la evaluación del rendimiento del AG aplicado a los DI. Esta etapa es fundamental para comprender la eficacia y los límites de nuestra metodología. Para ello, daremos inicio probando el algoritmo con una serie de diagramas básicos. Estos ejemplos simples nos permitirán observar y comprender las capacidades del algoritmo en condiciones controladas.

Una vez comprobado el funcionamiento del programa, se realizará una fase iterativa. Durante esta fase, se ajustará los hiperparámetros del algoritmo genético. Este proceso será llevado a cabo utilizando el DI que se ha descrito en el marco teórico. Es importante destacar que este diagrama posee dimensiones sustancialmente mayores y complejidades adicionales en comparación con los ejemplos básicos. Por lo tanto, su uso permitirá no solo evaluar el rendimiento del algoritmo en situaciones más desafiantes, sino también identificar posibles limitaciones y áreas de mejora.

Al explorar el comportamiento del AG en una variedad de escenarios y al ajustar cuidadosamente sus parámetros, se podrá obtener una visión más completa de su desempeño para abordar problemas de mayor envergadura. Este análisis permitirá no solo comprender las fortalezas del algoritmo, sino también establecer pautas claras para su aplicación efectiva en la resolución de problemas de decisión complejos.

4.1. Uso de algoritmo genético con diagramas reducidos

Los diagramas de influencia pueden llegar a ser muy complejos, ya que influyen una gran cantidad de variables en las decisiones a tomar. Por tanto, se han diseñado dos DI simples para poder comprobar las capacidades del AG.

4.1.1. Diagrama de prueba

El primer diagrama ha sido diseñado expresamente como prueba para este proyecto. Con este propósito presenta una estructura relativamente sencilla, con un único nodo de decisión. Esta característica simplifica el proceso de generación de reglas, ya que solo se requiere una tabla de reglas una vez que el diagrama ha sido evaluado. Además del nodo de decisión, el diagrama cuenta con un nodo de utilidad, una presencia habitual en este tipo de representaciones, y cuatro nodos de azar, que introducen elementos de incertidumbre en el proceso de toma de decisiones.

Al centrarnos en este diagrama inicial, podemos obtener información valiosa sobre cómo el algoritmo genético aborda la optimización de parámetros en un entorno más controlado. La simplificación de la estructura nos permite identificar y analizar con mayor precisión las decisiones tomadas por el algoritmo en cada paso del proceso.

La figura 2.1 muestra el DI de prueba, permitiendo esta representación gráfica probabilística el poder ver las independencias condicionales entre variables, el flujo de información y posibles relaciones causales entre ellas.

En relación con las reglas del experto, se ha llevado a cabo una prueba en la que se proporciona el 100 % de las reglas establecidas por dicho experto. Estas reglas constituyen el estándar con el cual el algoritmo genético intentará imitar sus resultados. Esta prueba permite estudiar la capacidad del algoritmo genético para modificar los valores de probabilidades y utilidades con el fin de adaptarse a las reglas establecidas por el experto.

Al utilizar el conjunto completo de reglas proporcionadas por el experto como referencia, podemos evaluar con mayor precisión la capacidad del algoritmo para ajustar sus parámetros y producir resultados que se alineen estrechamente con las decisiones tomadas por el experto. Esta comparación directa permite determinar la eficacia del algoritmo en imitar y, potencialmente, mejorar las decisiones establecidas por un experto humano.

Para extraer el resultado, se comparan las reglas dadas por el experto con la elección tomada por el DI a evaluar en cada caso. Si ha coincidido con la elección de las reglas, se sumará uno a la puntuación obtenida, de tal forma que la puntuación máxima será el número de reglas usadas en el entrenamiento, por lo que cuanto más puntuación se obtenga estará más cercano a la opinión del experto. Esto se realiza en cada individuo de una generación, guardando los datos en una tabla como 1, donde se muestra al mejor y peor individuo de una generación, además de calcular la media obtenida.

	D_NO	D_YES	D_MAYBE
B_NO	0.41010517	0.05931904	0.53057579
B_YES	0.73741116	0.12223549	0.14035334
B_MAYBE	0.54216732	0.39390495	0.06392773

Cuadro 4.1: Tabla de probabilidad condicionada del nodo D

	D_NO	D_YES	D_MAYBE
A_NO	NO	NO	NO
A_SI	YES	YES	NO

Cuadro 4.2: Tabla de decisiones

En las tablas 4.1 y 4.2 se muestran la tabla de probabilidad condicionada del nodo D y la tabla de decisión respectivamente. La tabla 4.1 presenta la probabilidad de que se obtenga cada valor de D según el valor de B. Por otro lado, en la tabla 4.2 se muestra la opción del nodo de decisión que presenta mayor utilidad según los valores de las variables aleatorias A y D

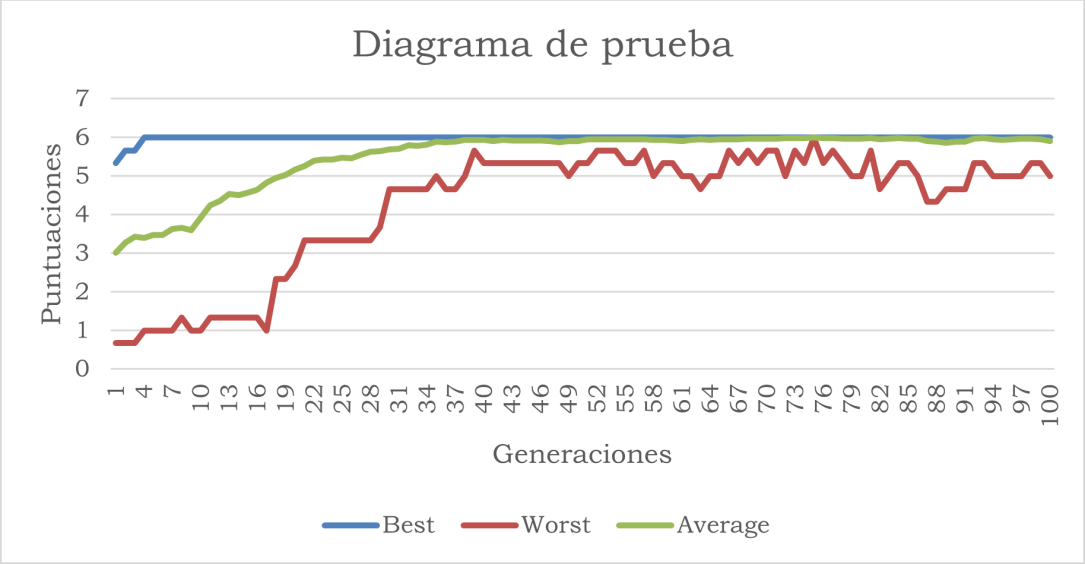

Figura 4.1: Gráfica usando el DI de prueba

En la gráfica 4.1 se muestra el transcurso de las tres medidas a estudiar durante las cien iteraciones del algoritmo genético. Estas medidas muestran la puntuación tanto del mejor y peor individuo de la generación como la media de puntuación alcanzada. Además, se ha realizado la ejecución tres veces y se han extraído la media de dichas ejecuciones para reducir la aleatoriedad. Este proceso ha durado algo más de 6 minutos. Cabe destacar que, si se aumentara el número de ejecuciones, se podría medir el error de estimación con una mayor precisión. Además, si se pudiera realizar este proceso en paralelo, no debería aumentar el tiempo necesario.

Como se puede observar en la gráfica, se logró replicar el conjunto total de reglas en menos de cinco generaciones. Además, la población en su conjunto demostró una rápida convergencia, evidenciada por el hecho de que incluso el peor individuo de cada generación alcanzó una puntuación mínima de cuatro sobre seis. Este resultado es especialmente notable dado que se trata de un problema relativamente simple, con un único nodo de decisión. Sin embargo, resalta la efectividad del algoritmo genético, que sin haber optimizado los hiperparámetros, demostró su capacidad para adaptarse a las reglas establecidas en muy pocas generaciones.

4.1.2. Diagrama de helicóptero

Antes de adentrarnos en el estudio del diagrama de influencia principal, se ha llevado a cabo una prueba adicional con el objetivo de explorar el desempeño del algoritmo genético desde una perspectiva diferente. Esta prueba complementaria nos brinda la oportunidad de analizar aspectos específicos del algoritmo o de la metodología empleada que pueden no haber quedado completamente claros en la primera prueba.

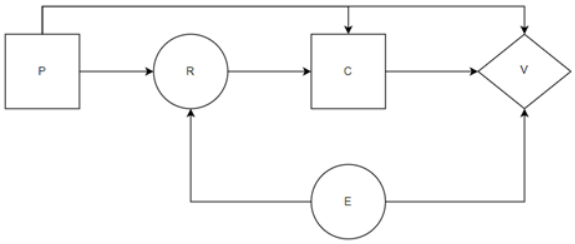

Figura 4.2: DI basado en la compra de un helicóptero [5]

En este escenario, se presenta el diagrama de influencia 4.2 que consta de dos nodos de decisión y dos nodos de azar, como se muestra en la imagen adjunta. Este diagrama simula un caso hipotético en el que se debe tomar la decisión de realizar una revisión y, posteriormente, la compra de un helicóptero. Este caso resulta especialmente interesante debido a que el modelo está completamente inicializado con todos sus parámetros, y se utilizará para generar pruebas con el fin de entrenar el algoritmo genético.

El objetivo principal de este ejercicio es evaluar si el algoritmo genético es capaz de generar el conjunto completo de reglas, tomando como referencia las reglas proporcionadas inicialmente. Este proceso implica que el algoritmo deberá ajustar los parámetros del modelo para que las reglas generadas coincidan lo más posible con las reglas de referencia.

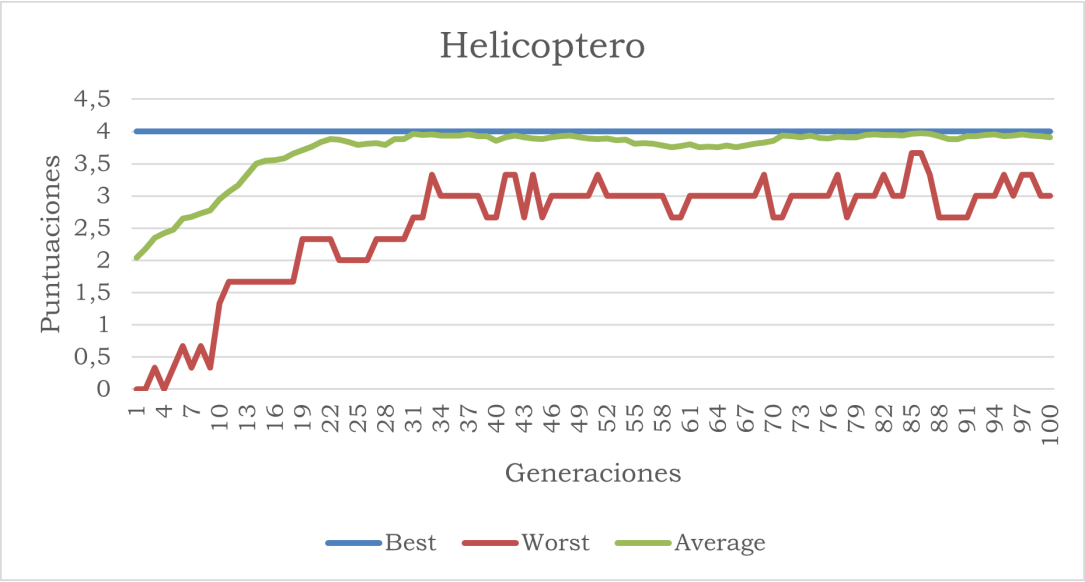

Figura 4.3: Gráfica usando el DI del helicóptero

La gráfica 4.3 muestra que, debido al menor número de reglas en comparación con la prueba anterior, se logra alcanzar la puntuación máxima desde la primera iteración. Además, dado que el modelo tiene menos parámetros, el peor individuo de cada generación exhibe saltos más pronunciados que en la prueba anterior.

En cuanto a la predicción de las reglas restantes, el algoritmo se entrenó con cuatro reglas, dejando tres por predecir. Los resultados obtenidos fueron diversos: una de las tres ejecuciones logró predecir satisfactoriamente las tres reglas restantes, mientras que en las otras dos iteraciones solo se predijo una de las tres reglas. Esta variabilidad en los resultados puede atribuirse, en parte, a la limitada cantidad de datos de entrenamiento proporcionados. Es posible que una mayor cantidad de datos de entrenamiento y la utilización de diagramas de influencia más complejos mejoren la capacidad del algoritmo para predecir con precisión las reglas restantes.

Es fundamental destacar que las reglas proporcionadas al AG fueron seleccionadas de manera arbitraria para este ejercicio. En un entorno real, sería el experto quien proporcionaría las reglas más relevantes y significativas para la toma de decisiones, lo que podría influir significativamente en los resultados obtenidos por el algoritmo. Por lo tanto, es importante tener en cuenta este factor al interpretar los resultados y considerar cómo la calidad y relevancia de los datos de entrada pueden afectar el rendimiento del algoritmo genético en aplicaciones prácticas.

4.2. Ajuste de hiperparámetros

Una vez explicada la implementación del proyecto y verificado el funcionamiento del AG mediante dos algoritmos de prueba, se procederá a la siguiente etapa: el ajuste de los hiperparámetros. Esta fase es fundamental para afinar el desempeño del AG y adaptarlo de manera óptima al problema específico que se pretende resolver, por lo que han habido artículos [12, 13] donde se han estudiado diferentes ajustes de AG para tipos de problemas específicos.

Es esencial destacar y registrar el tiempo empleado en llevar a cabo todas estas pruebas de ajuste de hiperparámetros. Este tiempo no solo es una medida del esfuerzo invertido en el proceso, sino que también representa el tiempo que se ahorra en comparación con el tiempo necesario para obtener soluciones de manera manual o mediante la consulta de un experto.

En estas pruebas de ajuste de hiperparámetros, se explorarán diferentes valores para cada uno de ellos. Se variarán sistemáticamente los parámetros relevantes, buscando un equilibrio entre la exploración del espacio de soluciones y la explotación de soluciones encontradas. Se llevará a cabo una evaluación comparativa de los resultados obtenidos con cada configuración de hiperparámetros para determinar cuál de ellas produce los mejores resultados en términos de calidad de la solución y eficiencia del algoritmo.

Para facilitar este proceso, se organizarán los hiperparámetros en grupos según el módulo al que pertenezcan y se ajustarán de manera individualizada. Esto permitirá una evaluación más precisa de cómo cada conjunto de hiperparámetros afecta al desempeño general del algoritmo genético.

4.2.1. Medidas generales

Una estrategia para mejorar los resultados es aumentar el número de generaciones y de individuos en cada generación. Sin embargo, nos encontramos con el factor limitante de la capacidad computacional, lo que dificulta incrementar estos valores de manera significativa. Por tanto, se ha tomado la decisión de utilizar cien generaciones, cada una con cincuenta individuos, y repetir este proceso tres veces, calculando luego la media de los resultados obtenidos.

Es importante señalar que sería ideal aumentar el número de iteraciones ejecutadas, ya que esto proporcionaría una mayor estabilidad a las decisiones tomadas con respecto a los hiperparámetros. Sin embargo, utilizando esta configuración se necesitan aproximadamente siete horas y media para completar el proceso, lo que hace inviable a priori aumentar los parámetros mencionados. Cabe destacar que el ajuste tradicional con el experto puede durar semanas o incluso meses, por lo que es una reducción de tiempo considerable.

Esta limitación también subraya la importancia que tendría poseer un equipo con una mayor capacidad computacional, ya que una mayor potencia de procesamiento podría traducirse en mejores resultados. Este aspecto es relevante para evaluar la viabilidad del uso de AG en el ajuste de los modelos de probabilidad y preferencias en diagramas de influencia. Un mayor poder de cómputo no solo aceleraría el proceso de optimización, sino que también permitiría explorar configuraciones más complejas de hiperparámetros y mejorar la calidad de las soluciones obtenidas. Otra forma

de reducir el tiempo sería realizar las pruebas de forma paralela, lo que también aceleraría el proceso de optimización.

4.2.2. Módulo de selección

Como se ha explicado anteriormente, este módulo tiene como objetivo determinar que individuos serán cruzados para generar la siguiente generación. Además, se ha empelado la elección por torneo, lo que permite escoger al individuo ganador de manera inmediata.

4.2.2.1. Tamaño del torneo

El primer hiperparámetro decide el número de individuos que participan en cada torneo. Los tres tamaños más comunes son dos, cuatro y ocho individuos. Se usan potencias de dos ya que permite realizar enfrentamientos entre dos individuos, avanzando las eliminatorias hasta obtener un ganador claro.

En principio usando tamaños de torneo pequeños promueve la diversidad y la exploración, dando la oportunidad a individuos peores a sobrevivir y transmitir sus características. Esto puede ser bueno debido a que evita caer en óptimos locales. Por otro lado, cuanto mayor sea el torneo será más seguro que los mejores individuos de una generación sean escogidos. La decisión de que tamaño se usará se tomará tras comparar los resultados obtenidos con cada valor.

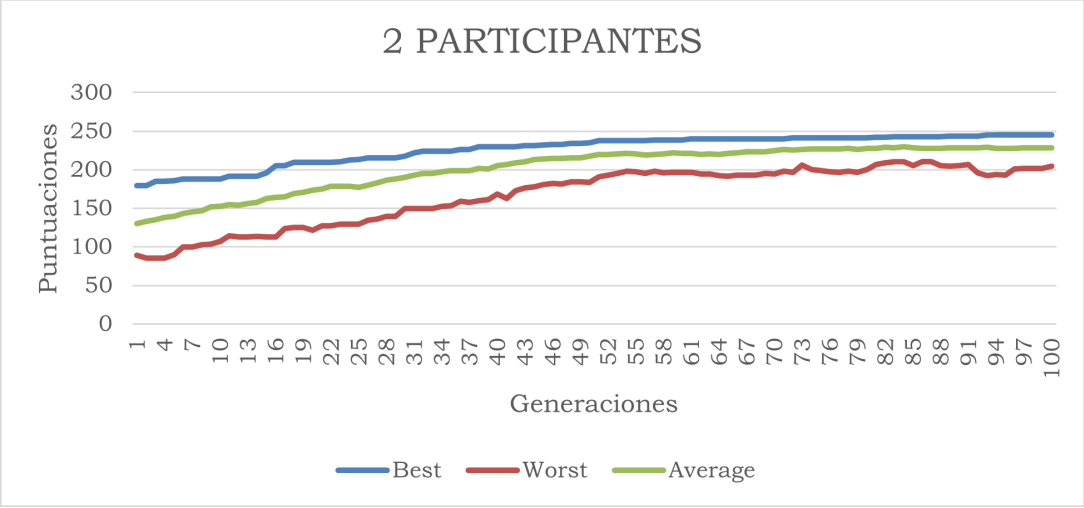

Figura 4.4: Gráfica en la que el torneo es de 2 participantes

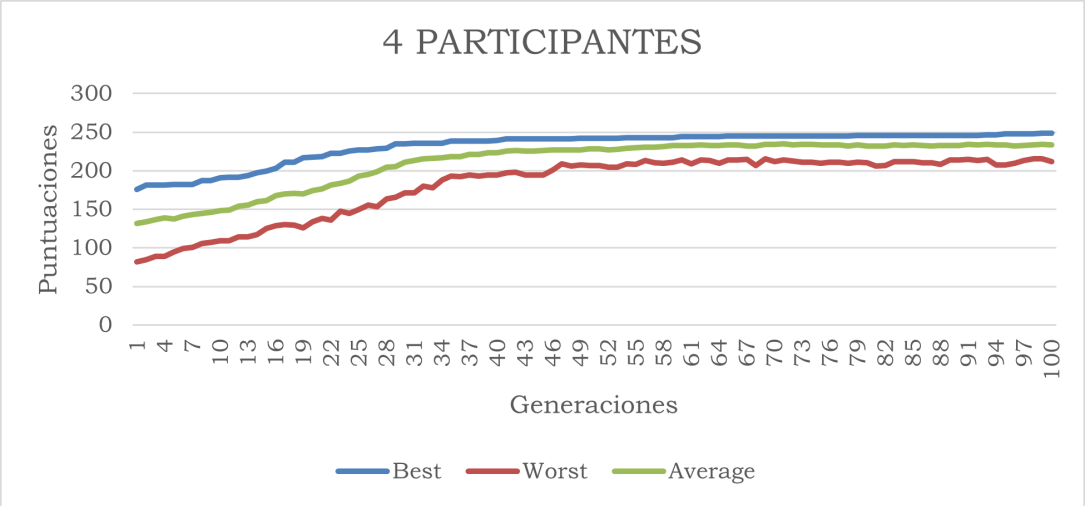

Figura 4.5: Gráfica en la que el torneo es de 4 participantes

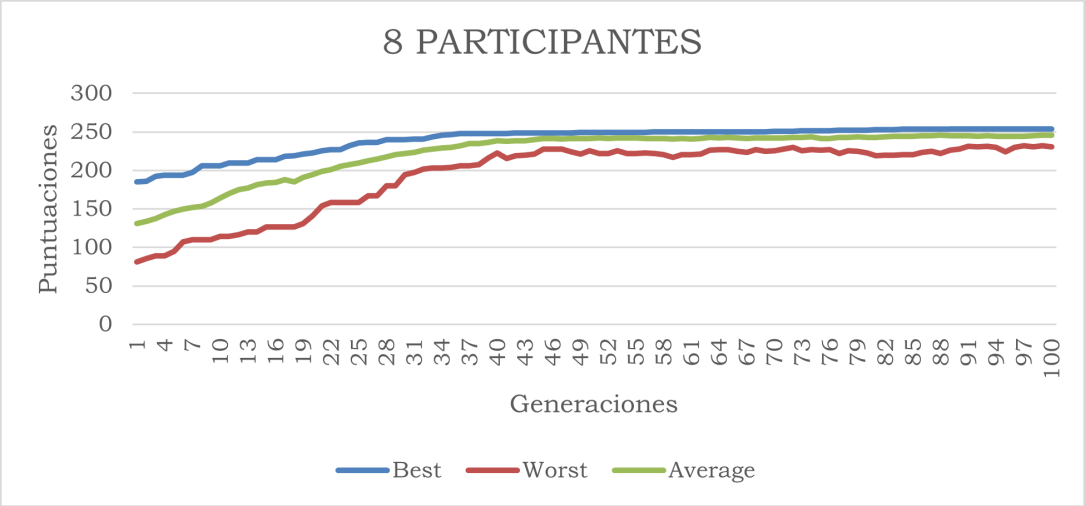

Figura 4.6: Gráfica en la que el torneo es de 8 participantes

En las tres figuras 4.4 4.5 4.6 vemos el rendimiento del algoritmo genético usando diferentes valores como tamaño del torneo. Cabe destacar que el resto de hiperparámetros no se han modificado entre ejecuciones pero que si pueden afectar al rendimiento ya que no son variables totalmente independientes. Además, el número de reglas con las que se ha entrenado el algoritmo es de 484 reglas, por lo que esta es la puntuación máxima que puede alcanzar el algoritmo. Como vemos en las gráficas, el número máximo que se alcanza es 250, por lo que actualmente se estaría prediciendo correctamente el 52 % de las reglas.

Ya comparando los resultados obtenidos, vemos como las gráficas son bastantes similares, siguiendo toda una estructura ascendente hasta estancarse sobre la generación cincuenta. Además, a medida a que se aumenta el número de participantes en un torneo, aumenta el valor máximo conseguido, llegando con un torneo de ocho participantes a 250 de puntuación media. Esto es lo que se había predicho, a mayor número de participantes en un torneo mayor es la probabilidad de escoger a los mejores individuos, y por tanto permite alcanzar más rápidamente cotas más altas.

Por otro lado, un menor número de participantes en los torneos debería mejorar la diversidad, haciendo que fluctúen más los resultados entre generaciones. Sin embargo, aparentemente no se detecta esa diversidad entre generaciones, ya que simplemente en sus gráficas hay más diferencia entre el mayor puntuado de una generación y el resto, pero no se nota un cambio entre generaciones que justifique el usar tamaños de torneo menores.

Cabe destacar que en todas las pruebas tiene como hiperparámetro base el elegir siempre a los dos mejores individuos de una generación para que formen parte de la siguiente generación. Sin embargo, aunque esto pueda modificar la gráfica azul, haciendo que esta siempre sea ascendente, no es el caso de las otras dos gráficas, en las que tampoco se aprecia la diversidad entre generaciones tal y como se ha comentado.

Por todo esto, se ha elegido los torneos con ocho participantes, ya que es la que da mejores resultados y la diversidad que daría usar torneos más pequeños no es apreciable en las gráficas.

4.2.2.2. Posibilidad de repetición

La segunda decisión para tomar en el módulo de selección es si permitir que un individuo ganador de un torneo pueda participar nuevamente en otro torneo. El permitir a un individuo volver a participar fomentaría los buenos resultados haciendo que la próxima generación sea más parecida a los individuos con mayor desempeño. Esto permitiría avanzar más rápidamente a buenas soluciones.

La contraparte de esto es que se perdería mucha capacidad de diversificación, ya que el número de padres diferentes se reduce en gran medida. Al perder esta diversidad es posible no alcanzar las mejores soluciones, ya que el algoritmo genético puede estancarse en óptimos locales que perjudiquen su mejoría.

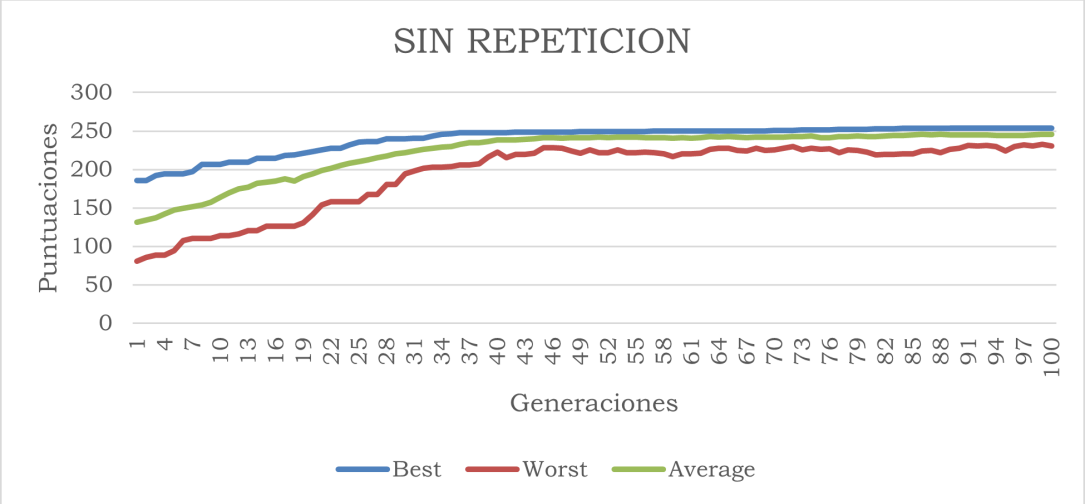

Figura 4.7: Gráfica en la que el torneo se realiza sin repeticiones

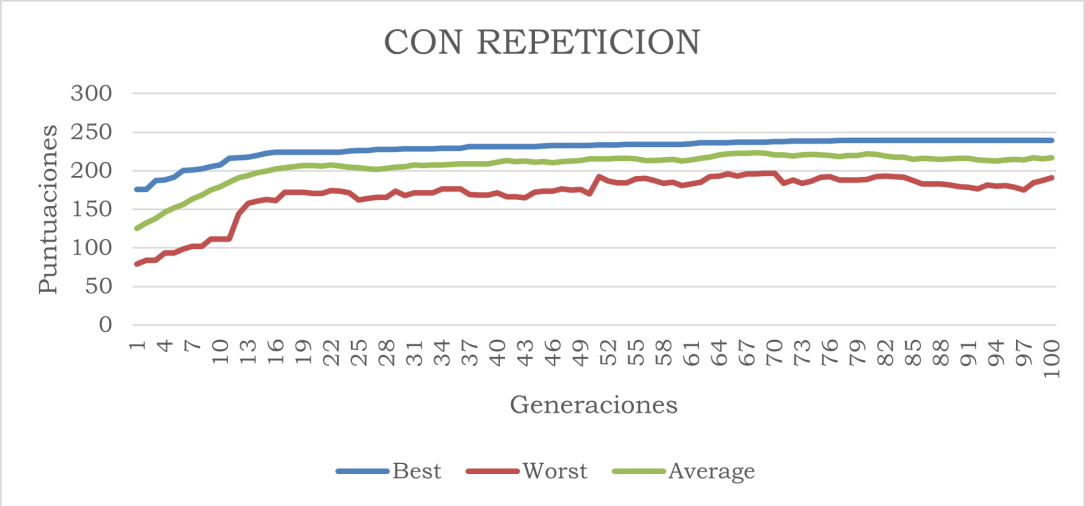

Figura 4.8: Gráfica en la que en el torneo se permite la repetición

Viendo las gráficas 4.7 y 4.8 se puede ver como la prueba donde no se permite repeticiones consigue alcanzar mejores resultados en las tres métricas estudiadas. Esto puede deberse a que se haya encontrado un óptimo local que no permita obtener resultados tan favorables como en la otra prueba. Sin embargo, hay que destacar la rapidez con que alcanza sus mejores soluciones la prueba donde se permite la repetición, ya que solamente ha necesitado unas diez generaciones en encontrar su mejor individuo, mientras que en la otra prueba ha necesitado unas treinta generaciones aproximadamente.

Otro tema que destacar es la inestabilidad de la métrica roja, que simboliza el peor individuo de la generación, en la prueba permitiendo la repetición. Esto puede deberse a que, al permitir que un mismo individuo pueda ganar varios torneos, se limita mucho las probabilidades de ser escogidos los peores individuos.

Sin embargo, debido a que la clara diferencia de puntuaje en las métricas entre las dos pruebas no se permitirá a un individuo ganador de un torneo volver a participar en otro torneo, negando la posibilidad de que un mismo individuo sea elegido más de una vez.

4.2.2.3. Cantidad de individuos seleccionados

La última decisión por tomar en el módulo de selección es decidir cuantos individuos se seleccionarán en cada generación. Bajar el número de individuos seleccionados implicaría disminuir el número de cruces y mutaciones que se realizan. En las pruebas realizadas se ha rellenado al azar la próxima generación entre los individuos de la generación pasada, y solamente se ha generado un individuo por cada pareja, por lo que esto puede afectar a los resultados obtenidos. Esto se decidió de esta manera para no aumentar el ya elevado tiempo de ejecución, y además esto podría disminuir la diversidad del algoritmo genético.

Se plantearán tres opciones diferentes, seleccionar un 25 %, 50 % o 75 % de la población. La diferencia entre cada opción radica, al igual que en la mayoría de las decisiones, si fomentar la diversidad o la convergencia del algoritmo genético.

Cuanto mayor sea el porcentaje escogido, menos importancia se le dará al módulo de selección, ya que se seleccionará una mayoría de la población, y por tanto su papel será el de evitar transmitir la información genética de los peores individuos. Por otro lado, un menor porcentaje de elegidos provocará que solo se transmita los genes de menos individuos, y donde la generación posterior será muy parecida a la anterior, ya que la mayoría de los individuos serán escogidos de forma azarosa del total de la población. Por tanto, se puede decir que el 25 % aumentaría la convergencia del algoritmo, ya que solo selecciona un grupo pequeño, mientras que el 75 % provocaría una diversificación entre los individuos generados, ya que se crearían con el material genético de buena parte de la población.

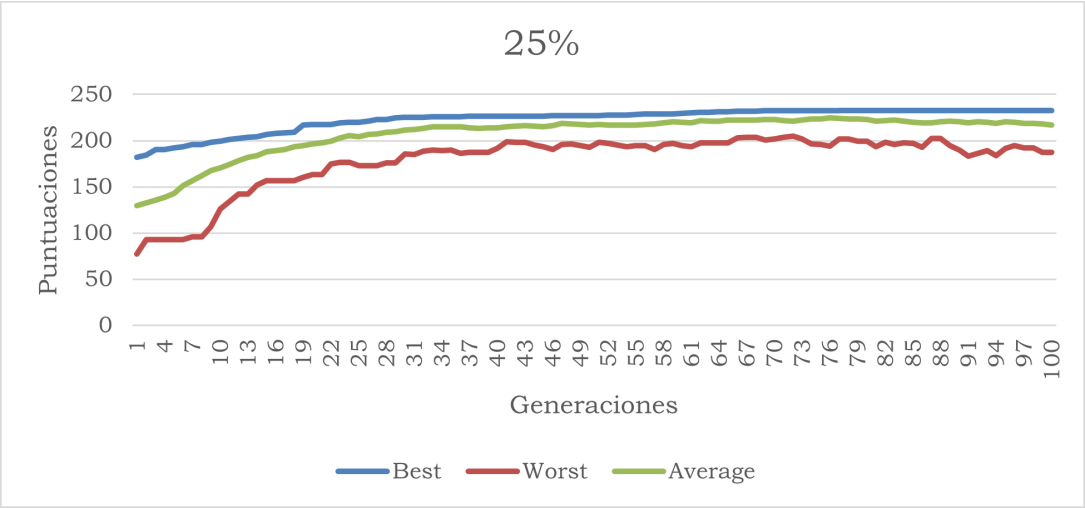

Figura 4.9: Gráfica donde se selecciona al 25 % de la población

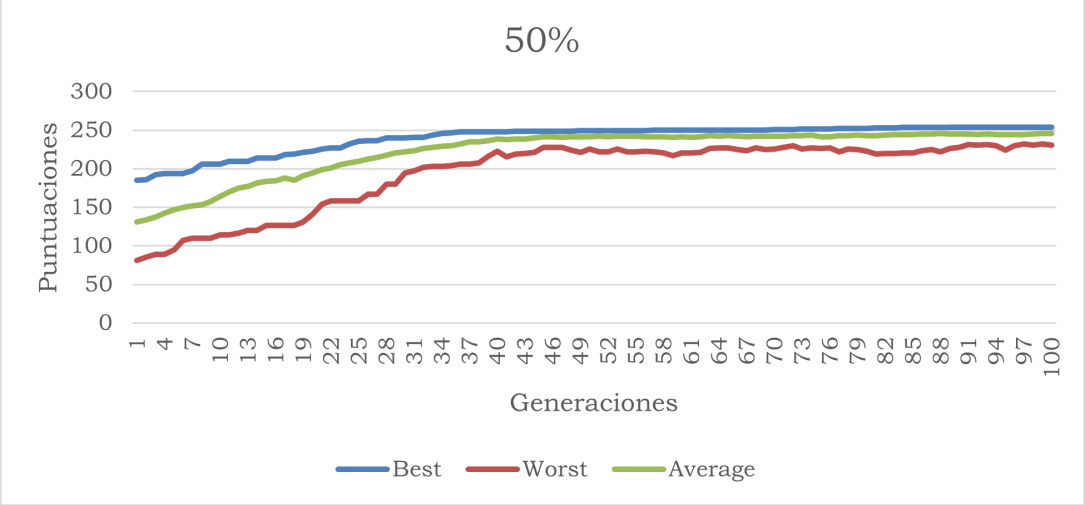

Figura 4.10: Gráfica donde se selecciona al 50 % de la población

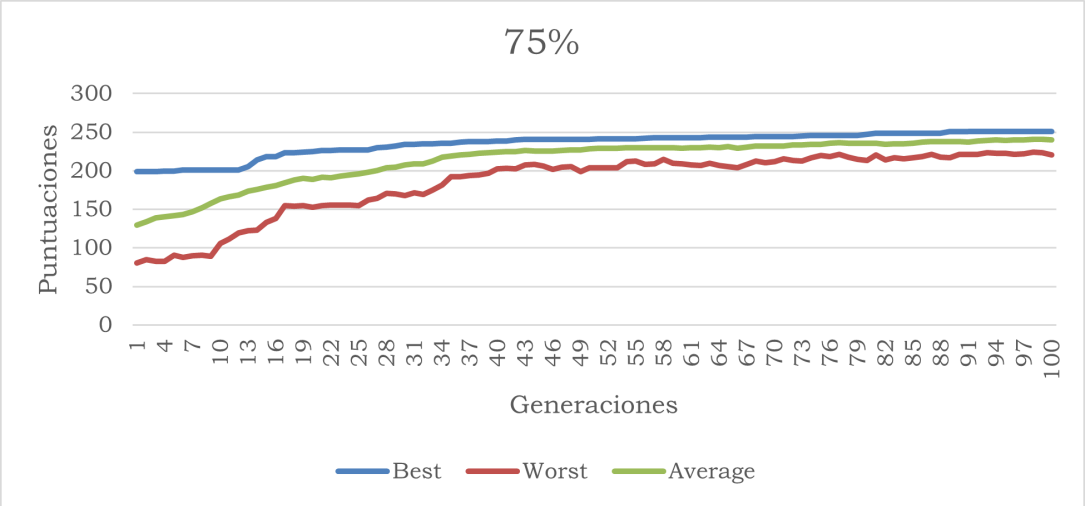

Figura 4.11: Gráfica donde se selecciona al 75 % de la población

Lo primero a comentar es el tiempo de ejecución de cada prueba, y es que, como el mayor tiempo empelado está en la evaluación de nuevos individuos, contra menos individuos se seleccionen menos tiempo tardará el algoritmo genético. Esto se refleja en que la primera prueba que corresponde a la figura 4.9 tardó unas cuatro horas, la de 4.10 tardó unas siete horas y medias, y por último la del 4.11 tardó algo más de once horas.

Mientras que la reducción de tiempo es un factor muy positivo para el desarrollo de estas pruebas, los resultados obtenidos por la primera prueba son muy inferiores a las otras dos. Esto puede deberse a que no se generan suficientes individuos en cada generación, y por tanto no tiene tantas oportunidades de generar individuos mejores.

Por otro lado, la figura 4.11 sí que ha obtenido resultados cercanos al de la prueba de 4.10, e incluso ha necesitado menos generaciones en alcanzar dichos resultados. Sin embargo, al igual que pasaba al permitir la repetición de ganadores en los torneos, se alcanza rápidamente un óptimo local que es difícil de superar en el resto de las generaciones, ya que en ese punto la generación es demasiado homogénea.

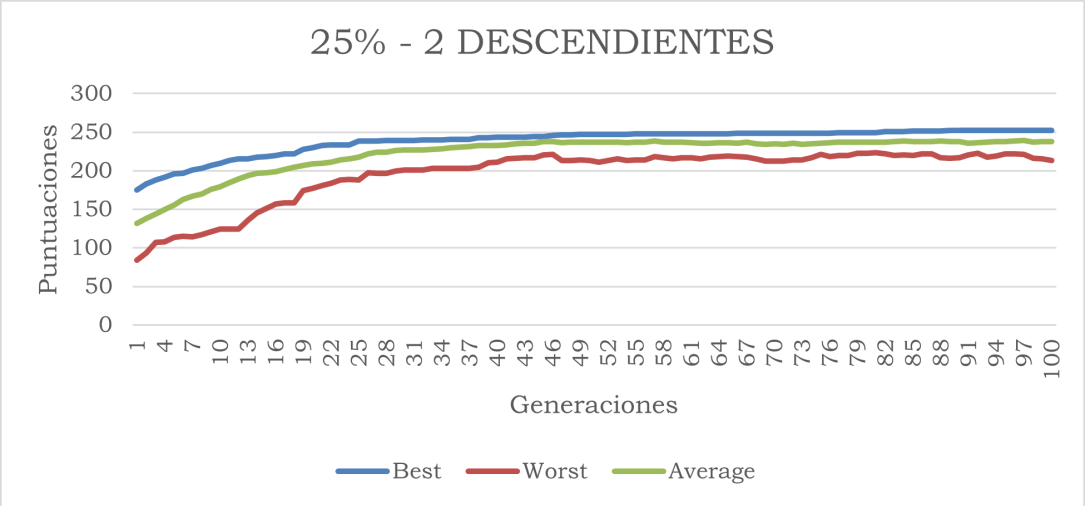

Figura 4.12: Gráfica donde se selecciona al 25 % de la población y se generar dos descendientes por pareja

Se ha realizado otra prueba en este apartado. La prueba de la figura 4.12 consiste en generar dos descendientes por cada par de individuos seleccionados, seleccionando un 25 % de la población. El motivo de esta prueba radica en el ahorro de tiempo que producía bajar el porcentaje de seleccionados. Cabe destacar que esta última prueba ha tardado en ejecutarse casi ocho horas, algo más que si solamente se seleccionase el 50 %.

Los resultados obtenidos son muy parecidos al de la figura 4.10, aunque unas pequeñas diferencias. La métrica del mejor individuo y la media son muy similares, mientras que la métrica roja es la que presenta las diferencias, ya que converge menos en la prueba 4.12 que en la 4.10. Esto no tiene por qué ser negativo, ya que puede dar más diversidad y que esta sea necesaria en las posteriores pruebas para alcanzar nuevos óptimos. El otro cambio es la velocidad de mejora en las primeras generaciones, ya que en esta gráfica se estabiliza más rápidamente que en la anterior.

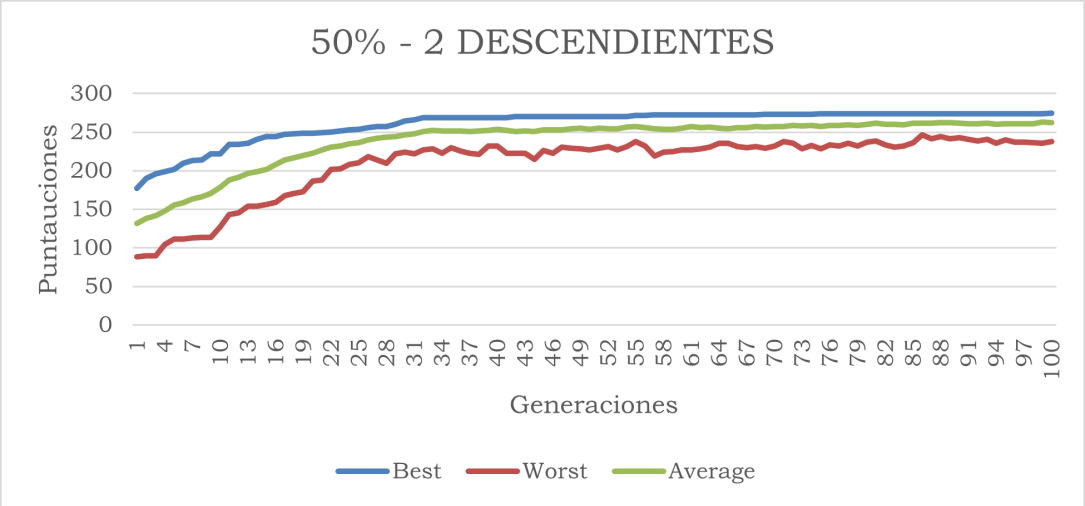

Figura 4.13: Gráfica donde se selecciona al 50 % de la población y se generar dos descendientes por pareja

Para poder tomar esta decisión se ha decidido realizar una última prueba 4.13, en donde se seleccionará el 50 % y se generarán dos individuos de cada pareja seleccionada. Esta prueba ha dado resultados bastante mejores que las anteriores, llegando a puntuaciones de más de 270. Además, ha conseguido alcanzar su valor máximo tras pasar las treinta generaciones, un número muy parecido al de la prueba de la figura 4.10, en donde se usaba el 50 % generando un solo descendiente. Sin embargo, esta última prueba ha tenido un tiempo de ejecución demasiado elevado, habiendo tardado casi quince horas en total. Por tanto, al dar mejores resultados, pero con un tiempo inviable, se ha decidido optar por generar dos individuos por pareja seleccionada, pero solamente dejando cincuenta generaciones, para así poder el tiempo de ejecución.

4.2.3. Módulo de mutación

El objetivo de este módulo es el de añadir diversidad a los individuos generados, lo que permite explorar diferentes zonas del conjunto de soluciones con las que poder alcanzar soluciones óptimas que de otra manera no sería posible con el resto de los módulos al caer en óptimos locales. Cabe destacar que no se mencionado el módulo de cruce ya que no tiene ningún hiperparámetro relevante que pueda ser ajustado, ya que simplemente varía la forma de implementación y sin cambios importantes en los resultados. Además, se han fijado las mejores opciones escogidas en la sección previa, por lo que los hiperparámetros de los módulos anteriores no se van a cambiar.

A continuación, se estudiará la implicación de dos hiperparámetros estrechamente relacionados de este módulo, estos son la probabilidad de mutación y el valor máximo de mutación. La probabilidad de mutación se aplicará a cada vector de probabilidades en cada matriz de cada nodo de azar. Si supera la probabilidad establecida, se modificará un valor de dicho vector sumando o restando como máximo el valor de mutación establecido por el segundo hiperparámetro. Tras esto, se ajustarán el resto de los valores para que el vector siga siendo un vector de probabilidades.

Por tanto, si se aumenta la probabilidad y el valor máximo de mutación, aumentarán el número de mutaciones provocadas y la fuerza de dichas mutaciones respectivamente. Esto hará que se incremente la diversidad de soluciones en la población, lo que puede evitar óptimos locales. Por otro lado, esto también puede afectar negativamente en la población, ya que los individuos generados estarán más lejos del camino seguido por sus predecesores, lo que bajará el rendimiento general.

Para realizar esta prueba se han probado tres conjuntos de valores diferentes, y que se estudiarán para comprobar cual da mejores resultados. De primeras, se optará por tener solamente un 5% de probabilidad de mutación, y manteniendo la mutación máxima en un valor de 0,05. En la segunda prueba se aumentará la probabilidad de mutación a un 15%, haciéndolas que ocurran más frecuentemente, aunque se mantendrá el valor máximo de mutación. Por último, se volverá a aumentar la probabilidad de mutación a un 20%, y además el valor máximo también se aumentará dejándolo en un 0,15. Con estas tres pruebas se tendrá un rango de valores diferenciados y que permitirá elegir la que me dé mejores resultados.

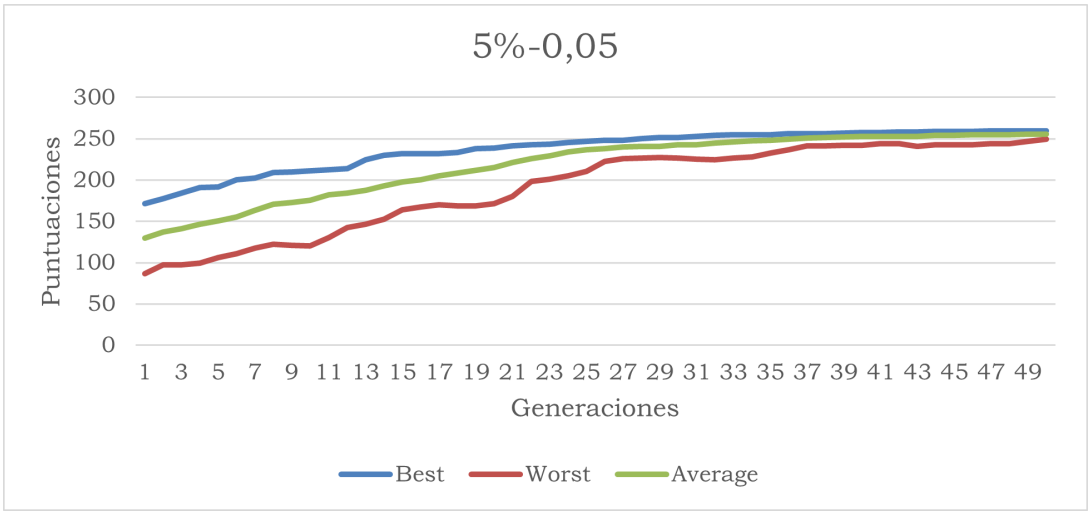

Figura 4.14: Gráfica con porcentaje de mutación de 5% y mutación máxima de 0.05

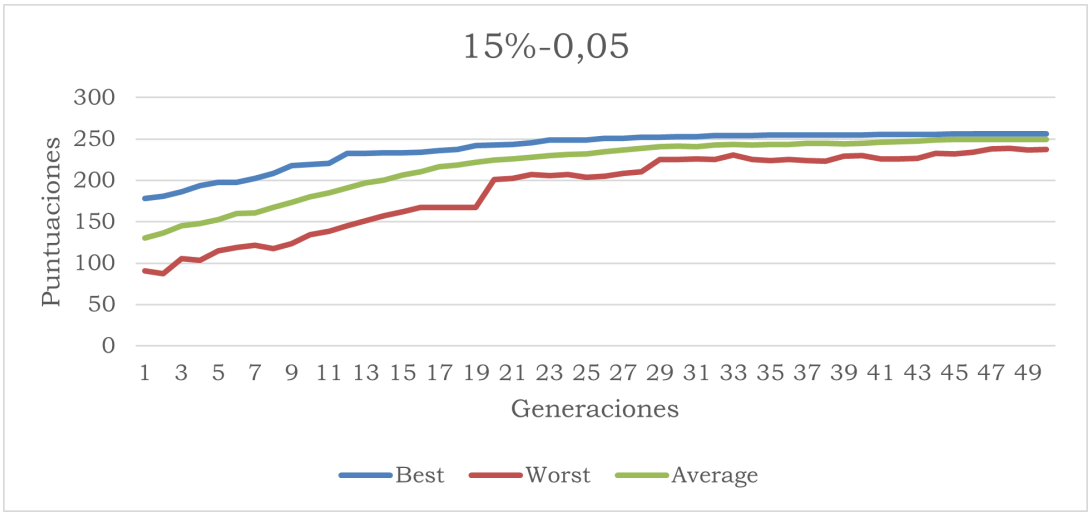

Figura 4.15: Gráfica con porcentaje de mutación de 15% y mutación máxima de 0.05

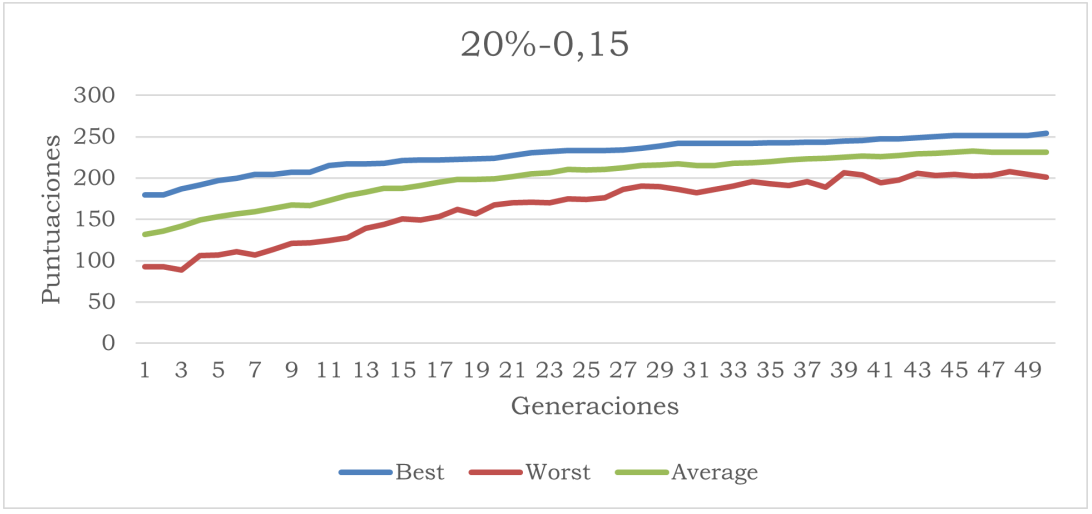

Figura 4.16: Gráfica con porcentaje de mutación de 20% y mutación máxima de 0.15

Como se puede ver en las figuras 4.14, 4.15, 4.16, a medida que se van aumentando los valores aumenta la diversidad, ya que se separan más las gráficas de las tres medidas. Esto provoca que las puntuaciones sean menores que en las otras dos gráficas, superando 250 de puntuación solamente en las últimas generaciones. Por otro lado, las otras dos pruebas alcanzan valores superiores, pero en las últimas generaciones los individuos son todos muy similares, haciendo que no haya mucha diferencia entre el mejor y el peor individuo.

Sin embargo, mientras que en las dos primeras pruebas 4.14 y 4.15 el resultado parece estancarse en una puntuación algo superior a 250, la figura 4.16 es más constante, pero parece que seguiría aumentando con un mayor número de generaciones, por lo que se generará otra prueba para comprobar esto.

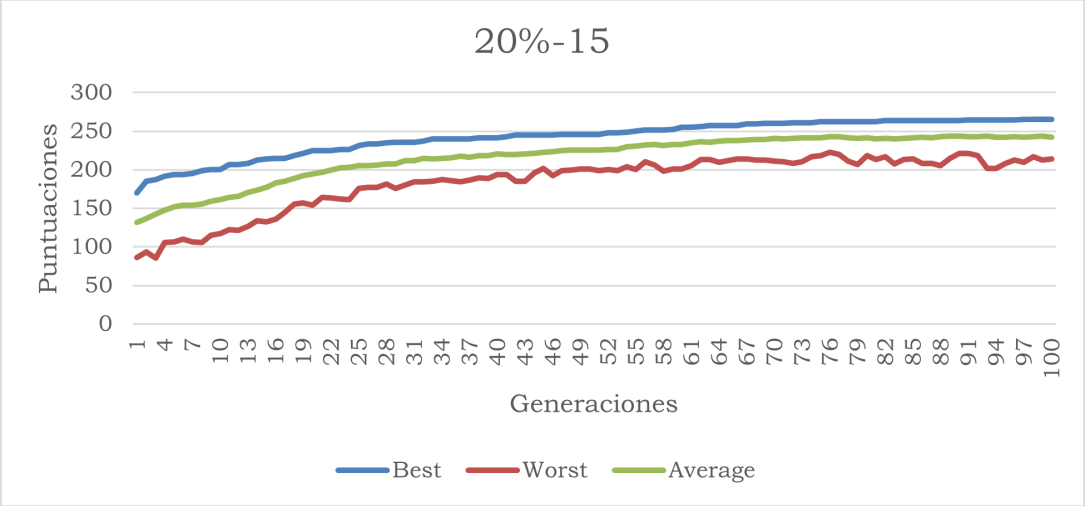

Figura 4.17: Gráfica con porcentaje de mutación de 20 % y mutación máxima de 0.15 extendido a 100 generaciones

En la prueba 4.17 se ha confirmado la hipótesis planteada, ya la puntuación ha ido avanzando al aumentar el número de generaciones a cien, llegando hasta los 265. Además, sigue habiendo mucha separación entre las tres métricas, lo que puede ser positivo para las siguientes pruebas, ya que permite aumentar aún más la puntuación y que no importe demasiado la disminución de diversidad al ser ya elevada. La parte negativa es el tiempo de ejecución, que ha aumentado considerablemente al poner a cien el número de generaciones.

4.2.4. Módulo de elección

En este módulo se decide que individuos serán parte de la siguiente generación. Este módulo se ha añadido ya que existen decisiones a tomar sobre los individuos que acompañarán a los nuevos individuos generados como parte de la siguiente generación.

4.2.4.1. Conjunto seleccionable

En este primer paso se decidirá de donde se seleccionarán los individuos que rellenarán la siguiente generación para que tenga el mismo número de individuos que las anteriores.

La primera opción y la utilizada durante todas las pruebas anteriores es la de seleccionarlos aleatoriamente del conjunto de individuos de la generación pasado. Esto permitirá darle otra oportunidad a los individuos que no han conseguido ganar un torneo, lo que mejoría la diversidad de las generaciones siguientes a costa de disminuir la calidad, ya que podrían entrar individuos con una puntuación muy baja.

La otra opción planteada es seleccionar a dichos individuos directamente entre los ganadores de los torneos. Esto haría aumentar la influencia de los mejores individuos, ya que estos son los que tienen más probabilidades de ganar un torneo, aumentando a priori la puntuación en las siguientes generaciones.

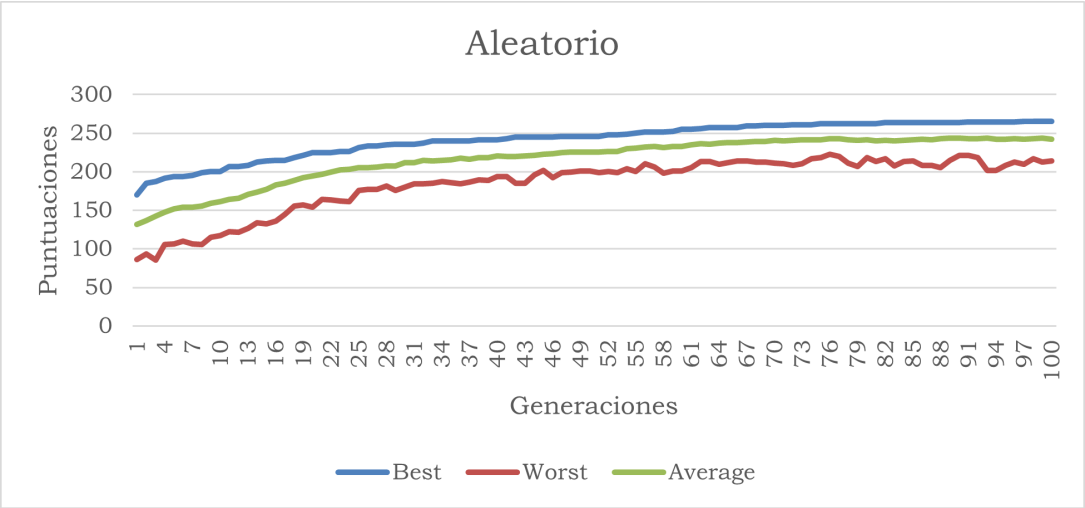

Figura 4.18: Gráfica en donde se seleccionan a los individuos del total de la población

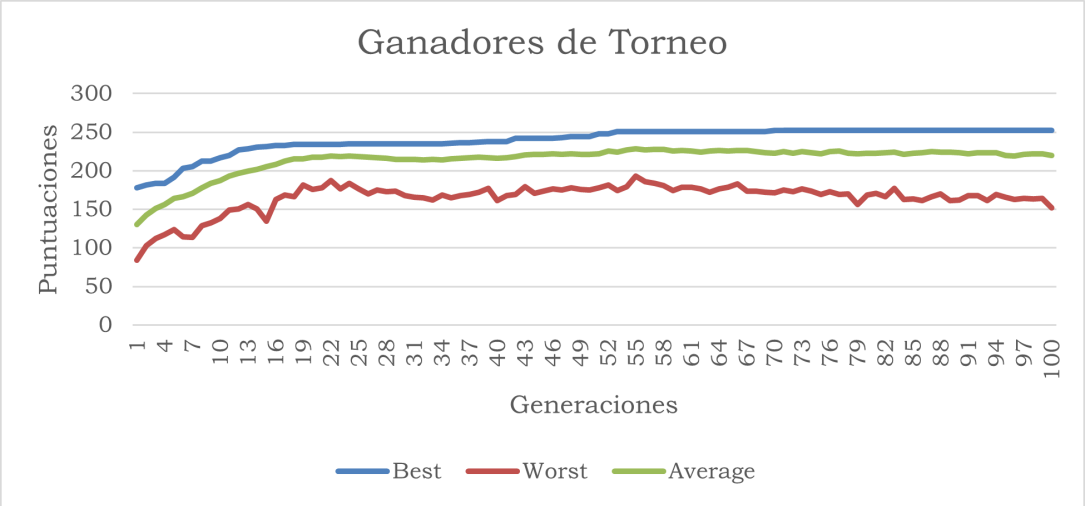

Figura 4.19: Gráfica en donde se seleccionan a los individuos entre los ganadores de los torneos

Tras realizar las pruebas se ve claramente en las figuras 4.18 y 4.19 como la mejor opción es dejar que se obtengan los individuos de la generación entera al tener 4.18 mejores resultados. Hay dos problemas a destacar en 4.19. El primero de ellos es el estancamiento temprano del mejor individuo de la generación, siendo el mismo puntuaje desde la generación quince hasta la cincuenta, y mejorando solamente en las siguientes diez generaciones para mantenerse casi sin cambios hasta la última generación. Por otro lado, en la primera prueba la gráfica se mantiene ascendente hasta las últimas generaciones, lo que significa que no se queda en ningún óptimo local. Por tanto, al restringir los individuos seleccionados a los ganadores del torneo lo que provoca es alcanzar un óptimo local muy rápidamente, lo que dificulta el progreso del algoritmo.

El otro punto que comentar es la caída de puntuación en la segunda gráfica de la métrica que muestra al peor individuo. Esta es la primera prueba en la que ha disminuido la puntuación en alguna métrica, lo que refleja el mal rendimiento de esta prueba. Esto puede deberse a que, como los individuos en una generación se vuelven muy similares, los cruces y las mutaciones no pueden conseguir individuos mejores, ya que se a alcanzado un óptimo local, por lo que es más probable que los individuos generados empeoren la calidad de la generación.

Debido a estas dos razones, se seguirá obteniendo los individuos del conjunto total. Sin embargo, esta prueba ha sido de utilidad, ya que ha mostrado las consecuencias de una concentración de individuos muy similares, lo que curiosamente también ocurre con el material genético biológico.

4.2.4.2. Número de individuos generados

A continuación, se volverá a una idea ya probada en módulos anteriores, ya que ahí no era el momento de probar con más detalle variar este parámetro, ya que se perdería la narrativa que se estaba siguiendo. Por tanto, en este apartado se probará a modificar el número de individuos generados por cada pareja de individuos formada tras ganar los torneos. Teóricamente, al aumentar el número de individuos gene-

rados, se ahondaría más en el material genético transmitido por sus padres, pero cruzándolo y mutándolo de manera diferente, lo que puede aumentar las probabilidades de conseguir individuos mejores como se vio en las pruebas anteriores. Cabe destacar que en la implementación realizada los individuos son cruzados de manera independientes, ya que existen implementaciones que agrupan el material genético de los padres en dos individuos, siendo estos completamente diferentes entre sí.

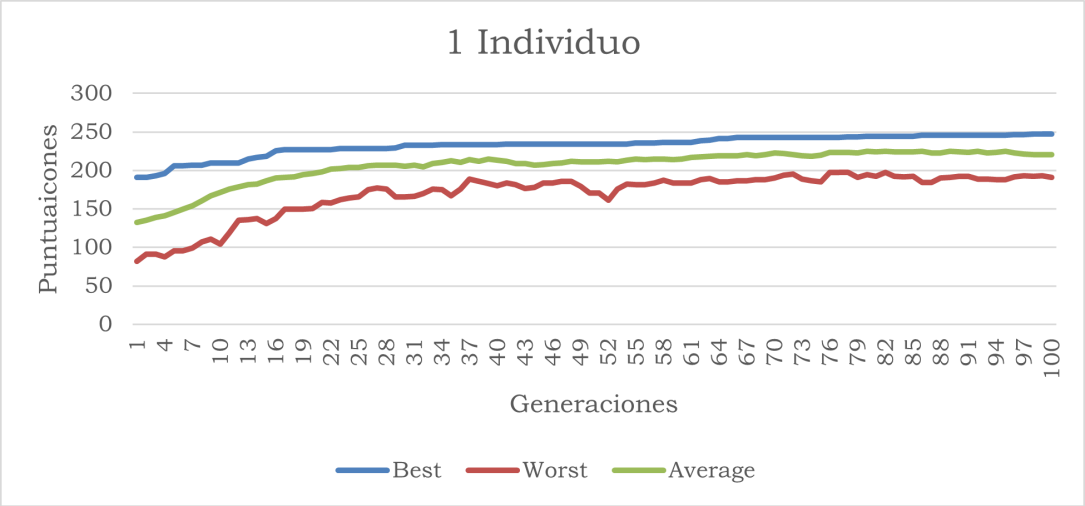

Figura 4.20: Gráfica en donde se genera un individuo por pareja

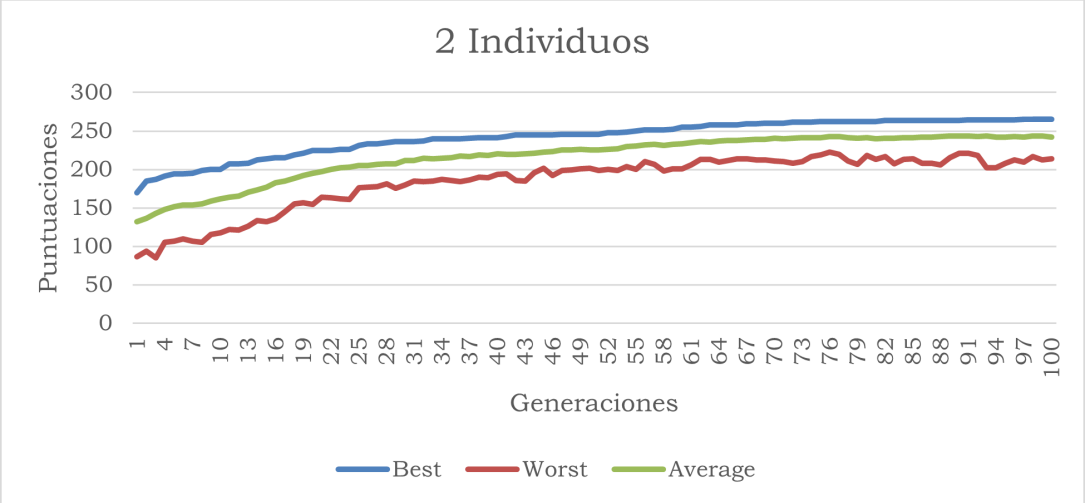

Figura 4.21: Gráfica en donde se generan 2 individuos por pareja

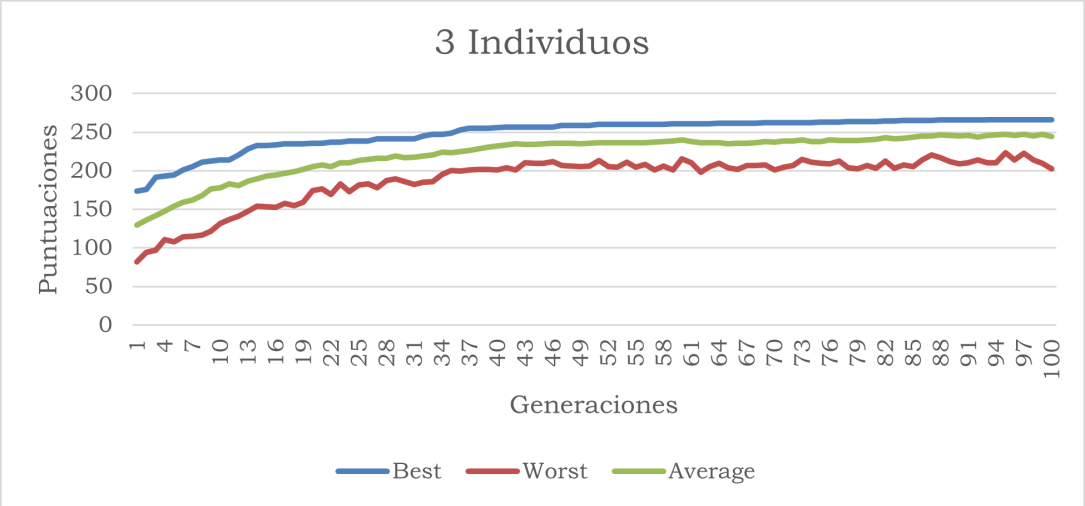

Figura 4.22: Gráfica en donde se generan 3 individuos por pareja

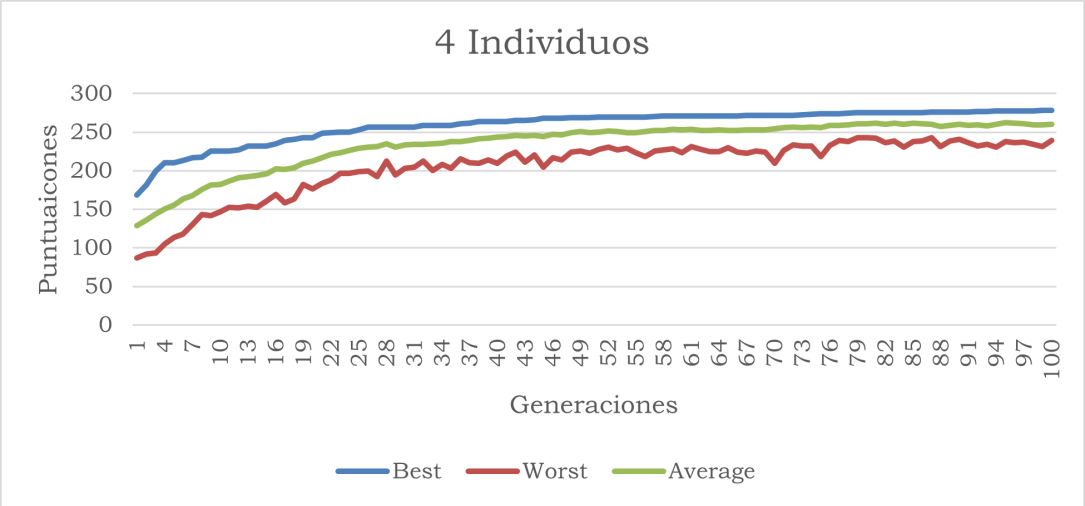

Figura 4.23: Gráfica en donde se generan 4 individuos por pareja

Como vemos en las gráficas 4.20, 4.21, 4.22, 4.23, a medida que aumenta el número de individuos generados por cada pareja, aumenta la puntuación máxima alcanzada, pero disminuye la diversidad, al acercarse cada vez más los resultados de las tres métricas. Sin embargo, los resultados obtenidos por la última prueba justifican esa pérdida de diversidad, ya que da resultados bastante favorables. Evidentemente, a mayor número de individuos generados mayor es el tiempo de ejecución, lo que puede decantar el elegir otra opción. Aun así, debido a que quedan solo unas pocas pruebas, se ha decido optar por generar cuatro individuos por pareja, aunque también se empleará la prueba en donde se generar tres individuos para realizar un conjunto de pruebas extra que no se puede realizar con la otra.

4.2.4.3. Adición de elitismo

Las pruebas anteriores se realizaron con elitismo. Esto quiere decir que los mejores individuos de cada generación se mantienen siempre a la siguiente generación, por

lo que el valor máximo de puntuación nunca puede bajar. En el caso de las pruebas realizadas se mantenía este hiperparámetro a dos, por lo que a continuación se probarán diversos valores que decidirán cuantos individuos son añadidos a la siguiente generación de forma obligatoria. El motivo de que no se pueda realizar esta prueba usando la gráfica con mayores resultados en la prueba anterior es debido a que si se generan cuatro individuos por pareja y se seleccionan a la mitad de los individuos, significa que solo con el cruce y la mutación la siguiente generación estaría completamente ocupada, por lo que no habría espacio para el elitismo. Por tanto, se comparará el valor obtenido por dicha prueba con tres valores de elitismo diferentes, lo que decidirá cual es el mejor modelo obtenido.

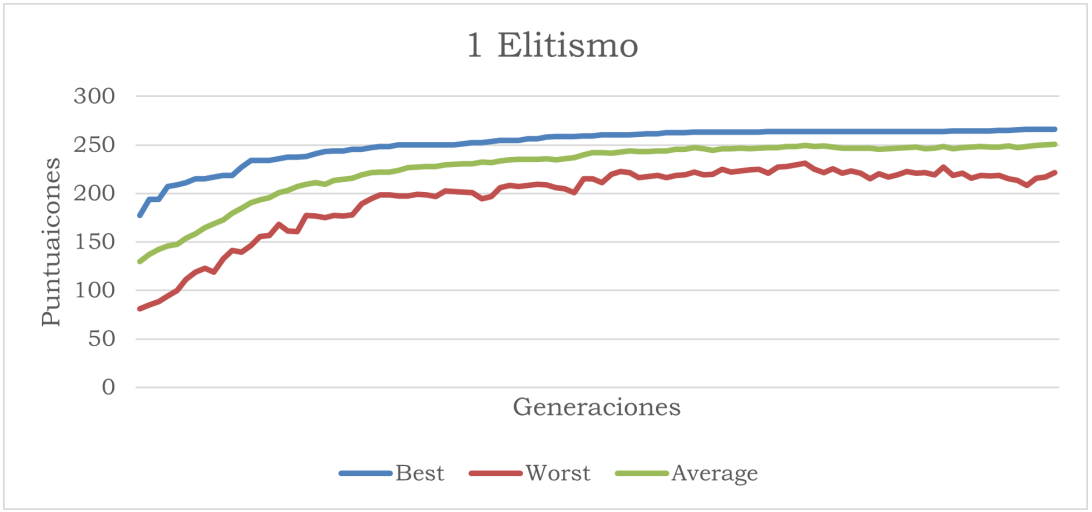

Figura 4.24: Gráfica en donde el mejor individuo pasa directamente a la siguiente generación

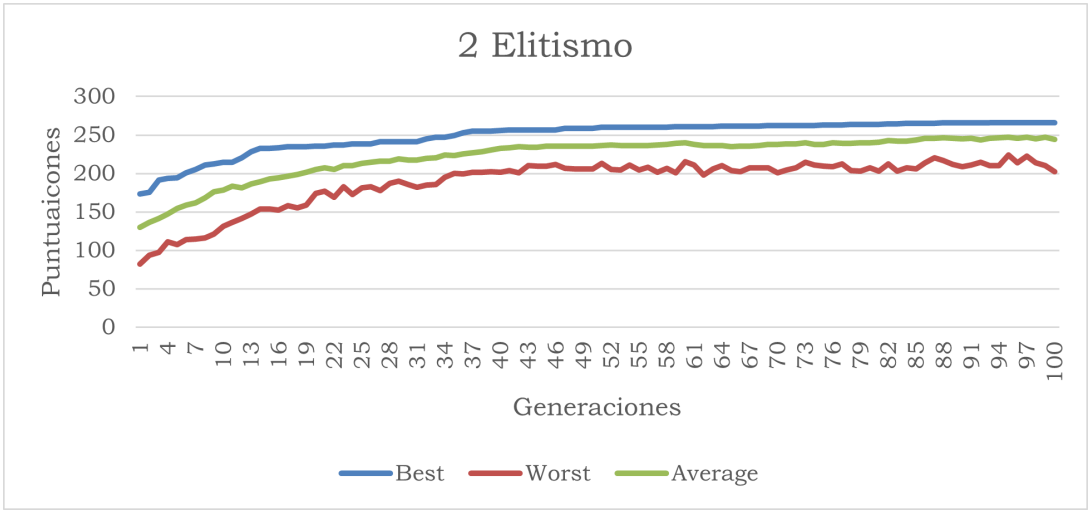

Figura 4.25: Gráfica en donde los 2 mejores individuos pasa directamente a la siguiente generación

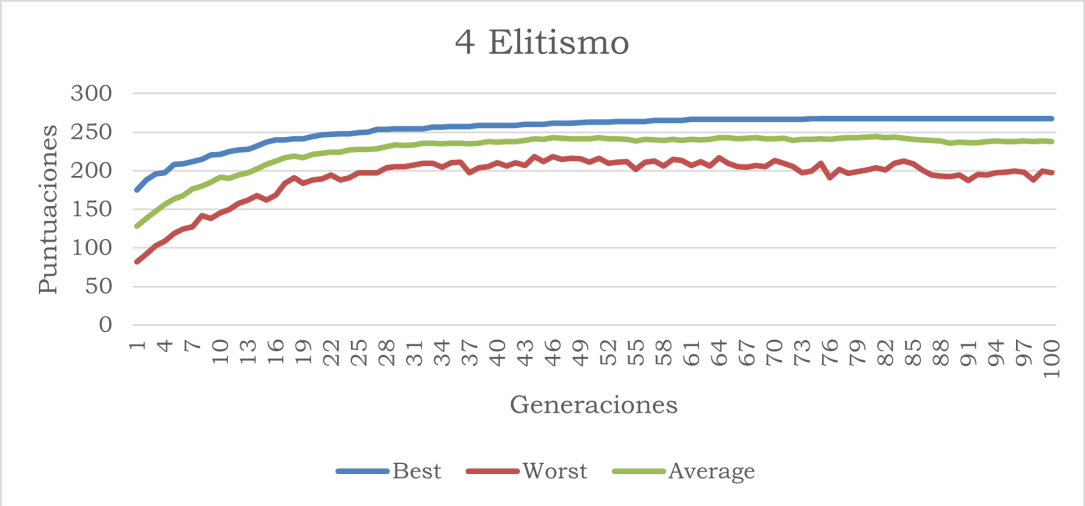

Figura 4.26: Gráfica en donde los 4 mejores individuos pasa directamente a la siguiente generación

Como se puede ver en las figuras 4.24, 4.25, 4.26, a medida que aumenta el elitismo en las pruebas, va empeorando las métricas del peor individuo y de la media. Esto puede deberse nuevamente a una alta concentración de individuos muy parecidos, lo que provoca que las mutaciones que provoquen empeoren el resultado al igual que pasaba en pruebas anteriores. Sin embargo, no se aprecia una mejora considerable en el mejor individuo de cada generación, lo que tiene sentido al presentar las tres pruebas al menos un individuo de elitismo.

Sin embargo, comparando estas tres pruebas con la prueba del apartado anterior 4.23, que consistía en generar cuatro individuos por cada pareja, se aprecia como esta sigue dando mejores resultados, e incluso la curva de la gráfica no parece estancada completamente, lo que podría dar indicios de que hubiera una mejora si se dejara durante más generaciones.

4.3. Etapa de generalización y estudio del resultado

Una vez que se han ajustado todos los hiperparámetros del algoritmo genético se estudiará el resultado usando el conjunto total de reglas. Como se ha comentado anteriormente, el entrenamiento del algoritmo genético se ha realizado usando un pequeño conjunto del número total de reglas. Esto es debido a que se quiere reducir la carga del experto, agilizando el desarrollo. Sin embargo, en las pruebas anteriores la puntuación se ha realizado usando únicamente las reglas que proporcionaría el experto, pero es interesante ver la puntuación usando como referencia el conjunto total de reglas.

Para ello, se hará una última prueba de la que saldrá el modelo final de este trabajo. En dicha prueba se comprobará la eficiencia del algoritmo prediciendo las reglas totales variando el porcentaje de reglas que se le otorgan. En principio, contra más reglas se le distribuya más puntuaje debería obtener, ya que tiene una mejor idea del conjunto de reglas total. Por otro lado, a mayor número de reglas otorgadas, más trabajo del experto es necesario, por lo que hay que tenerlo en cuenta para comparar el resultado.

Además, hay que destacar que el porcentaje de reglas se obtiene de manera aleatoria, por lo que los resultados variarían con la ayuda de un experto real, ya que este daría las reglas que más importancia tuvieran en el conjunto de reglas total. Además, lo mejor sería proceder de modo iterativo, aportando las reglas y posiblemente su importancia relativa en varias fases. Tal vez, en un ejemplo real el experto deba valorar las reglas que propone el AG en conjunto, tanto las que verifica como adecuadas (escala de ajuste) como las que identifica como error (escala de desajuste).

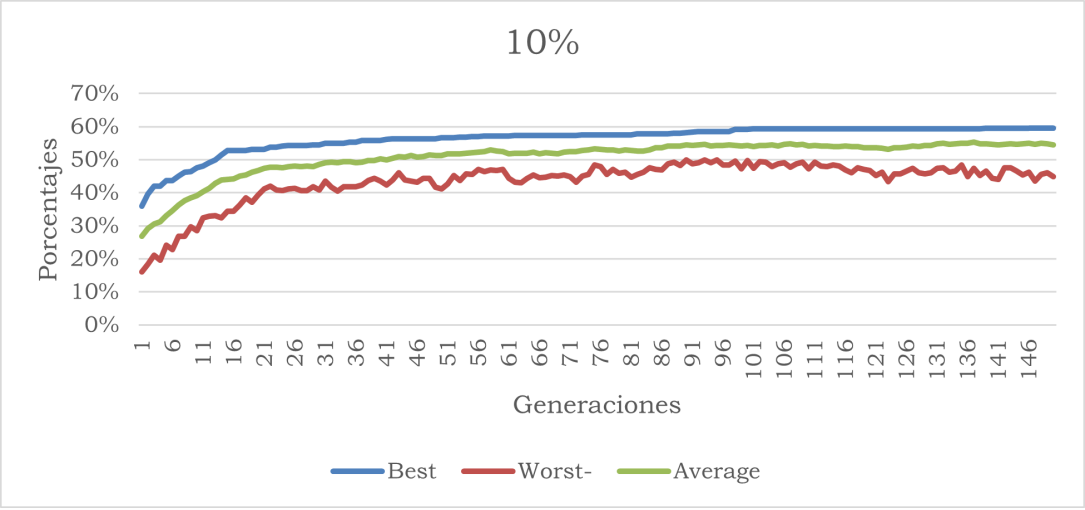

Figura 4.27: Gráfica porcentual en donde se usa el 10 % de la reglas para entrenar el modelo

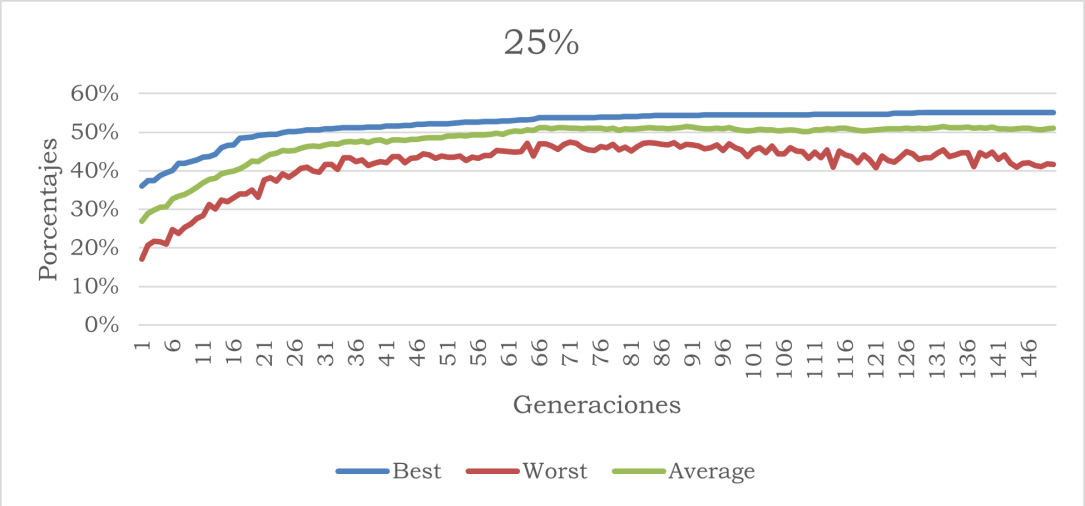

Figura 4.28: Gráfica porcentual en donde se usa el 25 % de la reglas para entrenar el modelo

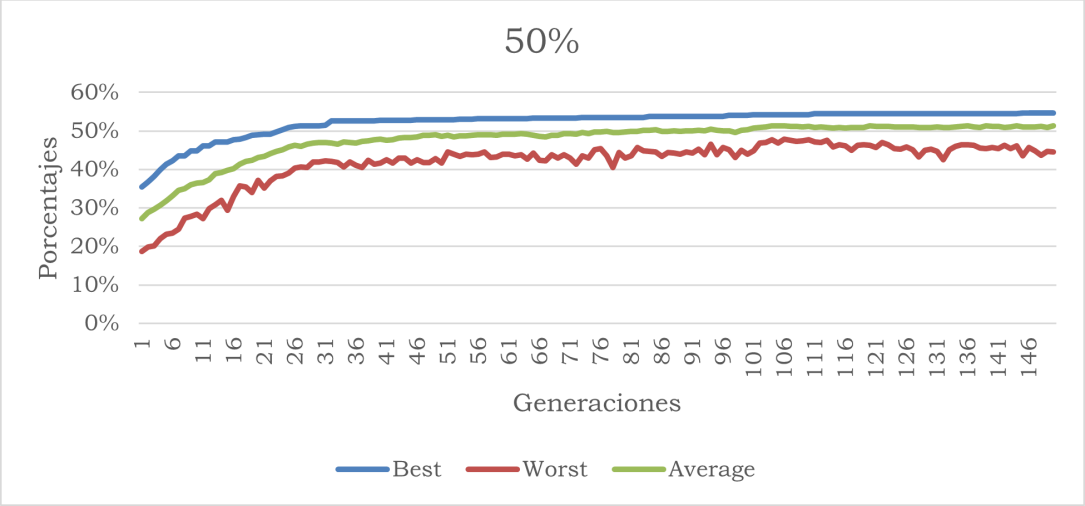

Figura 4.29: Gráfica porcentual en donde se usa el 50 % de la reglas para entrenar el modelo

En las gráficas 4.27, 4.28, 4.29 vemos el porcentaje de reglas conseguidas sobre el total de reglas aportado en cada prueba, usando un 10 %, 25 % y 50 % de las reglas totales respectivamente. Con esto podremos comprobar la importancia del número de reglas a la hora de ponderar los valores para poder predecirlas correctamente. Sin embargo, esta prueba tiene interés únicamente como medida de rendimiento teórico, ya que en general no se conoce la BC completa.

Lo primero a destacar es la caída de la gráfica roja en las tres pruebas al pasar de las cien generaciones, apreciándose más en la gráfica del medio. Esto es debido al sobreentrenamiento, que provoca que los individuos sean muy parecidos entre sí y hace que las mutaciones y los cruces solo puedan generar individuos peores.

Por otro lado, los resultados obtenidos por las tres pruebas son parecidos, consiguiendo todas un resultado porcentual entre el 60 % y 55 % al final del experimento.

Sin embargo, la primera prueba es la que consigue un mejor resultado que las otras, obteniendo el 60 % en las últimas generaciones. Este resultado nos indicaría que el algoritmo genético funciona mejor contra menor número de datos se utilicen, aunque no presenta un gran cambio entre la gráfica del 25 % y la del 50 %. Por tanto, el AG aprende mejor con pocas reglas, aunque sería menos robusto si el experto aporta reglas en conflicto o reglas redundantes. Sin embargo, esto no se ha probado, por lo que puede ser un tema de investigación a futuro.

4.3.1. Estudio sobre el total de reglas

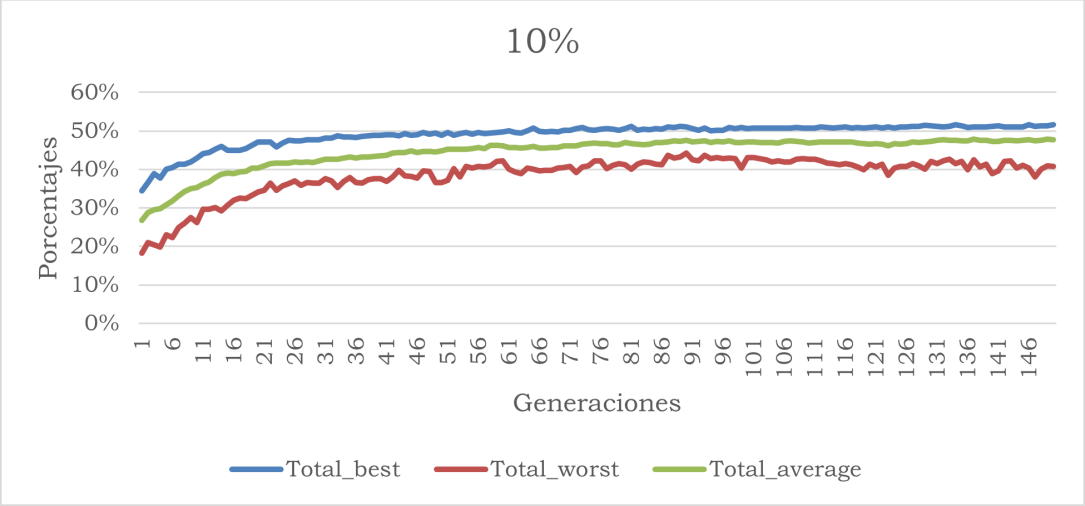

Figura 4.30: Gráfica porcentual sobre el total de reglas en donde se usa el 10 % de la reglas para entrenar el modelo

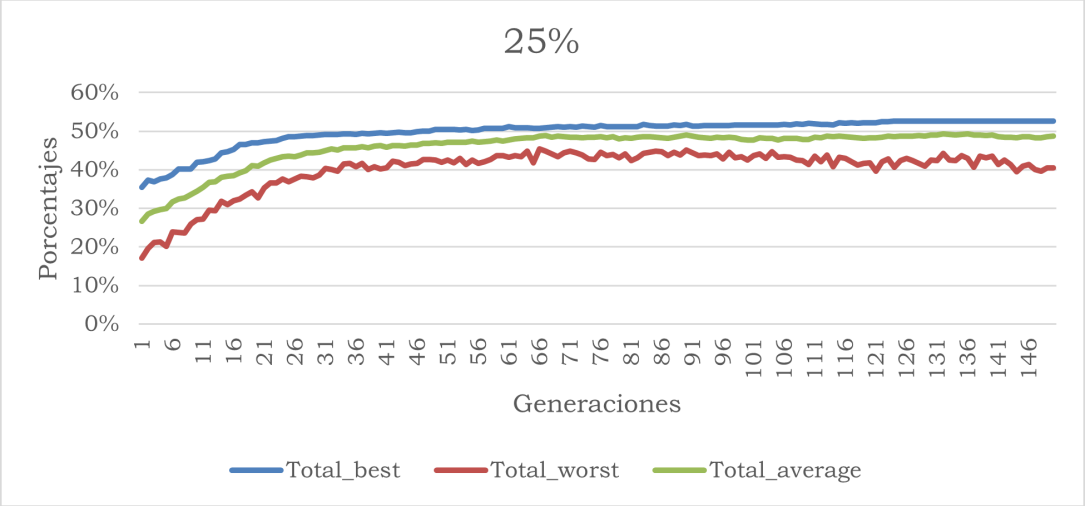

Figura 4.31: Gráfica porcentual sobre el total de reglas en donde se usa el 25 % de la reglas para entrenar el modelo

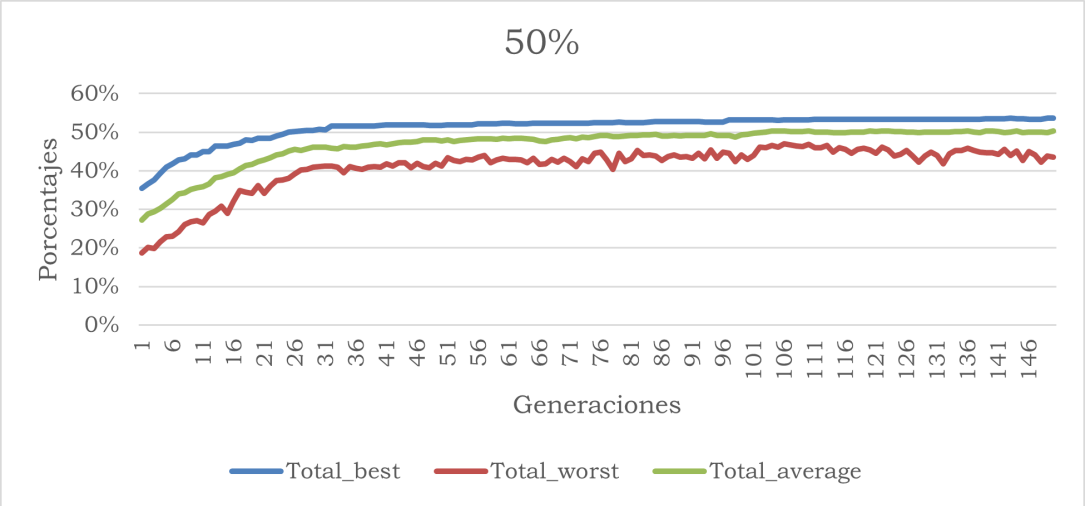

Figura 4.32: Gráfica porcentual sobre el total de reglas en donde se usa el 50 % de la reglas para entrenar el modelo

Estas tres figuras 4.30, 4.31, 4.32 son de las mismas pruebas que las anteriores, pero esta vez se muestra el porcentaje de reglas del total que se logra alcanzar. Con estas gráficas lo que se busca es ver de que forma se adaptan las reglas usadas en el entrenamiento al total de reglas, y ver cuanto sería el mejor porcentaje de reglas necesarias para predecir el total.

Como es lógico cuanto más porcentaje de reglas se use, más parecido será el resultado usando el total al obtenido anteriormente. Estas diferencias son muy pronunciadas, ya que la diferencia de puntos porcentuales más grande en la prueba del 50 % es únicamente un 1 %, mientras que en la prueba del 25 % la diferencia máxima está en un 3 % y en el 10 % esta diferencia se dispara, pudiendo alcanzar un 9 %. Además, en estas pruebas la puntuación máxima obtenida es justo al contrario que la anterior, pero con valores más bajo, donde la tercera gráfica se mantiene en los 55 %, la segunda baja a los 53 % y la tercera pasa del 60 % a un 51

Por tanto, el escoger el porcentaje de reglas a usar en el entrenamiento es una decisión importante a tomar con el experto, ya que dependerá de la importancia de la reglas dadas, donde se tendrá que diferenciar entre reglas genéricas o redundantes, reglas que recojan excepciones o reglas muy relevantes para resultado. Nuevamente, hay más variables en juego, como el usar las reglas más determinantes en las decisiones, que indudablemente mejorará la puntuación obtenida. Además, del tamaño del diagrama de influencia también influirá, ya que, como se ha visto, los algoritmos genéticos funcionan mejor en cantidades pequeñas de reglas, aunque esto podría cambiar con otra metaheurística.

Capítulo 5

Resultados y conclusiones

5.1. Evaluación de los objetivos del trabajo

En este TFM se ha probado el uso de metaheurísticas para ponderar los valores de utilidad y probabilidad de los diagramas de influencia. Para ello primero se ha modelizado los diagramas de influencia en un entorno de programación y se ha desarrollado el algoritmo de evaluación que permite obtener un conjunto de reglas del DI. Además, se implementó un algoritmo genético que permita usar los valores de probabilidad y utilidad como material genético para ir modificándolo a lo largo de una serie de generaciones para obtener un conjunto de reglas similar al otorgado por el experto.

EL objetivo final es comprobar la utilidad de las metaheurísticas, en este caso un algoritmo genético, para ponderar los valores del DI y reducir la dependencia del experto, al poder usar únicamente un reducido número de reglas para predecir el conjunto total de reglas. Con la implementación realizada, este proceso puede usar en cualquier DI de manera rápida y efectiva, solamente es necesario introducir las reglas en un fichero de texto y representar la estructura del DI de forma que pueda reconocerlo el programa. Esto se ha hecho en una interacción con el experto, que en este trabajo ha sido emplear el modelo NHL, ya que al tener todos sus parámetros estimados se ha podido extraer el conjunto de reglas.

Los resultados obtenidos han sido buenos, pudiendo predecir algo más de la mitad de las reglas usando un porcentaje bajo de estas. La inteligencia artificial desarrollada es capaz de aprender de ese conjunto de reglas y extenderlo al total. Aun así, seguramente los resultados obtenidos pueden ser mejorados, ya que, como se comentó en el apartado de líneas futuras, hay multitud de maneras de ampliar el trabajo realizado. A parte de eso, puede mejorarse el propio programa permitiendo el uso de variables continuas, lo que aumentaría los casos de uso en los que sería de utilidad, e incluso se podría añadir una interfaz gráfica que mejore la forma en la que se introducen los DI y las reglas proporcionadas por el experto.

Como conclusión personal, este proyecto ha sido una gran experiencia ya que he podido mezclar dos campos de la inteligencia artificial sumamente importantes, pudiendo abrir una nueva vía para mejorar los sistemas de ayuda a la decisión. Por otro lado, gracias al entrenamiento del algoritmo genético he podido ver la dificultad de ajustar correctamente una inteligencia artificial, lo que me servirá en el trabajo

una vez acabado este máster. Además, he obtenido experiencia al realizar un trabajo multidisciplinar, ya que se se han realizado tareas en ámbitos como la modelización del problema, la adquisición de conocimiento y posteriormente su representación, la inferencia probabilística, en optimización y en toma de decisiones.

5.2. Líneas futuras de investigación

En este trabajo se ha mostrado como se puede implementar metaheurísticas en la toma de decisiones usando los diagramas de influencia como base. Sin embargo, hay muchas maneras diferentes de ampliar, optimizar y mejorar los resultados obtenidos en este trabajo. Por tanto, se comentará diferentes aproximaciones que se pueden tomar para mejorar esto.

5.2.1. Optimización del tiempo de ejecución

El primer punto importante para tratar es el tiempo de ejecución. Al tener los diagramas de influencia una gran cantidad de parámetros en las matrices de probabilidad y utilidad condicionado, resolver el diagrama puede ser costoso, lo que ralentiza la realización de pruebas y obstruye el obtener mejores resultados.

Hay varias maneras de mejorar esto. La primera es la de mejorar el hardware en donde se realiza la simulación de la metaheurística, pero también se puede intentar usar la GPU para realizar todos los cálculos matriciales, que se toma una gran parte del tiempo total. En este trabajo se ha empleado una técnica de paralelización, que consiste en evaluar a todos los individuos de un algoritmo genético simultáneamente, lo que reducía enormemente el tiempo de desarrollo. Sin embargo, conseguir implementar esta paralelización usando una GPU muy potente posiblemente volvería a reducir el tiempo de cómputo. Otra forma es el de crear estructuras [17] que permitan procesar de forma más óptima los DI, reduciendo así su tiempo de cómputo.

5.2.1.1. Instanciación de variables

Existe otra manera de reducir el tiempo de ejecución, sin embargo, no se estaría ponderando los valores de todos los nodos de azar y decisión. Este método consiste en rellenar una matriz entera con valores reales, evitando tener que ponderar sus datos y permitiendo al algoritmo centrarse en el resto de los nodos. Sin embargo, es necesario preguntarle al experto dichos valores, lo que puede dificultar la realización de este método. Por tanto, la utilidad de este método depende del tema en cuestión como del experto del que se requiera los servicios.

5.2.1.2. Parametrización de las matrices de probabilidad y utilidad

Otro método con el que se reduciría enormemente el tiempo de ejecución es el de parametrizar las matrices al principio, sustituyendo sus valores por variables, y solo realizando el algoritmo de evaluación una única vez. Para poder conseguir esto, se tendría que remplazar los valores de todos los nodos por variables, para que posteriormente para obtener la puntuación de un individuo solamente habría que remplazar las variables por la información genética del algoritmo. Para poder realizar esto, habría que utilizar algún lenguaje de programación específico que permita parametrizar matrices con un gran número de variables.

Por ejemplo, para parametrizar una matriz de dos dimensiones que represente la utilidad de un DI, si cada variables aleatoria que sea predecesor del nodo tiene dos posibles valores, harían falta cuatro variables distintas para parametrizar la matriz, donde cada variable representa una combinación de cada valor posible de los nodos aleatorias.

5.2.2. Estudio de la importancia de las reglas

Un apartado importante de este proyecto es el número y la calidad de las reglas dadas por el experto. Se podría estudiar si hubiera alguna forma de medir la importancia de una regla, ya que el experto puede diseñar una regla general, que abarque una buena parte de las opciones posibles en una decisión o que se detalle una regla que actúe más como una excepción y que solo sea importante para unos casos determinados. Por otro lado, habría que ver si, con varias ejecuciones, se aprenden reglas diferentes y si estas son conflictivas o contradictorias entre sí. Además, también podrían haber reglas redundantes, por lo que habría que estudiar si dichas reglas se aprenden siempre aunque no aporten a la BC.

5.2.3. Implementación de diferentes metaheurísticas

En este proyecto se ha empleado un algoritmo genético como método para ponderar los valores de las matrices de los nodos. Sin embargo, existen multitud de metaheurísticas diferentes que realicen un mejor trabajo adaptado a este problema en concreto. Se podría encontrar una mejor metaheurística analizando la forma de proceder de los datos y adaptando el proceso de evaluación para determinar una forma óptima. Incluso se podría adoptar un enfoque híbrido, que combinara varias metaheurísticas para adaptarse mejor a los DI.

Otra posibilidad es la de usar el paradigma de aprendizaje por refuerzo [18], que es un área del aprendizaje automático donde un agente aprende a tomar decisiones mediante ensayo y error para maximizar una recompensa acumulada a lo largo del tiempo. Cabe destacar que en este proyecto se aprende un modelo del problema de decisión, interpretable y explicable, a diferencia de otros esquemas como el Deep Learning, cuyos pesos no tienen semántica o interpretación.

5.2.4. Otros problemas reales

El trabajo realizado en este proyecto se ha realizado usando un DI para el diagnóstico y tratamiento del linfoma gástrico NHL [10, 11]. Sin embargo, esto se puede aplicar a cualquier otro modelo que sea un DI, como por ejemplo IctNeo [19, 20].

Bibliografía

[1] D. Koller y N. Friedman, Probabilistic Graphical Models: Principles and Techniques. MIT Press, 2009.

[2] R. A. Howard and J. E. Matheson, “Influence Diagrams,” Decision Analysis, vol. 2, no. 3, pp. 127–143, septiembre de 2005

[3] R. D. Shachter, “Evaluating influence diagrams”, Operations Res., vol. 34, n.º 6, pp. 871–882, diciembre de 1986

[4] S. Insua, A. Jiménez-Martín, y A. Mateos, “Los Sistemas de Ayuda a la Decisión,” Anales de la Real Academia de Doctores de España, vol. 9, pp. 161-177, 2005.

[5] C. Bielza Lozoya, S. Rios Insua y A. Mateos Caballero, "Fundamentos de los sistemas de ayuda a la decisión". Madrid, España: RaMa, 2002.

[6] R. A. Howard y A. E. Abbas, Foundations of Decision Analysis. Prentice Hall, 2008.

[7] J. H. Holland, Adaptation in Natural and Artificial Systems. MIT Press, 1992.

[8] K. P. Murphy, Machine learning: A probabilistic perspective. Cambridge, MA: MIT Press, 2012.

[9] “Influence Diagrams – BayesFusion”. BayesFusion. Disponible: https://www.bayesfusion.com/influence-diagrams

[10] H. Boot, B. G. Taal y P. J. F. Lucas, “Computer-based decision support in the management of primary gastric non-hodgkin lymphoma”, Methods Inf. Medicine, vol. 37, n.º 03, pp. 206–219, julio de 1998

[11] C. Bielza, J. A. Fernández del Pozo y P. J. F. Lucas, “Explaining clinical decisions by extracting regularity patterns”, Decis. Support Syst., vol. 44, n.º 2, pp. 397–408, enero de 2008.

[12] C. Bielza, J. A. Fernández del Pozo y P. Larrañaga, “Parameter Control of Genetic Algorithms by Learning and Simulation of Bayesian Networks — A Case Study for the Optimal Ordering of Tables”, J. Comput. Sci. Technol., vol. 28, n.º 4, pp. 720–731, julio de 2013.

[13] C. Bielza, J. A. Fernández del Pozo, P. Larrañaga y E. Bengoetxea, “Multidimensional statistical analysis of the parameterization of a genetic algorithm for the optimal ordering of tables”, Expert Syst. with Appl., vol. 37, n.º 1, pp. 804–815, enero de 2010.

[14] F. J. Díez, “Parameter adjustment in Bayes networks. The generalized noisy OR–gate”, en Uncertainty in Artificial Intelligence. Elsevier, 1993, pp. 99–105.

[15] P. Larrañaga, C. Kuijpers, R. Murga, et al., "Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators,.ªrtificial Intelligence Review, vol. 13, pp. 129–170, 1999.

[16] L. Pedro y L. J. A. 1968-, Eds., Estimation of distribution algorithms: A new tool for evolutionary computation. Boston: Kluwer Acad. Publ., 2002.

[17] C. Pralet, T. Schiex, and G. Verfaillie, “From influence diagrams to multi-operator cluster DAGs,” arXiv.org, Jun. 27, 2012.

[18] R. S. Sutton y A. G. Barto, Reinforcement Learning: An Introduction. MIT Press, 1998.

[19] M. Gómez, C. Bielza, J. A. Fernández del Pozo y S. Ríos-Insua, “A graphical decision-theoretic model for neonatal jaundice”, Med. Decis. Making, vol. 27, n.º 3, pp. 250–265, mayo de 2007.

[20] Gómez M. ÏctNeo: A decision support system for newborn jaundice management"[dissertation, in Spanish]. Madrid (Spain): Department of Artificial Intelligence, Technical University of Madrid; 2002.

Anexo

.1. Tabla con los resultados de la prueba

Cuadro 1: Resultados de la prueba

	Best	**Worst**	**Average**
1	5.33	0.67	3.01
2	5.67	0.67	3.28
3	5.67	0.67	3.43
4	6.00	1.00	3.39
5	6.00	1.00	3.47
6	6.00	1.00	3.47
7	6.00	1.00	3.63
8	6.00	1.33	3.65
9	6.00	1.00	3.59
10	6.00	1.00	3.91
11	6.00	1.33	4.25
12	6.00	1.33	4.35
13	6.00	1.33	4.53
14	6.00	1.33	4.51
15	6.00	1.33	4.56
16	6.00	1.33	4.65
17	6.00	1.00	4.83
18	6.00	2.33	4.95
19	6.00	2.33	5.03
20	6.00	2.67	5.17
21	6.00	3.33	5.25
22	6.00	3.33	5.40
23	6.00	3.33	5.43
24	6.00	3.33	5.43
25	6.00	3.33	5.48
26	6.00	3.33	5.46
27	6.00	3.33	5.55
28	6.00	3.33	5.63
29	6.00	3.67	5.64
30	6.00	4.67	5.69
31	6.00	4.67	5.71
32	6.00	4.67	5.79
33	6.00	4.67	5.78

	Best	Worst	Average
34	6.00	4.67	5.81
35	6.00	5.00	5.89
36	6.00	4.67	5.88
37	6.00	4.67	5.89
38	6.00	5.00	5.94
39	6.00	5.67	5.93
40	6.00	5.33	5.93
41	6.00	5.33	5.91
42	6.00	5.33	5.93
43	6.00	5.33	5.92
44	6.00	5.33	5.93
45	6.00	5.33	5.93
46	6.00	5.33	5.92
47	6.00	5.33	5.91
48	6.00	5.33	5.87
49	6.00	5.00	5.91
50	6.00	5.33	5.91
51	6.00	5.33	5.95
52	6.00	5.67	5.95
53	6.00	5.67	5.96
54	6.00	5.67	5.96
55	6.00	5.33	5.95
56	6.00	5.33	5.95
57	6.00	5.67	5.96
58	6.00	5.00	5.93
59	6.00	5.33	5.93
60	6.00	5.33	5.93
61	6.00	5.00	5.91
62	6.00	5.00	5.94
63	6.00	4.67	5.95
64	6.00	5.00	5.93
65	6.00	5.00	5.95
66	6.00	5.67	5.96
67	6.00	5.33	5.95
68	6.00	5.67	5.97
69	6.00	5.33	5.97
70	6.00	5.67	5.97
71	6.00	5.67	5.97
72	6.00	5.00	5.98
73	6.00	5.67	5.99
74	6.00	5.33	5.99
75	6.00	6.00	6.00
76	6.00	5.33	5.99
77	6.00	5.67	5.98
78	6.00	5.33	5.97
79	6.00	5.00	5.97
80	6.00	5.00	5.97
81	6.00	5.67	5.99

	Best	Worst	Average
82	6.00	4.67	5.96
83	6.00	5.00	5.97
84	6.00	5.33	5.98
85	6.00	5.33	5.97
86	6.00	5.00	5.97
87	6.00	4.33	5.91
88	6.00	4.33	5.89
89	6.00	4.67	5.86
90	6.00	4.67	5.89
91	6.00	4.67	5.89
92	6.00	5.33	5.97
93	6.00	5.33	5.98
94	6.00	5.00	5.96
95	6.00	5.00	5.94
96	6.00	5.00	5.95
97	6.00	5.00	5.97
98	6.00	5.33	5.97
99	6.00	5.33	5.95
100	6.00	5.00	5.91

.2. Función de evaluación de DI

```
from influence_diagram import *
import copy
import numpy as np

def diagram_analysis(diagram):
    # Add memory arcs
    add_memory_arcs(diagram)

    # Find if there is a path from the utility node to all decision nodes
    if not search_all_decision_node(diagram):
        return None

    # Eliminar sumideros
    eliminate_sinks(diagram)

    while len(diagram.node_utility.predecessors) > 0:
        # Find if there is a chance node that can be eliminated
        if not eliminate_chance_node(diagram):
            # Find if there is a decision node that can be eliminated
            if not delete_decision_node(diagram):
                for node1 in diagram.nodes.values():
                    # Find if there is a chance node and it does not have a decision node as a
                        successor
                    if node1.type == "CHANCE" and not any([node2.type == "DECISION" for node2
                        in node1.sucessors(diagram.nodes)]):
                        # Verify that it does not have more successors than the utility node
                        while node1.sucessors(diagram.nodes).__len__() > 1:
                            for node2 in node1.sucessors(diagram.nodes):
                                if invertir_nodo(diagram, node1, node2):
                                    break
                        eliminate_chance_node(diagram)
                        break

    return diagram

# ------------------ Check conditions ------------------
```

```
35 # Add memory arcs
36 def add_memory_arcs(diagram):
37     update_node = True
38     while update_node:
39         update_node = False
40         for node in diagram.nodes.values():
41             if node.type == "DECISION":
42                 search_decision_node(diagram, node, node)
43     return diagram
44 
45 # Add memory arcs aux
46 def search_decision_node(diagram, decision_node, node):
47     for predecessor in node.predecessors:
48         # If it is a decision node, add that node and its predecessors to the search
               predecessors list and continue searching
49         if diagram.nodes[predecessor].type == "DECISION":
50             if predecessor not in decision_node.predecessors:
51                 decision_node.predecessors.append(predecessor)
52             for predecessor2 in diagram.nodes[predecessor].predecessors:
53                 if predecessor2 not in decision_node.predecessors:
54                     decision_node.predecessors.append(predecessor2)
55                 search_decision_node(diagram, decision_node, diagram.nodes[predecessor2])
56         else:
57             search_decision_node(diagram, decision_node, diagram.nodes[predecessor])
58     return None
59 
60 # Find if there is a path from the utility node to all decision nodes
61 def search_all_decision_node(diagram):
62     decision_total = [node for node in diagram.nodes.values() if node.type == "DECISION"]
63     decision_list = []
64     search_all_decision_node_aux(diagram,decision_list,[],diagram.node_utility)
65     if decision_list.__len__() == decision_total.__len__():
66         return True
67     else:
68         return False
69 
70 # Search all decision nodes aux
71 def search_all_decision_node_aux(diagram,decision_list,passed_nodes,node):
72     if node not in passed_nodes:
73         passed_nodes.append(node)
74         if node.type == "DECISION":
75             decision_list.append(node)
76         for predecessor in node.predecessors:
77             search_all_decision_node_aux(diagram,decision_list,passed_nodes,diagram.nodes[
                   predecessor])
78     return decision_list
79 
80 # Eliminate sinks
81 def eliminate_sinks(diagram):
82     global_sink = True
83     while global_sink:
84         global_sink = False
85         nodes_copy = diagram.nodes.copy()
86         for node in nodes_copy.values():
87             if node.type == "UTILITY":
88                 continue
89             sucessors = node.sucessors(nodes_copy)
90             if sucessors.__len__() == 0:
91                 global_sink = True
92                 del diagram.nodes[node.name]
93                 #print("Eliminando nodo sumidero ", node.name)
94     return diagram
95 
96 # ------------------ Eliminate nodes ------------------
97 # Eliminate decision node
98 def delete_decision_node(diagram):
99     for predecessor in diagram.node_utility.predecessors:
100         node = diagram.nodes[predecessor]
101         # Verify if the conditions to eliminate the node are met
```

```
102         if node.type == "DECISION" and all(item in node.predecessors for item in diagram.
                node_utility.predecessors if item != node.name):
103             #print("Eliminando nodo de decision ", node.name)
104 
105             # Adjust order of the matrix
106             utility = diagram.node_utility
107             decision_index = utility.pointer.index(node.name)
108             utility.matrix = np.moveaxis(utility.matrix, decision_index, -1).tolist() # Move
                    the decision node to the last position
109             utility.pointer = utility.pointer[:decision_index] + utility.pointer[
                    decision_index+1:] + [node.name] # Move the decision node to the last position
110 
111             utility.matrix, matrix = delete_decision_node_aux(diagram, utility.matrix, utility
                    .pointer, node.value)
112             utility.pointer = utility.pointer[:-1]
113 
114             utility.add_decision(Decision_matrix(node.name, matrix, utility.pointer.copy()))
115 
116             # Delete the node
117             utility.predecessors.remove(node.name)
118             del diagram.nodes[node.name]
119 
120             # Eliminate sinks
121             eliminate_sinks(diagram)
122             return True
123 
124     return False
125 
126 def delete_decision_node_aux(diagram, matrix, pointer, decision_values):
127     if np.ndim(matrix) > 1:
128         values = []
129         decision_matrix = []
130         for i in range(diagram.nodes[pointer[0]].value.__len__()):
131             val, del_val = delete_decision_node_aux(diagram, matrix[i], pointer[1:],
                    decision_values)
132             values.append(val)
133             decision_matrix.append(del_val)
134         return values, decision_matrix
135     else:
136         maximum = max(matrix)
137         index = matrix.index(maximum)
138         return maximum, decision_values[index]
139 
140 # Invert node
141 def invertir_nodo(diagram, node1, node2):
142     # Check that there is no other path directed from node1 to node2
143     for node in node2.predecessors:
144         if node != node1:
145             if search_path(diagram, node1, node) == True:
146                 return False
147 
148     # Save Y
149     old_node2 = copy.deepcopy(node2)
150 
151     # Obtain the new table of Y
152     multipy_probability(diagram, node2, node1)
153 
154     # Obtain the new table of X
155     bayes_theorem(diagram, node1, node2, old_node2)
156 
157     return True
158 
159 # Eliminate chance node
160 def eliminate_chance_node(diagram):
161     for predecessor in diagram.node_utility.predecessors:
162         node = diagram.nodes[predecessor]
163 
164         # Verify if the conditions to eliminate the node are met
165         if node.type == "CHANCE" and len(node.sucessors(diagram.nodes)) == 1:
166 
```

```
167             # Adjust the utility
168             utility = diagram.node_utility
169             multipy_probability(diagram, utility, node)
170 
171             # Delete the node
172             del diagram.nodes[node.name]
173 
174             return True
175 
176     return False
177 
178 # ------------------ Auxiliar functions ------------------
179 def multipy_probability(diagram, node1, node2):
180     A = [x for x in node1.predecessors if x not in node2.pointer] # Predecessors of node1 that
            are not in node2
181     B = [x for x in node2.predecessors if x in node1.predecessors] # Predecessors of node2
        that are in node1
182     C = [x for x in node2.predecessors if x not in node1.predecessors] # Predecessors of node2
            that are not in node1
183 
184     # node1 -> B,A,value1,value2 -> B,A_value1,C_1,C_2
185     index_node1 = [node1.pointer.index(x) for x in B]
186     index_node1.extend([node1.pointer.index(x) for x in A])
187     if node1.type == "CHANCE":
188         index_node1.append(node1.pointer.index(node1.name))
189     index_node1.append(node1.pointer.index(node2.name))
190 
191     node1.matrix = np.transpose(node1.matrix, index_node1).tolist()
192     node1.pointer = [node1.pointer[x] for x in index_node1]
193 
194     # node2 -> B,C_1,value2,C_2
195     index_node2 = [node2.pointer.index(x) for x in B]
196     index_node2.append(node2.pointer.index(node2.name))
197     others_index2 = [node2.pointer.index(x) for x in C]
198     if(others_index2.__len__() > 0):
199         index_node2 = index_node2[:-1] + others_index2[:-1] + [index_node2[-1]] + [
                others_index2[-1]]
200     node2.matrix = np.transpose(node2.matrix, index_node2).tolist()
201     node2.pointer = [node2.pointer[x] for x in index_node2]
202 
203     node1.matrix = multiply_matrix(diagram, B, node1.matrix, node2.matrix)
204     # Adjust the pointers
205     node1.pointer.pop() # Delete the pointer of node2
206     add_pointer = [x for x in node2.pointer if x in C] # Add the pointers of node2 that are
            not in node1
207     if len(add_pointer) > 0:
208         node1.pointer = node1.pointer[:-1] + add_pointer[:-1] + [node1.pointer[-1]] + [
                add_pointer[-1]]
209 
210     # Adjust the predecessors
211     node1.predecessors = node1.pointer.copy()
212     if node1.type == "CHANCE":
213         node1.predecessors.remove(node1.name)
214     return 0
215 
216 def multiply_matrix(diagram, common_predecessors, utility_values, node_values):
217     if common_predecessors.__len__() > 0:
218         values = []
219         for i in range(diagram.nodes[common_predecessors[0]].value.__len__()):
220             values.append(multiply_matrix(diagram, common_predecessors[1:], utility_values[i],
                    node_values[i]))
221         return values
222     if isinstance(utility_values, list) and len(np.shape(utility_values)) > 2:
223         values = []
224         for i in range(len(utility_values)):
225             values.append(multiply_matrix(diagram, common_predecessors, utility_values[i],
                    node_values))
226         return values
227     else:
228         return np.matmul(utility_values, node_values).tolist()
```

```
229
230 def search_path(diagram, node1, node2):
231     # Search if there is a path from node1 to node2
232     for node in diagram.nodes[node2].predecessors:
233         if node != node1:
234             if search_path(diagram, node1, node) == True:
235                 return True
236     return False
237
238 def bayes_theorem(diagram, node1, node2, node2_old):
239     # obtain the common predecessors
240     B = [x for x in node1.predecessors if x in node2.predecessors and x in node2_old.
        predecessors]
241     A = [x for x in node2.predecessors if x in node1.predecessors and x not in B]
242     C = [x for x in node2_old.predecessors if x in node2.predecessors and x not in B]
243
244     # node1 -> B,A,value1
245     index_node1 = [node1.pointer.index(x) for x in B]
246     index_node1.extend([node1.pointer.index(x) for x in A])
247     index_node1.append(node1.pointer.index(node1.name))
248
249     node1.matrix = np.transpose(node1.matrix, index_node1)
250     node1.pointer = [node1.pointer[x] for x in index_node1]
251
252     # node2_old -> B,C,value2,Node1
253     index_node2_old = [node2_old.pointer.index(x) for x in B]
254     index_node2_old.extend([node2_old.pointer.index(x) for x in C])
255     index_node2_old.append(node2_old.pointer.index(node2_old.name))
256     index_node2_old.append(node2_old.pointer.index(node1.name))
257
258     node2_old.matrix = np.transpose(node2_old.matrix, index_node2_old)
259     node2_old.pointer = [node2_old.pointer[x] for x in index_node2_old]
260
261     # node2 -> B,C,A,value2
262     index_node2 = [node2.pointer.index(x) for x in B]
263     index_node2.extend([node2.pointer.index(x) for x in C])
264     index_node2.extend([node2.pointer.index(x) for x in A])
265     index_node2.append(node2.pointer.index(node2.name))
266
267     node2.matrix = np.transpose(node2.matrix, index_node2)
268     node2.pointer = [node2.pointer[x] for x in index_node2]
269
270     # B,C,A,Value2,Value1
271     node1.matrix = bayes_multiply(diagram, A, B, C, node1.matrix, node2.matrix, node2_old.
        matrix)
272     node1.pointer = B + C + A + [node2.name] + [node1.name]
273
274     # Adjust the predecessors
275     node1.predecessors = node1.pointer.copy()
276     if node1.type == "CHANCE":
277         node1.predecessors.remove(node1.name)
278
279
280 def bayes_multiply(diagram, A, B, C, node1_values, node2_values, node2_old_values):
281     if B.__len__() > 0:
282         values = []
283         for i in range(diagram.nodes[B[0]].value.__len__()):
284             values.append(bayes_multiply(diagram, A, B[1:], C, node1_values[i], node2_values[i
                ], node2_old_values[i]))
285         return values
286     if C.__len__() > 0:
287         values = []
288         for i in range(diagram.nodes[C[0]].value.__len__()):
289             values.append(bayes_multiply(diagram, A, B, C[1:], node1_values, node2_values[i],
                node2_old_values[i]))
290         return values
291     if A.__len__() > 0:
292         values = []
293         for i in range(diagram.nodes[A[0]].value.__len__()):
```

```
294             values.append(bayes_multiply(diagram, A[1:], B, C, node1_values[i], node2_values[i
                    ], node2_old_values))
295         return values
296     values = np.zeros(np.shape(node2_old_values))
297     for i in range(len(node2_old_values)):
298         values[i] = np.divide(node2_old_values[i], node2_values[i]).tolist()
299         values[i] = np.multiply(values[i], node1_values).tolist()
300 
301     return values
302 
303 def init_individual(diagram):
304     c = copy.deepcopy(diagram)
305     c.instance_probabilities()
306     return c
```

.3. Función del Algoritmo Genético

```
1  from influence_diagram import *
2  from diagram_prueba.diagram_prueba import *
3  import random
4  from nh1v1.nh1v1 import *
5  from evaluate_rules import *
6  from txtConverter import *
7  from genetic_algorithm.population import *
8  
9  
10 # Tournament selection
11 def tournament_selection(population, tournament_size, n_selections, replacement = False):
12     individuals = copy.copy(population.individuals)
13     selected = []
14     for i in range(n_selections):
15         tournament = []
16         for j in range(tournament_size):
17             tournament.append(individuals[np.random.randint(0, len(individuals))])
18         tournament.sort(key = lambda x: x.fitness, reverse = True)
19         selected.append(tournament[0])
20         if not replacement:
21             individuals.remove(tournament[0])
22     return selected
23 
24 # Crossover
25 def crossover(parent1, parent2):
26     children = Individual(parent1.diagram)
27 
28     for name in parent1.diagram.nodes.keys():
29         if not parent1.diagram.nodes[name].type == "DECISION":
30             children.diagram.nodes[name].matrix = crossover_aux(children.diagram, parent1.
                    diagram.nodes[name].matrix, parent2.diagram.nodes[name].matrix)
31 
32     return children
33 
34 # Crossover auxiliar
35 def crossover_aux(diagram, node1_values, node2_values):
36     if len(np.shape(node1_values)) == 1:
37         return random.choice([node1_values, node2_values])
38     else:
39         values = []
40         for i in range(len(node1_values)):
41             values.append(crossover_aux(diagram, node1_values[i], node2_values[i]))
42         return values
43 
44 # Mutation
45 def mutation(individual, mutation_rate, mutation_variance):
46     for name in individual.diagram.nodes.keys():
47         if not individual.diagram.nodes[name].type == "DECISION":
```

```
48              individual.diagram.nodes[name].matrix = mutation_aux(individual.diagram.nodes[name
                    ].matrix, mutation_rate, mutation_variance)
49
50  # Mutation auxiliar
51  def mutation_aux(matrix, mutation_rate, mutation_variance):
52      if len(np.shape(matrix)) == 1:
53          if np.random.rand() < mutation_rate:
54              return mutation_vector(matrix, mutation_variance)
55          else:
56              return matrix
57      else:
58          values = []
59          for i in range(len(matrix)):
60              values.append(mutation_aux(matrix[i], mutation_rate, mutation_variance))
61          return values
62
63  # Mutation vector
64  def mutation_vector(matrix, mutation_variance):
65      mutation_index = random.randint(0, len(matrix) - 1)
66      new_probability = matrix[mutation_index] + random.uniform(-mutation_variance,
            mutation_variance)
67
68      # Ensure the new probability is in the range [0, 1]
69      new_probability = max(0, min(new_probability, 1))
70
71      # Adjust the other probabilities to keep the total sum in 1
72      diference = new_probability - matrix[mutation_index]
73      matrix[mutation_index] = new_probability
74      for i in range(len(matrix)):
75          if i != mutation_index:
76              matrix[i] -= diference / (len(matrix) - 1)
77              matrix[i] = max(0, min(matrix[i], 1))
78
79      return matrix
80
81  # Genetic algorithm
82  def genetic_algorithm(diagram, population_size, n_generations, elite_size,
83                        selection_porcent, tournament_size, replacement, mutation_rate,
                              mutation_variance, expert_rules, total_rules):
84
85
86      # Init population
87      population = Population(diagram, population_size, expert_rules, count_rules(expert_rules,
            diagram), total_rules, count_rules(total_rules, diagram))
88
89
90      results = []
91
92      for i in range(n_generations-1):
93          print("Generation: ", i+1)
94
95          new_population = []
96
97          # Evaluate population
98          population.evaluate()
99
100         # Save results
101         results.append(population.get_puntuations())
102
103         # Select parents
104         parents = tournament_selection(population, tournament_size, int(population_size*
                selection_porcent),replacement)
105
106         # delete last parent if odd
107         if len(parents) % 2 != 0:
108             parents.pop()
109
110         for j in range(0, len(parents), 2):
111             for k in range(0,4):
112                 # Crossover
```

```
113                 children = crossover(parents[j], parents[j+1])
114                 # Mutation
115                 mutation(children, mutation_rate, mutation_variance)
116                 new_population.append(children)
117 
118         # Selection
119         # Order population by fitness
120         population.individuals.sort(key = lambda x: x.fitness, reverse = True)
121 
122         # Search if there is space for the elite
123         if len(new_population) + elite_size < population_size:
124             new_population.extend(population.individuals[:elite_size])
125         else:
126             new_population.extend(population.individuals[:population_size - len(new_population
                )])
127 
128         # Fill the population with individuals from the previous generation
129         while len(new_population) < population_size:
130             new_population.append(population.individuals[np.random.randint(0, population_size)
                ])
131 
132         # new population
133         population.individuals = new_population
134 
135     population.evaluate()
136 
137     results.append(population.get_puntuations())
138 
139     return results
```