UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Diseño y aplicación de técnicas inteligentes en problemas de logística

Trabajo Fin de Máster

Máster Universitario en Ingeniería Informática

AUTOR/A: Calatayud Giner, Javier

Tutor/a: Garrido Tejero, Antonio

CURSO ACADÉMICO: 2022/2023

Resumen

En este trabajo de final de máster, hemos planteado un problema logístico en el que habrá que encontrar la forma de suministrar unos fármacos a los pacientes de unos hospitales teniendo un personal limitado y un número determinado de dosis de los fármacos. A partir del enunciado vamos a explorar distintos tipos de técnicas inteligentes, principalmente algoritmos heurísticos y metaheurísticos, que podremos emplear para su resolución. Durante este proceso llevaremos a cabo el diseño e implementación de un algoritmo genético, explicando como las distintas configuraciones de este afectan a sus resultados y rendimiento.

Palabras clave: Inteligencia Artificial, Algoritmos heurísticos, Algoritmos Metaheurísticos, Algoritmos genéticos, Logística.

Abstract

In this master's thesis, we have posed a logistical problem in which we will have to find a way to supply medicine to patients in a hospital with a limited number of staff and a certain number of doses. From the statement we will explore different types of intelligent techniques, mainly heuristic and metaheuristic algorithms, that we can use for its resolution. During this process we will carry out the design and implementation of a genetic algorithm, explaining how the different configurations of the algorithm affect its results and performance.

Keywords: Artificial Intelligence, Heuristics Algorithms, Metaheuristics Algorithms, Genetic Algorithms, Logistics.

Tabla de contenidos

Tabla de ilustraciones

1. INTRODUCCIÓN

La Real Academia Española (Real Academia Española de la Lengua, 2021) define la logística como *un conjunto de medios y métodos necesarios para llevar a cabo la organización de una empresa o de un servicio, especialmente de distribución*. Esta ha sido un área que ha ido ganando más y más importancia desde la revolución industrial y la aparición de las grandes corporaciones, ya que las empresas descubrieron que organizando de una manera óptima sus procesos y recursos podrán ahorrar mucho tiempo y capital; y por inercia, ser más competitivos. Un ejemplo de esto fue la cadena de montaje propuesta por Henry Ford (Motorpasión, 2021), que permitió la fabricación en masa de vehículos de la marca homónima, y la cual podemos seguir viendo implantada en cientos de empresas de todo el mundo.

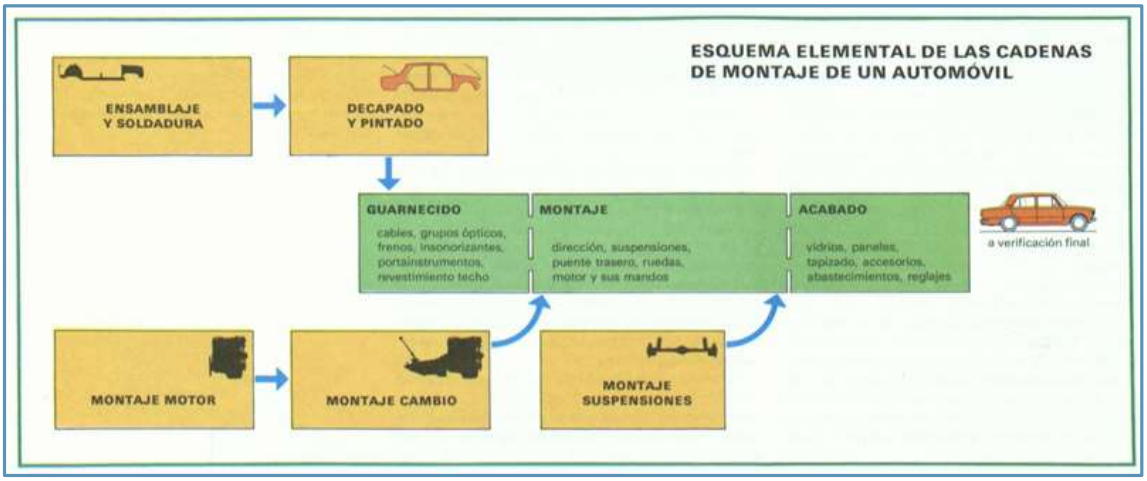


Ilustración 1-Esquema de la cadena de montaje

Los problemas de índole logístico no solo los encontramos en compañías del sector industrial, sino que son comunes en muchos otros ámbitos, desde el control de los accesos al puerto de una población hasta la administración masiva de un fármaco a la población. La aparición de este tipo de problemas ha coincidido con la popularización, y consecuente expansión de la informática, siendo natural que se intente emplear los sistemas computacionales, junto a distintos tipos de algoritmos de inteligencia artificial, para su resolución especialmente cuando la talla de los problemas alcanza cuotas imposibles de resolver por el ser humano corriente.

Así mismo, la inteligencia artificial es la disciplina informática que busca dotar a los sistemas computacionales de capacidades intrínsecamente humanas, como el razonamiento lógico o el reconocimiento de formas con el objetivo de poder abordar problemas que serían inasumibles por un ser humano corriente. La inteligencia artificial tiene una gran variedad de campos y aplicaciones, aunque recientemente han acaparado toda la atención del público general los LLM (Schade, 2023), sistemas capaces de analizar billones de archivos de texto y producir salidas idénticas a las que produciría un ser humano. Sin embargo, en este trabajo nos centraremos en analizar un área de la inteligencia artificial que, a pesar de haber perdido relevancia (Google,

2023), no deja de ser uno de los pilares fundamentales de este campo, como son las técnicas heurísticas y metaheurísticas para abordar el problema que plantearemos más adelante.

Teniendo en mente los dos puntos anteriores, decidimos plantear un problema vigente que respondiera a una problemática logística real y resolverlo mediante técnicas inteligentes. El reto en cuestión sería la planificación hospitalaria ante una crisis médica donde muchos pacientes deben ser medicados. El reto sería la organización de los recursos médicos para que dichos pacientes puedan recibir una dosis del fármaco en el menor tiempo posible. Para resolverlo, tal y como veremos más adelante, hemos explorado muchos tipos de algoritmos para terminar decantándonos por un algoritmo genético que, no solo nos parece más interesante, sino que además se puede ajustar mejor al tipo de problema que vamos a resolver.

1.1 MOTIVACIÓN

La motivación detrás de este proyecto era seleccionar una situación real donde se produjeran conflictos de ámbito logístico, y aportar una solución propia utilizando técnicas inteligentes. Para ello tendríamos que buscar casos concretos donde se pueda vislumbrar la vigencia y transversalidad de este tipo de problemas. Reflexionando sobre esto llegamos a la conclusión de que los casos más claros y actuales surgen de la crisis creada por el virus COVID-19 (Organizacion Mundial de la Salud, 2021).

La crisis del COVID-19, conocido también como coronavirus, fue una pandemia ocasionada por la aparición del patógeno SARS-CoV-2. El primer caso se dio en Wuhan en diciembre de 2019, y se extendió rápidamente por todo el mundo durante el año 2020, poniendo el jaque al mundo entero. Su alta capacidad de transmisión fue lo que hizo que fuese muy difícil de contener y provocó que los gobiernos tuvieran que tomar medidas excepcionales, como la restricción de movilidad o la obligatoriedad de llevar mascarilla quirúrgica.

Esta situación puso de manifiesto muchos problemas logísticos que nos podrían servir de inspiración a la hora de seleccionar el caso a resolver, como el control de personas dentro de espacios cerrados para minimizar el riesgo de contagio o el reparto de vacunas entre las distintas poblaciones del territorio español. Finalmente decidí estudiar la optimización de la vacunación de la población, asumiendo ciertas restricciones. Este nos pareció el caso más interesante ya que es un proceso que se intentó acelerar en su momento para inmunizar los antes posible al 70% de la población, y así alcanzar la inmunidad de grupo (Wikipedia, 2020).

Algunos casos, que no están relacionados con la temática anterior, que también nos parecieron interesantes fue el cálculo de rutas en la ciudad de valencia utilizando el transporte público o bici; y el posicionamiento óptimo de estaciones de bomberos dentro de una ciudad. La optimización de la infraestructura pública es un tema que va ganando mucha relevancia debido a que dentro de los objetivos de desarrollo sostenible de la agenda 2030 (Naciones Unidas, 2023) está la transformación de las ciudades para hacerlas más sostenibles.

Además, a nivel personal, siempre me ha interesado el campo de la inteligencia artificial, de forma que mi trabajo de final de grado (Giner, 2020) está también relacionado con esta área. Además, la asignatura de Sistema inteligentes fue la que más me llamó la atención, lo que me motivó a escoger concretamente esta propuesta, y así tener la oportunidad de seguir aprendiendo sobre esta fascinante área de la informática.

1.2 Objetivos

El fin último de este trabajo es el planteamiento y resolución de un problema de carácter logístico mediante el uso de técnicas de inteligencia artificial, el cual se puede estructurar en los siguientes objetivos:

- Definir un problema a resolver, dejando claro cuál es el desafío que plantea, que agentes y factores intervienen en el proceso, y que se intenta conseguir mediante la aplicación de técnicas inteligentes.
- Explorar los principales algoritmos que podemos aplicar a nuestro problema concreto, y justificar la elección de un algoritmo genético para llevar a cabo la resolución de nuestro problema concreto.
- Demostrar cómo vamos a implementar dicho algoritmo, especificando los distintos tipos de configuraciones que nos permite el mismo, dejando claras las ventajas y desventajas de cada una.
- Realizar la implementación del algoritmo, el testeo y el análisis de resultados, donde comprobaremos los aciertos y fallos de nuestra elección, y cuál es la configuración óptima para nuestro caso.

1.3 Beneficios esperados

Mediante el desarrollo de este proyecto espero demostrar los beneficios que te puede aportar el uso de inteligencia artificial, como herramienta para abordar un problema contemporáneo, como es la gestión de recursos médicos durante una crisis sanitaria, pudiendo ser un motor de innovación y aportar un gran beneficio social. Además, quiero mostrar cómo se pueden seguir empleando técnicas de inteligencia artificial menos imperantes para resolver problemas aún muy vigentes; y así evidenciar la importancia de que no caigan en el olvido.

A nivel personal, espero adquirir un mayor conocimiento en el área de la inteligencia artificial, y más concretamente en las técnicas inteligentes que estudiaremos durante el desarrollo de este proyecto; además de tener la oportunidad de implementar desde cero un algoritmo genético, con todas las decisiones que ello conlleva.

1.4 Estructura de la memoria

La estructura que sigue este documento responde al desarrollo natural de un trabajo de estas características, dividido en capítulos.

El primero de estos capítulos lo dedicaremos a la exposición del problema que vamos a resolver, especificando sus distintas características. Con este apartado espero que el lector comprenda el problema que vamos a intentar resolver y su magnitud.

El segundo capítulo del trabajo lo dedicaremos a explorar el estado del arte, exponiendo los conocimientos teóricos que vamos a aplicar. En este se explorarán las técnicas que se han tomado en cuenta para la resolución del problema y sus ventajas e inconvenientes. Esta exploración comenzará con las técnicas de búsqueda más simples y, a partir de estas, avanzará hacia aquellas más complejas, como son las búsquedas heurísticas, para terminar en las búsquedas estocásticas.

Siguiendo a la teoría, vendrá el capítulo donde abordaremos el problema, seleccionando el tipo de algoritmo que vamos a utilizar para resolverlo, y explicando el razonamiento detrás de dicha elección. Además, expondremos como vamos a implementar dicho algoritmo, y todas las decisiones que se han tomado respecto a este.

Una vez expuesta la implementación, tendremos un capítulo exponiendo y valorando los resultados de la ejecución del algoritmo. Los casos que se estudiarán serán distintas configuraciones del algoritmo y como los parámetros de este afectan a la solución final y al rendimiento.

El último capítulo de este documento la dedicaremos a exponer las conclusiones del proyecto, además de posibles cambios que se podrían realizar sobre el mismo, teniendo como referencia los objetivos expuestos en la introducción.

2 DEFINICIÓN Y ANÁLISIS DEL PROBLEMA

Como se ha expuesto en la introducción, el objetivo a la hora de plantear el problema es que este se relacione con un reto contemporáneo como es la crisis sanitaria causada por la Covid-19. También se buscaba que el problema supusiese un reto real, y se ajustase al tiempo y especificaciones típicas de un trabajo de final de máster. Por último, tanto en el planteamiento como la solución, que diseñaré posteriormente, tendrá que ser susceptible de cambios, sin que estos impliquen una gran reestructuración del resto de elementos originales.

La gestión hospitalaria ha supuesto un quebradero de cabeza desde tiempos inmemoriales, que acrecentada por una crisis sanitaria supone todo un reto llevar a cabo. Al final con unos recursos médicos limitados es fundamental saber organizarlos para poder resolver la crisis en el menor tiempo posible. Inspirados por las campañas de vacunación llevadas a cabo durante la crisis del Covid-19 se nos ocurrió plantear un problema donde, con un número determinado de pacientes y médicos, hubiese que encontrar la manera óptima de gestionar las dosis de un medicamento. Los pacientes y médicos están sujetos a restricciones como puede ser el hecho de que no cualquier paciente puede recibir un fármaco concreto, a raíz de posibles alergias o por la naturaleza del propio medicamento que a veces lo restringe a determinados grupos de edad.

El caso resultante del análisis consistirá en un problema de satisfacción de restricciones. Los problemas de satisfacción de restricciones, o *constraint satisfaction problem* (Brailsford, Potts, & Smith, 2011), son ejercicios matemáticos donde hay que encontrar una solución que satisfaga una serie de condiciones definidas en el planteamiento del problema. Este tipo de retos se asocian directamente con tareas de planificación y asignación de recursos.

Nuestro ejercicio tendrá el siguiente enunciado:

El gobierno de una ciudad tiene la difícil tarea de suministrar un medicamento a su población. Para ello contará con una serie de médicos que se distribuirán por los distintos centros médicos donde atenderán a la ciudadanía.

Los centros médicos tendrán asignados una lista de pacientes, que, dependiendo de su edad, llevará un tiempo medicarlos; y que pueden no tolerar alguno de los medicamentos disponibles. También tendrán un valor asignado que indicará su prioridad. Además, dispondrán de una cantidad limitada de fármacos de cada tipo. La lista de pacientes está ordenada. Los pacientes se tienen que atender en ese orden obligatoriamente; pero no hace falta que se haya terminado de atender a un paciente para que otro médico atienda al siguiente, excluyendo aquellos que no se vayan a medicar.

Los médicos podrán desplazarse entre los distintos hospitales con un coste temporal fijo para todos los desplazamientos. Además, los médicos reducirán el tiempo que tardan en medicar a un paciente, en función a su veteranía.

Los medicamentos, a su vez, también tardarán un tiempo determinado en ser suministrados.

El objetivo del gobierno es optimizar los recursos de los que dispone para suministrar el fármaco al mayor número de ciudadanos con la mayor prioridad y en el menor tiempo posible.

Para asignar un fármaco a un paciente se tendrán en cuenta sus intolerancias. En caso de que tolere más de una opción, se le asignará cualquiera de las posibles.

El tiempo que se tarda en medicar a un paciente se calcula como la suma del tiempo asignado a este más el tiempo que se tarda en suministrar dicho fármaco, menos el tiempo que puede reducir el médico.

Propiedades de cada uno de los factores del problema.

Médicos:

- Localización actual: Centro de vacunación en el que se haya actualmente.
- Veteranía: Cuanta más veteranía más se reduce el tiempo que se tarda en medicar a un paciente

Centro:

- Pacientes: número de pacientes, clasificados por tipo
- Suministros: número de dosis de cada uno de los medicamentos
- Localización

Fármaco

- Tiempo que se tarda en suministrar

Pacientes

- Tiempo
- Tipo de medicamento que no tolera su sistema
- Prioridad

2.1 Ejemplos

Para ilustrar mejor el problema, voy a mostrar un ejemplo simple de cómo podría ser el problema.

Pongamos que tenemos que administrar un solo tipo de fármaco(F1), que tarda 10 minutos en ser suministrado, en dos hospitales (H1 y H2), donde tendremos un médico y una lista de dos pacientes en cada uno. Todos los pacientes tienen la misma prioridad y tardan 10 minutos en ser atendidos. También cabe destacar, que los médicos tienen un 0% de veteranía por lo que no reducen el tiempo de medicación y que cada hospital dispone de 2 unidades del fármaco F1.

Con este input, el problema nos quedaría algo así:

Médicos		
Nombre	Localización inicial	Veteranía
M1	**H1**	0%
M2	**H2**	0%

Fármacos	
Nombre	Tiempo
F1	10 minutos

Hospital 1 (**H1**)			
Unidades de **F1**		2	
Nombre Paciente	Tiempo	Prioridad	Intolerancias
P11	10	1	-
P12	10	1	-

Hospital 2 (**H2**)			
Unidades de **F1**		2	
Nombre Paciente	Tiempo	Prioridad	Intolerancias
P21	10	1	-
P22	10	1	-

Ante estos datos de entrada el algoritmo resultante deberá encontrar que la mejor solución para este problema pasa por que el Médico **M1** medique con el fármaco **F1** a los pacientes del hospital **H1** y que **M2** haga lo respectivo con los del hospital **H2**.

Si tomamos el ejemplo anterior y aumentamos la veteranía del médico M1 hasta el 50%, la solución del problema ya cambiaría porque lo óptimo sería que dicho médico, que ahora tardaría la mitad en medicar a los pacientes, se encargase de suministrar el fármaco a tres de los cuatro pacientes de forma que, tras 15 minutos, todos los pacientes estén medicados.

Es posible complicar aún más el problema y reducir la cantidad de unidades del fármaco F1 a solo una en el hospital H1. Si además de esto aumentamos la prioridad del paciente P11 en 1, aumentarán el número de soluciones, siendo aquellas en las que el paciente P11 es medicado, las admisibles.

Sobre ese problema base se puede seguir aumentando la talla, y por tanto la complejidad, añadiendo más fármacos, médicos y pacientes. Por ejemplo, podemos añadir un segundo fármaco(F2) y un tiempo de desplazamiento entre hospitales de 5 minutos. Además, vamos a mantener al médico M1 con un 50% de veteranía. Por último, vamos a añadir más pacientes en cada hospital con intolerancias a alguno de los fármacos, quedándonos el siguiente problema:

Fármacos	
Nombre	Tiempo
F1	10 minutos
F2	20 minutos

Médicos		
Nombre	Localización inicial	Veteranía
M1	**H1**	50%
M2	**H2**	0%

Hospital 1 (**H1**)			
Unidades de **F1**		5	
Unidades de **F2**		5	
Nombre Paciente	Tiempo	Prioridad	Intolerancias
P11	5	1	F2
P12	30	1	F1
P13	10	1	F1
P14	10	1	F1
P15	40	1	F1
P16	20	1	F2

Hospital 2 (**H2**)			
Unidades de **F1**		5	
Unidades de **F2**		5	
Nombre Paciente	Tiempo	Prioridad	Intolerancias
P21	10	1	F1
P22	10	1	F2
P23	20	1	F2
P24	20	1	F1
P25	5	1	F1
P26	30	1	F1

Dado este planteamiento, la solución pasa porque el médico M1 medique a los pacientes P11, P12, P15, P16, P21, P24, P25 y P26; y el médico M2 al resto, tardando un total 147.5 minutos. Es posible que un ser humano halle esta solución a mano, pero es también existe la posibilidad que le lleve una cantidad de tiempo considerable, teniendo en cuenta la gran cantidad de combinaciones que existen; generando así la necesidad de encontrar un método de resolución dentro de la inteligencia artificial. Si tomamos el ejemplo anterior y eliminamos las intolerancias de los pacientes de repente nos encontramos que al paciente P11 pueden medicarle con el fármaco 1 y el fármaco 2, además puede hacerlo tanto el médico 1 como el médico 2, dando lugar a cuatro posibilidades para un paciente. Con el paciente 2 pasaría lo mismo dando lugar a otras cuatro posibilidades, resultando en 16(4^2) posibles soluciones del problema. Si extendemos esto a los 12 pacientes del ejemplo tendríamos un total de 16.777.216 posibles soluciones.

Nuestro objetivo es tomar esta estructura de problema y aumentar la talla del problema a una escala mucho mayor, en torno a los 500 pacientes por hospital para comprobar la eficiencia de un algoritmo genético.

La dificultad del problema residirá en el hecho de que, al aumentar el número de pacientes, la cantidad de posibles soluciones crecerá exponencialmente dando lugar a una gran explosión combinatoria, de forma que no nos bastará con realizar una simple búsqueda o emplear técnicas de fuerza bruta para su resolución. Además, al tener un espacio de búsqueda tan grande, lo más probable es que no demos con la mejor solución del problema, siendo nuestro objetivo ver cómo podemos aplicar técnicas de inteligencia artificial para dar con la mejor solución posible.

3 ESTADO DEL ARTE

Durante la introducción y el planteamiento del problema he comentado como vamos a aplicar técnicas de inteligencia artificial para resolver el caso planteado, pero en ningún caso hemos explicado lo que es este campo, ni cómo funcionan las técnicas que vamos a evaluar.

3.1 INTELIGENCIA ARTIFICIAL

La inteligencia artificial (Norvig & Russell, 1995), como hemos adelantado en la introducción, busca dotar a los sistemas computacionales de habilidades propias de los seres vivos, como el reconocimiento de formas o el razonamiento lógico. Debido a la ambigüedad en su definición, este campo puede englobar una gran cantidad de técnicas que van desde una búsqueda simple hasta sistemas de procesamiento de lenguaje natural; y su nacimiento se puede datar junto al de la misma computación, en el siglo XIV; y su popularización a mediados del siglo pasado.

Esto va a ser un arma de doble filo ya que, por una parte, vamos a disponer de muchas opciones a la hora de resolver el problema planteado; pero, por otra parte, vamos a tener que escoger con precisión a que área vamos a ceñirnos para evitar que el esfuerzo sea en balde. Por ejemplo, sería poco plausible resolver el ejercicio mediante una red neuronal ya que no disponemos de suficientes datos de muestras; por lo que sería poco fructífero emplear horas de trabajo diseñando una arquitectura compleja, que se viera lastrada por esta cuestión.

Después de analizarlo, se llegó a la conclusión lógica de que la manera más eficiente de resolver nuestro problema sería empleando métodos de búsqueda, y se va a plantear el problema como un espacio de búsqueda compuesto por todas las soluciones que satisfagan las restricciones especificadas en el enunciado.

Es por ello que este apartado comenzará con los dos tipos de búsquedas disponibles, la búsqueda a ciegas y la búsqueda informada.

3.2 BÚSQUEDAS A CIEGAS

Las búsquedas a ciegas (R, 2019), o no informadas, son todas aquellas estrategias que no disponen de información sobre el problema que están resolviendo más allá del propio estado. También saben si su estado actual es una solución, o estado objetivo; pero no cuan cerca están de una solución. Este tipo de búsquedas iteraran sobre los distintos estados en un espacio de búsqueda en forma de árbol. Todas tendrán en común que serán búsquedas completas, ya que exploraran todas las posibles soluciones, y que tendrán un coste computacional que escalará exponencialmente respecto al espacio de búsqueda, haciendo que el coste para talla grandes sea inasumible.

Los dos tipos de búsquedas no informadas más populares son la búsqueda en anchura y en profundidad. La búsqueda en anchura expande todos los nodos de un mismo nivel antes de empezar a expandir los del siguiente, garantizando que encontrará la solución más cercana en términos de profundidad. La búsqueda en profundidad, por el contrario, buscará expandir siempre el nodo más profundo, teniendo como principal

ventaja que no necesita almacenar una cola de estados por explorar, pero siendo susceptible de entrar en bucles.

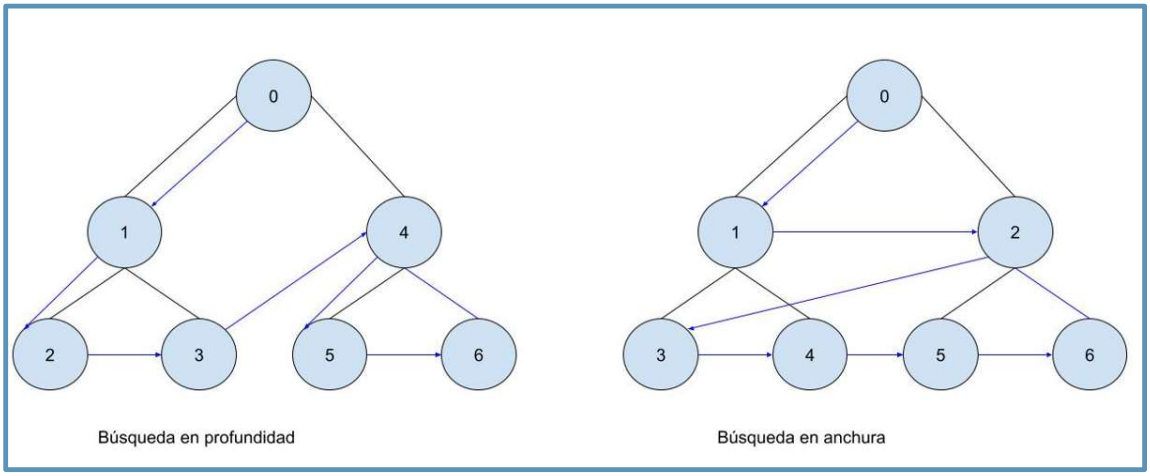


Ilustración 2-Búsqueda en profundidad(izquierda) y búsqueda en anchura(derecha)

Estas dos aproximaciones son las más básicas, y existen variaciones de estas que buscaran paliar las desventajas de cada uno de los algoritmos, como la búsqueda en profundidad limitada, donde se expandirá hasta cierta profundidad; la búsqueda bidireccional, que recorrerá el árbol a la vez desde el nodo raíz y un nodo objetivo (Intriago, 2022); o la búsqueda con profundidad iterativa (Shivam, 2022), que seguirá un procedimiento similar a la profundidad limitada, pero aumentando el límite cuando se llegue a un nodo hoja.

Las estrategias de búsqueda a ciegas son adecuadas cuando no es posible evaluar cuan prometedores son los nodos que vamos expandiendo. Históricamente se han aplicado en áreas como los juegos de mesa, la planificación de rutas o la optimización de recursos.

En caso de que dispongamos de alguna manera de evaluar los distintos nodos es muy recomendable utilizar esa información, no solo por ser una buena práctica, sino porque puede ahorrar muchos recursos computacionales. Este tipo de estrategias más eficientes se conocen como búsquedas informadas.

3.3 BÚSQUEDAS INFORMADAS

Cuando trabajamos con espacios de búsqueda complejos habitualmente requeriremos técnicas más sofisticadas para poder encontrar una solución satisfactoria. Es por ello por lo que se deben utilizar búsquedas informadas, que se diferencian de las anteriores en que se emplea la información que nos proporciona el entorno para guiarnos hacia la meta.

El principal mecanismo de este tipo de búsquedas es la función de evaluación o heurística (Reeves, 1991). Esta función se utiliza para calcular la idoneidad de los distintos nodos a expandir, asignándoles un valor numérico, lo que permite que la búsqueda tome aquellos caminos que se consideran más prometedores, dando lugar a muchas estrategias de búsqueda.

Primero surgieron las técnicas voraces que se basan en expandir siempre el nodo más idóneo, presentando la desventaja de que es muy susceptible de entrar en bucles.

Además, si no disponen de un sistema de backtracking (Bhalla, Lynce, Sousa, & Marqueés-Silva, 2003), que le haga retroceder para probar opciones que a priori no son tan prometedoras, hacen que este tipo de búsquedas no sean completas.

A raíz de esta estrategia aparece la búsqueda A* (Hart, Nilsson, & Raphael, 1968), que mediante una ligera modificación de la función heurística solventa gran parte de los problemas de la búsqueda voraz. Esta calcula el coste de expandir un nodo como el coste de expandirlo más el coste de expandir a sus antecesores, lo que provoca que este tipo de búsquedas sean completas. Por desgracia, al igual que las otras búsquedas que he repasado en este documento, tiene problemas de rendimiento cuando la talla del problema crece. La función de evaluación es el factor más importante de este tipo de algoritmo porque de ella va a depender que se encuentre una solución factible. Al igual que con la función heurística, no existe una función de evaluación universal para todo tipo de problemas, lo que provoca que sea responsabilidad del ingeniero diseñar una función optima.

Las búsquedas anteriormente descritas presentan el problema de que es fácil que se estanquen en mínimos locales, lo que quiere decir que alcanzarán un valor mínimo de una parte del espacio de búsqueda, que por la naturaleza de las funciones de evaluación les impedirá salir de ese espacio acotado. Es aquí donde tenemos que empezar a plantearnos utilizar búsquedas que no recorran de forma sistemática el árbol de búsqueda y que utilicen métodos más abstractos para encontrar la solución, adentrándonos en el campo de las técnicas metaheurísticas.

Tanto las búsquedas tradicionales como las heurísticas tienen un enfoque determinista y son capaces de encontrar una solución óptima en un espacio de búsqueda acotado. Las búsquedas metaheurísticas, por el contrario, son algoritmos estocásticos que se basan en introducir elementos aleatorios en el proceso de búsqueda lo que hace que trabajen mejor con espacios de búsqueda más grande, siendo capaces de encontrar soluciones óptimas en un tiempo razonable. No obstante, cabe recalcar que en ningún caso garantizan que puedan encontrar dichas soluciones por su carácter aleatorio. Los tres tipos de búsqueda metaheurística más expandidos son el temple simulado, la búsqueda tabú y los algoritmos genéticos.

3.4 Temple simulado

El algoritmo del temple simulado (Sancho, 2020) está inspirado, y recibe el nombre, de las técnicas de temple que se utilizan en la industria metalúrgica. Este proceso consiste en aumentar mucho la temperatura del metal para luego ir reduciéndola poco a poco, con el objetivo de que adquiera la forma que nosotros deseemos. De la misma forma, en una búsqueda informada, podemos saltar a un estado aleatorio cuando el proceso de búsqueda, de se encuentre estancado, para así salir de mínimos, o máximos, locales en los cuales podemos habernos parado por la propia entropía de la búsqueda. El algoritmo permite establecer un parámetro de temperatura para controlar el número de iteraciones del algoritmo, con la desventaja de que cuanto mayor sea este valor, mayor será la probabilidad de generar un estado peor al actual.

En la ilustración 3, se puede observar el proceso de salir de mínimos locales mediante el temple simulado. La línea negra representa las distintas soluciones del problema y la línea roja los intentos del algoritmo de salir del mínimo local en el que se halla estancado. En la gráfica también se representa el parámetro de temperatura, que afectará directamente a la distancia que se recorra con dichos saltos.

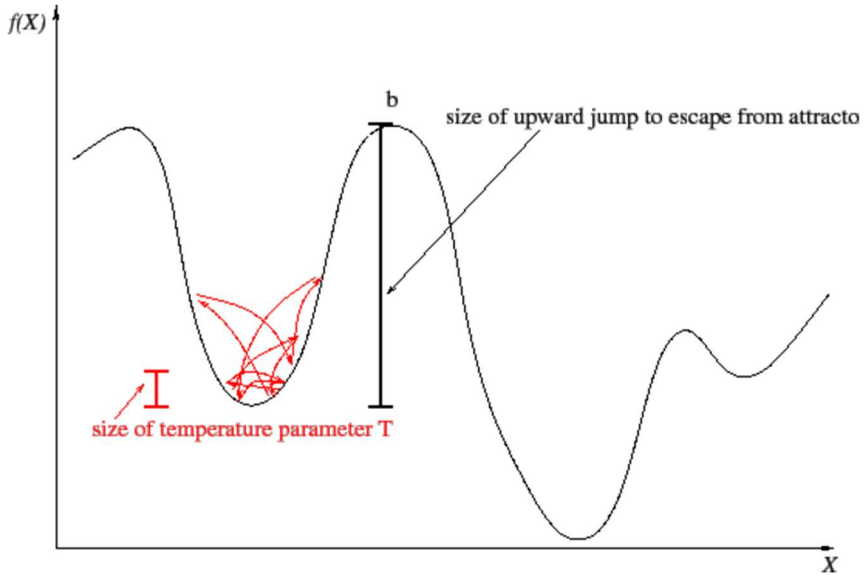


Ilustración 3-Gráfica del temple simulado

Como es lógico, esta técnica no siempre mejorará la calidad de nuestras soluciones, pero es útil emplearla para aumentar la variedad de estas, con un coste insignificante.

3.5 BÚSQUEDA TABÚ

La búsqueda tabú (Glover & Laguna, 1997) es otra técnica metaheurística cuyo objetivo es encontrar soluciones fuera de mínimos locales. Este tipo de búsqueda utiliza una estructura de memoria a corto plazo, llamada lista tabú, donde almacena las soluciones que fueron visitadas recientemente, para así excluirlas. Esta lista puede almacenar también soluciones que contengan determinadas propiedades, para así prevenir ciertos movimientos. Al igual que la estrategia de temple simulado, se trata de una estrategia metaheurística, por lo que en ningún caso puede asegurarte encontrar la mejor de las soluciones, ya que su utilidad reside en poder llegar a encontrar soluciones que las búsquedas heurísticas no podrían hallar por su naturaleza iterativa.

En la siguiente ilustración, se ha añadido una gráfica para mostrar el comportamiento de la búsqueda tabú. La línea azul representa las posibles soluciones, los puntos rojos los mínimos locales y el punto verde es el mínimo global; la mejor solución del problema. Mediante flechas se puede ver como el algoritmo recorre el espacio de soluciones y avanza hacia el mínimo global, recorrido que solo es posible si se tienen en cuenta soluciones peores a las ya encontradas.

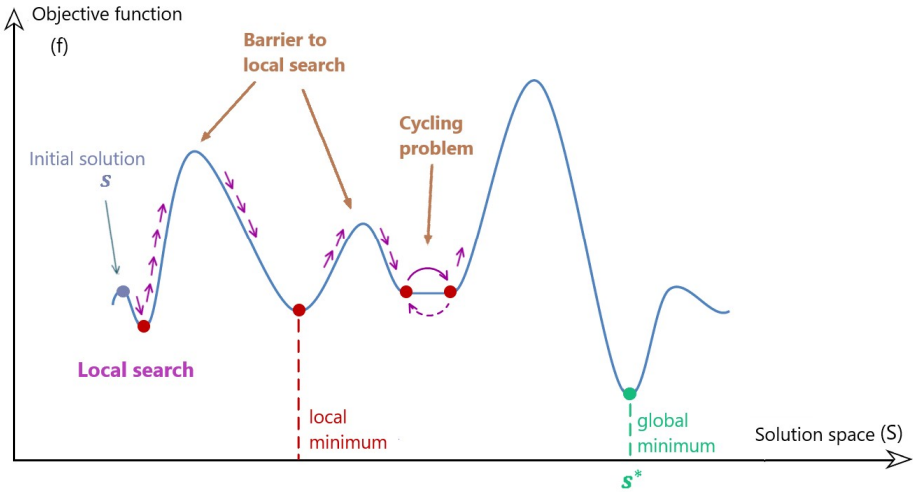


Ilustración 4-Búsqueda Tabú

3.6 ALGORITMOS GENÉTICOS

Los algoritmos genéticos (Yang, 2021) son un método para resolver problemas de optimización con restricciones que se basan en el modelo de Charles Darwin (khan academy, 2021) sobre la evolución de las especies.

El biólogo británico explicó que cuando aparecen rasgos en una especie que favorecen su supervivencia, estos se volverán comunes al resto de la población con el paso del tiempo, debido a un mecanismo conocido como selección natural. Este proceso se da cuando los individuos con estas características especiales veían como aumentaban sus posibilidades de sobrevivir al entorno y, por lo tanto, de llegar a reproducirse, haciendo que su progenie pudiese heredar dicha característica.

En 1975, el científico John Henry Holland expuso al mundo (Holland, 1975) como esta idea de la evolución puede llevarse al campo de la inteligencia artificial, dando lugar a los algoritmos genéticos.

Este tipo de algoritmos consideran al conjunto de soluciones admisibles como su población, y emplean distintas técnicas no deterministas para generar nuevas soluciones a partir de los ya existentes.

Los algoritmos genéticos brindan muchos mecanismos para resolver los problemas que plantean las anteriores técnicas. Por ejemplo, pueden implementar un proceso de mutación para modificar de manera aleatoria las soluciones encontradas, mitigando así el problema de mínimos locales. Además, a la hora de reproducir a los individuos (las soluciones) podemos seleccionar a los individuos que pasaran su material genético siguiendo distintos tipos de emparejamiento, que afectarán tanto al rendimiento como a la calidad de la solución, siendo el criterio del investigador el que deba decidir cuál es la óptima dependiendo del tipo de problema.

A su vez esto será el principal inconveniente de estos, será la necesidad de una configuración optima que responda a las necesidades del problema, para evitar que la

población sufra una convergencia prematura, lo que en última instancia significa que el diseñador debe tener un gran conocimiento del problema a resolver.

Los algoritmos genéticos se dividen en 5 fases: Evaluación, emparejamiento, cruzamiento o reproducción, mutación y cribado.

3.6.1 Evaluación

El primer paso de los algoritmos genéticos es saber cómo se evalúa a los individuos de la población inicial, que al fin y al cabo serán las soluciones del problema. Para ello será necesario disponer de una función de evaluación (Kour, Sharma, & Abrol, 2015) que nos indicará cuan bueno es dicho individuo, típicamente otorgándoles un valor numérico llamado métrica. Este valor sirve principalmente para saber cuan buenas son las soluciones del problema además de servirnos como referencia en el resto de las fases.

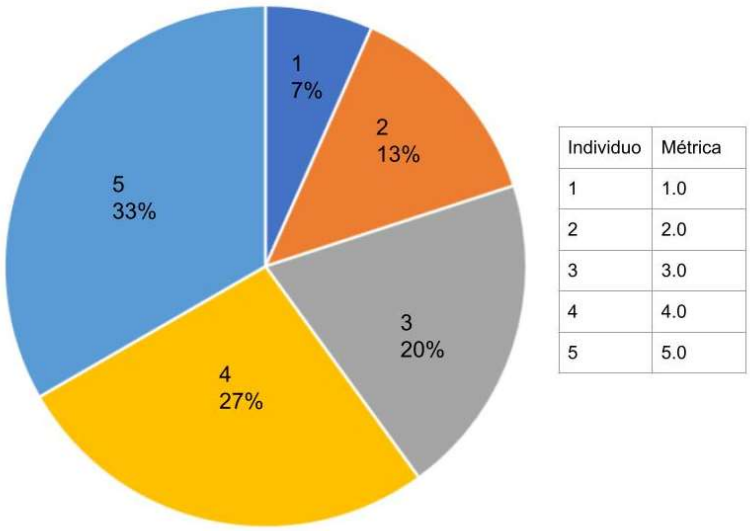
No existe un criterio fijo sobre cómo debe ser dicha métrica, ya que dependerá del problema abordado, pudiendo hacer que esta función sea creciente para problemas de maximización o decreciente para problemas de minimización. Además, también se podrá utilizar para penalizar a los individuos en caso de que se salten alguna restricción del problema.

3.6.2 Selección

El proceso de selección será el encargado de formar parejas en la población para que estas se reproduzcan y den lugar a nuevos individuos. Al igual que con el resto de las fases, existen distintas maneras de implementar la selección que afectarán al resultado.

Los mecanismos más comunes son los emparejamientos basados en ruleta (Pencheva, Atanassov, & Shannon, 2009). Por su nombre se puede intuir que con este tipo de emparejamiento se intenta emular una ruleta del casino donde se escogen dos individuos de forma aleatoria. Para darle más peso a la métrica, esta selección se puede ponderar de forma que los individuos con mejor métrica tienen mayor probabilidad de ser seleccionados.

En la siguiente ilustración, se puede ver el proceso de emparejamiento por ruleta. Dependiendo de lo prometedora que sea la solución, tendrán más o menos probabilidades de ser seleccionados para reproducirse y así pasar sus genes a la siguiente generación de individuos.



Individuo	Métrica
1	1.0
2	2.0
3	3.0
4	4.0
5	5.0

Ilustración 5-Seleccion por Ruleta

También tendremos los procesos de selección basados en torneo (Fang & Li, 2010). Con estos buscaremos simular una tabla de un torneo deportivo donde se realizarán "enfrentamientos" individuales, que ganará el individuo con mejor métrica, que pasará a la siguiente fase. En la fase final solo quedarán dos individuos que serán los que realicen el cruce.

En la siguiente imagen, se ilustra un proceso de selección de este tipo, donde ocho individuos compiten por ser elegidos para el cruce. Al finalizar la ejecución son el individuo 1 y el 7 los "ganadores" que procederán a reproducirse dando lugar a dos nuevos individuos, tal y como hemos visto en el apartado anterior.

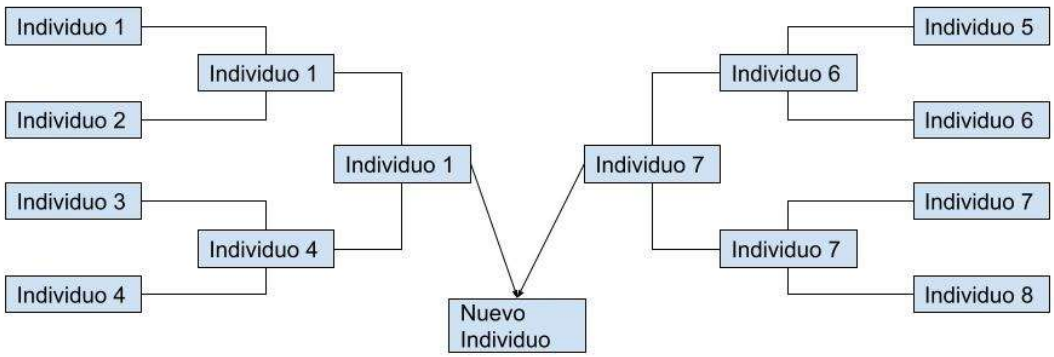


Ilustración 6 Selección por torneo

3.6.3 Reproducción

Para llevar a cabo el proceso de reproducción típicamente se escogen dos individuos aleatorios de la población, un punto de cruce también aleatorio, y se forma un individuo nuevo con una parte de la cadena de cada uno de los progenitores, tal y como se ve en la imagen inferior.

En la ilustración 4, se muestra un ejemplo de una operación de cruce. Cada una de las cadenas está formada por ocho genes. Durante el cruce se generan dos nuevos individuos(cadenas), donde cada uno de ellos contendrá la mitad de los genes de uno de los progenitores.

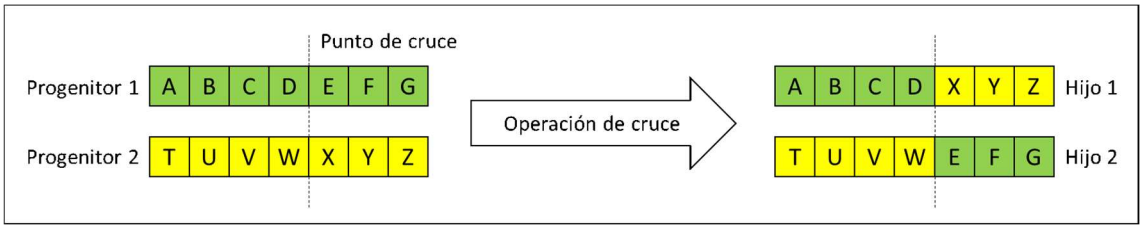


Ilustración 7 operación de cruce

Al igual que el emparejamiento, el cruce tiene distintas variaciones. El cruce aritmético es el que más se utiliza, pero presenta inconvenientes en cuanto a la variación genética, que crearan patrones en las secuencias de cromosomas. Como solución, se propone que el punto de cruce no sea aleatorio, y que dependa de la ponderación de los individuos progenitores. Esto sirve como solución parcial porque puede provocar que los individuos con mejor métrica se adueñen de las cadenas impidiendo grandes variaciones y estancando el progreso, existiendo como solución el cruce uniforme (Xiao-Bing Hu, 2021).

Este tipo de cruce recorre gen por gen la cadena del hijo, asignando a cada uno, el valor del mismo cromosoma de uno de los padres. Esta manera de proceder evita los problemas derivados de las técnicas anteriores ya que los cromosomas podrían adquirir cualquier valor siempre que esté en presente en un individuo de la población. Para ver de cuál de los padres coge cada cromosoma, se pueden seguir varias estrategias, pero lo más común es darle la misma probabilidad a los dos padres de forma que sea equiprobable. También se puede variar el número de cromosomas que se le da a cada uno para que el hijo tenga más genes del progenitor más prometedor.

3.6.4 Mutación

A pesar de los intentos de que los genes puedan tener todos los valores posibles, cubriendo más soluciones del espacio de búsqueda, sigue estando el hecho de que mediante la reproducción únicamente podrán tener valores existentes en los individuos de la población inicial. Es aquí donde entra en juego la mutación (Falco, Cioppa, & Tarantino, 2002).

La mutación es otro proceso esencial en la evolución de las especies ya que hace que estas adquieran características que no están en la línea genética. Un caso muy claro lo vemos en los virus. Estos entes debido a lo rápido que se reproducen sufren muchas alteraciones en sus cadenas genómicas. Estas mutaciones provocaban que cambiase la viralidad o los efectos, de forma que cuando una de las células del virus aumentaba su capacidad de replicarse se expande rápidamente dando lugar a una variante del virus e incluso inutilizando las vacunas existentes contra el mismo. Esto causa por ejemplo que todos los años se tengan que desarrollar nuevas vacunas contra la gripe ya que las existentes dejan de ser efectivas.

La mutación, por lo tanto, lo definiremos como el proceso mediante el cual vamos a introducir de manera aleatoria alteraciones en la cadena de los individuos, de forma que aumentará la variación genética, porque dará la posibilidad de que se generen genes nuevos que por el factor de aleatoriedad no estuviesen presentes en los

individuos de la población; y a su vez evitará un estancamiento en el proceso de mejora, por el mismo motivo.

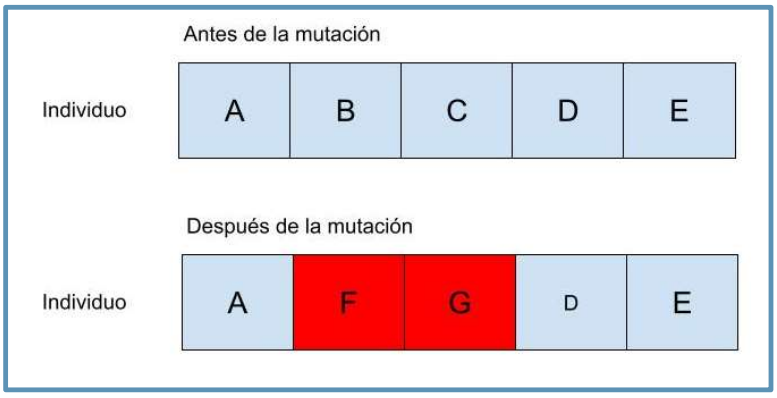


Ilustración 8-Ejemplo de Mutación en una cadena cromosómica

Este proceso se llevará a cabo justo después del cruce genético y la generación de individuos nuevos.

3.6.5 Cribado

El proceso de cribado (Shukla, Pandey, & Mehrotra, 2015) es el último paso del algoritmo genético y su propósito es eliminar parte de la población, lo cual no es imprescindible en todas las implementaciones de este tipo de algoritmos.

Podremos optar por tres tipos de cribado. El primero será el cribado aleatorio, que elegirá tantos individuos como se quiera de forma totalmente aleatoria y los descartará. El problema de esta configuración es que se corre el riesgo de eliminar los mejores individuos.

Luego tendremos el cribado de peores individuos donde se descartarán los peores individuos de la población. Esta ejecución no tiene el problema de la anterior de descartar individuos buenos, pero disminuye la variabilidad genética.

Por último, tenemos el cribado ponderado, donde los mejores individuos tendrán menos posibilidades de ser cribados. Esta opción es un punto medio entre las dos anteriores, ya que respeta más la variabilidad y le das más peso a los mejores individuos, sin llegar a sacrificar ninguna de las dos.

3.7 DIFERENCIAS ENTRE LAS TÉCNICAS METAHEURÍSTICAS

Si bien es cierto que la filosofía detrás de las tres técnicas es emplear métodos aleatorios para explorar el espacio de búsqueda, no serán igual de eficaces. Los algoritmos genéticos se inspiran en la evolución natural, utilizando conceptos asociados a esta como la reproducción y la mutación, para encontrar soluciones. Esto le permite trabajar con una amplia gama de problemas de optimización, además de buscar soluciones en espacios de búsqueda inmensos. Por otro lado, el temple simulado se basa en aceptar soluciones peores con el objetivo de escapar de mínimos locales, por lo que lo óptimo es utilizarlo junto a otro tipo de búsqueda heurística que encuentre primero la solución de la que quieres escapar. Con las búsquedas tabú pasa lo mismo,

ya que para que el algoritmo funcione es recomendable conocer las soluciones que quieres almacenar en la lista, teniendo además el inconveniente de que una talla demasiado grande puede causar problemas de rendimiento y memoria.

Cada una de estas técnicas tiene sus propias ventajas y desventajas en términos de eficacia y eficiencia, por lo tanto, la elección de la técnica adecuada dependerá en gran medida de la naturaleza del problema y los recursos disponibles

4 DISEÑO DE LA SOLUCIÓN

A partir de aquí voy a trabajar en el desarrollo del algoritmo. Además, hemos visto la sección del estado del arte donde he expuesto el estado actual de las técnicas heurísticas, esbozando los algoritmos que evaluaremos

En la exposición del problema he hablado de como los casos de prueba van a suponer una gran carga combinatoria, lo que nos da a entender que el algoritmo que escojamos va a tener que rendir bien en este tipo de situaciones.

4.1 ELECCIÓN DEL ALGORITMO

Las técnicas de búsqueda a ciegas o no informada tienen su interés ya que, para espacios de búsqueda más pequeños, nos permiten generar todos los nodos posibles, asegurándonos que la solución obtenida es la mejor posible. Obviamente, como nosotros pretendemos crear casos de prueba con una gran carga combinatoria, este tipo de algoritmos quedan totalmente descartados, ya que podrían tardar eones en terminar su ejecución. Además, es considerado una buena práctica utilizar la información del problema siempre que sea posible.

Una vez hemos especificado que la búsqueda a aplicar va a ser informada, tendremos que seleccionar aquella estrategia de búsqueda que vaya a resolver nuestro problema mejor.

La estrategia de búsqueda voraz es la primera que he tenido en cuenta. Utilizando la información dada por el problema, principalmente el tiempo de suministro de los fármacos, se hace factible aplicar esta estrategia ya que, si asignamos siempre el fármaco de menor coste a cada paciente, podemos llegar a obtener soluciones factibles siempre que haya fármacos suficientes. El problema viene cuando no disponemos de una gran cantidad de fármacos, lo cual provocaría que los primeros pacientes de la lista sean los que medicamos, siendo imposible cumplir con el objetivo de medicar al máximo número de pacientes posible.

La siguiente estrategia heurística que se valoró fue el algoritmo A*. Con esta técnica podemos resolver los problemas de la búsqueda voraz, ya que al expandir el nodo que minimice la función de coste, podemos dar lugar a soluciones más interesantes. A pesar de esto, nos topamos con el problema intrínseco a las heurísticas tradicionales, el rendimiento. Al aumentar la talla del problema, el espacio de búsqueda crece exponencialmente, llegando rápidamente a situaciones donde el tiempo es inasumible. Dado estos problemas pensamos en aplicar técnicas de poda para reducir rápidamente el espacio de búsqueda, pero dada la naturaleza del problema, que básicamente no nos da ningún indicador de que una rama es susceptible de ser eliminada, es imposible implementarlos.

Teniendo en cuenta todo lo mencionado anteriormente, es obvio que lo que más va a limitarnos es el gran espacio de búsqueda siendo imposible recorrerlo de una forma sistemática, surgiendo la necesidad de utilizar técnicas más abstractas como son los algoritmos de búsqueda local, parte de la metaheurística. Con algoritmos de búsqueda local nos referimos a todas aquellas técnicas que no como un conjunto de caminos a

recorrer, sino que tienen consciencia del estado actual y generan nuevas soluciones a partir de este estado.

Dentro de todos los algoritmos el que más me llamó la atención fueron los algoritmos genéticos y me pareció la mejor elección por varios motivos. El principal beneficio de utilizar un algoritmo genético es su forma de recorrer el espacio de búsqueda. A diferencia de las búsquedas convencionales, los AG no siguen un patrón determinista y dependen ligeramente del azar durante la ejecución. Esta aleatoriedad, siendo un rasgo que pueda parecer contraproducente, hará que alcancemos soluciones que no sería posible alcanzar con las búsquedas heurísticas, por la talla del problema.

Las otras técnicas metaheurísticas que hemos explicado también podrían aplicarse a nuestro problema por los mismos motivos. El inconveniente de la búsqueda tabú es que necesita una gran lista en memoria para almacenar las soluciones, haciendo difícil trabajar con una talla de problema tan grande. En cuanto al temple simulado, también presenta problemas a la hora de trabajar con espacios de búsqueda tan grandes como el de nuestro problema, además de que computacionalmente puede ser muy costoso.

Si nos centramos en los beneficios de aplicarlo a nuestro problema concreto también podemos discernir unos cuantos. El primero es que, con el número de pacientes vacunados, la prioridad media y el tiempo transcurrido, es fácil diseñar una función de evaluación que califique a cada uno de los individuos. Además, el hecho de que los pacientes del problema se presenten en forma de lista ordenada, facilita mucho la creación de cadenas cromosómicas y su manipulación. Por último, a nivel personal, ya había trabajado con este tipo de algoritmos con anterioridad por lo que conocía su funcionamiento, a grandes rasgos.

Como se ha visto en el estado del arte, será indispensable diseñar el algoritmo genético teniendo en mente el problema que tratamos de resolver. Es por ello por lo que voy a recorrer cada una de las partes que lo componen explicando como lo voy a implementar.

4.2 INDIVIDUO

La unidad básica de un algoritmo genético es el individuo, que representa una posible solución del problema, y puede tener muchas formas distintas dependiendo del problema. En nuestro caso se decidió que un individuo se representase como una cadena. Dicha cadena será una solución del problema y estará compuesta por una sucesión de genes. Cada uno de los genes representará una parte de la solución que para nuestro caso será un paciente del problema, es decir, una solución de nuestro problema consistirá en una sucesión de pacientes y el tratamiento que se le ha aplicado. Por ejemplo, si el paciente P1 recibe el fármaco F1 de mano del médico M1, su gen se representará como P1-F1-M1. En la imagen inferior, se puede ver de manera gráfica la cadena de un individuo.

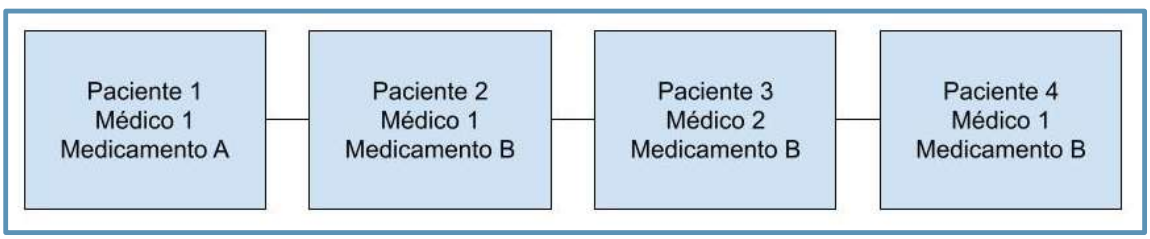


Ilustración 9-Representación de la cadena de un individuo.

En caso de que no queden medicamentos para un paciente, no tendrá ni medico ni fármaco asignado, otorgándole un valor de 'no medicado'.

En cada ejecución habrá cientos de estos individuos agrupados en lo que conoceremos como la población del algoritmo. En nuestro caso vamos a limitar el tamaño de esta población para evitar problemas de rendimiento o de memoria, mediante un parámetro que podremos cambiar en cada ejecución.

Al comenzar la ejecución de algoritmo, se debe generar una población inicial de manera aleatoria sobre la que aplicaremos las técnicas propias de los algoritmos genéticos para avanzar hacia una solución óptima del problema. Para generar esta población inicial, vamos a recorrer la lista de pacientes que recibimos en el enunciado, asignándole a cada uno un fármaco aleatorio. El tamaño de la población inicial también estará fijado mediante un parámetro.

En la ilustración 12, se puede ver representados esos tres conceptos de individuo, gen y población.

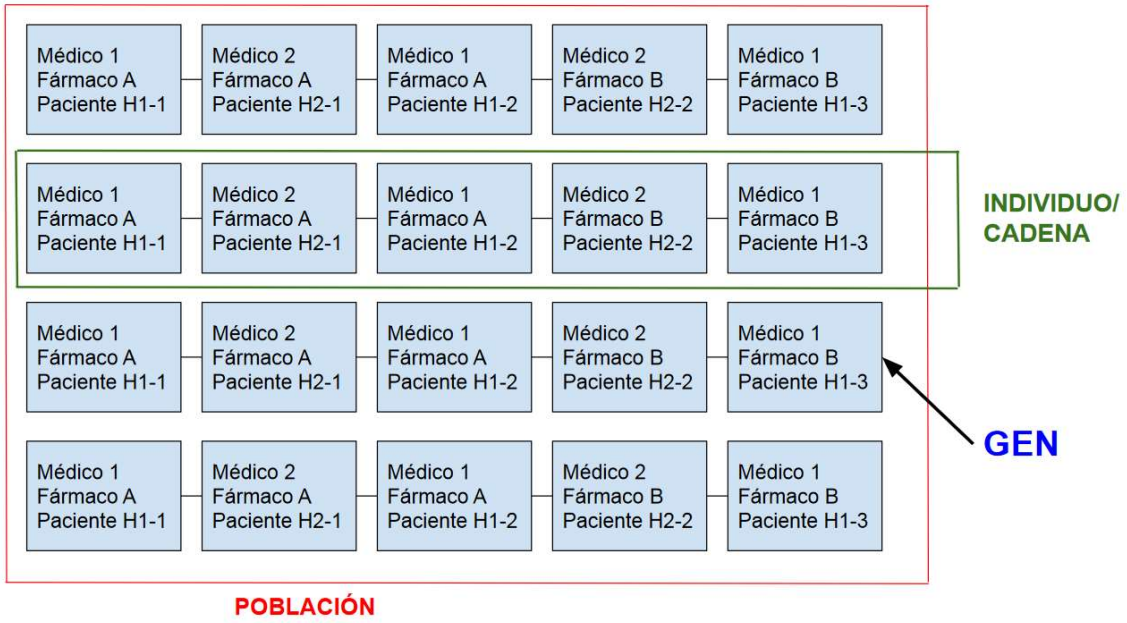


Ilustración 10: Cadenas cromosómicas

4.3 FUNCIÓN DE FITNESS

Una vez definido la forma que van a adoptar los individuos de nuestro problema, tendremos que buscar una manera de evaluar como de buenas son las soluciones que estos representan. Para ello vamos a definir lo que se conoce como una función fitness que se utilizará para asignar a cada individuo un valor numérico, que llamaremos métrica, para conocer la calidad de su solución y poder compararlos.

Para ello se tendrá que identificar los elementos del problema que hacen que una solución sea mejor o peor. El enunciado del problema establece que los objetivos son medicar al mayor número de pacientes, maximizando la prioridad y minimizando el tiempo que lleva medicarlos. También recalca que los más importante es maximizar el número de pacientes vacunados, después la prioridad de dichos pacientes y lo menos relevante será el tiempo total.

Tomando estos tres elementos, se optó por crear una sola métrica combinándolos, ya que facilitaría mucho la comparación de los individuos y la representación gráfica de las soluciones del problema. La otra opción que valoramos fue considerar esos tres valores como métricas distintas, lo que dificultaría la comparación de los individuos.

La dificultad de combinarlos en una sola cifra residía en el hecho de que no todos los objetivos tienen la misma importancia. Para respetar esta restricción del problema, decidimos que las cifras más cercanas a las unidades representarían el tiempo que se ha tardado en vacunar, que asumíamos que, a no ser que aumentásemos la talla del problema a cotas inasumibles, nunca debería llegar a superar la cifra de 99999. Es por ello por lo que, en la métrica, desde las decenas de millar hasta las unidades representarían el tiempo de vacunación. Lo siguiente que se debe añadir es la prioridad media que ha alcanzado la solución. Como la prioridad de los pacientes se mueve entre 0 y 99, bastará con otorgarle las centenas de millar y las unidades de millón. Por último, el número de pacientes vacunados, que es el objetivo más importante, se representa desde las decenas de millón en adelante. Es por ello por lo que la métrica se calculará como los pacientes vacunados *10000000 + la prioridad media * 100000 + el tiempo total de medicación.

Para que quede más claro, si suponemos que en una solución se ha medicado a 5 pacientes con una prioridad media de 80 y en un lapso de 500 minutos, su métrica se calculará como 5*10000000 + 80*100000 + 500, dando como resultado una métrica de 58000500.

Como apunte, el tiempo siempre será múltiplo de 5 por lo que se debe dividir está cifra entre 5 antes de introducirla en la métrica para ahorrar memoria. También cabe recalcar que el tiempo total se corresponderá a lo que haya tardado el médico que termine el último de suministrar fármacos. Por ejemplo, si tenemos un médico que ha tardado 50 minutos y otro que ha tardado 80, el tiempo total de ese individuo será igual a 80 ya que se asume que los médicos trabajan en paralelo.

En la siguiente ilustración se puede observar un esquema de cómo se configura la métrica del problema.

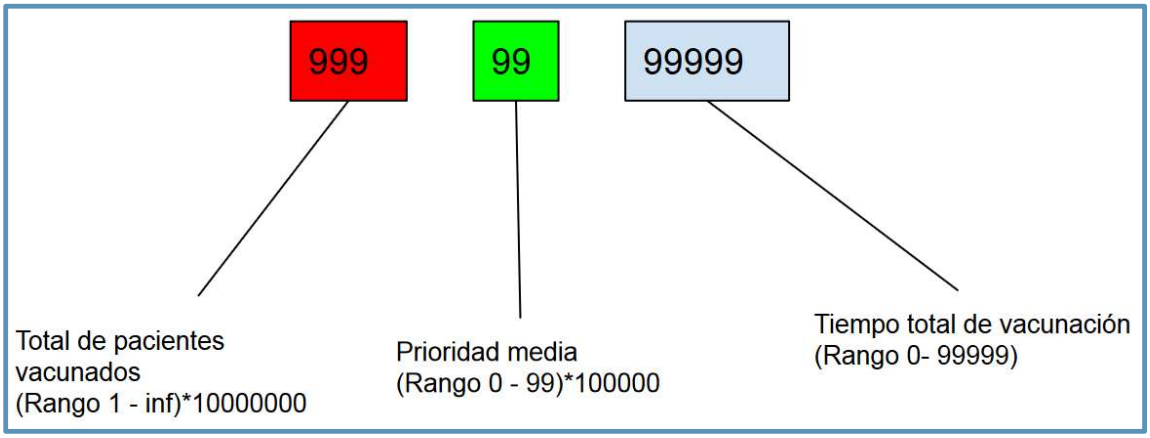


Ilustración 11 Esquema de la métrica

En este punto, nos percatamos de que con el enunciado podemos estimar una supuesta métrica ideal, y darle la vuelta a la métrica de cada individuo, de forma que esta midiese lo mucho que se ha aproximado a este valor ideal. Por ejemplo, si estableciésemos que la métrica ideal es 100 y la de uno de los individuos es 90, su métrica pasaría a ser 10 indicándonos que está a 10 puntos del valor que se busca. La principal utilidad de esto reside en que convertimos el ejercicio en uno de minimización de la métrica, estableciendo cero como el valor al que vamos a querer aproximarnos.

Además, nos permite tratar con números más pequeños, que serán más fáciles de entender, y ahorran espacio en términos de memoria.

Para realizar la aproximación de la métrica ideal, se tomará el número de pacientes total como el máximo de pacientes que se pueden medicar, en caso de que haya medicamentos para todos; sino será el número total de dosis lo que lo determine. Para la prioridad tomaremos el dato anterior y realizaremos la media de las prioridades de los n pacientes con prioridad más alta, siendo n el número máximo de pacientes que se pueden medicar. Por último, para el tiempo, se asignará el máximo que puede adoptar ese valor, es decir 99999. Viendo esto último se puede deducir que los individuos nunca van a poder tener una métrica de cero, así que su función será aproximarse lo máximo posible a este valor.

El proceso de calcular la métrica será muy simple. Se recorrerá la cadena del individuo sumando los pacientes vacunados, las prioridades y el tiempo del tratamiento, a los médicos que corresponda. Entonces dividiremos la prioridad entre los pacientes, para calcular la media, y sacaremos el tiempo máximo de los médicos disponibles. Este proceso se llevará a cabo después de la reproducción, sobre los nuevos individuos, y después de la mutación, sobre los individuos mutados.

Con el individuo y la función fitness definidos podemos empezar a estudiar cómo se van a diseñar las cuatro fases del algoritmo.

- Emparejamiento
- Cruce/Reproducción
- Mutación
- Relevo Generacional/Cribado

4.4 Emparejamiento

Para llevar a cabo el emparejamiento, vamos a implementar las dos técnicas explicadas en el estado del arte, por ruleta y por torneo; además de una adicional que llamaremos emparejamiento de mejores individuos.

Aquí se tiene que tomar la decisión de cuantas parejas queremos que se formen durante el emparejamiento, lo cual impactará directamente en el rendimiento y la calidad de las soluciones. Con esto en mente se decidió que nuestro algoritmo ofrecerá dos versiones de estos mecanismos. Con una formará tantas parejas como sea posible, que será típicamente la mitad del número de individuos; y con la otra formará una sola pareja. Generando todas las parejas posibles lo que se busca es que la totalidad de los individuos participen en el proceso de reproducción generando mayor variedad genética. En cambio, con la pareja única ganaremos en rendimiento, ya que el proceso de selección de parejas será presumiblemente el más costoso a nivel computacional, pero la mejora de las métricas será más lenta.

El emparejamiento por ruleta lo implementaremos con una versión aleatoria y otra ponderada. La versión aleatoria simplemente extraerá dos individuos cualesquiera de la población, mientras que la versión ponderada requerirá un mecanismo un poco más complejo donde se tenga en cuenta la métrica para esta selección. El algoritmo de ponderación será común siempre que queramos hacer una selección ponderada y tendrá la siguiente estructura:

```python
def seleccionPonderada(poblacion):
    #Se suman todas la metricas de la población
    maximo = sum([metrica_ideal - i.metrica for i in poblacion])

    #Se escoge un valor entre 0 y el sumatorio anterior
    pick = random.uniform(0, maximo)

    #Inicializamos la variable actual
    actual = 0

    #Por cada individuo de la población sumamos
    #la inversa de la métrica a la variable 'actual'
    #y si supera el valor aleatorio pick, entonces
    #será el individuo escogido.
    for i in range(0, len(poblacion)):
        #Es importante trabajar con la inversa de
        #la métrica ya que, al ser un problema
        #de minimización, estariamos dando más
        #peso a las peores métricas.
        actual += metrica_ideal - poblacion[i].metrica
        if actual > pick:
            return i
```

Ilustración 12-Algoritmo de selección ponderada

Tal y como se explica en los comentarios del código, va a ser importante trabajar con la inversa de la métrica en este proceso, ya que sino la ponderación funcionaria al revés de lo esperado. La inversa de una métrica se calcula como la métrica ideal menos la métrica del individuo.

El emparejamiento por torneo requerirá que se recorra la población formando los enfrentamientos típicos de estos, almacenando los ganadores que serán los que pasen a la siguiente ronda. Los emparejamientos serán totalmente aleatorios por lo que existe la posibilidad que no lleguen a la última ronda los mejores individuos de la población. Este proceso será el más costoso computacionalmente ya que su implementación requerirá que iteremos sobre los mismos individuos varias veces.

El emparejamiento de los mejores individuos va a seleccionar los individuos con mejor métrica y va a emparejarlos sucesivamente, de forma que el mejor se cruce con el segundo mejor, el tercero con el cuarto y así sucesivamente. Esta variante confía en que el cruce de los mejores individuos vaya a dar como resultado una progenie más prometedora, sin dejar por ello a los individuos con peor valoración fuera del proceso, dándoles la oportunidad de engendrar. Como ventaja, este reducirá la entropía de este proceso, pero, por otra parte, al no dar pie a que los peores individuos se reproduzcan con los mejores, puede que la mejora se estanque.

Si hablamos de rendimiento, la ruleta aleatoria será el que mejores tiempos dé mientras que el emparejamiento por torneo va a ser el más costoso computacionalmente debido a la complejidad del proceso.

4.5 Reproducción

Una vez seleccionados los padres, habrá que aplicarles un operando de cruce para dar lugar a la siguiente generación. En este punto, nos topamos con el dilema de seleccionar el operador de cruce que utilizaremos en nuestro caso.

Para implementar esta parte del algoritmo, optamos por utilizar un algoritmo de cruce uniforme. Este método de cruce consiste en que cada uno de los genes de las cadenas resultantes se escoge de manera aleatoria de uno de los padres, lo que permite explorar todas las posibles recombinaciones de los genes. Por esto se considera uno de los más potentes.

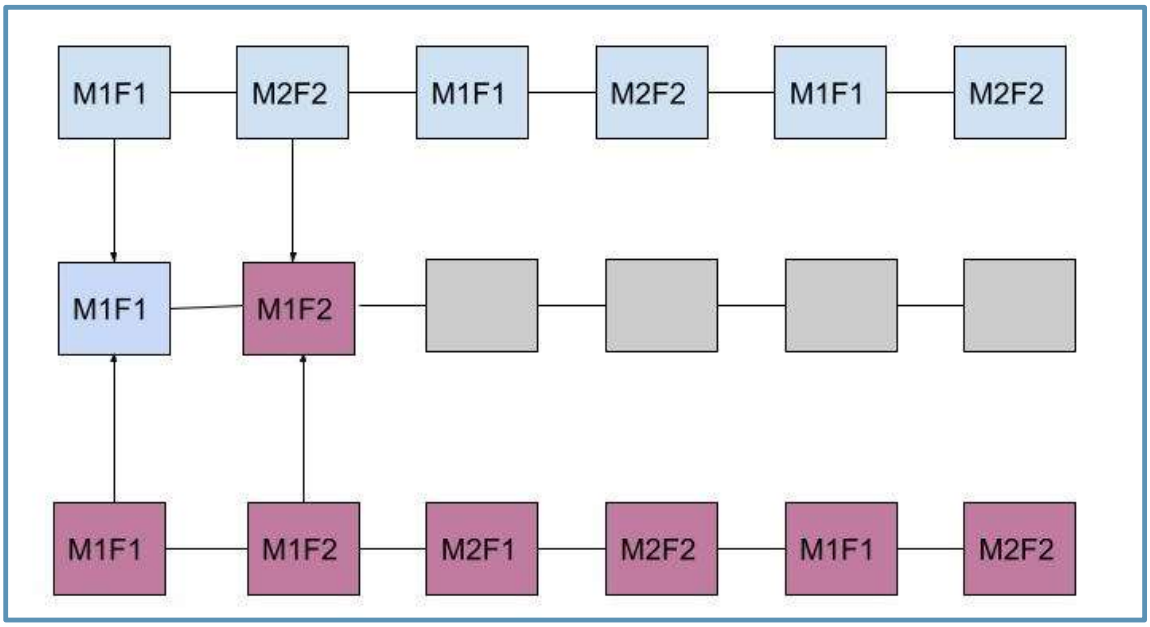


Ilustración 13 Esquema del cruce uniforme

La ilustración 15 trata de representar este concepto. La primera fila y última fila son los individuos que se están reproduciendo, y la cadena entre ellos, el individuo que se va a generar. Dicho individuo no está formado al 100% y ya posee un gen de cada progenitor, reforzando la idea de que este tipo de cruce generará más variabilidad que uno simétrico.

Aquí se decidió que la cadena resultante no sería la mitad de cada uno de los progenitores, sino que el número de genes que le corresponda trasmitir a cada uno dependerá del valor de su métrica, por lo que se puede definir como un cruce aritmético ponderado; y como consecuencia, antes de comenzar a formar la cadena del hijo el algoritmo tendrá que calcular cuantos genes le corresponden a cada uno de los progenitores.

Una vez, se conoce cuántos genes van a salir de cada padre, recorreremos la cadena hijo asignando de manera aleatoria a cada cromosoma el valor del correspondiente a uno de los padres, de manera aleatoria, siempre teniendo en cuenta que no se pueden pasar del límite impuesto por el reparto de genes.

De todo este proceso surge la problemática de que la cadena resultante del cruce es posible que no cumpla con las restricciones del problema, es decir, que cabe la posibilidad de que no se respete el número de fármacos existente en los hospitales. Es por ello por lo que hay que validar que la cadena de los nuevos individuos es correcta después de este proceso y en caso de que no lo sea arreglarla desasignándole tantos medicamentos como sea necesario a los pacientes. Una vez concluido este proceso será cuando se calcule la métrica y comience el proceso de mutación.

4.6 MUTACIÓN

Este punto será especialmente problemático ya que una ligera modificación puede hacer variar mucho la métrica, lo que provocará que sea necesario escoger bien cómo vamos a diseñar este proceso.

Para el proceso de selección de individuos a mutar, hemos considerado escogerlos de forma aleatoria, pero tras realizar algunas pruebas se llegó a la conclusión de que una selección pseudoaleatoria ponderada era la mejor opción, de manera que sea posible que cualquier individuo mute, pero que lo probable sea que los peores individuos se vean sometidos a este proceso.

Por otra parte, los individuos no tendrán número máximo de genes mutados, siendo posible que mute toda la cadena de un individuo concreto o que permanezca igual. Esta probabilidad dependerá de la parametrización del algoritmo, al igual que el máximo de individuos mutados, y será tarea nuestra encontrar los valores que mejor se ajusten.

Los datos que podrán mutar en un determinado gen son el medicamento y el medico asignado al paciente que corresponda dicho gen. En ningún caso se cambará el paciente correspondiente, ya que dificultaría el proceso de reproducción.

De todo este proceso surge un problema, debido al caso que estamos estudiando. Y es que si uno de los genes, que asigna un determinado fármaco a un paciente, ve alterado dicho medicamento, existe el riesgo de que se sobrepase el número de unidades de este. Entonces tendremos que lanzar el mismo proceso de validación que ejecutamos después de la reproducción.

Por último, cabe comentar que, al alterarse la cadena de los individuos, su métrica ya no será la misma, por lo que habrá que calcularla de nuevo. Una vez termina todo el proceso de mutación, empezará el cribado de individuos.

4.7 CRIBADO

Como mecanismos de cribado, también se van a implementar aquellos explicado en el estado del arte. De manera adicional, consideramos que es necesario una manera de eliminar aquellos individuos repetidos, por lo que se ha desarrollado un cribado adicional que elimina estas soluciones de la población. Como el tamaño de la población será limitado, esto permitirá que nuevos individuos ocupen su posición dentro de esta. Además, el resto de los mecanismos de cribado solo los aplicaremos en caso de que la población, después de reproducirse, supere el límite establecido lo cual añadirá variabilidad a la misma, y como consecuencia hará que sea más sencillo encontrar una solución óptima.

El cribado aleatorio y ponderado se implementarán igual que la selección por ruleta, siendo la única diferencia que los descartaremos de la población al terminar el proceso en vez de reproducirlos. Para el cribado de peores individuos, haremos lo contrario que en la selección de mejores individuos, recorreremos la población extrayendo los individuos con peor métrica hasta que el número de individuos no supere el máximo establecido en los parámetros.

4.8 CONDICIONES DE PARADA

Para terminar el desarrollo del algoritmo, falta determinar cuáles serán las condiciones que harán que este pare. La importancia de este apartado reside en que una mala calibración de estas condiciones supondrá un desperdicio de recursos, en caso de que se ejecute más de lo necesario; o que obtengamos peores soluciones, en caso de que se detenga antes de tiempo.

Para ello podemos tener en cuenta diferentes aproximaciones, como establecer un número fijo de generaciones, un tiempo máximo de ejecución o detectando cuando deja de haber mejora en el algoritmo. Las dos primeras soluciones tienen como desventaja que se puede llegar a desperdiciar poder computacional en iteraciones sin mejora o quedarse corto y parar cuando se sigue produciendo mejora. La última requiere que se establezcan muchos parámetros que pueden ser difíciles de calibrar como la mejora mínima o el número óptimo de generaciones sin mejora. Lo óptimo termina siendo combinar varios de estos mecanismos con un número máximo de generaciones y de generaciones sin mejora. También es conveniente establecer un número mínimo para que no pare entes de tiempo.

4.9 IMPLEMENTACIÓN DEL ALGORITMO

Para implementar el algoritmo he elegido el lenguaje de programación Python. Esta decisión se debe a que estoy bastante familiarizado con el lenguaje y además ofrece una versatilidad que va a hacer que me pueda enfocar en la implementación del diseño en lugar de preocuparme de los detalles de la programación de bajo nivel. Además, es un lenguaje que favorece la portabilidad del código y en caso de necesitar ejecutarlo en un servidor no debería de darme ningún problema.

Como contraparte no ofrece el mismo rendimiento que otros lenguajes de más bajo nivel como java o c, aunque en las versiones más recientes se han centrado en mitigar este problema.

A pesar de ser un lenguaje con una comunidad inmensa y tener muchos recursos a mi disposición, no planeo utilizar ninguna biblioteca de algoritmos genéticos para implementar mi solución. Las únicas que utilizaré serán las que me ofrezcan utilidades específicas, como pandas (pydata, 2023) para leer ficheros CSV o Matplotlib (matplotlib, 2023) para generar las gráficas de los resultados de la ejecución.

5 ANÁLISIS DE RESULTADOS

En este penúltimo apartado del trabajo, ejecutaremos el algoritmo, desde varios ángulos, y analizaremos los resultados que nos proporciona, comprobando como responde este ante los distintos tipos de configuración. También cambiaremos la talla del problema para ver que rendimiento ofrece cuando el número de posibles soluciones se dispara.

5.1 PARAMETRIZACIÓN DEL ALGORITMO

Para iniciar su ejecución, el algoritmo necesita, por una parte, unos ficheros con la información relativa al problema, véase el número de pacientes por hospital, el número de fármacos, la situación de cada médico, etc.; y una serie de parámetros que condicionarán su actuación, que se pasarán en otro fichero.

Los ficheros con la información del problema consisten en un fichero main donde se especifica los fármacos que hay disponibles, la situación de los médicos al iniciar la ejecución y los hospitales existentes con las dosis de las que disponen. Este fichero lo modificaremos cuando haya que aumentar la talla del problema. Por ejemplo, si se añaden más pacientes a un hospital, habrá que aumentar el número de dosis de los fármacos para que este aumento tenga sentido. Adicionalmente también habrá que pasarle un fichero CSV por cada hospital con los pacientes asignados. Cada fila de estos ficheros tendrá un valor con la prioridad de los pacientes, una lista de fármacos a los que son intolerantes y el tiempo que se va a emplear medicándolos, tal y como aparece en la imagen inferior.

```
1   Tiempo;Intolerancias;Prioridad
2   50;E,D,C;33
3   50;C,E,B;88
4   5;E,D,C;90
5   45;E,C,D;7
6   35;E,C,B;72
7   35;E,D,B;39
8   45;E,D,C;5
9   40;C,B,E;71
10  50;E,B,D;46
11  50;E,C,B;46
12  40;D,E,C;43
13  50;C,D,B;2
14  30;D,E,C;79
15  15;D,E,B;21
16  30;B,C,D;41
17  30;B,C,D;3
18  15;C,E,B;16
19  10;C,E,D;52
20  30;C,B,E;34
21  45;B,E,C;11
```

Ilustración 14-Fichero de configuración de un hospital.

El fichero de configuración contendrá la parametrización que va a emplear, es decir, el tipo de selección, el tipo de cribado y el número de parejas que se generaran en la selección, entre otras cosas.

```
Numero_Individuos_Poblacion_Inicial: 200
Numero_Generaciones: 500
Maximo_Individuos_Poblacion: 500
Numero_Minimo_Generaciones: 500
Generaciones_Sin_Mejora: 300
#1-Seleccion por torneo
#2-Seleccion por ruleta aleatoria
#3-Seleccion por ruleta ponderada
#4-Seleccion de mejores individuos
Tipo_Emparejamiento: 5
#1-Aleatorio
#2-Ponderado
#3-Siempre peores
Tipo_Cribado: 2
Porcentaje_Mutacion: 0.5
Probabilidad_Mutacion: 0.2
#1-una sola pareja
#2-todas las posibles
Parejas_Torneo: 2
Nombre_Ejecucion: Test
Dominio: Test
```

Ilustración 15-Fichero de parametrización del algoritmo.

Nuestro objetivo será probar las distintas combinaciones de estos parámetros para ver como estos afectan al resultado. Los más importantes serán el tipo proceso de selección, el tipo de cribado, el porcentaje de la población que podrá mutar, el número de generaciones de la población y el número de individuos de dicha población.

Como tipos de Emparejamiento, se ha implementado la selección por ruleta aleatoria, la selección por ruleta ponderada, la selección por torneo y el emparejamiento de los mejores individuos. Además, también podremos establecer el número de parejas que se generen durante el emparejamiento, dándonos como opciones generar una pareja o todas las parejas posibles.

Como tipos de Cribado, se dispone de un cribado aleatorio, un cribado ponderado y un cribado solo de los peores individuos.

En cuanto a la probabilidad de Mutación, hemos establecido un parámetro para definir qué porcentaje de la población va a mutar y otro para definir cuál es la probabilidad de que mute cada gen de la cadena.

El número de individuos limitará el tamaño de la población, siendo muy importante ya que afecta tanto a las soluciones como al rendimiento. Una población con pocos individuos no cubrirá muchas de las posibles soluciones, pero si no lo limitamos el tiempo de ejecución sería inasumible, además de provocar fallos de memoria.

Por último, el número de generaciones estará definido en otros dos parámetros, el número de generaciones máximo y número de generaciones máximo sin mejora, que provocará que el algoritmo finalice en caso de que no consiga encontrar un individuo con mejor función fitness después del número de generaciones definido.

Cada uno de los parámetros afectará de una forma a la mejor solución encontrada, a lo rápido que converjan las métricas a dicha solución y al rendimiento del programa; valores que me servirán para evaluar dichas configuraciones.

5.2 CASO DE ESTUDIO

En el apartado 2, se ha explicado el problema que vamos a resolver y como aumentaremos su talla para que su resolución no sea trivial. Para ello hemos diseñado un caso donde nos encontraremos un total de 5 hospitales con 500 pacientes cada uno, haciendo un total de 2500 pacientes; y un médico por hospital, que tendrá un grado de veteranía que variará entre 0 y 100. Habrá 5 tipos distintos de fármaco y cada paciente será intolerante a 3 de ellos. Por último, en cada hospital se dispondrá de 400 unidades de fármacos en total, repartidas de forma aleatoria, de forma que será tarea del algoritmo encontrar aquellos pacientes que mejoren más la métrica, ya que no será posible que todos reciban su fármaco.

Además de modificar los parámetros del problema, en una de las pruebas aumentaremos el número de pacientes a 5000 para ver cómo responde el algoritmo y que soluciones nos devuelve ante tal explosión combinatoria.

5.3 EXPOSICIÓN DE RESULTADOS

Para empezar, se ejecutará el algoritmo con todas las posibles combinaciones de tipos de emparejamiento y de cribado y ver cómo afectan. En cuanto a la mutación he establecido que mutará el 20% de la población y cada uno de sus genes tendrá un 50% de probabilidad de mutar. Estableceremos un máximo de 500 generaciones, 100 sin mejora y una población máxima de 300 individuos.

Como he comentado en las secciones anteriores, cada una de las partes del algoritmo puede tener distintas configuraciones que afectaran directamente al rendimiento y resultado del problema. Por ejemplo, a la hora de cribar individuos de una población vamos a poder elegir entre un cribado aleatorio o uno ponderado. El primero será más rápido, pero será el segundo el que posiblemente ofrezca mejores resultados. Por esto, nosotros vamos a modelar casos específicos de entrada, sobre los cuales probaremos todas las posibles combinaciones en cuanto a configuración.

Los resultados se expondrán mediante una tabla de resultados donde aparece la mejor métrica encontrada, las iteraciones que ha tardado en encontrar esa solución y el tiempo de ejecución de la prueba. Además, aquellas ejecuciones que puedan ser interesantes, las complementaré con una gráfica linear donde el eje Y representará la métrica del mejor individuo encontrado hasta el momento y el eje X el número de generación, o iteración, del algoritmo. Este tipo de representación nos indicará, por una parte, la mejor solución encontrada, que podríamos considerar el resultado de la ejecución; y por otra, las generaciones que ha tardado en converger hacia esta. De esta forma podremos evaluar la eficacia de las técnicas empleadas.

5.3.1 Métodos de selección con todas las parejas posibles

La primera tanda de ejecuciones se ha lanzado con todos los métodos de selección, un cribado aleatorio y creando todas las parejas posibles para la reproducción, dando lugar a la siguiente tabla de resultados.

Tabla 1-Metodos de selección con todas las parejas posibles

Prueba	Mejor métrica	Tiempo aprox.	Generaciones hasta converger
Mejores individuos	802797	60 min	32 generaciones
Ruleta aleatoria	902559	55 min	45 generaciones
Ruleta ponderada	902509	60 min	59 generaciones
Selección por torneo	902617	220 min	32 generaciones

Comparando los métodos de selección por ruleta, se observa que las métricas finales son muy similares en ambos casos, teniendo como única diferencia que la ruleta ponderada ha encontrado una solución que emplea 50 minutos menos que la aleatoria en medicar a los pacientes. También destaca que la ruleta aleatoria ha convergido en menos iteraciones, bastándole apenas 45 para encontrar la solución final, mientras que la ruleta ponderada ha necesitado 59 iteraciones.

Si observamos la gráfica resultante de la ejecución con ruleta ponderada es interesante ver ha sufrido un ligero estancamiento en la vigésimo quinta generación durante 20 iteraciones y partir de ahí ha empezado a mejorar más lentamente. Va a ser común en todas las configuraciones, que mejore mucho en las primeras generaciones del algoritmo y posteriormente se vaya estancando poco a poco la mejora.

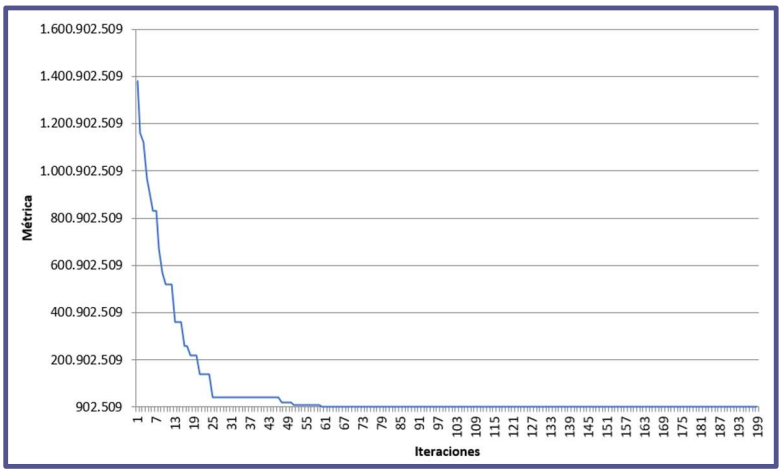


Ilustración 16-Ruleta Ponderada(I)

En cuanto a la ejecución con selección por torneo, la solución que se ha encontrado en este caso es ligeramente peor a las dos anteriores, pero como contraparte ha encontrado la solución final en tan solo 32 iteraciones.

Lo más destacable del emparejamiento de mejores individuos es que la métrica final es significativamente mejor que las anteriores, debido a que la prioridad media de los pacientes que ha medicado es un punto inferior a las anteriores ejecuciones. Además, en la gráfica, solo se percibe un ligero estancamiento en la décimo novena generación, encontrando la solución final en 32 iteraciones al igual que la selección por torneo. Es interesante ver como a medida que avanzaban las generaciones, la mejor solución cambia poco a poco.

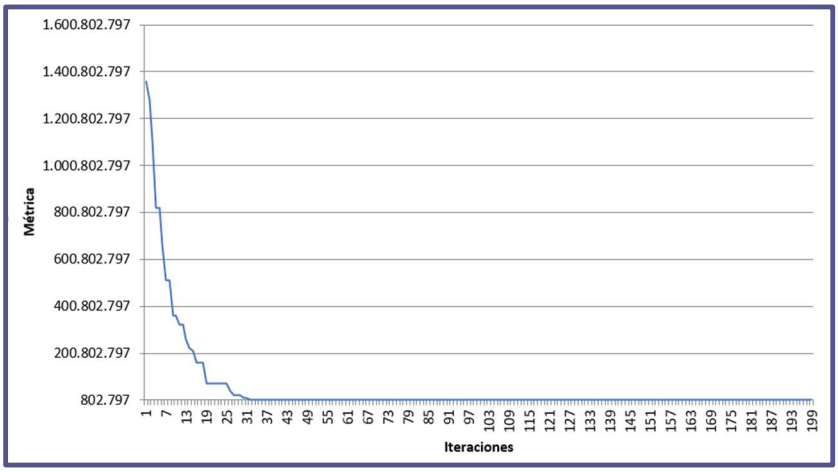


Ilustración 17- Emparejamiento de Mejores Individuos(I)

Con los resultados anteriores podemos llegar a varias conclusiones. Lo primero y más destacable es que los algoritmos dejan de mejorar sobre las 50 iteraciones, hecho que servirá para calibrar las siguientes pruebas del algoritmo. También se ha confirmado que un cruce ponderado de los mejores individuos proporciona mejores resultados en menos iteraciones que los mecanismos que se apoyan en la aleatoriedad. Teóricamente si se dejase ejecutar infinitamente las soluciones aleatorias estas deberían poder llegar a superar las soluciones ponderadas, cosa que no hemos visto en 200 iteraciones del algoritmo.

Si hablamos de tiempos de ejecución, los emparejamientos por ruleta y de mejores individuos han terminado tras una hora de ejecución. En cambio, el emparejamiento por torneo ha tardado cuatro veces más, suceso esperable dado que su implementación es más compleja.

5.3.2 Métodos de selección con pareja única

Sobre este mismo caso, he decidido ejecutar las mismas configuraciones, pero cambiando el proceso de emparejamiento para que solo se genere una nueva pareja en cada iteración; y que, por lo tanto, solo haya un individuo nuevo por generación. Tal y como habíamos adelantado en secciones anteriores, esta decisión provoca que el tiempo de ejecución sea mucho menor, ya que los procesos de selección son bastante costosos computacionalmente hablando.

Tabla 2-Métodos de selección con pareja única

Prueba	Mejor métrica	Tiempo aprox.	Generaciones hasta converger
Mejores individuos	902410	35 min	33 generaciones
Ruleta aleatoria	902860	30 min	45 generaciones
Ruleta ponderada	902620	30 min	45 generaciones
Selección por torneo	902550	90 min	106 generaciones

En el caso del emparejamiento de los mejores individuos (configuración que ha dado la mejor métrica anteriormente), la solución final es peor que la encontrada con todas las parejas posibles ya que la prioridad media de los pacientes medicados baja en un

punto. Por otro lado, ha encontrado dicha solución con el mismo número de iteraciones.

Observando el resto de las ejecuciones tanto la selección por ruleta aleatoria como la selección por torneo han mejorado ligeramente, mientras que la selección por ruleta ponderada ha empeorado un poco. Algo a destacar es que en estos casos han seguido mejorando pasadas las 50 iteraciones, indicándonos que para el emparejamiento único puede ser interesante aumentar el número, tal y como podemos observar en la gráfica a continuación.

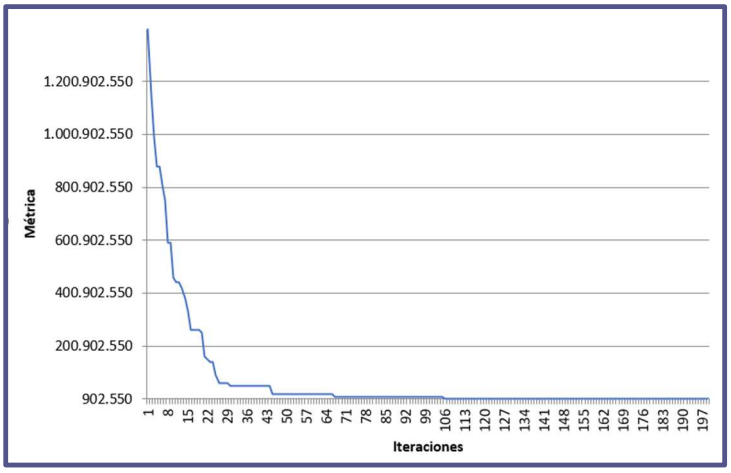


Ilustración 18- selección por Torneo y Una única Pareja(I)

El tiempo de ejecución también ha disminuido, completando los emparejamientos por ruleta y de mejores individuos en 30 minutos, y la selección por torneo en hora y media.

Con esta configuración suponía que se podrían encontrar soluciones igual de buenas que las anteriores pero que esta se haya encontrado con el mismo número de iteraciones que las ejecuciones con múltiples parejas es sorprendente.

5.3.3 Método de cribado ponderado y de peores individuos

Con estos resultados en mente, se ha lanzado otra vez el algoritmo, pero con el segundo tipo de cribado: el cribado ponderado. Este cribado teóricamente debe de aumentar la velocidad con la que las soluciones convergen a la mejor métrica; y hemos vuelto a establecer que el proceso de selección genere múltiples parejas. Además, hemos bajado el número de generaciones sin mejora a 50, de forma que, si no mejora la solución, pare el proceso antes de tiempo.

Tabla 3- Método de cribado ponderado

Prueba	Mejor métrica	Tiempo aprox.	Generaciones hasta converger
Mejores individuos	911290	20 min	13 generaciones
Ruleta aleatoria	912150	20 min	45 generaciones
Ruleta ponderada	911680	15 min	25 generaciones
Selección por torneo	911570	50 min	23 generaciones

Tabla 4-Método de cribado de peores individuos

Prueba	Mejor métrica	Tiempo aprox.	Generaciones hasta converger
Mejores individuos	912050	20 min	13 generaciones
Ruleta aleatoria	911230	20 min	20 generaciones
Ruleta ponderada	912325	15 min	25 generaciones
Selección por torneo	911810	50 min	18 generaciones

Con las tablas de la ejecución podemos comprobar que se cumple nuestra hipótesis, ya que la mejora más destacable es el número de iteraciones que tardan en encontrar el mejor individuo. También cabe resaltar que, en el caso de mejores individuos, la métrica ha empeorado. En cuanto a tiempos de ejecución, estos han sido menores debido a que, con el límite de 50 generaciones sin mejora, el algoritmo ha parado en todos los casos sobre la generación 70, en vez de llegar a las 200 de los casos anteriores.

5.3.4 Caso de ejecución pesada

En este punto, hemos querido complicar un poco el problema para ver cómo reaccionan los distintos mecanismos ante una talla de problema superior. Para ello se ha doblado el número de pacientes en cada hospital, con lo que espero poder ver si se ralentiza la convergencia, a la vez que pruebo el algoritmo.

Tabla 5-Caso de ejecución pesada con cribado aleatorio

Prueba	Mejor métrica	Tiempo aprox.	Generaciones hasta converger
Mejores individuos	831101237	200 min	21 generaciones
Ruleta aleatoria	1247593575	180 min	71 generaciones
Ruleta ponderada	1239082676	200 min	28 generaciones
Selección por torneo	213460509	720 min	37 generaciones

Tabla 6-Caso de ejecución pesada con cribado ponderado

Prueba	Mejor métrica	Tiempo aprox.	Generaciones hasta converger
Mejores individuos	1015879619	210 min	97 generaciones
Ruleta aleatoria	404098699	180 min	35 generaciones
Ruleta ponderada	1159810987	200 min	18 generaciones
Selección por torneo	986943447	450 min	17 generaciones

Tabla 7-Caso de ejecución pesada con cribado de peores

Prueba	Mejor métrica	Tiempo aprox.	Generaciones hasta converger
Mejores individuos	1010662	210 min	30 generaciones
Ruleta aleatoria	369553323	180 min	15 generaciones
Ruleta ponderada	841830459	220 min	61 generaciones
Selección por torneo	1018437	860 min	65 generaciones

El tiempo de ejecución en este caso se ha disparado tardando tres horas las ejecuciones sin torneo y estas hasta ocho horas cada una. Por otro lado, nos ha ofrecido unos cuantos resultados interesantes. Algo que también cabe destacar en las ejecuciones con esta talla aumentada, es que el ritmo de convergencia se ha ralentizado, continuando la

mejora pasada las 30 generaciones, lo cual es lógico porque hemos aumentado el número de posibles soluciones. A raíz de esto también vemos que las métricas han empeorado considerablemente, dejando pacientes por medicar en la mayoría de las ejecuciones.

Para la configuración con selección de mejores individuos con cribado ponderado, ha seguido un descenso habitual hasta la iteración 20, donde ha encontrado un individuo cuya solución medicaba a todos los pacientes, produciendo un descenso de la métrica enorme, tal y como se puede observar en la gráfica inferior.

Ilustración 19-Mejores individuos con cribado ponderado (II)

Algo similar ha ocurrido en la selección por torneo y cribado ponderado, donde en la generación 59 ha encontrado un individuo con características similares. En este caso la mejora se ha estancado durante 40 generaciones, aunque ha terminado dando la mejor solución al problema

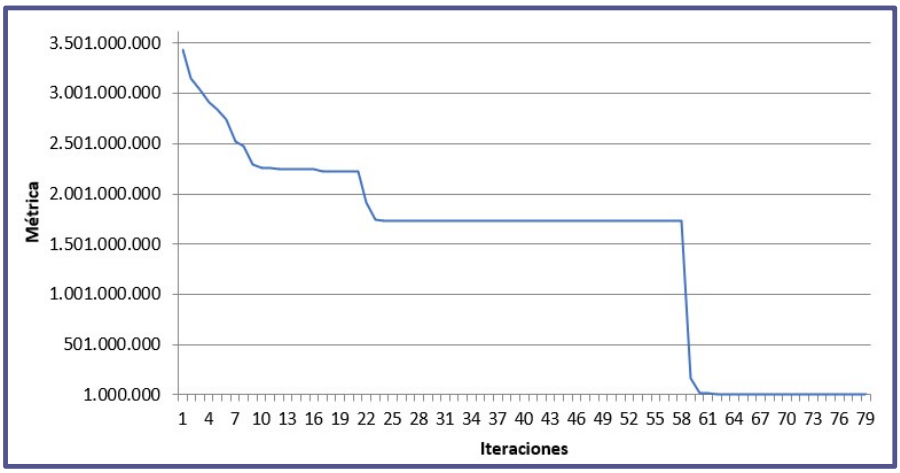


Ilustración 20-Selección por torneo y cribado ponderado

La selección de mejores individuos con cribado de peores es la que ha brindado la mejor solución, medicando a todos los pacientes de los hospitales. La selección por torneo también ha conseguido esto, pero su tiempo ha sido ligeramente peor.

5.3.5 Métodos con mutación alterada

Por último, hemos querido comprobar el efecto que tienen las distintas configuraciones del proceso de mutación en los resultados. Para ello se ha tomado la configuración inicial y lanzado todos los casos, primero con unos porcentajes de mutación muy altos y luego suprimiendo completamente este proceso.

Para aumentar la mutación, he establecido que el 80% de los individuos de la población muten el 80% de su cadena.

Tabla 8-Métodos con mutación alta y cribado aleatorio

Prueba	Mejor métrica	Tiempo aprox.	Generaciones hasta converger
Mejores individuos	911840	35 min	119 generaciones
Ruleta aleatoria	4111320	30 min	76 generaciones
Ruleta ponderada	911350	30 min	96 generaciones
Selección por torneo	911500	90 min	46 generaciones

Tabla 9-Métodos con mutación alta y cribado ponderado

Prueba	Mejor métrica	Tiempo aprox.	Generaciones hasta converger
Mejores individuos	902410	35 min	53 generaciones
Ruleta aleatoria	912105	30 min	54 generaciones
Ruleta ponderada	1011345	30 min	47 generaciones
Selección por torneo	911500	90 min	40 generaciones

Tabla 10-Métodos con mutación alta y cribado de peores individuos

Prueba	Mejor métrica	Tiempo aprox.	Generaciones hasta converger
Mejores individuos	911905	35 min	20 generaciones
Ruleta aleatoria	912170	30 min	41 generaciones
Ruleta ponderada	911955	30 min	50 generaciones
Selección por torneo	911285	90 min	22 generaciones

Los resultados obtenidos han sido sorprendentes ya que, en todos los casos, el alto porcentaje de mutación no ha sido impedimento para encontrar un individuo que medicase a todos los pacientes. El número de iteraciones que han necesitado para converger ha variado de una ejecución a otra, ya que el proceso de mutación recae mucho en la aleatoriedad.

Por ejemplo, para el caso con ruleta aleatoria y cribado de peores individuos, ha tardado 30 iteraciones en dar con un buen individuo, pero ha mejorado tanto que el descenso en la métrica ha sido muy pronunciado, tal y como podemos observar en la gráfica inferior.

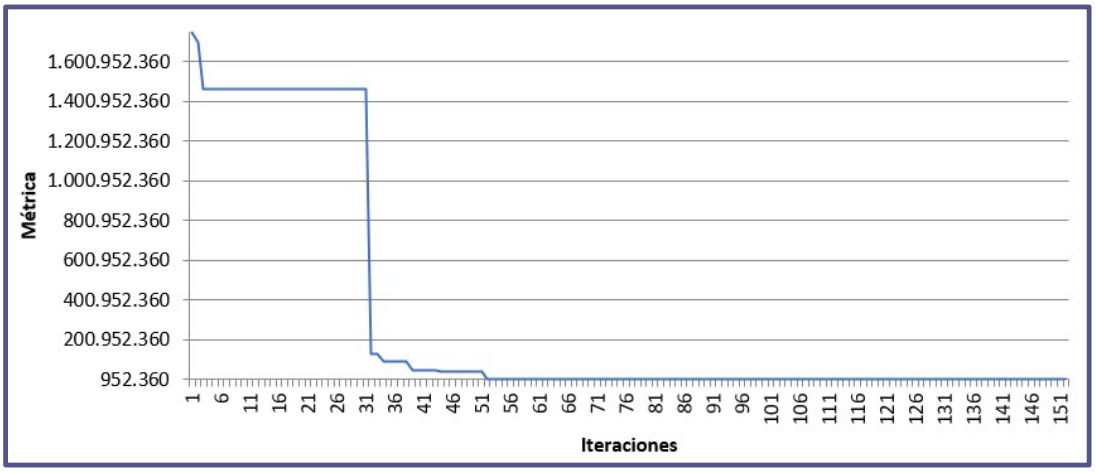


Ilustración 21-Seleccion por ruleta aleatoria y cribado de peores individuos.

Para el caso de selección de mejores individuos con cribado ponderado, que ha sido el que mejores resultados nos ha brindado en la mayoría de los casos, la métrica ha mejorado repentinamente en la décima iteración, casi encontrando el individuo que ha devuelto como resultado, como se observa en la ilustración 24.

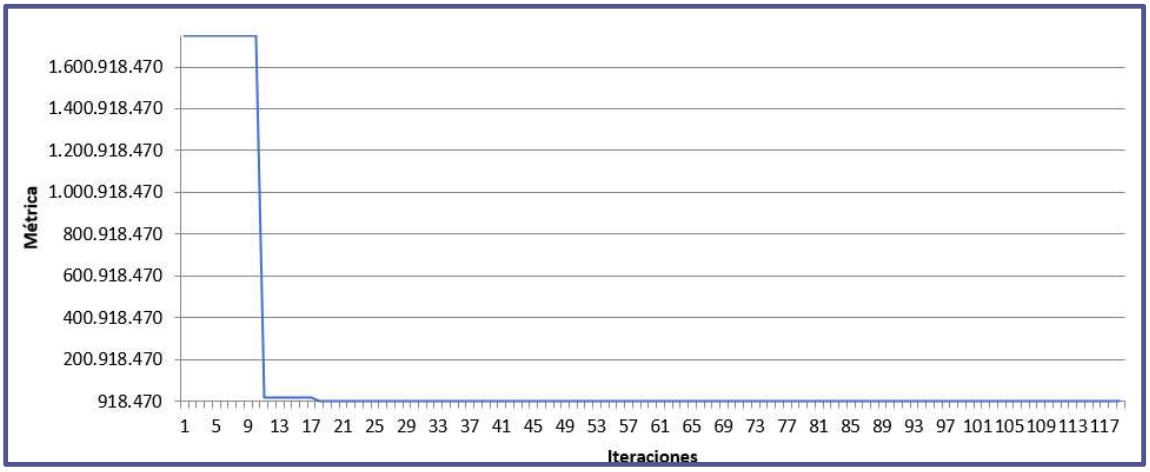


Ilustración 22-Selección de mejores individuos con alta mutación

Por el contrario, los resultados obtenidos de las ejecuciones donde prescindíamos de la mutación han sido catastróficos. En estos casos, la métrica apenas ha mejorado durante el proceso. Por ejemplo, en el caso de la selección de mejores con cribado aleatorio ha comenzado con una métrica de 1731712551 y la mejor solución que ha encontrado tiene una métrica de 1620661148. En el caso de selección de mejores con cribado ponderado, directamente no ha mejorado. Esto pone de manifiesto la importancia que tiene el proceso de mutación en el algoritmo ya que cuando suprimimos este las soluciones son significativamente peores.

Tabla 11-Métodos sin mutación con cribado aleatorio

Prueba	Mejor métrica	Tiempo aprox.	Generaciones hasta converger
Mejores individuos	1620661148	35 min	9 generaciones
Ruleta aleatoria	1581018345	30 min	38 generaciones
Ruleta ponderada	1591017480	30 min	64 generaciones
Selección por torneo	1281017960	90 min	188 generaciones

En la gráfica, resultado de una selección de mejores individuos con cribado ponderado, destaca como las soluciones sí que han ido teniendo mejora poco a poco, pero estos saltos en la métrica han sido muy ligeros. La mejor solución la ha encontrado en la iteración 63, pero está ni siquiera era una que medicase a todos los pacientes de los hospitales.

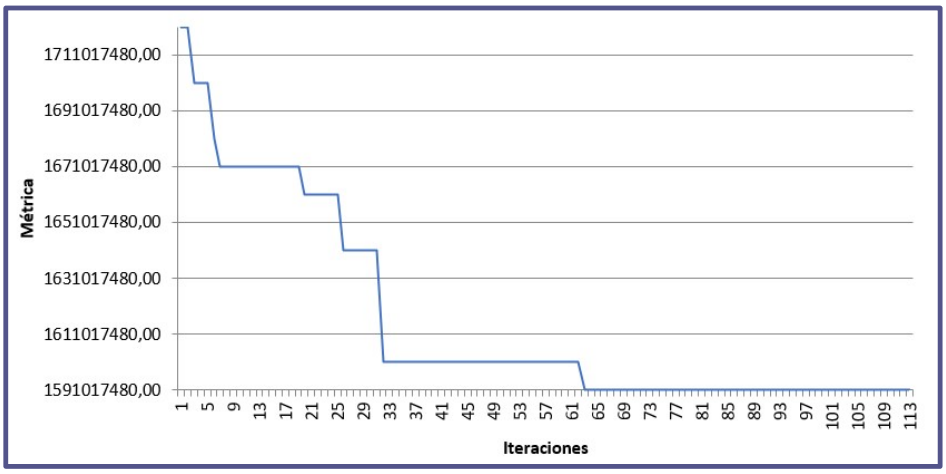


Ilustración 23-Selección de mejores individuos sin mutación

5.4 ANÁLISIS DE LOS RESULTADOS

Observando los resultados obtenidos he sacado las siguientes conclusiones.

Por una parte, se ha confirmado la hipótesis de que aquellas técnicas que recaían en cruzar los mejores individuos han encontrado mejores soluciones que aquellas que dependían de la aleatoriedad, pero en un espacio de tiempo infinito estas segundas deberían ser capaces de superar a las primeras.

Me ha sorprendido el efecto que tiene aumentar la mutación dentro de la población, ya que en un principio se esperaba que esta tuviese un efecto negativo, pero tal y como se ha visto en los resultados, ha provocado justo lo contrario, encontrando soluciones igual de buenas o incluso mejores, eso sí, con un número superior de iteraciones del programa.

En el caso del cribado, ha sido el cribado aleatorio el que mejor ha solucionado el problema, aunque el cribado de peores individuos también ha ofrecido buenas soluciones, siendo el cribado ponderado el que ha dado peores resultados.

Si hablamos de la velocidad con la que han convergido hacia una solución óptima, ha sido también la selección de mejores individuos la que ha mostrado un mejor comportamiento. Es difícil decir que configuración ha dado peores resultados en este sentido, ya que ha ido variando, dependiendo del caso.

En cuanto a tiempos de ejecución, la selección por torneo ha sido la que peores resultados ha ofrecido con diferencia, tardando de media 4 veces más que el resto de los métodos. La batería de pruebas tardaba en total entre 14 y 24 horas en ejecutarse, dependiendo del caso, siendo casi la mitad de este intervalo ocupado por la selección por torneo.

En conclusión, si tuviese que implementar este algoritmo para otro caso de uso, sería difícil decantarme por una configuración porque, a pesar de que por lo general ha sido

la selección de mejores individuos la que mejor ha rendido, no ha sido tan superior al resto como para justificar su elección. Esta decisión debería tomarse pensando en cómo sería el caso por resolver y los recursos disponibles. Lo que sí se puede afirmar es que lo recomendable sería un alto porcentaje de mutación porque ha quedado claro que afecta enormemente a la mejora de las soluciones.

6 CONCLUSIONES

A modo de conclusión, con este proyecto hemos querido mostrar como la aplicación de técnicas inteligentes puede resolver problemas de logística del día a día. El problema lo hemos intentado plantear de forma que hiciese referencia a un problema real de la sociedad, y con el diseño de la solución estudiar las distintas alternativas que teníamos para resolverlo.

6.1.1 Problemas encontrados

Durante la elección del problema a resolver, nos topamos con la dificultad de plantearlo de manera que este supusiese un reto real, pero que a la vez no fuese tan complicado como para que el desarrollo imposibilitase cumplir los plazos. Además, que el enunciado tuviese que estar relacionado con un problema real también dificultaba la tarea.

También supuso un reto diseñar el algoritmo ya que su maleabilidad hacía que tuviésemos que tomar muchas decisiones que hasta su implementación no sabíamos si eran las correctas, como la configuración de la métrica, por ejemplo. Además, estas no eran fáciles de corregir ya que un cambio mínimo suponía tener que reestructurar el algoritmo entero.

A nivel de programación, nos hemos encontrado problemas con el lenguaje escogido, Python, ya que el mal rendimiento aumento mucho el tiempo de ejecución de los casos de prueba. Para mitigar el impacto de esto, investigue las herramientas que nos ofrece el lenguaje para lanzar ejecuciones en paralelo, pudiendo realizar hasta 3 pruebas simultaneas en el mismo equipo.

6.1.2 Objetivos cumplidos

Durante el desarrollo del proyecto se ha abordado cada uno de los objetivos planteados en la introducción, con el fin de resolver un problema concreto mediante la aplicación de técnicas inteligentes.

En primer lugar, he definido el problema a resolver, tomando como inspiración un tema actual, y he establecido las variables de este para que dicho problema supusiera un reto complejo y su resolución no fuese trivial.

A continuación, he explorado los distintos algoritmos disponibles para abordarlo, haciendo un análisis exhaustivo de las principales opciones disponibles, estableciendo la viabilidad de cada una de las opciones, optando finalmente por los algoritmos genéticos, debido a su capacidad de resolver problemas de tallas grandes.

La implementación del algoritmo se ha llevado a cabo analizando cuidadosamente los distintos mecanismos que nos ofrecen los AG, explicando y estableciendo los diversos parámetros de acuerdo con la naturaleza del problema.

Durante el proceso de testeo y análisis de resultados, hemos comprobado como la distinta parametrización del algoritmo afecta a la ejecución de este, estableciendo las ventajas e inconvenientes de cada implementación.

En conclusión, este proyecto ha demostrado como hay técnicas de IA que se alejan de los modelos actuales y aun así pueden seguir siendo igualmente vigentes a la hora de resolver problemas actuales. Además, como una correcta parametrización es fundamental para poder trabajar con este tipo de tecnologías.

6.1.3 Valoración personal

A nivel personal, este ha sido un proyecto muy interesante en el que he estado trabajando 9 meses junto a mi tutor. Este me ha servido para ver la inteligencia artificial desde otra perspectiva y a profundizar más en este campo.

Este trabajo está relacionado con la asignatura de Sistemas Inteligentes (SIN) cursada en el máster. A pesar de no estudiar las técnicas metaheurísticas, sí que aprendimos técnicas de planificación inteligente, que tuvimos que aplicar a un problema dentro del dominio del transporte.

Estoy muy satisfecho con el resultado de este proyecto, y considero que hemos cumplido con los objetivos propuestos en la introducción.

Como opinión, y teniendo en cuenta la concienciación que existe actualmente respecto a la sostenibilidad, este tipo de algoritmos pueden ser un gran aliado, ya no solo por su capacidad de resolver problemas que pueden mejorar la sociedad, sino por la gran cantidad de energía que consumen las técnicas de aprendizaje actuales en comparación. Igual volver a emplear este tipo de técnicas menos sofisticadas puede seguir siendo una opción.

6.1.4 Futuras mejoras

Las futuras mejoras de este proyecto deberían girar en torno a la implementación de otros algoritmos de búsqueda local para comprobar si ofrecen mejores resultados que el genético.

Relacionado con lo anterior, también sería interesante utilizar la salida de nuestro algoritmo como entrada de otro, como un algoritmo de temple simulado, con el objetivo de obtener resultados mejores.

También se podría añadir nuevas características al enunciado del problema, como por ejemplo que los médicos tengan un horario de trabajo o que los medicamentos pudiesen caducar. Todo esto serviría para darle más capas de complejidad al problema y hacerlo más interesante.

Por último, sería interesante la migración del código Python a otro lenguaje que ofrezca un mayor rendimiento, para reducir el tiempo de ejecución, a costa de perder versatilidad.

7 Referencias

Bhalla, A., Lynce, I., Sousa, J. T., & Marqueés-Silva, J. (2003). *Heuristic Backtracking Algorithms for SAT*.

Brailsford, S. C., Potts, C. N., & Smith, B. M. (13 de Enero de 2011). *https://www.sciencedirect.com*. Obtenido de https://www.sciencedirect.com/science/article/abs/pii/S0377221798003646

Falco, I. D., Cioppa, A. D., & Tarantino, E. (2002). *Mutation-based genetic algorithm: Performance evaluation*.

Fang, Y., & Li, J. (2010). *A Review of Tournament Selection in Genetic Programming*.

Giner, J. C. (20 de Junio de 2020). *https://riunet.upv.es*. Obtenido de https://riunet.upv.es/handle/10251/127157

Glover, F., & Laguna, M. (1997). *Tabu Search*.

Google. (25 de Julio de 2023). *https://trends.google.com*. Obtenido de https://trends.google.com/trends/explore?date=today%205-y&q=LLM,genetic%20algorithm&hl=es

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). *A Formal Basis for the Heuristic Determination of Minimal Cost Paths*.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. MIT.

Intriago, J. A. (07 de 02 de 2022). *Advance Intelligence*. Obtenido de https://advanceintelligence.wordpress.com/2014/10/16/busqueda-bidireccional/

khan academy. (6 de junio de 2021). *khan academy*. Obtenido de khan academy: https://es.khanacademy.org/science/ap-biology/natural-selection/natural-selection-ap/a/darwin-evolution-natural-selection

Kour, H., Sharma, P., & Abrol, P. (2015). *Analysis of Fitness Function in Genetic Algorithms*.

matplotlib. (10 de Septiembre de 2023). *matplotlib*. Obtenido de https://matplotlib.org/

Motorpasión. (15 de Junio de 2021). *www.motorpasion.com*. Obtenido de www.motorpasion.com: https://www.motorpasion.com/industria/100-anos-de-ford-en-cadena-o-cuando-ford-reinvento-la-industria

Naciones Unidas. (10 de 08 de 2023). *unric.org*. Obtenido de unric.org: https://unric.org/es/agenda-2030/

Norvig, P., & Russell, S. J. (1995). *Inteligencia Artificial: Un Enfoque Moderno*. Prentice Hall.

Organizacion Mundial de la Salud. (16 de Junio de 2021). *www.who.int*. Obtenido de https://www.who.int/es/news-room/q-a-detail/coronavirus-disease-covid-19

Pencheva, T., Atanassov, K., & Shannon, A. (2009). *Modelling of a Roulette Wheel Selection Operator in Genetic Algorithms Using Generalized Nets*.

pydata. (10 de Septiembre de 2023). *pydata.org*. Obtenido de https://pandas.pydata.org/

R, G. (2019). *Comparative Analysis of Various Uninformed Searching Algorithms in AI*.

Real Academia Española de la Lengua. (15 de Junio de 2021). *rae.es*. Obtenido de https://dle.rae.es/log%C3%ADstico

Reeves, C. (1991). *Heuristic Search Methods: A Review*.

Safe, M., Carballido, J., Ponzoni, I., & Brignole, N. (2014). *On Stopping Criteria for Genetic Algorithms*.

Sancho, F. (12 de Noviembre de 2020). *www.cs.us.es*. Obtenido de http://www.cs.us.es/~fsancho/?e=205

Schade, M. (25 de Julio de 2023). *openai.com*. Obtenido de openai.com: https://help.openai.com/en/articles/7842364-how-chatgpt-and-our-language-models-are-developed

Shivam. (07 de 02 de 2022). *geeksforgeeks*. Obtenido de https://www.geeksforgeeks.org/iterative-depth-first-traversal/

Shukla, A., Pandey, H. M., & Mehrotra, D. (2015). *Comparative review of selection techniques in genetic algorithm*.

Wikipedia. (16 de Junio de 2020). *https://es.wikipedia.org*. Obtenido de https://es.wikipedia.org: https://es.wikipedia.org/wiki/Inmunidad_de_grupo

Xiao-Bing Hu, E. D. (25 de junio de 2021). *sciencedirect*. Obtenido de sciencedirect: https://www.sciencedirect.com/science/article/abs/pii/S0305054807001748

Yang, X.-S. (2021). Nature-Inspired Optimization Algorithms. En X.-S. Yang, *Nature-Inspired Optimization Algorithms (Second Edition)*. Obtenido de sciencedirect: https://www.sciencedirect.com/topics/engineering/genetic-algorithm

ANEXO

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	**Alto**	**Medio**	**Bajo**	**No Procede**
ODS 1. **Fin de la pobreza.**				**X**
ODS 2. **Hambre cero.**				**X**
ODS 3. **Salud y bienestar.**	**X**			
ODS 4. **Educación de calidad.**				**X**
ODS 5. **Igualdad de género.**				**X**
ODS 6. **Agua limpia y saneamiento.**				**X**
ODS 7. **Energía asequible y no contaminante.**				**X**
ODS 8. **Trabajo decente y crecimiento económico.**				**X**
ODS 9. **Industria, innovación e infraestructuras.**			**X**	
ODS 10. **Reducción de las desigualdades**.				**X**
ODS 11. **Ciudades y comunidades sostenibles**.		**X**		
ODS 12. **Producción y consumo responsables.**				**X**
ODS 13. **Acción por el clima.**		**X**		
ODS 14. **Vida submarina.**				**X**
ODS 15. **Vida de ecosistemas terrestres.**				**X**
ODS 16. **Paz, justicia e instituciones sólidas.**				**X**
ODS 17. **Alianzas para lograr objetivos.**				**X**

Los objetivos que se pueden relacionar con mi trabajo de final de máster son:

ODS 3. Salud y bienestar: El problema que he tratado en este trabajo de final de máster ha sido la gestión de recursos médicos en unos hospitales. El algoritmo genético que he empleado es perfectamente exportable a una situación real de crisis donde hubiese que medicar a una población, como fue el COVID-19. Además, aplicándolo a un hospital podríamos aumentar su eficiencia y asegurarnos que los pacientes reciben los tratamientos lo antes posible.

ODS 9. Industria, innovación e infraestructuras: La relación que guarda mi trabajo con este ODS es que se pueden aplicar técnicas inteligentes a la industria de nuestro país para mejorar la logística de estas, no solo aumentando el rendimiento, sino que también haciendo que sean más respetuosas con el medio ambiente.

ODS 11. Ciudades y comunidades sostenibles: Durante el planteamiento de mi trabajo, he comentado como una de las ideas que tuvimos fue utilizar los algoritmos genéticos para mejorar el flujo de transporte publico en la ciudad. Este planteamiento es muy factible; y aplicar estas técnicas inteligentes podría mejorar mucho la calidad de vida de los ciudadanos y ayudaría a dejar de despender de los vehículos propios en favor del transporte público.

ODS 13. Acción por el clima: Este último ODS, está muy relacionado con el anterior. Como he comentado una posible aplicación de las técnicas que hemos estudiado en este trabajo consiste en la planificación del transporte. Mejorar la logística del transporte a nivel global es fundamental para poder frenar entre todos el calentamiento global.

En conclusión, las técnicas inteligentes tienen una gran utilidad y pueden emplearse para resolver multitud de problemas logística y ayudarnos a hacer un uso eficiente de los recursos. He comentado los cuatro ODS que me parecían que se relacionaban más con mi trabajo, pero una buena logística afectaría positivamente y facilitaría cumplir con todos los objetivos en mayor o menor medida.