UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC
DEPARTAMENT DE SISTEMES INFORMÀTICS I COMPUTACIÓ

Departamento de sistemas informáticos y computación

Búsqueda de mejores estrategias de entrada a un mercado

Trabajo Técnicas de Inteligencia Artificial

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

Autor: Francisco Javier Gil-Terrón Rodríguez

2021 - 2022

Tabla de contenidos

1. Introducción

En este trabajo se tratará de ofrecer soluciones a un problema propuesto mediante el uso de técnicas metaheurísticas, en este caso, se abordarán resoluciones basadas en el desarrollo de algoritmos genéticos y enfriamiento simulado.

1.1 Dilema del prisionero

En primer lugar, se explicará brevemente en qué consiste el dilema del prisionero ya que el problema propuesto se basa en este mismo con un enfoque ligeramente distinto.

El dilema del prisionero es un problema clásico de teoría de juegos ampliamente usado en el ámbito de la economía, en el que dos individuos deben tomar una decisión binaria, en este contexto, colaborar o delatar, al contrario, y recibirán una recompensa o pena en consecuencia tanto de su decisión como la del otro individuo. Los resultados obtenidos por A se pueden representar mediante la siguiente matriz de pagos, donde A y B son dichos individuos. Esta matriz es la que ha sido utilizada durante todo este trabajo debido a su uso estandarizado en este problema.

Decisiones	**B Toma la decisión**	**B No la toma**
A Toma la Decisión	1	5
A No la toma	0	3

Tabla 1 - Matriz de pagos

Aunque tradicionalmente los pagos de este problema representen valores negativos (años de prisión) consideraremos que es un valor positivo para homogeneizar la notación utilizada en nuestro problema. De esta manera, las reglas del problema son las siguientes:

- A delata a B y B colabora con A debe ser la situación más favorable para A.
- A colabora con B y B delata a A debe ser la situación más desfavorable para A.

- Los casos de delatarse mutuamente o colaborar ambos deben tener valores comprendidos entre los dos casos anteriores, siendo el caso de colaboración mutua más favorable que el de delatarse ambos.
- No puede haber dos situaciones con el mismo valor.

Mientras se cumplan estas condiciones, la matriz de pagos puede tener cualquier valor, pero como ya se ha dicho, se emplearán los anteriormente citados debido a su estandarización.

En este punto, la decisión más lógica es la 'egoísta' ya que, aunque colaborar podría llegar a obtener más puntos, delatar siempre es la decisión que minimiza el riesgo, siendo esta opción la preferible en prácticamente cualquier situación, incluso en la versión iterada del problema, es decir, donde no se va a tomar una única vez la decisión, sino que se tomará una y otra y el resultado final será la suma de los resultados obtenidos tras cada decisión.

No obstante, si a la anterior problemática le añadimos memoria sobre las n jugadas anteriores y que la decisión actual dependa de estas, en un entorno donde no solo compiten dos individuos, sino que puede haber una gran cantidad de ellos, la situación cambia. En este contexto, si el objetivo es maximizar el beneficio propio, la estrategia de elegir siempre la opción 'egoísta' puede llegar a no ser la mejor (aunque desde luego será potencialmente mejor que decidir siempre la otra opción), pues, aunque minimice el riesgo y lo más probable es que garantice un resultado aceptable, en determinadas ocasiones no será la opción que reciba más puntos, situación que no es raro que suceda teniendo en cuenta que habrá numerosos individuos con diferentes estrategias, y frecuente en nuestro problema ya que el número de veces que se puede elegir la opción 'egoísta' puede llegar a estar limitado como se comentará más adelante, caso en el que la estrategia de tomar siempre la decisión 'egoísta' no es siquiera una opción.

1.2 Problema actual

El problema que se tratará de abordar en este trabajo es una de las muchas posibles interpretaciones que se le puede dar al dilema del prisionero iterado con decisiones basadas en el historial de decisiones anteriores, en este caso con restricciones para tomar la decisión 'egoísta'.

Bajo esta premisa, se tendrá por objetivo encontrar aquella estrategia de toma de decisión binaria para una empresa, por ejemplo, realizar o no una acción comercial o campaña de marketing en un periodo dado, que maximice el beneficio propio al entrar a un mercado conocido, en otras palabras, se conocen las estrategias de las empresas que ya existen en dicho mercado (esta situación no dista demasiado de la realidad, pues mediante técnicas de

estudios de mercado se puede conocer en profundidad el comportamiento del mismo).

De esta manera, la decisión 'egoísta' sería la de realizar esa acción comercial, obteniendo el máximo beneficio en el caso en que el contrincante no la hubiera hecho, y obteniendo el mínimo en el caso contrario. En caso de que ninguno de los dos decidiera realizar dicha acción comercial, ambos obtendrían un resultado intermedio y superior al caso en que ambos hubieran decidido hacerlo, pues el hecho de realizar una acción comercial supone una inversión de recursos.

En esta línea, tanto por motivos internos, por ejemplo, los recursos disponibles, como por otros externos, como regulaciones, es posible que el número de veces que se puede tomar la decisión 'egoísta' sea limitado de alguna manera, por lo que se deberá tener en cuenta un factor de limitación durante la implementación.

Con todo lo anterior, se buscará implementar un programa flexible en cuanto a la introducción de parámetros, para encontrar la mejor estrategia A posible, entendiendo estrategia como el conjunto de decisiones que habrá que tomar contemplando todas las combinaciones posibles de situaciones anteriores, tal que A maximice los resultados obtenidos frente a toda estrategia B presente en una población conocida.

1.3 Tecnología utilizada

La metodología seleccionada es la de implementación propia, desarrollando los algoritmos enteramente, y únicamente haciendo uso de librerías externas con el fin de recurrir a funciones concretas. Estas librerías con sus respectivos usos son:

- Random: Generación de números aleatorios.
- Numpy: Selección aleatoria de elementos de una lista en base a una lista de pesos de la misma longitud.
- Itertools: Generación de todas las combinaciones posibles de unas entradas dadas.
- Copy: Creación de copias de elementos de forma segura para evitar cambios en la original.
- Statistics: Cálculo de la mediana de los elementos de una lista.
- Matplotlib: Creación de gráficos.
- Scipy: Generar pesos de manera suavizada en función del fitness dando más valor a los más altos y menos a los más bajos.

El lenguaje de programación elegido para el proyecto es Python, y el entorno utilizado es el ofrecido por Google: Colaboratoy, servicio cloud basado en Notebooks de Jupyter.

1.4 Codificación

La codificación para el problema que se describirá a continuación será compartida tanto para la resolución mediante el uso del algoritmo genético desarrollado como para la técnica de enfriamiento simulado.

Así pues y siguiendo la notación original del dilema del prisionero, tomar la decisión 'egoísta', en este caso, llevar a cabo la acción comercial, será representada con un cero, mientras que la decisión de no realizarla será representada con un uno. De esta manera, tanto las estrategias como las jugadas anteriores serán representadas por listas binarias de ceros y unos, y la matriz de pagos que se empleará es la propuesta en la tabla 1.

En primer lugar, se encontrará el parámetro **n**, que tendrá el valor del número de jugadas anteriores que se recordarán y que se le asignará un valor entero mayor que cero. La longitud de la lista donde se encuentran las jugadas anteriores es de 2***n**, una por cada uno de los dos individuos que se enfrentarán, y la longitud de las estrategias será de 2^{2*n}, pues dependerá de esta misma **n,** tal como se mostrará a continuación con algunos ejemplos para distintos valores de **n**.

n = 1

Si recordamos una única jugada se necesitarán 2 bits para representarla, por ejemplo, (0,1), que significa que en la jugada anterior el primer individuo tomó la decisión 0 y el segundo tomó la decisión 1. Puesto que para representar todas las posibles jugadas anteriores con esta talla (uno) se necesitan 2 bits, podrá haber $2^2 = 4$ situaciones anteriores posibles ((0,0), (0,1), (1,0), (1,1)).

Por ello, se necesitarán 4 bits para representar todas las estrategias posibles en base a una jugada anterior, es decir, habrá $2^4 = 16$ estrategias distintas posibles. Una posible de estas estrategias sería [0,0,1,1], que contiene todas las decisiones para cualquier valor de las jugadas anteriores: en caso de que las jugadas anteriores tomaran el valor (0,0) la decisión de esta estrategia sería 0 (la posición 0 de la lista), si fuera (0,1) tomaría la decisión 0 (posición 1), en el caso (1,0) tomaría la decisión 1 (posición 2 de la lista) y por último el caso (1,1) tomaría la decisión 1 (posición 3 de la lista).

En otras palabras, el valor de las jugadas anteriores en decimal es equivalente al índice de la posición de la decisión que tomará cada una de las estrategias de esa talla.

n = 2

En esta situación y siguiendo la lógica anteriormente expuesta, recordando dos jugadas anteriores se necesitarán 4 bits dando lugar a 2^4 = 16 jugadas anteriores posibles, y, por tanto, se requerirán 16 bits para representar las estrategias dando lugar a 2^{16} = 65536 estrategias diferentes posibles.

Por ejemplo, las jugadas anteriores (1,0,0,1) significan que en la jugada t-1 el primer individuo decidió 1 y el segundo 0, y en t-2 el primero decidió 0 y el segundo 1 (es decir, los primeros bits en la lista son las jugadas más cercanas en el tiempo).

Para estas jugadas anteriores (1,0,0,1), la estrategia, por ejemplo, [1,0,1,1,1,0,0,0,1,1,1,1,0,0,1,0], decidirá 1, pues es el valor que tiene la estrategia en la posición 9 de la lista (1001 en decimal).

n = 3

Para una talla del problema de **n** = 3, en otras palabras, recordando las 3 jugadas anteriores, se necesitarán 6 bits, que dan lugar a 2^6 = 64 posibles jugadas anteriores. A su vez, serán necesarios 64 bits para representar las estrategias, es decir habrá 2^{64} estrategias diferentes para esta talla.

Un ejemplo de jugadas anteriores podría ser (0,0,1,1,0,0) y una estrategia sería una lista de 64 ceros y unos. Una vez más, la decisión tomada por una estrategia cualquiera de esta talla sería el valor que se encontrara en la posición de la estrategia que fuera 001100 en decimal, es decir, la decisión será el valor que se encuentre en la posición 12 de la estrategia.

2. Algoritmo genético

2.1 Diseño y desarrollo

Mediante el uso de un algoritmo genético, buscaremos las mejores estrategias en una población inicial de la siguiente manera: iterativamente, se hará una selección, cruce entre individuos, mutación de estos y reemplazo sobre la población inicial de estrategias siguiendo un criterio determinado en función de su fitness.

Cabe destacar que existirá una población fija de estrategias para una talla dada que no se modificará durante la ejecución y que representará el mercado al que se pretende entrar. Esta población supone una instancia concreta de este problema y se empleará para ejecutar la función de evaluación sobre la población en la que trataremos de encontrar las mejores soluciones.

En definitiva, los individuos que se manejarán en el problema y que serán los miembros de la población son las estrategias, mientras que las jugadas anteriores, la matriz de pagos y la población fija que represente el mercado se utilizarán sobre estos individuos para evaluarlos.

A continuación, se detallará, entre otros, el comportamiento de las funciones de selección, cruce, mutación y reemplazo, y posteriormente, en la evaluación, se describirá por qué se ha elegido este comportamiento, contrastando con los resultados obtenidos.

2.1.1 Generación de poblaciones

Por lo que respecta a la población fija o mercado que servirá para evaluar nuestra población, sería interesante realizar un estudio de mercado para crear una instancia del problema que tratase de solucionar un caso real, pero ya que escapa a los objetivos de este trabajo, se generará de manera aleatoria con una longitud de **pm** estrategias.

En cuanto a la población inicial sobre la que trabajaremos y trataremos de mejorar, tendrá un tamaño de **p** estrategias, que no tiene por qué coincidir con el tamaño del mercado, y que serán creadas aleatoriamente respetando el factor de limitación **r**. El factor de limitación será un valor entero positivo cuyo significado es el número máximo de veces seguidas que se podrá tomar

la decisión 'egoísta', o cero si queremos que no exista ninguna limitación. Este parámetro debe ser inferior o igual a **n**, pues no podemos controlar que no se decida más de **n** veces seguidas la opción 'egoísta' si no se recuerdan al menos ese número de jugadas anteriores. Por ello, el problema será más restrictivo cuanto menor sea **r** (sin ser cero), pues con **r** = 1, las estrategias estarán obligadas a decidir uno siempre que en la jugada inmediatamente anterior hubieran sido cero; mientras que será menos restrictivo con **r** = n, donde sólo estará obligadas a decidir uno cuando se hubiera decidido cero en todas las jugadas anteriores.

Por último, en cuanto a las jugadas anteriores, ya que para que una estrategia pueda tomar una decisión debe hacerlo con respecto a unas jugadas anteriores, se creará una lista de jugadas anteriores aleatorias también respetando el factor de limitación **r** y todas las estrategias se evaluarán comenzando desde esa misma situación para que se encuentren en las mismas condiciones. Aunque se pueda seleccionar tanto el tamaño de la población inicial como del mercado, el tamaño de las jugadas anteriores vendrá determinado por **n** (número de jugadas anteriores a recordar), como ya se comentó en la sección de codificación, siendo su tamaño concretamente **n***2.

2.1.2 Función de evaluación (Fitness)

El algoritmo ejecutará **k** veces, siendo **k** un parámetro de entrada, tanto esta función como las que serán comentadas tras esta misma.

Para evaluar a una estrategia de nuestra población, esta se enfrentará a cada uno de los miembros del mercado (la población fija) y realizarán un proceso de toma de decisiones iterativo en base a sus respectivas estrategias y a las jugadas anteriores. Cada uno obtendrá la puntuación correspondiente de la matriz de pagos en consecuencia de su decisión y de la del contrincante. Además, tras cada jugada, se actualizarán las jugadas anteriores, eliminando los 2 bits últimos de la lista (la jugada anterior más alejada en el tiempo) y añadiendo delante la jugada actual.

El número de jugadas que harán cada par de individuos es controlable mediante un parámetro del algoritmo (**l**), aunque es recomendable que sea del orden de 10***n** (número de jugadas a recordar) para que haya suficientes valores como para que el resultado sea representativo y no se tome el valor de unas pocas decisiones de manera aislada, ya que el valor que interesa es la media a la que debería converger a largo plazo. No obstante, se debe tener en consideración que si el número de jugadas es muy elevado puede llegar a ralentizar enormemente el algoritmo.

De esta manera, cada estrategia de la población, tras haberse enfrentado a cada estrategia del mercado, habrá obtenido una puntuación media para cada uno de estos, siendo el sumatorio de estas medias el fitness resultante de dicha estrategia. Es decir, el resultado de la función de evaluación de una estrategia en concreto es la suma de sus resultados medios contra cada una de las estrategias del mercado.

2.1.3 Selección

La selección será la primera función que se llevará a cabo tras la evaluación en cada iteración del algoritmo y consiste en elegir a una cantidad determinada de individuos para generar otra cantidad de estrategias nuevas a partir de los elegidos mediante cruces entre estos, tal como se verá en el siguiente apartado.

El criterio de selección de individuos vendrá dado por unos pesos en función de su fitness, siendo el peso la probabilidad de ser seleccionado. Estos pesos son calculados mediante la función *softmax,* que dará más peso cuanto mayor sea el fitness y menos cuanto menor sea siguiendo una función exponencial normalizada.

En cuanto a la cantidad de estrategias que serán seleccionadas, se podrá controlar mediante el parámetro **t**, que podrá tener valores entre 0 y 1 y representa el tamaño que tendrá la lista en porcentaje del tamaño de la población inicial. Por ejemplo, para una **t** = 0.4, el tamaño de la selección de estrategias para generar nuevas mediante cruce será igual al del 40% de la población inicial. Este parámetro está estrechamente vinculado con las etapas de cruce y reemplazo, como se comentará a continuación.

2.1.4 Cruce

La siguiente etapa, la de cruce, tiene como objetivo la de crear nuevos individuos a partir de la selección anterior, con el fin de introducirlos en la población para que mejore iterativamente.

La cantidad de individuos nuevos que se generarán mediante cruce dependerá también del parámetro **t** y se comportará de la siguiente manera: cuando **t** tenga un valor de 0.5 o inferior, es decir, la selección anterior tuviera un tamaño del 50% de la población inicial o menos, se generará esa misma cantidad de individuos; mientras que en el caso contrario, únicamente se generarán los necesarios para volver a alcanzar el 100% del tamaño de la

población, es decir, se generarán (1-**t**) multiplicado por el tamaño de la población, por ejemplo, para **t** = 0.8, es decir, la selección es del 80% del tamaño de la población, sólo se generarán individuos mediante cruce para completar el 20% restante. Este comportamiento en función de la variable **t** se tratará con más detalle en la sección de reemplazo.

Las estrategias nuevas se generarán eligiendo a dos padres aleatorios de la selección, y eligiendo aleatoriamente cada uno de los bits entre el del padre y el de la madre para esa posición, es decir, el cruce se hará de manera uniforme.

Puesto que ambas estrategias, el padre y la madre, se generaron inicialmente respetando las restricciones, cualquier combinación de ellos las seguirá respetando, por lo que en ese sentido las restricciones iniciales no suponen una limitación durante el proceso de cruce.

2.1.5 Mutación

Tras el cruce, viene la etapa de mutación, donde, a cada uno de los individuos nuevos generados durante el cruce, se le aplicará, con una probabilidad de **mt**, una mutación. La mutación consistirá en cambiar de manera aleatoria uno de los bits que forma la estrategia en cuestión.

En contraposición a lo que ocurre con el cruce, podría darse el caso en que una mutación diera una estrategia no válida de acuerdo con el factor de limitación, por lo que antes de mutar uno de los bits de la estrategia se asegurará que tras este proceso siga respetando las restricciones.

2.1.6 Reemplazo

En esta última etapa se sustituirán una cierta cantidad de individuos de la población inicial de una iteración determinada por otros, obtenidos mediante los tres procesos inmediatamente anteriores o generando nuevos de manera aleatoria. Es decir, el reemplazo será elitista con posibilidad de inclusión de nuevos individuos. La cantidad que será reemplazada dependerá, una vez más, del parámetro **t**, el mismo que el de selección y cruce, y será también el porcentaje de individuos que se mantendrán de la población inicial y el resto serán rellenados con hijos a partir del cruce y nuevos generados aleatoriamente.

En otras palabras, la cantidad de individuos que son seleccionados para reproducirse y para sobrevivir y pasar a la siguiente generación es la misma, pero cabe destacar que la elección que se hace de los individuos de la

población inicial para que sobrevivan NO tiene por qué coincidir con la selección de padres. Es decir, en la selección inicial únicamente se están eligiendo para reproducirse, y en esta etapa se vuele a seleccionar sobre la población inicial la misma cantidad de individuos siguiendo el mismo criterio en función de los pesos obtenidos aplicando la función *sotmax* a los fitness obtenidos.

Respecto a la nueva población que se generará mediante el reemplazo, la distribución entre hijos resultado del cruce y nuevos generados aleatoriamente dependerá este mismo valor **t**. Cuando **t** tenga valores inferiores a 0.5, es decir, sobrevivirá menos del 50% de la población inicial, y se generará esa misma cantidad de hijos mediante cruce, se rellenará con individuos nuevos aleatorios hasta alcanzar de nuevo el 100% del tamaño de la población inicial; mientras que cuando tenga valores por encima de 0.5, es decir, sobrevive más del 50% de la población, se completará enteramente con hijos generados mediante cruce.

Por ejemplo, si **t** = 0.3, tanto el tamaño de hijos generados por cruce como el número de estrategias que sobrevivirán será del 30% de la población inicial. La población nueva resultante será ese 30% que sobrevive, el 30% resultado del cruce, y el 40% restante será generado aleatoriamente. Mientras que si **t** = 0.7, sobreviven el 70% de las estrategias y el 30% restante será enteramente fruto del cruce de la selección, de manera que no se añaden individuos nuevos aleatoriamente. En otras palabras, si **t** tiene valores inferiores a 0.5, se está dando margen para que en cada generación entren nuevos individuos aleatorios a la población (además de los generados por cruce), mientras que si tiene valores superiores a 0.5 los nuevos serán generados únicamente por cruce.

2.2 Evaluación

Finalmente, para el algoritmo genético, evaluaremos el modelo descrito con distintos parámetros y se explicará, acompañado de gráficas, por qué se ha decidido este comportamiento. Todas las las gráficas que se muestran en esta sección presentarán el fitness del mejor individuo de la respectiva generación.

2.2.1 Tamaño del problema y convergencia

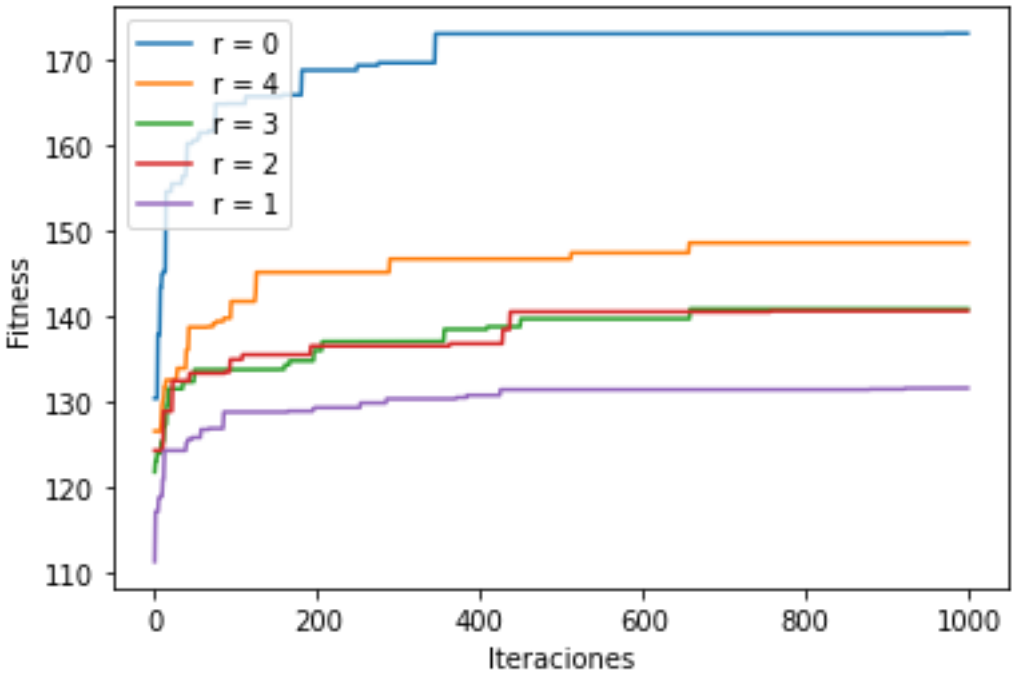
La talla del problema dependerá directamente de **n** (nº de jugadas anteriores a recordar) y reflejará la cantidad de estrategias posibles que existirán para una **n** dada. El tamaño del problema será de 2**2**2*n (2 elevado a 2 elevado a

2 multiplicado por **n**). Por ejemplo, con **n** = 1 habrá 2^4 estrategias, con n = 2 habrá 2^{16}, etc.

Como se puede deducir, a menor **n**, antes convergerá el problema, pues el espacio de soluciones a explorar será menor. No obstante, no tiene sentido comparar resultados de convergencia con distintas tallas, ya que se estaría tratando de resolver problemas totalmente distintos. Esto se debe a que en cada talla tendríamos que cambiar el mercado, y, por tanto, cambia la función de evaluación, por lo que la medida del fitness no sería contrastable entre diferentes tallas del problema.

Pese a esto, sí que es interesante estudiar el comportamiento del factor de limitación **r**. Como se había mencionado, cuanto menor sea **r**, más restrictivo será el problema (sin contar el caso de 0, en cuyo caso no habría restricción alguna) pues este parámetro condiciona la cantidad de veces seguidas que se puede tomar la decisión 'egoísta'.

Ilustración 1 - Evaluación de r



Tal como se puede observar en la gráfica, cuanto menor sea **r**, en el caso de que exista restricción (sea distinto de cero), antes convergerá el algoritmo. Esto es debido a que cuanto más restrictivo sea, menos estrategias viables existirán, reduciendo el espacio de exploración del problema.

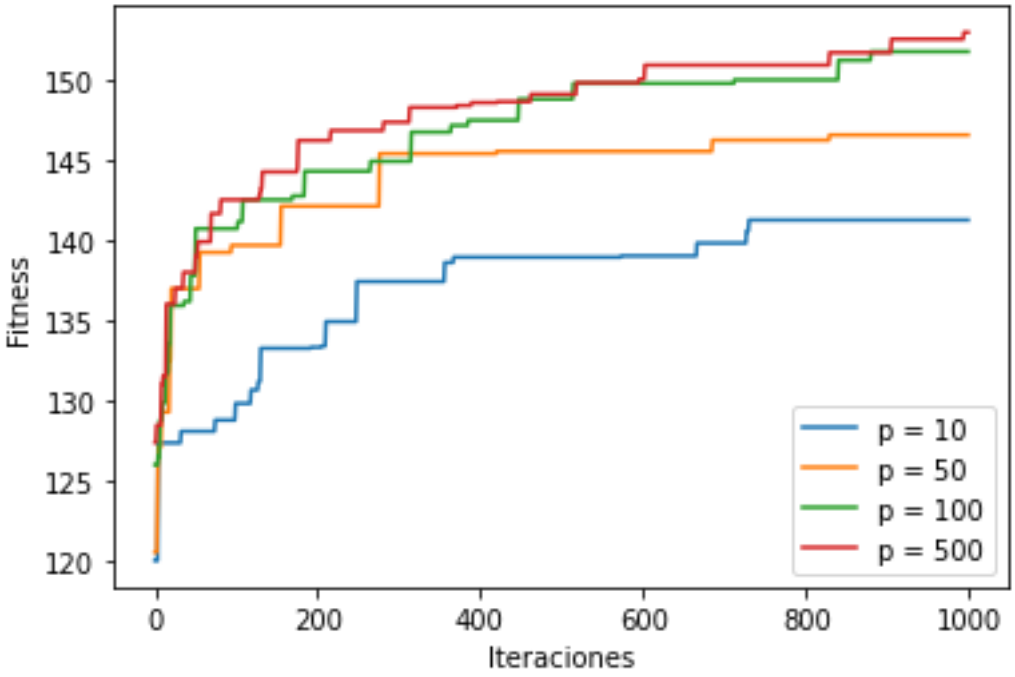
Cabe destacar que este factor NO es un parámetro de optimización, sino una variable que modifica el problema a resolver, es decir, no se debe interpretar en la gráfica que esté aportando mejores o peores resultados, simplemente es una instancia diferente del problema, con mayor o menor dificultad para encontrar mejores estrategias.

De esta manera, la talla elegida para las siguientes pruebas y evaluaciones será de **n** = 4 y **r** = 4, es decir, se recordarán cuatro jugadas anteriores y se podrá tomar la decisión 'egoísta' 4 veces seguidas como máximo, ya que esta talla obtiene los resultados esperados en un tiempo razonable.

2.2.2 Contraste y evaluación de parámetros

El primer parámetro que se ha evaluado es el tamaño de la población inicial **p**, o número de estrategias que existirá en la primera generación, y en las sucesivas. A continuación, en la ilustración 2, se muestran gráficamente los resultados que se han obtenido para distintos tamaños iniciales de población.

Ilustración 2 - Evaluación de p



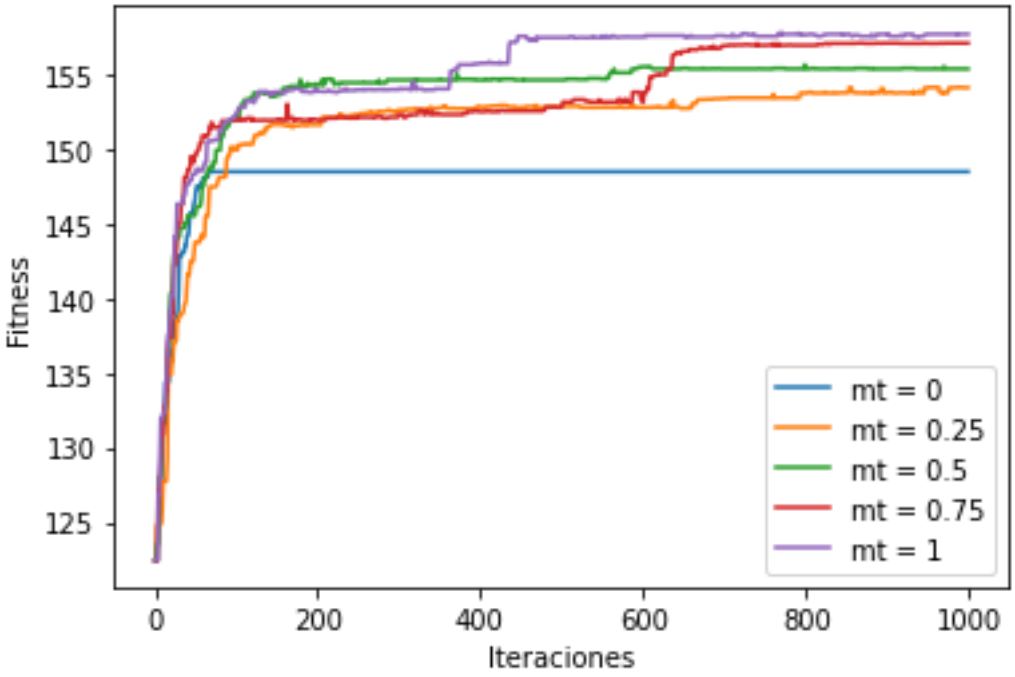
Como se puede observar, a mayor tamaño de población, además de generar un mayor número de soluciones, encontrará mejores resultados en menos iteraciones, pero, por otro lado, se incrementará a su vez el coste computacional de cada iteración.

Puesto que en este problema la cantidad de soluciones generadas no es tan relevante como lo es encontrar una solución mejor, se ha elegido el tamaño de población 50 como el más adecuado debido a que se ha considerado que poseía la mejor relación resultados/tiempo de computo.

Los siguientes parámetros que se evaluarán son los que controlan los procesos de selección, cruce, mutación y reemplazo, empezando por la mutación, tal como se explicará a continuación.

Para el proceso de mutación, existe un parámetro (**mt**) que controla la probabilidad de que se produzca una mutación en cada una de las estrategias resultantes del cruce (cabe recordar que la mutación consistía en invertir un único bit aleatorio de la estrategia), y el impacto de la variación de este parámetro en el fitness de cada generación se ve reflejado en la siguiente gráfica.

Ilustración 3 - Evaluación de mt



Como se puede ver, la mejora del fitness tras cada generación es muy dependiente de la probabilidad de mutación, de hecho, sin mutación, el problema converge de manera prematura. Pero, si se obtienen estas mejoras a mayor probabilidad de mutación, ¿por qué no mutar más bits en lugar de limitar la mutación a un único bit? Porque la probabilidad de que invertir un bit sobre una estrategia relativamente buena conduzca a una estrategia mejor es mucho menor de que provoque una estrategia peor. Por eso, en el caso de mutar más de un bit, la probabilidad de obtener mejores estrategias es ínfima.

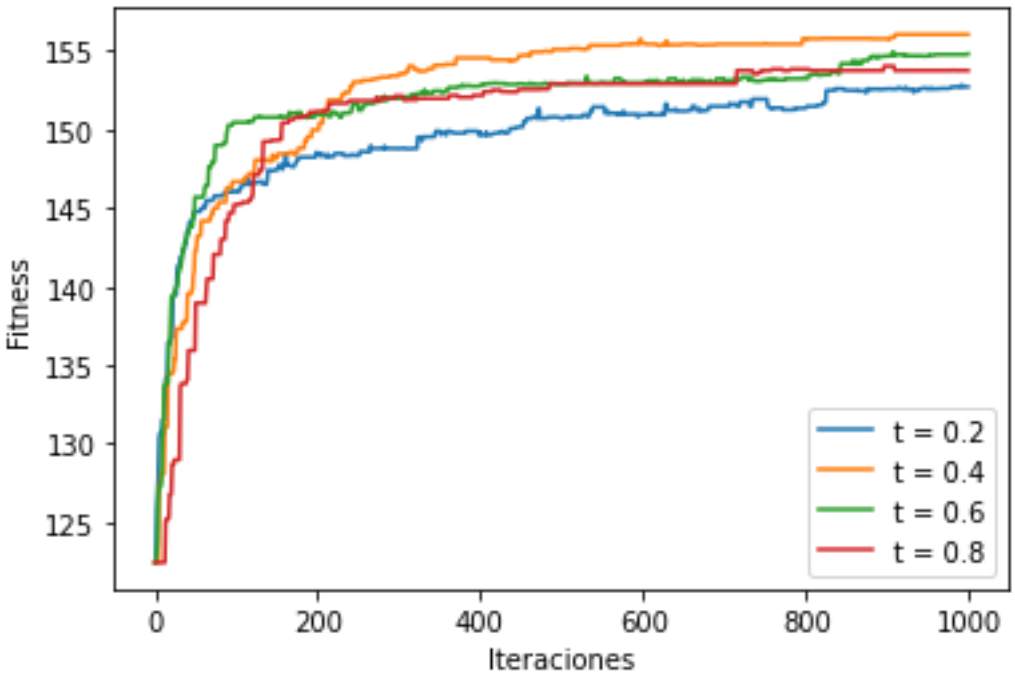
Es por esto por lo que se decidió este criterio de mutación, ya que se probó con diferentes cantidades de bits a mutar (constantes, en función de **n** y en función de **r**), y la única que ofrecía mejoras consistentes en el fitness tras cada iteración era esta.

Tras establecer este criterio de mutación, y entender que para alcanzar mejores resultados era necesaria la inclusión de cierta aleatoriedad, se consideró añadir nuevos individuos aleatorios en cada generación. Bajo esta premisa, se configuró un parámetro **t** que controlara de manera conjunta los procesos de selección, cruce y reemplazo y que se comporta de la siguiente manera: **t** será el porcentaje de supervivientes de la generación anterior que

pasan a la siguiente y cuando este parámetro es superior a 0.5, no se añadirán nuevos aleatorios, siendo el porcentaje que falte para llegar al total, el número de hijos resultado del cruce y mutación; mientras que cuando sea inferior a 0.5, **t** será tanto el porcentaje de supervivientes como de hijos en la siguiente generación, y el porcentaje que resta hasta alcanzar el total será rellenado con nuevos individuos aleatorios.

Por ejemplo, para **t** = 0.6, la siguiente generación se compondrá de 60% de supervivientes de la anterior y 40% de hijos; mientras que si, por ejemplo, tomara el valor **t** = 0.4, a siguiente generación será 40% de supervivientes, 40% de hijos y 20% de individuos nuevos generados de manera aleatoria. En la siguiente gráfica se puede observar cómo se comporta el fitness con la variación de dicho parámetro.

Ilustración 4 - Evaluación de t



Como se puede ver, los mejores resultados se obtienen con **t** = 0.4, donde se añaden en cada generación un 20% de individuos nuevos, ya que, si la **t** es muy cercana a 0, se añadirán demasiados individuos aleatorios como para que los mejores puedan tener impacto en la evolución de las generaciones, mientras que cuando la **t** es próxima a 1, se producirá una convergencia prematura en consecuencia a que los muy aptos dominan la población y estancando la búsqueda.

3. Enfriamiento simulado

3.1 Diseño y desarrollo

Como ya se había comentado, la codificación será la misma para la resolución mediante enfriamiento simulado y para el algoritmo genético, al igual que este método seguirá con la función de evaluación propuesta para el algoritmo genético y el mercado o población de estrategias fija es el mismo que se ha empleado para la evaluación del algoritmo genético para que los resultados sean contrastables.

En cuanto al enfriamiento simulado, se ha definido como vecino de una estrategia, esa misma estrategia con cualquiera de sus bits mutados, una vez más, teniendo en cuenta el factor de restricción no pudiendo generar los vecinos con los índices de la estrategia restringidos. De esta manera, se cogerá un vecino aleatorio (no el mejor) en cada iteración respecto al individuo actual.

En cuanto a la solución inicial sobre la que partirá el algoritmo, se empleará una solución aleatoria, ya que, debido a la gran cantidad de máximos locales existentes en este problema, la calidad de la primera solución no debería tener gran impacto en los resultados, puesto que, desde cualquier solución posible, se alcanzará un máximo local rápidamente.

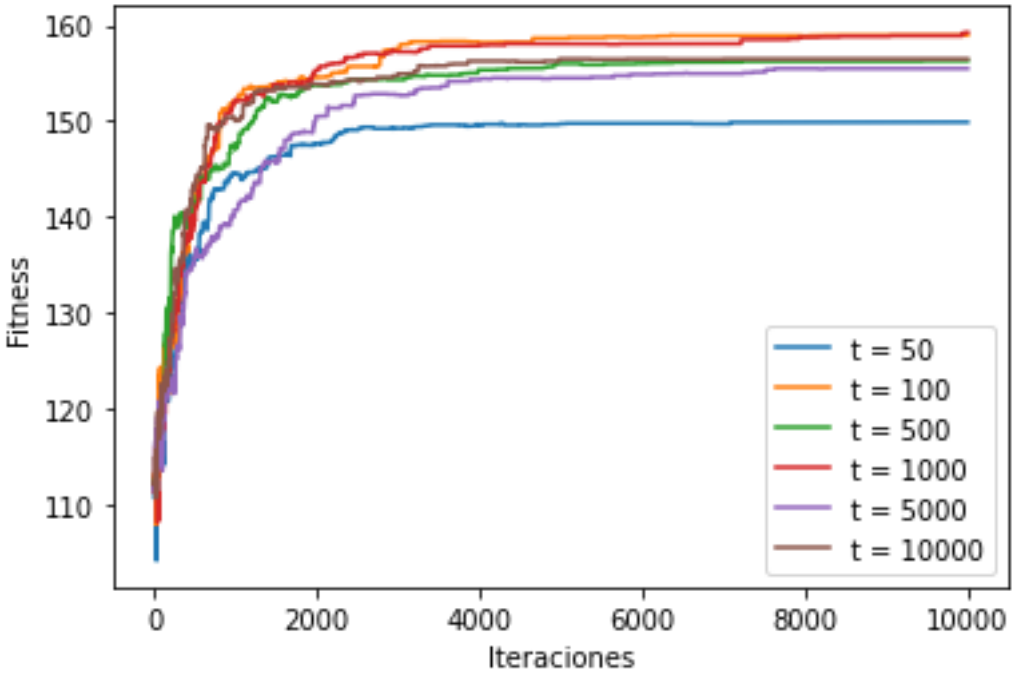
3.2 Evaluación

Por lo que respecta a la evaluación de parámetros del enfriamiento simulado, se comentarán algunas de las pruebas realizadas sin acompañarlas de sus respectivas gráficas en los casos en que no fuera relevante para el problema si inclusión, como, por ejemplo, la función de decremento de la temperatura, habiendo utilizado como función $T = T/(1+k*T)$, siendo el resto de las funciones de decremento muy inferiores en cuanto a resultados obtenidos.

Ya que se emplea esta función, los valores de k deberán ser pequeños y próximos a cero para obtener los mejores resultados, pero de nuevo, este es el comportamiento genérico de esta función de decremento y no es relevante para el problema, por lo que se evaluó superficialmente la variación de k simplemente para confirmar el comportamiento.

Con esta configuración, se procedió estudiar el comportamiento del fitness en función de la temperatura inicial, cuyos resultados se pueden ver en la siguiente gráfica.

Ilustración 5 - Evaluación de Temperatura



Como se puede ver, cuando la temperatura es baja (menor que 50), no se toma un vecino aleatorio como el actual (en lugar del mejor) suficientes veces como para alcanzar posteriormente resultados mejores, mientras que, a partir de ese valor de temperatura inicial, el algoritmo converge de manera similar.

4. Comparativa de métodos

Finalmente, se mostrarán los mejores resultados obtenidos para ambos algoritmos con las mejores parametrizaciones encontradas para cada uno.

Ilustración 6 – Mejores resultados AG

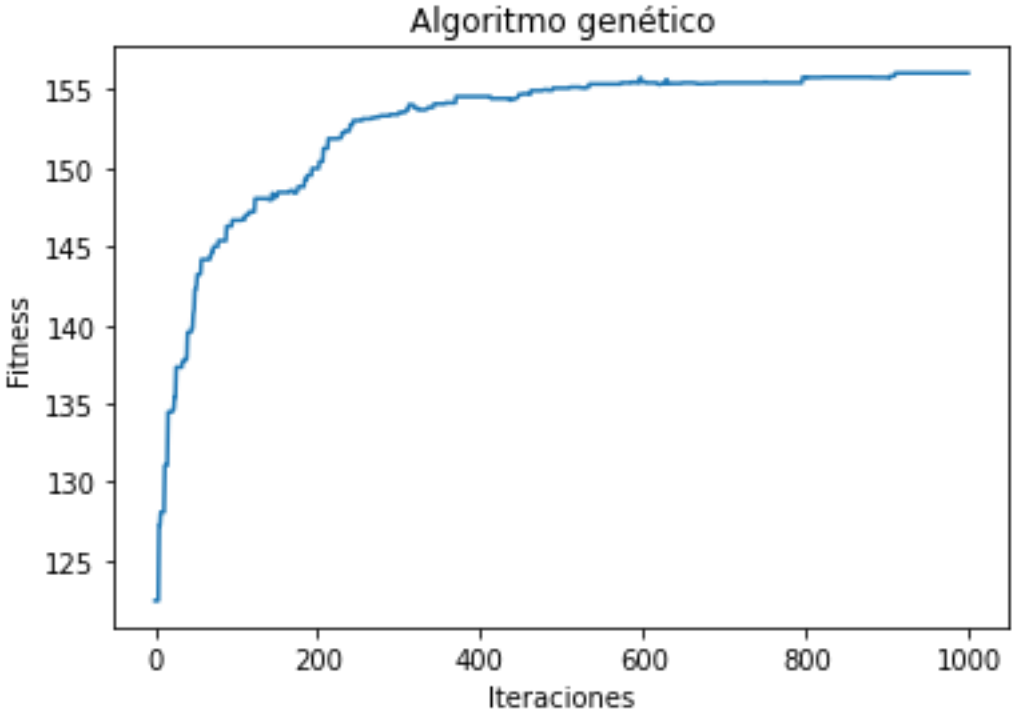
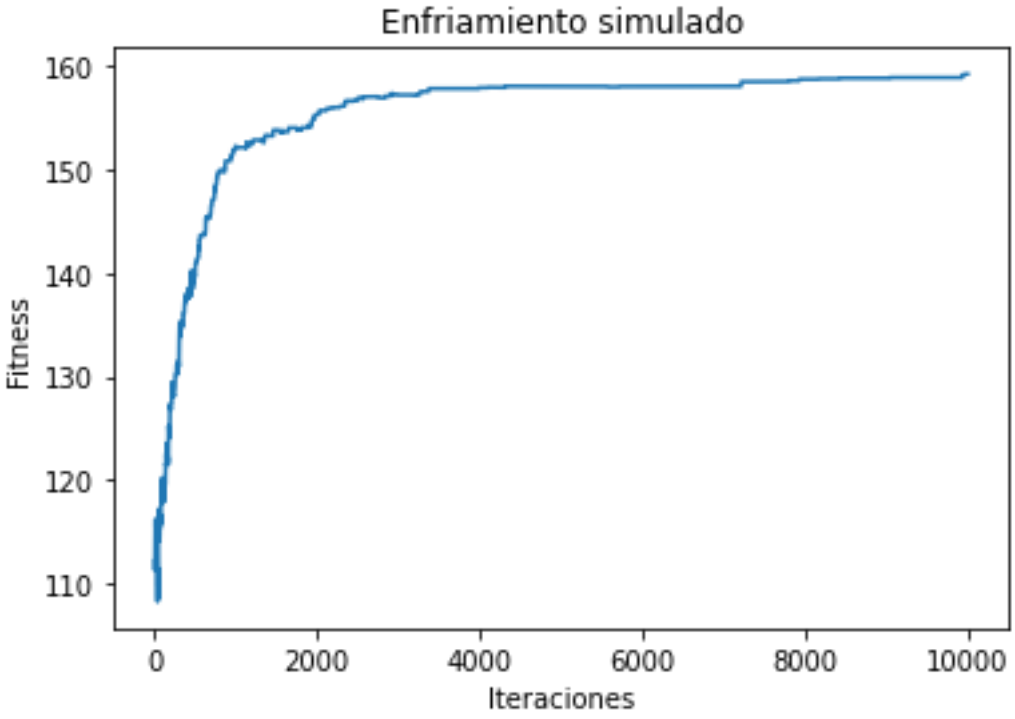


Ilustración 7 – Mejores resultados ES



Tal como se puede apreciar, el enfriamiento simulado obtiene mejores resultados que el algoritmo genético. Esto es debido, en parte, a que probablemente la parametrización y diseño del algoritmo genético no sea la ideal para resolución del problema ya que la complejidad del diseño del algoritmo genético es considerablemente superior a la del enfriamiento simulado. Pese a esto, y recordando que el problema presentado está repleto de máximos locales, ya que el enfriamiento simulado suele ofrecer buenos resultados en este tipo de problemas, tiene sentido que efectivamente el enfriamiento simulado resulte mejor para la resolución del problema, donde cabe citar que también es mejor en cuanto a coste computacional ya que las iteraciones del enfriamiento simulado son más sencillas.