Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

# Desarrollo de Algoritmos Dirigidos por Retos: Optimización de un modelo de simulación para la planificación de la capacidad y recursos de hospitales durante la pandemia de COVID-19

*Challenge-driven Algorithms Development: Optimization of a simulator model for a capacity and resource planning task for hospitals during COVID-19*

Néstor Torres Díaz

La Laguna, 8 de julio de 2022

D. **Eduardo Manuel Segredo González**, con N.I.F. 78.564.242-Z, Profesor Ayudante Doctor adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor.

D. **Alejandro Marrero Díaz**, con N.I.F. 78.649.404-F, Investigador Predoctoral adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como cotutor.

**C E R T I F I C A (N)**

Que la presente memoria titulada:

"*Desarrollo de Algoritmos Dirigidos por Retos: Optimización de un modelo de simulación para la planificación de la capacidad y recursos de hospitales durante la pandemia de COVID-19*"

ha sido realizada bajo su dirección por D. **Néstor Torres Díaz**, con N.I.F. 54.064.450-Y.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 8 de julio de 2022.

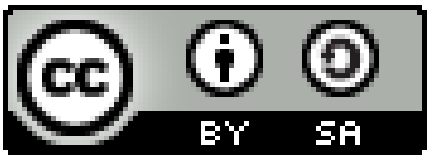
# Agradecimientos

Agradecer principalmente a mis padres, hermano y amigos, por apoyarme siempre y confiar en mí en todo momento.

A mis tutores Eduardo y Alejandro, por el excepcional trato y atención que me han dado, por orientarme y ayudarme cuando lo necesitaba.

Y a todas las personas que, de una forma u otra, han hecho que este camino sea más ameno.

# Licencia



**Resumen**

*La "Optimización de un modelo de simulación para la planificación de la capacidad y recursos de hospitales durante la pandemia de COVID-19", es un problema de optimización muy reciente cuyo objetivo es encontrar una configuración de parámetros óptima para un modelo de simulación. Este modelo de simulación hace uso de eventos discretos para simular el flujo de pacientes de COVID-19 en el hospital desde que se infectan hasta que se recuperan o mueren. El simulador permite llevar a cabo una estimación de los recursos necesarios para atender a los pacientes.*

*La computación evolutiva brinda los recursos necesarios para abordar problemas de optimización que requieren un alto coste computacional como es este caso. Por lo tanto, se ha tomado la decisión de realizar la implementación de un algoritmo genético para resolver el problema y posteriormente realizar un estudio de los resultados obtenidos en los experimentos. Para poder llevar a cabo el desarrollo, se han realizado adaptaciones sobre la librería GeneticsJS, la cual facilita estructuras básicas y métodos que permiten realizar la implementación de algoritmos evolutivos.*

**Palabras clave:** Computación Evolutiva, Algoritmo Genético, Simulación por Eventos Discretos, Ajuste de Parámetros, Optimización de Caja Negra

**Abstract**

*The Optimization of a simulator model for a capacity and resource planning task for hospitals during COVID-19 is a very recent optimization problem whose objective is to find an optimal parameter configuration for a simulation model. This simulation model makes use of discrete events to simulate the flow of COVID-19 patients in the hospital from infection to recovery or death. The simulator allows to estimate the resources needed to care for the patients.*

*Evolutionary computation provides the necessary resources to tackle optimization problems that require a high computational cost as in this case. Therefore, the decision has been made to implement a genetic algorithm to solve the problem and then to study the results obtained in the experiments. In order to carry out the development, adaptations have been made to the GeneticsJS library to meet the requirements of the problem.*

**Keywords:** Evolutionary Computation, Genetic Algorithm, Discrete-Event Simulation, Parameter Setting, Black-box Optimization

# Índice general

# Índice de Figuras

# Índice de Tablas

# Capítulo 1

# Introducción

En este capítulo se presentará brevemente el problema de optimización que se abordará en este trabajo, así como los detalles sobre la tipología del mismo. Por otro lado, se dará una breve introducción a la Computación Evolutiva, los objetivos marcados para el proyecto y las contribuciones realizadas.

## 1.1. ¿Qué es un *modelo de caja negra*?

Un **modelo de caja negra** o **black-box model** (2) se puede ver como un sistema que espera una o varias entradas procedentes de alguna fuente exterior al mismo. Cuando este sistema obtiene la información, la procesa a través de algún modelo computacional cuyos detalles son desconocidos (de ahí el término de caja negra) y tras procesar la entrada, se obtienen una o varias salidas.

Una caja negra está compuesta, básicamente por tres partes principales: la entrada, el modelo y la salida. Según las partes conocidas de nuestro modelo de caja negra, podremos clasificarlo en tres tipos distintos de problemas: optimización, modelización y simulación.


Figura 1.1: Esquema general de un modelo computacional de caja negra

Nos encontramos ante un **problema de optimización** cuando lo que se conoce es el modelo y la salida o una descripción de la misma, en base a lo cual se deberán de calcular la entrada o entradas que devuelven la salida deseada.

En un **problema de modelización** se conocen las entradas y sus salidas asociadas, pero no se tiene conocimiento del modelo de computación que se debería usar para conseguir dicho comportamiento.

Por último, cuando lo que se conoce es el modelo y las entradas pero se desconoce cual es la salida, nos encontramos ante un **problema de simulación**.

## 1.2. Modelo de simulación

Un modelo de simulación puede verse como una representación de la realidad que busca probar o evidenciar sucesos que no se han dado o que no se dan con frecuencia.

### 1.2.1. Utilidad y ventajas

Los modelos de simulación son utilizados principalmente para realizar predicciones a futuro, es decir, llevar a cabo pruebas sobre situaciones o eventos concretos para posteriormente obtener los resultados de dichas predicciones. También son utilizados para tareas de investigación de manera que permita a los investigadores tomar medidas sin tener que recurrir a experimentos reales, ahorrando una gran cantidad de tiempo.

Una de las principales ventajas del uso de un modelo de simulación es el abaratamiento de costes respecto de la implementación, ya que es más económico, seguro y factible que estudiar los efectos del mundo real.

### 1.2.2. Reto

El reto titulado *Optimización de un modelo de simulación para la planificación de la capacidad y recursos de hospitales durante la pandemia de COVID-19* (3) es un costoso problema de optimización basado en la simulación por ordenador de eventos discretos de alta dimensionalidad, es decir, el modelo de simulación consiste en un número elevado de entradas.

Este reto de optimización fue propuesto en la *Genetic and Evolutionary Computation Conferencee (GECCO)* de 2021, uno de los congresos más importantes a nivel mundial sobre computación evolutiva (4) y viene derivado de la necesidad de optimizar los parámetros de entrada del simulador BaBSim.Hospital (5). Este es un simulador de eventos discretos (DES) de 29 dimensiones que simula el curso típico del tratamiento de los pacientes de COVID-19 bajo escenarios configurados por el usuario, gracias a los cuales el simulador puede calcular la demanda de camas del hospital para atender a los pacientes. Aunque principalmente, la estimación es relativa a las camas, en base a esta información se pueden estimar otra serie de recursos necesarios.

Este simulador, es capaz de manejar muchos aspectos de la planificación de recursos en hospitales, como son las camas de la *Unidad de Cuidados Intensivos (UCI)*, los respiradores o el personal necesario, teniendo en cuenta factores influyentes como la edad o el estado de salud de los pacientes.

Otro aspecto que tiene en cuenta el simulador para realizar sus estimaciones son los parámetros de entrada, como podrían ser el número de días transcurridos desde que un paciente se infecta hasta que va al hospital o el factor utilizado para ese mismo caso, el cual se utiliza internamente para calcular la probabilidad de que se produzca la transición de ese paciente infectado al hospital.

Los parámetros de entrada son clave, ya que encontrar una configuración apropiada de los mismos propiciará mejores resultados en las simulaciones y para ello se implementarán una serie de algoritmos que tienen como finalidad hallar una configuración óptima de los parámetros.

## 1.3. Computación Evolutiva

La **Computación Evolutiva** es un área de investigación dentro de las Ciencias de la Computación que se inspira principalmente en el proceso de la evolución natural (6). Esta nos proporciona un medio alternativo para abordar problemas complejos de diversa índole, basándose en un estilo de resolución de problemas idéntico al empleado por los individuos en la selección natural, donde teniendo en cuenta la adaptación de un individuo o solución, se puede saber si tiene mayor probabilidad de sobrevivir o ser la solución que se está buscando.

### 1.3.1. Evolución Natural

La *evolución natural* puede ser descrita como un proceso o situación, donde existe una población de individuos cuyo objetivo es adaptarse al entorno en el que conviven, para poder sobrevivir y tener descendencia, pero con el matiz de que cada individuo tiene una capacidad de supervivencia dependiendo del medio en el que vive, los rasgos que tiene y de su eficacia para conseguir sus objetivos.

En este punto es donde entra el concepto de la *supervivencia del más apto*, introducido en la *teoría evolutiva de Charles Darwin* (7), donde aquellos individuos que mejor se adaptan son los que tienen descendencia, por lo que los rasgos que han favorecido su supervivencia, pasan a la siguiente generación, mientras que el resto de individuos no sobreviven y por ende tampoco lo hacen sus rasgos. Cuando hablamos de rasgos que pasan de una generación a otra, hay que tener en cuenta que estos no serán idénticos a los de sus progenitores, sino que es probable que sufran ciertos cambios, conocidos como mutaciones.

Todo el proceso de la evolución, la selección natural y la aplicabilidad de los mismos dentro del campo de la Computación es lo que hace que la Computación Evolutiva llame tanto la atención frente a otras posibles opciones para la resolución de problemas que cada vez son más complejos.

### 1.3.2. Algoritmos Genéticos

Los *algoritmos genéticos* son algoritmos que pertenecen a la familia de los algoritmos evolutivos, que son aquellos inspirados en la computación evolutiva (6). Estos algoritmos son una herramienta muy útil para la resolución de problemas complejos, como podría ser un problema de optimización de rutas o de planificación de tareas, por ejemplo.

Al ser algoritmos aproximados, destacan porque permiten hallar soluciones en tiempos razonables, con resultados óptimos y la implementación de los mismos es relativamente fácil puesto que siempre tienen la misma estructura con la diferencia de que la función de fitness se tiene que adaptar según el problema que se aborda.

El funcionamiento de un algoritmo genético no difiere mucho con respecto al funcionamiento de cualquier otro tipo de algoritmo evolutivo, donde ambos tipos representan una analogía directa con el comportamiento natural.

Para ilustrar los pasos que sigue el funcionamiento básico de un algoritmo genético, se ha planteado el pseudocódigo descrito en el Algoritmo 1.

**Algoritmo 1** Fases de un algoritmo genético

1: Se parte de una población inicial de individuos, donde cada uno de estos representa una solución factible al problema.
2: **while** no se cumpla la condición de parada **do**
3: La población de individuos es evaluada, asignando una puntuación a cada individuo según su valor de fitness.
4: Tras esto, se seleccionan individuos para que tengan descendencia, donde tienen mayor probabilidad de ser seleccionados aquellos individuos que representen soluciones más óptimas que el resto.
5: Estos individuos seleccionados se cruzan, dando lugar a nuevos individuos
6: A estos individuos obtenidos se les aplica una mutación sobre el genotipo, el cual es básicamente la codificación de la solución
7: Posteriormente, se seleccionan los individuos que van a formar parte de la siguiente generación, entre padres e hijos recién generados
8: **end while**

El equivalente a esto en la naturaleza sería la aptitud de un organismo para adaptarse a un entorno, conseguir recursos y sobrevivir, por lo que cuanto más apto sea un individuo, más probabilidad hay de que este sobreviva y tenga descendencia junto con otro individuo seleccionado de la misma manera, cruzando así los genes de ambos. En contraposición, cuanto menos apto sea un individuo, menos probabilidades hay de que sobreviva y tenga descendencia.

El cruce entre individuos produce nuevos individuos, los cuales tienen algunas de las características de sus predecesores, lo que se traduce en que tras cada generación, los individuos de la población tienen en general mejores características que los individuos de anteriores generaciones, lo cual propicia que al abordar problemas con esta técnica se exploren las soluciones del espacio de búsqueda que son más prometedoras.

Por lo tanto, si se diseña de la manera correcta y el caso de uso lo favorece, un algoritmo genético podría ser capaz de dar la solución óptima a un problema, ya que la población irá convergiendo hasta llegar a la misma.

## 1.4. Objetivos

Para tener definido un alcance aproximado del proyecto, se plantearon una serie de objetivos que permitieran dividir las diferentes fases de desarrollo del proyecto, así como también permitieron un orden a la hora de proceder con el. Dichos objetivos se encuentran listados a continuación, además de tener una breve descripción de cada uno de ellos:

- **Investigación y estado del arte del problema:** Se busca ampliar información sobre el problema para tratar de detectar posibles enfoques viables para resolverlo, además de averiguar si una aproximación haciendo uso de un algoritmo genético podría llegar a proporcionar resultados prometedores.

- **Estudio de la adecuación de las tecnologías a utilizar:** Una buena elección de tecnologías puede ayudar mucho en lo que a tiempos de desarrollo, ejecución y rendimiento se refiere, por lo que también se planteó un estudio de las tecnologías disponibles para seleccionar aquellas más adecuadas para el desarrollo del proyecto.
- **Implementación del problema a optimizar, además de un algoritmo básico como, por ejemplo, búsqueda aleatoria, que lo resuelva:** El trabajo gira en torno al desarrollo de algoritmos dirigidos por retos, por lo que para tener una primera aproximación al problema y tener nociones acerca de las soluciones al mismo, se propuso implementar un algoritmo de búsqueda aleatoria.
- **Implementación de un algoritmo evolutivo que permita llevar a cabo la resolución del problema:** Por otro lado, se ha propuesto también el diseño de un algoritmo genético que resuelva el problema y que nos permita obtener mejores soluciones. Teniendo en cuenta que para el proyecto se hace uso Typescript como lenguaje de programación, se ha decidido utilizar la librería GeneticsJS (8), la cuál se ha extendido según las necesidades del proyecto.
- **Diseño, ejecución y análisis del rendimiento de los algoritmos a través de la evaluación experimental:** Por último, se propuso realizar una evaluación experimental tanto del algoritmo genético planteado, como de la búsqueda aleatoria. Esta evaluación servirá para tener un análisis completo del rendimiento de los mismos.

## 1.5. Tecnologías utilizadas

Dentro de los objetivos se planteaba el desarrollo de un algoritmo genético, para el cual se ha tenido que extender la librería *GeneticsJS* (8). Esta librería está implementada en *Typescript*, lo que ha motivado en gran parte la utilización de este lenguaje para el desarrollo del trabajo, además de las ventajas que este proporciona.


Figura 1.2: Logo de GeneticsJS

*Typescript* (9) es un superconjunto de *Javascript*, lo que significa que todo el código escrito en *Javascript* es válido para *Typescript*, pero no al revés. Esto se debe a que el código *Typescript* necesita de una "transpilación" a *Javascript* para poder ser ejecutado. Como característica destacable, *Typescript* incluye tipado estático.


Figura 1.3: Logos de Typescript y Javascript

En concreto, la inclusión de tipado en un código que va a ser transpilado a *Javascript* para poder ser ejecutado podrá parecer de poca utilidad. Sin embargo, el potencial del tipado que ofrece *Typescript* se encuentra principalmente en el tiempo de desarrollo, donde nos permite desarrollar de manera más rápida y segura, puesto que si en el proceso de compilación se encuentran errores, se nos informará de ello.

Por otro lado, tendríamos las ventajas que nos ofrece en lo que respecta a la documentación y a la realización de pruebas sobre las aplicaciones que desarrollemos.

Este lenguaje, en combinación con un IDE como *Visual Studio Code* (10), el cual ha sido utilizado en este caso para el desarrollo del trabajo, proporciona una gran cantidad de información a medida que utilizamos sus características, como podría ser información acerca de métodos, parámetros, objetos e incluso sugerencias sobre los métodos pertenecientes a dichos objetos, entre otras muchas cosas. Además, tendríamos la ventaja, si hacemos uso de Visual Studio Code, de que nos avise en tiempo de desarrollo de los errores que tiene nuestro código Typescript, ahorrándonos el proceso de compilación para tener que descubrirlos.

Figura 1.4: Logo de Visual Studio Code

Puesto que el código *Javascript* resultante de la compilación necesita ser ejecutado, se hará uso de *NodeJS* (11) como entorno de ejecución. Además, también se ha utilizado el API de NodeJS para lanzar procesos y para el manejo de ficheros.


Figura 1.5: Logo de NodeJS

Teniendo en cuenta que para el desarrollo del reto se proveía de una imagen Docker para poder lanzar el simulador, hemos utilizado dicha herramienta para realizar pruebas durante la mayor parte del desarrollo del proyecto. Sin embargo, debido a los elevados tiempos de ejecución de esta imagen, se ha implementado un script en R (12) que permite lanzar el simulador sin necesidad de montar la imagen de Docker.

Figura 1.6: Logo de R

## 1.6. Contribuciones

En este apartado se van a enumerar y detallar las contribuciones realizadas durante el transcurso del proyecto, tanto las que se habían fijado como objetivo en el plan de trabajo como las que no.

- **Desarrollo de un algoritmo genético:** Para el desarrollo del algoritmo genético en Typescript se ha usado como base todo lo disponible en la librería GeneticsJS, la cual se ha ido extendiendo según las necesidades derivadas durante la realización del proyecto. Además de ir realizando las adaptaciones necesarias, también se han revisado y planteado pruebas para las nuevas implementaciones.

- **Lanzamiento de simulaciones haciendo uso de Docker:** Durante el inicio del desarrollo, se implementó el código necesario en Typescript para lanzar simulaciones haciendo uso de la imagen de Docker proporcionada por la competición del GECCO. De esta manera, se podrían analizar los resultados de esas evaluaciones según los parámetros de entrada introducidos.

- **Lanzamiento de simulaciones haciendo uso directo de R:** Con el objetivo de mejorar los tiempos de ejecución y permitir un mayor número de simulaciones, se decidió realizar una instalación local del paquete Babsim.Hospital. Además de esto, se implementó un script en R que emula el comportamiento de la imagen Docker proporcionada por la competición del GECCO y se adaptó para poder ejecutar simulaciones sobre el código fuente instalado de manera local pero con unos tiempos de ejecución mejorados.

- **Manejo de las ejecuciones mediante línea de comandos:** Por último, para facilitar la ejecución de los algoritmos implementados, se ha incluido una herramienta para la gestión de las mismas mediante línea de comandos y también se han incluido ficheros con las configuraciones ya planteadas para que no sea necesario configurar un elevado número de parámetros.

# Capítulo 2

# Estudio del Estado del Arte

Para llevar a cabo la resolución del problema, se ha realizado previamente un estudio del estado del arte del mismo. Dicho estudio se ha basado, en primera instancia, en la definición de una serie de palabras clave que, en conjunto, puedan darnos una idea base de los aspectos que se tratan junto con el problema.

Por otro lado, se ha realizado una revisión sistemática en base a esas palabras clave, aplicando la metodología *"Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA)"* (13), la cual es utilizada comúnmente en el ámbito científico para la evaluación y el análisis de artículos.

A continuación, se detallará el procedimiento seguido para llevar a cabo dicha metodología en lo que a nuestro análisis del estado del arte respecta.

## 2.1. Objetivo

Con la definición de las palabras clave y el análisis sistemático lo que se busca es ampliar la información que se tiene acerca del problema, así como también encontrar diversos planteamientos o estrategias que puedan ayudar a resolver el mismo.

Debido a que el problema es bastante reciente y no hay apenas implementaciones del mismo o artículos relacionados más allá de los que ha publicado el equipo encargado del desarrollo del modelo de simulación, se han decidido estudiar todos aquellos recursos y técnicas que puedan ser de utilidad para poder llegar a implementar una solución que permita optimizar el modelo de simulación. Dentro de estos, destacaríamos todo lo relacionado con algoritmos evolutivos y más concretamente los algoritmos genéticos, así como también técnicas de optimización de parámetros, problemas basados en simulaciones por computador y problemas de caja negra.

## 2.2. Palabras clave

Para llevar a cabo la búsqueda y revisión sistemática de artículos, se han realizado búsquedas en diversas bases de datos de carácter científico. Para dicha revisión se utilizaron las siguientes palabras clave en los buscadores de las bases de datos:

- Optimization-via-simulation
- Parameter Optimization
- Genetic Algorithms
- COVID-19
- Sensitivity analysis
- Hospital resource planning

Las consultas realizadas en los buscadores consisten básicamente en la utilización de cada palabra clave por separado o combinaciones de las mismas, de manera que haya una mayor diversidad de recursos para su revisión.

## 2.3. Bases de datos

La búsqueda ha sido realizada sobre las siguientes bases de datos:

- IEEE (`https://ieeexplore.ieee.org/Xplore/home.jsp`)
- ScienceDirect (`https://www.sciencedirect.com/`)
- SpringerLink (`https://link.springer.com/`)
- Scopus (`https://www.scopus.com/`)

## 2.4. Criterios de inclusión

De entre todos los artículos encontrados, se incluirán dentro del análisis en profundidad aquellos artículos que cumplan alguna de las siguientes condiciones:

- Artículos con propuestas de algoritmos
- Artículos con análisis y pruebas de rendimiento
- Artículos relacionados de forma directa con el problema
- Artículos con propuestas para la optimización de parámetros
- Artículos en español o inglés

## 2.5. Criterios de exclusión

Al igual que se han definido unos criterios de inclusión, también realizaremos lo correspondiente con los criterios de exclusión:

- Artículos duplicados
- El artículo consta solo de una introducción o resumen
- El artículo no está en español o inglés
- El artículo dista del ámbito del problema a abordar

## 2.6. Búsqueda y selección de artículos

En la siguiente figura se muestra el diagrama de flujo que ilustra en detalle cómo se ha llevado a cabo la búsqueda y selección de los artículos. Siguiendo lo indicado en la metodología *PRISMA* (13), el flujo se divide en diferentes etapas, en las cuales se van excluyendo aquellos artículos que no cumplen los requisitos establecidos.

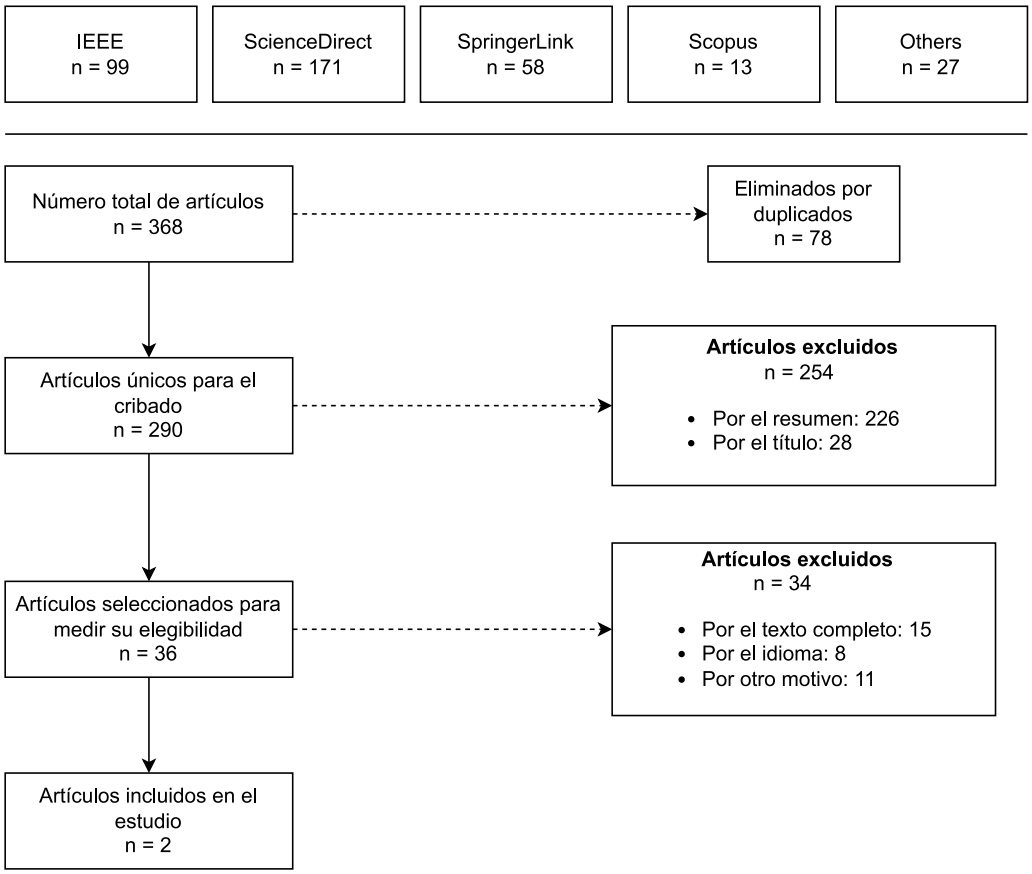



Figura 2.1: Diagrama de metodología PRISMA

## 2.7. Artículos seleccionados

Como indicamos en el apartado anterior, los artículos incluidos para el estudio son aquellos que cumplen los requisitos establecidos para la revisión sistemática y para los cuales se presentarán una breve descripción de cada uno.

**Model-based methods for continuous and discrete global optimization** (14)

En este artículo se realiza un estudio de las aplicaciones que tienen sobre la optimización global los métodos de optimización basados en modelos, y más concretamente la "Surrogate-based Optimization (SBO)" u Optimización basada en sustitutos. Son considerados de los métodos más eficientes para problemas de optimización cuyas demandas de tiempo o recursos son elevadas.

En términos generales, se trata de definir una taxonomía comprensible para algoritmos de optimización basados en el modelo, ya que la gran diversidad de clasificaciones aplicables tanto a los algoritmos como a los propios tipos de modelos lleva en muchas ocasiones a confusión, por lo que deciden enfocar esta taxonomía desde un punto de vista algorítmico.

El artículo versa también sobre el algoritmo SPOT (15), el cual define en sus inicios como una caja de herramientas para la optimización secuencial de parámetros que fue originalmente desarrollado como una "toolbox" para el análisis y compresión de los principios de funcionamiento de los algoritmos evolutivos. Este implementa un enfoque secuencial basado en modelos sustitutos para la optimización de los parámetros y en la actualidad es visto como un algoritmo de optimización y además sirve para ajustar parámetros.

En este se tratan muchos aspectos de relevancia para con este trabajo como pueden ser la selección de un modelo adecuado para la optimización de la simulación, la correcta evaluación del rendimiento de los propios algoritmos y modelos o el uso de algoritmos de búsqueda estocásticos.

Estos últimos son introducidos como algoritmos de fácil implementación y robustos. Por último, también se comenta que en ciertas ocasiones los algoritmos estocásticos requieren la especificación de varios parámetros de configuración, por lo que es necesario realizar un ajuste de estos parámetros. Esto tiene relación directa con el problema que se pretende abordar en este trabajo, puesto que precisamente el simulador tiene un comportamiento estocástico y requiere además de la configuración de una serie de parámetros concretos para funcionar adecuadamente. Además, según el artículo un buen ejemplo de algoritmos de este tipo podrían ser la búsqueda tabú o los algoritmos evolutivos.

**Resource Planning for Hospitals Under Special Consideration of the COVID-19 Pandemic: Optimization and Sensitivity Analysis** (16)

Este artículo versa sobre el software *BaBSim.Hospital* (5) y el gran reto de optimización que este plantea debido al gasto computacional y la alta complejidad del problema que aborda.

En primera instancia se destaca la importancia de la recopilación y el análisis de los datos utilizados para las predicciones, por lo que el software implementa un proceso "Extract Trasnform Load (ETL)" para analizar los datos de diversos institutos de los cuales obtienen información anónima sobre todos los casos registrados en Alemania y que están en constante actualización de manera automática gracias a los procesos de "Continuous Integration/Deployment (CI/CD)" que ayudan a minimizar la interacción humana.

El proceso de modelización incluye cuatro tipos de parámetros: probabilidades de transición, duraciones, propiedades de la distribución y factores de riesgo. Los resultados obtenidos en las simulaciones pueden servir para realizar ejecuciones de optimización que permitan mejorar los ajustes de los parámetros propuestos por los expertos.

Con respecto a la optimización del problema, se descubrió que el uso de optimizadores "out-of-the-box" no funciona debido a la dimensión y lo ruidoso del problema. Sin embargo, el uso de optimizadores basados en modelos si que proporciona buenos resultados. Es por ello que se eligió el algoritmo *SPOT* (15) el cual sirve para realizar optimizaciones basadas en modelos y ajustes de algoritmos. Además, se lanzaron simulaciones en paralelo y además se aprovecharon los modelos sustitutos para realizar un análisis de sensibilidad.

Para los experimentos se siguió un enfoque *"Simulation Model-Based Sensitivity Analysis" (SMBSA)* , probando a partir del análisis de sensibilidad que se puede ejecutar el optimizador con menos parámetros sin una pérdida de calidad significativa ya que tienen poca relevancia sobre el modelo y que hay ciertos parámetros que dominan la calidad de la simulación, lo cual puede ser de utilizad al proceder a realizar nuevas optimizaciones sobre el modelo.

# Capítulo 3

# Descripción del problema

## 3.1. Genetic and Evolutionary Computation Conference (GECCO)

Tal y como se comentaba con anterioridad, el problema *Optimization of a simulator model for a capacity and resource planning task for hospitals during COVID-19* se propone como un *reto* (3) durante la GECCO 2021, siendo esta una conferencia de renombre a nivel mundial en el ámbito de la computación evolutiva y genética, donde cada año se presentan los últimos avances en estos campos y muchos otros relacionados también con la computación.

## 3.2. Descripción del problema

En la descripción proporcionada para el reto, se presenta como una herramienta que hace uso de un modelo de simulación para la estimación de recursos y la planificación de tareas para hospitales, enfocada de manera explícita en las dificultades causadas por la pandemia de COVID-19. Cada evaluación simula las trayectorias que siguen los pacientes infectados durante su estancia en un hospital, lo que permite al simulador modelar el número de camas UCI en función del número de individuos infectados.

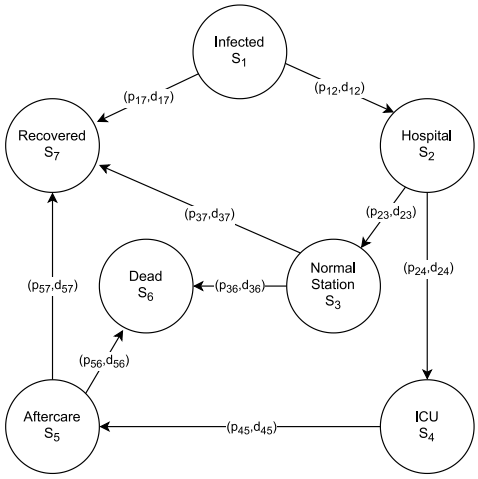



Figura 3.1: Diagrama simplificado del flujo de un paciente infectado (1)

BabSim.Hospital (5) hace uso de la simulación discreta de eventos (DES) ya que proporciona información sobre el riesgo y la eficacia del proceso al realizar cambios en las configuraciones, o sobre las consecuencias de las estrategias de rediseño. El DES procesa cada una de las infecciones registradas hasta la recuperación o muerte de los pacientes. Tal y como podemos observar en la Figura 3.1, los pacientes siguen una trayectoria, moviéndose del estado $S_i$ al estado $S_j$ con probabilidad $p_{i,j}$ tras una duración específica de la transición $d_{i,j}$.

La complejidad de este problema de optimización reside en el coste computacional de las evaluaciones que realiza, de manera que encontrar una configuración óptima de los parámetros de entrada del simulador es una tarea difícil.

### 3.2.1. Parámetros de entrada del simulador

El simulador recibe como entrada 29 parámetros, presentados en la Tabla 3.1, donde cada parámetro tiene un rango de valores recomendado que ha sido acordado con profesionales de la medicina, así como unos valores por defecto para establecer una configuración inicial de los mismos. El objetivo es encontrar un conjunto de valores lo más óptimo posible para este conjunto de parámetros de entrada al simulador.

Tabla 3.1: Parámetros de entrada del simulador con sus rangos recomendados

| Parameter | Range |
|---|---|
| AmntDaysInfectedToHospital | 6-14 |
| AmntDaysNormalToHealthy | 7-13 |
| AmntDaysNormalToIntensive | 3-7 |
| AmntDaysNormalToVentilation | 3-9 |
| AmntDaysNormalToDeath | 3-7 |
| AmntDaysIntensiveToAftercare | 5-9 |
| AmntDaysIntensiveToVentilation | 3-5 |
| AmntDaysIntensiveToDeath | 3-7 |
| AmntDaysVentilationToIntensiveAfter | 25-35 |
| AmntDaysVentilationToDeath | 17-25 |
| AmntDaysIntensiveAfterToAftercare | 2-5 |
| AmntDaysIntensiveAfterToDeath | 1-7 |
| GammaShapeParameter | 0.25-2 |
| FactorPatientsInfectedToHospital | 0.05-0.15 |
| FactorPatientsHospitalToIntensive | 0.07-0.11 |
| FactorPatientsHospitalToVentilation | 0.005-0.02 |
| FactorPatientsNormalToIntensive | 0.07-0.13 |
| FactorPatientsNormalToVentilation | 1e-04-0.002 |
| FactorPatientsNormalToDeath | 0.08-0.12 |
| FactorPatientsIntensiveToVentilation | 0.25-0.35 |
| FactorPatientsIntensiveToDeath | 0.08-0.12 |
| FactorPatientsVentilationToIntensiveAfter | 0.5-0.9 |
| FactorPatientsIntensiveAfterToDeath | 1e-06-0.01 |
| AmntDaysAftercareToHealthy | 2-4 |
| RiskFactorA | 1e-06-1.1 |
| RiskFactorB | 1e-06-0.0625 |
| RiskMale | 1-2 |
| AmntDaysIntensiveAfterToHealthy | 2-5 |
| FactorPatientsIntensiveAfterToHealthy | 0.5-0.75 |

Estos parámetros de entrada del simulador coinciden con las variables de decisión de nuestro problema, es decir, que cada solución que se plantee para este problema va a estar compuesta de una configuración concreta de estos 29 parámetros.

Otras de las variables que usa el simulador internamente contienen datos que describen la propagación de la pandemia a lo largo del tiempo, el uso de recursos, el número de camas de UCI disponibles o el número de pacientes con respiradores.

Todos estos datos se obtienen del Instituto Robert Koch (RKI) y el Instituto Deutsche Interdisziplinare (DIVI) y son actualizados de manera automática en el repositorio oficial del paquete mediante el uso de scripts, aunque si utilizamos una versión concreta del simulador los datos que se utilizarán son los de esa versión concreta. Por otro lado, los riesgos asociados que tiene en cuenta el simulador, como la edad y el sexo de los pacientes, también se obtienen en esta etapa.

### 3.2.2. Salida del simulador

Como ya se ha comentado previamente, el simulador permite obtener una estimación del número de camas de UCI, personal y recursos en general que son necesarios para atender a los pacientes infectados por COVID-19. Esta estimación viene dada como una única salida numérica del simulador representando la demanda de recursos.

La salida del simulador, es sin duda alguna, la que permite evaluar si una configuración de parámetros podría llegar a ser en cierta manera, mejor que otra configuración dada, debido a que el conjunto de datos sobre el que se está trabajando es el mismo. Esto es, si con un conjunto de parámetros concretos del simulador se requiere una menor movilización de recursos que con otra configuración, se hará uso de la configuración que consiga que esa demanda de recursos sea mínima.

Sin embargo, un aspecto importante a tener en cuenta es la naturaleza estocástica del simulador, el cual hace uso de semillas distintas en cada simulación para proporcionar aleatoriedad en los resultados obtenidos.

Esto lo que quiere decir, es que para los mismos parámetros de entrada del simulador, vamos a obtener diferentes salidas. Por lo tanto, aunque la salida del simulador sea algo determinante a la hora de definir si una configuración es más óptima que otra, no es fiable debido a esta naturaleza estocástica y esto implica que para poder llegar a obtener resultados robustos, habría que lanzar una gran cantidad de réplicas para cada conjunto de parámetros de entrada que se plantee.

## 3.3. Participación en el reto del GECCO

Aquellas personas que quieran participar en el reto, podrán aportar tantos algoritmos de optimización del problema como deseen, en el lenguaje de programación que gusten, y podrán participar de manera online, sin necesidad de realizar ningún tipo de instalación.

La competencia permite que los competidores elijan si quieren participar en todas las modalidades del reto o sólo en una de ellas, pero para cualquiera de las opciones, el algoritmo que encuentre el mejor valor de la función objetivo, es el ganador. Las modalidades que hay son las siguientes:

- **Track 1 - Evaluaciones ilimitidadas:** En esta modalidad la cantidad de evaluaciones de la función objetivo a realizar será determinada por el tiempo de computación que permita la máquina en donde se ejecuta el optimizador. La manera de participar en esta modalidad es haciendo uso de la imagen de Docker proporcionada para realizar evaluaciones.
- **Track 2 - Evaluaciones limitadas:** En esta otra modalidad, los algoritmos que se suban sólo podrán usar 200 evaluaciones.

En concreto, durante el desarrollo del trabajo se ha optado por seguir el “Track 1” debido a que el sitio Web habilitado para la segunda modalidad está fuera de servicio, además de que para poder experimentar adecuadamente con un algoritmo genético necesitaremos realizar una gran cantidad de evaluaciones.

Aunque la modalidad seleccionada permite realizar evaluaciones ilimitadas, hay que recalcar que estas dependen exclusivamente de la máquina donde se ejecuten, por lo que aunque en etapas iniciales del desarrollo se hizo uso de la imagen de Docker, luego se buscaron alternativas que permitían realizar un mayor número de evaluaciones en la misma cantidad de tiempo.

## 3.4. Analogía con la Computación Evolutiva

En este apartado se va plantear una analogía del problema de optimización a abordar y su contexto con la computación evolutiva, lo que nos dará una visión clara de la aplicabilidad de la computación evolutiva sobre el mismo.

Como habíamos comentado anteriormente, en la evolución natural se parte de una población de individuos que conviven en un entorno concreto, en el cuál viven y tratan de adaptarse para poder sobrevivir. Por lo general, aquellos individuos que tienen mejores rasgos o cuya adaptación al entorno es mejor, son los que mayor probabilidad tienen de sobrevivir y generar descendencia junto a otros de los individuos que sobrevivan. Los individuos resultantes de la descendencia heredan rasgos de sus predecesores, pero también sus genes sufren ciertos cambios que hacen que tengan características únicas, que es lo que se conoce como mutaciones.

Si extrapolamos esta lógica al problema que queremos abordar, el **entorno** sería el simulador, donde introduciremos una configuración de parámetros de entrada concretos, que en este caso conformarían un **individuo** o **solución** factible de nuestro problema.

Tras introducir los parámetros de entrada, el simulador los procesa y realiza la simulación correspondiente, obteniendo así un resultado o salida como respuesta.

La salida obtenida es la que nos dirá la calidad de esa solución, ya que cuanto más bajo sea el valor de la salida, menos recursos hay que movilizar, por lo que la solución es mejor y por lo tanto podríamos tener esto en cuenta para plantear una **función de fitness** que evalúe la aptitud de los individuos en torno a esto.

Teniendo en cuenta que se busca plantear un algoritmo genético que solucione nuestro problema, los individuos que manejemos en el problema serán configuraciones distintas de los parámetros de entrada del simulador, ya que lo que se tiene como objetivo es encontrar una configuración de parámetros óptima para el simulador.

Por lo tanto, partiendo de una serie de individuos o configuraciones iniciales se irán creando nuevas configuraciones a través de operaciones de cruce y mutación, las cuales serán evaluadas y seleccionadas en base a su calidad hasta llegar a una configuración óptima de los parámetros de entrada.

# Capítulo 4

# Algoritmos de resolución

## 4.1. Simulador

El simulador se encarga de modelar el número de camas de UCI en función del número de individuos infectados. Esto lo consigue simulando las trayectorias típicas de los pacientes con COVID-19 durante su estancia en el hospital, desde la infección hasta la recuperación o la muerte.

Para poder realizar estas simulaciones, requiere de una serie de datos que describan la propagación de la pandemia a lo largo del tiempo, así como también el uso o la disponibilidad de recursos. Aparte de disponer de todos estos datos, el simulador también requiere que se establezcan una serie de parámetros de entrada, que serán utilizados a modo de variables de decisión para la estimación de los recursos en conjunto con todo lo indicado anteriormente.

En apartados anteriores se había comentado que para la participación en el reto a través del "Track 1" se dispone de una imagen de Docker, por lo que en este apartado se detallará todo lo relativo a la configuración, codificación y simulación de las ejecuciones con docker, así como también el incremento de velocidad de ejecución llevado a cabo con respecto al simulador.

### 4.1.1. Ejecuciones del simulador

Para poder hablar de la configuración necesaria para realizar ejecuciones, primero debemos abordar lo necesario para poder lanzar una ejecución en el simulador.

En el apartado del reto, proporcionan un *paquete de recursos* (1), donde se encuentra un documento con una breve explicación del mismo, así como también una descripción de lo que aporta el software BabSim.Hospital. Dentro de este mismo documento, se encuentra detallada la forma en la que se debe de ejecutar el comando de Docker concreto para lanzar una simulación:

```
docker run --rm mrebolle/r-geccoc:Track1 -c 'Rscript objfun.R "6,7,3,
3,3,5,3,3,25,17,2,1,0.25,0.05,0.07,0.005,0.07,1e-04,0.08,0.25,0.08,
0.5,1e-06,2,1e-06,1e-06,1,2,0.5"'

68.92036
```

Tal y como se puede observar, para poder lanzar una simulación, necesitaremos montar una imagen de docker para lo que usaremos el comando $run$, al que le pasaremos como parámetros la imagen concreta a utilizar, además del comando que se va a ejecutar en la misma, el cual invoca a un script en R a través de RScript, al cual se le pasan a su vez todos los parámetros de entrada que requiere el simulador. Una vez se ejecute el comando, obtendremos el resultado y finalizará la ejecución en Docker.

Teniendo en cuenta que para el desarrollo del trabajo no se realizará una sola simulación, sino una gran cantidad de ellas, es ideal automatizar el proceso de generación de las configuraciones (conjunto de parámetros de entrada al simulador), para posteriormente ser ejecutadas.

Debido a esto, la configuración de una simulación se ha gestionado de manera que cada vez que se quiera probar un conjunto de parámetros de entrada concreto, se generará una cadena de texto con el comando de Docker a ejecutar sobre la correspondiente imagen. Cuando hayamos terminado de montar esta cadena de texto, ya tendremos lista nuestra configuración, puesto que sólo habría que lanzar un proceso de NodeJS, por ejemplo, que se encargue de ejecutar lo indicado en la misma.

### 4.1.2. Incremento de la velocidad de ejecución del simulador

En este apartado se van a introducir algunos de los problemas encontrados durante el desarrollo del trabajo que han llevado a plantear el realizar un incremento en la velocidad de ejecución de las simulaciones.

Primero que nada y para entrar en contexto, el tiempo que pasa desde que se lanza una simulación hasta que se obtiene un resultado, es de aproximadamente unos 30 segundos en una máquina convencional y, como en este caso, se desean ejecutar cientos de simulaciones, esperar tal cantidad de tiempo por cada una de ellas es algo inasumible.

```
time docker run --rm mrebolle/r-geccoc:Track1 -c’Rscript objfun.R "10,10,4,
5,7,8,3,5,29,17,5,6,1.25,0.06,0.1,0.014,009,0.001,0.11,0.32,0.1,0.74,0.007,
3,0.37,0.05,2,2,0.67"’
69.86047
0.11s user 0.07s system 0% cpu 30.735 total
```

Para tener una idea más clara de lo que esto puede significar, supongamos que se desean probar 100 configuraciones distintas de parámetros. Este proceso llevaría casi una hora en una máquina convencional, cuando realmente no se está ejecutando un número elevado de simulaciones, mientras que, si se desean ejecutar 10.000 simulaciones para poder obtener un mayor número de resultados y poder analizarlos, conllevaría una espera de, aproximadamente, más de 80 horas. Además de todo esto, habría que sumar el tiempo de ejecución de los algoritmos implementados, lo que se resume en unos tiempos de espera muy elevados para pruebas poco significativas.

Para poder resolver esta problemática, es necesario buscar una alternativa a Docker para la ejecución del simulador. Por lo tanto, una opción viable es hacer uso del paquete de R de BabSim.Hospital mediante una instalación local, de manera que los tiempos de ejecución se vean reducidos en comparación a lo que tardaría el contenedor en descargar la imagen, procesar los comandos, ejecutarlos y obtener una salida.

En este proceso realizado para reemplazar a Docker como herramienta para la ejecución del simulador, se ha hecho uso de un *repositorio público* (5) con todo el código fuente del proyecto y de lo proporcionado en la *página oficial del paquete en CRAN* (17). Un inconveniente encontrado durante este proceso, es que en el código fuente del repositorio no se disponía del script que usa la imagen de Docker proporcionada para el reto, por lo que se ha tenido que implementar un script en R que trate de emular dicho funcionamiento.

Tras haber llevado a cabo la implementación del script, el tiempo de ejecución de una simulación ha pasado de 30 segundos a 2 segundos en la misma máquina, lo cual se traduce en una mejoría de tiempo de aproximadamente un 93.3 %.

```
time Rscript babsim/BabsimHospital.R 10 10 4 5 7 8 3 5 29 17 5 6 1.25 0.06
0.1 0.014 009 0.001 0.11 0.32 0.1 0.74 0.007 3 0.37 0.05 2 2 0.67
[1] 123.1999
2.16s user 0.33s system 76% cpu 3.262 total
```

Esta implementación ha derivado en algunos otros problemas, como son la aleatoriedad de los resultados, ya que tras realizar la mejora, todos los resultados estaban siendo iguales para las mismas entradas, con lo cual se han tenido que realizar ciertas modificaciones en lo que respecta a la semilla que usa el simulador para gestionar su naturaleza estocástica.

## 4.2. Librería GeneticsJS

En este apartado se explicarán las implicaciones de utilizar GeneticsJS para resolver el problema, los requerimientos que esto supone y las adaptaciones realizadas sobre la librería para poder cumplir con dichos requerimientos.

Tal y como se ha comentado en sucesivas ocasiones, GeneticsJS es la librería que se ha elegido para llevar a cabo el desarrollo del algoritmo genético. Esta librería está centrada en proveer clases de utilidad que permitan crear individuos, poblaciones, generaciones, mutaciones y gran parte de las operaciones que conforman un algoritmo evolutivo. Ha sido creada, extendida y mantenida por compañeros del Grado en Ingeniería Informática de la Universidad de La Laguna, los cuales han hecho un gran esfuerzo porque la librería esté lo más documentada y probada posible.

#### Requerimientos

En esta ocasión, el problema a abordar tiene unos requerimientos bastante específicos, los cuales obligan a replantear el funcionamiento actual de la librería. Estos requerimientos son los siguientes:

- Cada individuo está compuesto por 29 genes, donde los genes se corresponden con los parámetros de entrada del simulador.
- Como el genotipo de los individuos es mixto, cada gen está formado por una variable entera o por una variable real.
- Cada gen tiene un rango recomendado de valores.
- Puesto que cada gen tiene un rango recomendado de valores, al inicializar una población de individuos, no se podrá hacer uso de un mismo rango para cada uno de los genes de los individuos.

Estos requisitos difieren en funcionamiento con respecto de la implementación actual de la librería en ciertos aspectos, como lo son por ejemplo la generación de individuos con variables mixtas, ya que hasta el momento, sólo habían implementados individuos para variables enteras, flotantes y binarias.

### Adaptaciones necesarias

En el apartado anterior se han comentado los requerimientos que tiene el problema de optimización que se aborda en este trabajo y en este apartado se indicarán las adaptaciones realizadas en la librería **GeneticsJS** en base a esos requerimientos.

Para poder usar la librería en la resolución del problema, se han realizado las siguientes adaptaciones:

- Se ha añadido el manejo de individuos mixtos, puesto que hasta el momento sólo se disponía de individuos cuyas variables eran enteras, flotantes o binarias.
- Se ha modificado el manejo de los rangos de los genes, para poder especificar un rango de valores con cada gen del individuo, en vez de manejar un rango de valores para todos los genes del individuo. Esto afectaba en absolutamente todos los aspectos del algoritmo, desde la creación de individuos o la inicialización de la población, hasta los cruces o mutaciones de los mismos.
- Se ha modificado la lógica planteada en el código fuente del algoritmo evolutivo que proporciona la librería para poder pasarle una población ya inicializada. Esto interesa puesto que de lo contrario solo se podría especificar un rango dentro de los parámetros de esta clase, el cual sería usado para generar todos los genes de cada individuo de la población inicial.

En el apartado 5 se detalla un poco más en profundidad todo lo relativo a estas adaptaciones y parte de las implementaciones realizadas durante el trabajo.

## 4.3. Algoritmos

### 4.3.1. Algoritmo de Búsqueda Aleatoria

La implementación del algoritmo de búsqueda aleatoria, más conocido como *"Random Search"*, se planteó con el objetivo de probar el funcionamiento del simulador haciendo uso de la imagen de Docker proporcionada para la realización de pruebas con respecto al reto de optimización.

Los resultados obtenidos en las pruebas con este algoritmo no son realmente significativos, ya que simplemente se generan posibles soluciones al problema y se van evaluando, pero no se sigue una técnica concreta que permita converger hacia una solución óptima del problema. Sin embargo, si es de destacar que al elevar el número de iteraciones del algoritmo se brinda una mayor posibilidad de obtener mejores soluciones.

El pseudocódigo del algoritmo de búsqueda aleatoria es el que se muestra en el Algoritmo 2.

**Algoritmo 2** Búsqueda aleatoria

```
1: Best = Inicializar solución candidata aleatoriamente
2: BestFitness = Evaluar Best
3: while not condición de parada do
4:     Current = Seleccionar aleatoriamente una solución candidata
5:     CurrentFitness = Evaluar Current
6:     if CurrentFitness menor que BestFitness then
7:         Best = Current
8:         BestFitness = CurrentFitness
9:     end if
10: end while
11: return Best
```

### 4.3.2. Algoritmo Genético

La implementación del algoritmo genético tiene como objetivo tratar de obtener configuraciones de parámetros lo más cercanas posibles a la óptima. Al carecer de configuraciones de referencia, no se pueden realizar comparaciones con los resultados obtenidos para ver la calidad de estos.

En este caso, la implementación del algoritmo genético ha sido planteada de manera que se puedan configurar gran parte de los elementos del mismo a la hora de realizar ejecuciones, aunque no está totalmente parametrizado.

Al igual que en los algoritmos evolutivos, un algoritmo genético parte de una población inicial de individuos generada de manera aleatoria. Tras esto, los individuos de esta población inicial son evaluados haciendo uso de la función de fitness la cual establece el fitness de un individuo como la media obtenida tras lanzar un número predeterminado de ejecuciones del simulador con los mismos parámetros de entrada del individuo. Por lo tanto, se define si un individuo es mejor que otro, si la media del fitness de las réplicas de un individuo es menor que la media del fitness de las réplicas del otro individuo.

Luego, de entre esta población inicial se seleccionan aleatoriamente dos individuos para generar la descendencia. Tras aplicarles a estos individuos un operador de cruce con la probabilidad de cruce configurada, se obtienen dos individuos. Esto se repite hasta que la descendencia tenga el mismo tamaño que la población actual. A cada individuo obtenido durante este proceso se le aplicará también una mutación con la probabilidad de mutación establecida que, en este caso, se corresponde con $1/29$, donde $29$ es el número de genes de un individuo. Con esta probabilidad de mutación nos aseguramos de que, al menos, se mute un gen de cada individuo.

Tras haber generado la descendencia, se realiza un reemplazo donde se unen todos los individuos de la población actual y la descendencia, para luego dejar en la nueva población los $n$ mejores individuos, es decir, que se eliminan los peores individuos de ambas generaciones. Esta técnica de reemplazo es conocida como “replace worst” (18).

Por último, cuando se cumple la condición de parada, se va a devolver el mejor individuo, el cual se corresponde con el mejor individuo encontrado durante toda la ejecución del algoritmo genético.

El pseudocódigo que se corresponde con la implementación del genético es el que se muestra en el Algoritmo 3.

**Algoritmo 3** Algoritmo Genético

```
1: Inicializar la población
2: Evaluar cada individuo o solución de la población
3: while not condición de parada do
4:     while tamaño de descendencia menor que tamaño población actual do
5:         Seleccionar padres aleatoriamente
6:         Cruzar padres para obtener nuevos hijos
7:         Mutar los hijos obtenidos
8:         Añadir hijos a la descendencia
9:     end while
10:    Seleccionar supervivientes entre padres y descendientes
11: end while
12: return mejor individuo
```

## 4.4. Modo de uso

Para facilitar la utilización de los algoritmos implementados se ha hecho uso de la librería *Yargs* (19), la cual permite manejar comandos de una manera sencilla a través de línea de comandos (CLI). Como los parámetros que se manejan para los algoritmos desarrollados son distintos, se ha decidido separar la implementación del manejo de comandos en dos implementaciones.

### 4.4.1. Algoritmo de Búsqueda Aleatoria

A continuación, se puede observar un ejemplo del modo de uso de la búsqueda aleatoria:

```
npm run random-search -- -i 100 -t 5 -o random_search_result
```

Los parámetros configurados para la búsqueda aleatoria son los siguientes:

- **i | iterations:** Indica el número de iteraciones a realizar. Tiene como valor por defecto 5.
- **t | times:** Indica el número de veces que se lanzará la búsqueda aleatoria. Tiene como valor por defecto 1.
- **o | output | outputFile:** Indica el nombre del fichero donde se almacenarán los resultados de la ejecución. Tiene como valor por defecto “randomSearch”.

### 4.4.2. Algoritmo Genético

A continuación, se puede observar un ejemplo del modo de uso del algoritmo genético:

```
npm run genetic -- -p 2 -g 2 -c 0.8 -f uniform.ts -o ga_Result
```

Los parámetros configurados para el algoritmo genético son los siguientes:

- **p | population | populationSize:** Indica el tamaño de la población. Tiene como valor por defecto 6.
- **g | generations | maxGenerations:** Indica el número máximo de ejecuciones del algoritmo. Tiene como valor por defecto 5.
- **r | replics:** Indica el número de réplicas a lanzar por cada individuo. Tiene como valor por defecto 1.
- **f | file:** Indica el fichero de configuración a utilizar para establecer la mutación. Pudiendo tomar como valores *uniform.ts*, *nonUniform.ts*, *polynomial.ts*, los cuales son los ficheros de configuración establecidos con las mutaciones implementadas. Tiene como valor por defecto *uniform.ts*.
- **c | crossoverRate:** Indica la probabilidad de crossover. Tiene como valor por defecto 0,8.
- **o | output | outputFile:** Indica el nombre del fichero donde se almacenarán los resultados de la ejecución. Tiene como valor por defecto “genetic”.

## 4.5. Enlaces de interés

En este apartado se incluyen los enlaces a los repositorios que han formado parte del desarrollo de este proyecto:

- **Organización de GeneticsJS:** `https://github.com/GeneticsJS`
- **Repositorio de GeneticsJS:** `https://github.com/GeneticsJS/GeneticsJS`
- **Fork realizado de GeneticsJS:** `https://github.com/dtote/GeneticsJS`
- **Repositorio con los algoritmos implementados:** `https://github.com/dtote/tfg-optimization-of-simulation-model-for-resource-planning-in-hospitals`

# Capítulo 5

# Detalles de Implementación

En este apartado se entrará al detalle de las implementaciones realizadas tanto para el algoritmo genético como para la búsqueda aleatoria, así como todos los componentes implicados en las mismas.

### Constantes

Durante el desarrollo del trabajo se fueron definiendo una serie de constantes para aquellos valores estáticos que eran utilizados en diferentes sitios.

- *Número de parámetros utilizados por el simulador*
- *Rangos recomendados para los valores de los parámetros*

Los 29 parámetros con sus rangos recomendados son los que se mostraban en la Tabla 3.1, donde veíamos las cotas inferiores y superiores que definen estos rangos.

La definición y utilización de estas constantes en el código facilita la legibilidad del mismo, permitiendo además una mayor adaptabilidad al cambio, puesto que si se tuviera que modificar algún valor, este cambio se aplicaría sólo en las constantes y el código seguiría funcionando correctamente.

### Motores de ejecución

Con el término *motor de ejecución* nos referimos a el encargado de ejecutar la simulación, donde podríamos llegar a distinguir dos motores distintos en nuestro caso.

- **Docker**: Es la herramienta principal que se ha utilizado durante el desarrollo del proyecto para hacer pruebas y es quien se encarga de lanzar la simulación haciendo uso de una imagen concreta. Cabe destacar que en etapas posteriores se ha descartado su utilización por lo elevados que son los tiempos de ejecución de las simulaciones con la imagen de docker.
- **Rscript**: Es la herramienta utilizada para realizar las simulaciones haciendo uso del script en R planteado para tratar de reducir los tiempos de ejecución que teníamos con docker. Este es el motor utilizado en todas las simulaciones que se lanzan durante el algoritmo genético.

### Manejadores de comandos

Los manejadores de comandos son las clases encargadas de proporcionar los métodos necesarios para generar y ejecutar un comando. A su vez, se hace referencia con comando a el conjunto de todos los parámetros que recibe un motor concreto donde vayamos a lanzar la ejecución, distinguiendo en este caso dos motores como ya se ha comentado previamente, los cuales son Docker y Rscript.

En base a estos motores de ejecución se han planteado dos clases, pensadas específicamente para trabajar con la construcción de comandos y la ejecución de los mismos en dichos motores. Estas clases son *DockerCommand* y *RscriptCommand*, de las cuales se está utilizando principalmente la de Rscript por las ventajas significativas en el tiempo de ejecución.

### Individuos

El elemento base de un algoritmo genético es el individuo, el cual tiene en su genotipo la codificación de una posible solución al problema que se aborda. Como en el problema que nos atañe el objetivo es obtener una configuración óptima de los parámetros de entrada de los cuales hace uso el simulador, un individuo de la población codificará en su genotipo una configuración de estos 29 parámetros, que en este caso se componen de valores mixtos, es decir, tanto enteros como decimales, por lo que en principio se hará uso de un individuo numérico ya que su genotipo será numérico.

Aparte de la codificación de la solución, se debe tener en cuenta que al ser un algoritmo genético lo que se va a utilizar, necesitaremos poder mutar los genes de los individuos tras cada cruce.

Para poder cumplir con estas necesidades, se ha implementado un individuo **MixedIndividual** para poder representar individuos cuyos genes están compuestos por variables mixtas, y esta clase hereda de la clase **NumericIndividual**, la cual aparte de permitir la modificación de genes, ya que hereda de **MutableIndividual**, permite también definir un rango para los genes del individuo.

El individuo mixto, al igual que el resto de individuos, recibe un parámetro obligatorio y otro opcional con un valor por defecto, siendo el opcional el rango entre el que se generarán los genes del individuo, y siendo el primer parámetro la representación del individuo, ya sea en formato string o como array de números.

En este caso, en dicha representación podremos pasar tantos valores enteros como flotantes, los cuales se interpretarán adecuadamente gracias a la implementación realizada de la clase **MixedReader**, cuya finalidad es precisamente interpretar esta representación y generar el genotipo del individuo en base a la misma.

### Generador de individuos mixtos

Otro de las clases implementadas es la clase **MixedGenerator**, encargada de generar los genes de los individuos mixtos. En esta clase se ha establecido la lógica necesaria para generar los genes con los rangos recomendados para cada uno de los parámetros, en vez de hacer uso de un mismo rango para todos los genes de un individuo. Sin embargo, se sigue permitiendo crear individuos con un único rango establecido para los genes del mismo.

Para conseguir esto, se ha utilizado la clase **Generator** de la librería GeneticsJS, la cuál implementa métodos para generar valores enteros, flotantes, booleanos, probabilidades, normalizaciones y valores según distribuciones.

En primera instancia, se ha establecido un array que contiene los límites inferiores de los rangos de cada uno de los 29 parámetros y otro con los límites superiores. Con esto se pretende poder tener un control sobre los valores qué tienen que manejar los genes dependiendo de su índice.

Si se tiene en cuenta que cada gen del individuo se corresponde con cada uno de los 29 parámetros de entrada del simulador, el rango recomendado para generar un gen en un índice $X$ será cualquier valor que se mueva entre el valor contenido en el índice $X$ del array de límites inferiores y el contenido en el mismo índice del array de límites superiores.

Tras esto, en la clase **Generator** se ha creado un método estático que comprueba si los límites inferior y superior de un gen son ambos enteros o ambos flotantes, de manera que se genere un valor del mismo tipo y dentro del rango indicado.

### Cruces

Para los cruces hubo que hacer ciertos ajustes puesto que no se estaba controlando la probabilidad de cruce entre individuos, al menos en el cruce por un solo punto y en el cruce por n puntos, que son los cruces que se han utilizado principalmente durante todo el desarrollo por su simplicidad y velocidad.

En concreto, el cruce por $n$ puntos ó *"n points crossover"* consiste en elegir precisamente $n$ puntos de cruce entre dos padres, donde estos puntos de cruce son elegidos aleatoriamente. Los individuos que se obtengan del cruce, serán formados escogiendo de forma correlativa los genes de sus predecesores.

Por otro lado, se tendría el cruce por un solo punto o *"one point crossover"*, el cual es un caso concreto del crossover por $n$ puntos donde $n = 1$. Su funcionamiento al ser un caso concreto del cruce por n puntos, es idéntico.

### Mutaciones

Las mutaciones numéricas implementadas dentro de la librería que más encajan con la implementación realizada para el **MixedIndividual** serían la mutación uniforme y la mutación no uniforme (6) puesto que en la codificación de los genes se usan valores enteros o reales, por lo que se han decidido adaptar ambas para que puedan ser utilizadas por individuos mixtos. Además, se ha implementado también una mutación polinómica, la cual no existía en la librería hasta el momento.

El funcionamiento de las mutaciones implementadas es el siguiente:

- **Mutación uniforme**: Esta mutación consiste en seleccionar de manera uniforme un valor de entre un rango de valores y sustituir el gen mutado por este valor. Este tipo de mutación es el más sencillo, siendo análogo a la inversión de bits para las codificaciones binarias o al reajuste aleatorio para las codificaciones de enteros. Normalmente, esta mutación se usa con una probabilidad de mutación por gen., o lo que es lo mismo, $\sigma = 1/29$, siendo $29$ el número de genes de los individuos.
- **Mutación no uniforme**: Esta mutación calcula un valor a partir de una distribución normal de probabilidad, la cual se añade al gen para modificarlo. En concreto, en esta mutación se usa una desviación típica que suele ser llamada "step size" y que determina que genes son perturbados por el operador de mutación. Este valor también suele fijarse a una probabilidad por gen.
- **Mutación polinómica**: Esta mutación calcula un valor a partir de una distribución polinomial, para lo cual se tienen en cuenta varios aspectos como son la variabilidad de la perturbación, la probabilidad de mutación y el índice de la distribución, entre otros.

**Selección y reemplazo**

En lo que respecta a la selección de padres y de supervivientes, tendríamos los métodos de selección o de reemplazo de individuos, que en este caso no se han modificado, sin embargo, si que se ha acotado la cantidad de estrategias utilizadas en los algoritmos, por lo que se hará una breve introducción a las mismas.

El criterio seguido para decidir que estrategias de selección y de reemplazo se han de utilizar es bastante simple, puesto que se han seleccionado aquellas estrategias que giran en torno a la calidad de las soluciones.

Es por ello, que como **estrategia de selección** se ha elegido el método de la ruleta (6), también conocido como *"Roulette Wheel"*, el cual es un tipo de selección donde los individuos son seleccionados de manera proporcional a su fitness ó *"fitness proportional selection"*, de manera que la probabilidad de que un individuo sea seleccionado para tener descendencia depende de su valor de fitness absoluto comparado con el valor de fitness absoluto del resto de individuos (6).

De esta manera, se asegura de que los individuos mejor adaptados tengan una mayor probabilidad de ser seleccionados, pero dejando también la oportunidad a individuos peor adaptados.

La ventaja que aporta esta estrategia de selección, es que evita en cierta manera que las búsquedas se queden atascadas en un óptimo local gracias a que garantiza variedad en las poblaciones que se van generando.

Por otro lado, en lo que respecta a la **estrategia de reemplazo**, se ha elegido el método de reemplazo basado en el fitness ó *"Fitness Based Replacement"* (6), donde como su propio nombre indica se basa en el fitness de los individuos para quedarse con los mejores.

En concreto, la implementación de este método sigue una estrategia donde se ordenan los individuos de la anterior generación y los obtenidos tras el cruce, se juntan, y de entre estos se quedan en la nueva generación los $n$ mejores. Esta estrategia se conoce como la selección $\mu + \delta$, donde $\mu$ se corresponde con la población actual, $\delta$ se corresponde con la descendencia y donde los mejores $\mu$ individuos son los que pasan a la siguiente generación. En la implementación concreta realizada, el tamaño de $\mu$ es igual al tamaño de $\delta$, así que podemos ver la selección $\mu + \delta$ como una generalización de la estrategia "replace worst", donde se reemplazan los peores individuos de entre el conjunto de la población actual y la descendencia.

### Inicialización de la población

Debido a la necesidad de poder controlar los valores entre los cuales se mueven los genes de los individuos, es necesario manipular la manera en la que se generan los individuos, pero además también se requiere modificar la inicialización de la población al lanzar el algoritmo. Esto se debe a que el funcionamiento actual de la librería se basa en el rango establecido dentro de los parámetros del generador para inicializar la población al lanzar el algoritmo genético, no permitiendo además pasar una población ya inicializada.

Para solventar esto, se ha modificado la clase **EvolutionaryAlgorithm**, para que reciba en el constructor un parámetro opcional de tipo **Population**, de manera que ahora si se desea, se puede instanciar un algoritmo genético pasando una población ya inicializada.

### Condición de parada

Como condición de parada, se hace uso de la implementación de el número máximo de generaciones como condición para que finalice la ejecución del algoritmo. Concretamente para este caso, tampoco se han realizado modificaciones sobre la implementación dada en la librería.

# Capítulo 6

# Experimentos y resultados

En este apartado se mostrarán los resultados obtenidos tras realizar experimentos tanto con el algoritmo genético como con el algoritmo de búsqueda aleatoria, para los cuales se han definido una serie de configuraciones básicas. Cabe destacar que en las tablas de los resultados de las ejecuciones de los algoritmos se presentan valores con un máximo de tres decimales sin redondear.

Los experimentos se ejecutaron en una máquina con una distribución centOS 8 que incluye dos procesadores AMD® Epyc (modelo 7502) a 2.5 GHz y 128 GB de memoria RAM.

## 6.1. Algoritmo de búsqueda aleatoria

Para poner a prueba la búsqueda aleatoria se han planteado tres configuraciones para el lanzamiento de la misma, de manera que siempre se lancen 30 ejecuciones del algoritmo pero variando el número de iteraciones para poder ver si se obtienen mejores resultados con un mayor número de iteraciones.

Las configuraciones establecidas para el algoritmo de búsqueda aleatoria se presentan en la Tabla 6.1.

Tabla 6.1: Configuraciones de la búsqueda aleatoria

| Configuración | Número de lanzamientos | Número de iteraciones |
|---|---|---|
| conf1 | 30 | 100 |
| conf2 | 30 | 200 |
| conf3 | 30 | 300 |

En los resultados que se muestran en la Tabla 6.2, se puede observar como el mejor valor obtenido en la búsqueda aleatoria es encontrado por la configuración donde se realiza un mayor número de iteraciones, y además se puede ver como en promedio, los valores obtenidos mejoran con cada configuración a medida que se aumenta el número de iteraciones.

Tabla 6.2: Resultados (fitness) obtenidos por la búsqueda aleatoria

| Mejor solución encontrada: **36,005** | | | | | | |
|---|---|---|---|---|---|---|
| Configuración | Mínimo | Cuartil 1 | Mediana | Cuartil 3 | Máximo | Media |
| conf1 | 36,999 | 41,326 | 42,939 | 44,399 | 46,733 | 42,700 |
| conf2 | 37,106 | 39,743 | 41,099 | 42,033 | 44,03 | 40,997 |
| conf3 | **36,005** | **37,943** | **39,506** | **40,667** | **42,637** | **39,365** |

## 6.2. Algoritmo genético

Para llevar a cabo la experimentación con el algoritmo genético, se han definido cuatro configuraciones variando algunos parámetros del mismo, teniendo en cuenta un número máximo de generaciones o iteraciones como condición de parada.

En todas estas configuraciones, los individuos son seleccionados en proporción a su adaptación al medio, haciendo uso del método conocido como *"roulette wheel"* y para el reemplazo se hace uso de un reemplazo basado en el fitness conocido como *"replace worst"*. Por otro lado, para los cruces de los individuos, en todas las configuraciones se está haciendo uso del *crossover de un solo punto*, el cual realmente es un caso de uso concreto del *crossover de n puntos*. En el apartado 5 se profundiza acerca de estos métodos y su funcionamiento.

Además, las configuraciones planteadas para el algoritmo genético van a tener una serie de características en común, los cuales son:

- Tamaño de población: 25
- Número de réplicas de simulación: 10
- Número máximo de generaciones: 100

En concreto, se han utilizado estos valores debido a que cada lanzamiento con estos parámetros tarda, aproximadamente, unas 18 horas, por lo que tratar de lanzar experimentos más grandes ha sido inviable debido a las restricciones temporales con las que contamos.

Por otro lado, el resto de aspectos que se han tenido en cuenta para establecer las configuraciones del algoritmo genético serían la probabilidad de cruce y el tipo de mutación utilizada (Tabla 6.3), donde se ha decidido descartar la mutación no uniforme para reducir el número de configuraciones a utilizar.

Tabla 6.3: Configuraciones del algoritmo genético

| Configuración | Probabilidad de cruce | Mutación | Tamaño de la población | Número de réplicas | Número de Generaciones |
|---|---|---|---|---|---|
| conf1 | 0,8 | Uniforme | 25 | 10 | 100 |
| conf2 | 0,8 | Polinomial | 25 | 10 | 100 |
| conf3 | 1 | Uniforme | 25 | 10 | 100 |
| conf4 | 1 | Polinomial | 25 | 10 | 100 |

Como se puede observar en la Tabla 6.4, las configuraciones 2 y 4 son las que, aparentemente, son capaces de proporcionar mejores resultados, siendo estos bastante similares entre si. Cabe mencionar que dichas configuraciones hacen uso de la mutación polinomial, frente al uso de la mutación uniforme por parte de las configuraciones 1 y 3, las cuales proporcionan resultados peores. Al mismo tiempo, también es importante resaltar que pareciera que la probabilidad de cruce no es un factor tan determinante como el operador de mutación, dado que las configuraciones 2 y 4 proporcionan resultados similares, a pesar de utilizar probabilidades de cruce iguales a 0,8 y 1, respectivamente. En la Figura 6.1, se muestra, de manera gráfica, los mismos resultados descritos en la Tabla 6.4.

Finalmente, también cabe resaltar la mejoría de los resultados obtenidos con el algoritmo genético respecto a los obtenidos por la búsqueda aleatoria. La mejor solución proporcionada por la búsqueda aleatoria (Tabla 6.2) tiene un fitness igual a 36,005, mientras que la mejor solución proporcionada por el algoritmo genético tiene un fitness igual a 7,158.

Tabla 6.4: Resultados (fitness) obtenidos por el algoritmo genético

| 25x10x100 | | | Mejor solución encontrada: **7,158** | | | |
|---|---|---|---|---|---|---|
| Configuración | Mínimo | Cuartil 1 | Mediana | Cuartil 3 | Máximo | Media |
| conf1 | 8,361 | 12,219 | 13,220 | 14,708 | 18,136 | 13,189 |
| conf2 | 7,277 | 7,737 | 10,895 | **11,04** | 14,821 | 9,92 |
| conf3 | 8,102 | 9,276 | 12,128 | 13,304 | 16,788 | 11,828 |
| conf4 | **7,158** | **7,301** | **10,775** | 11,078 | **11,801** | **9,413** |

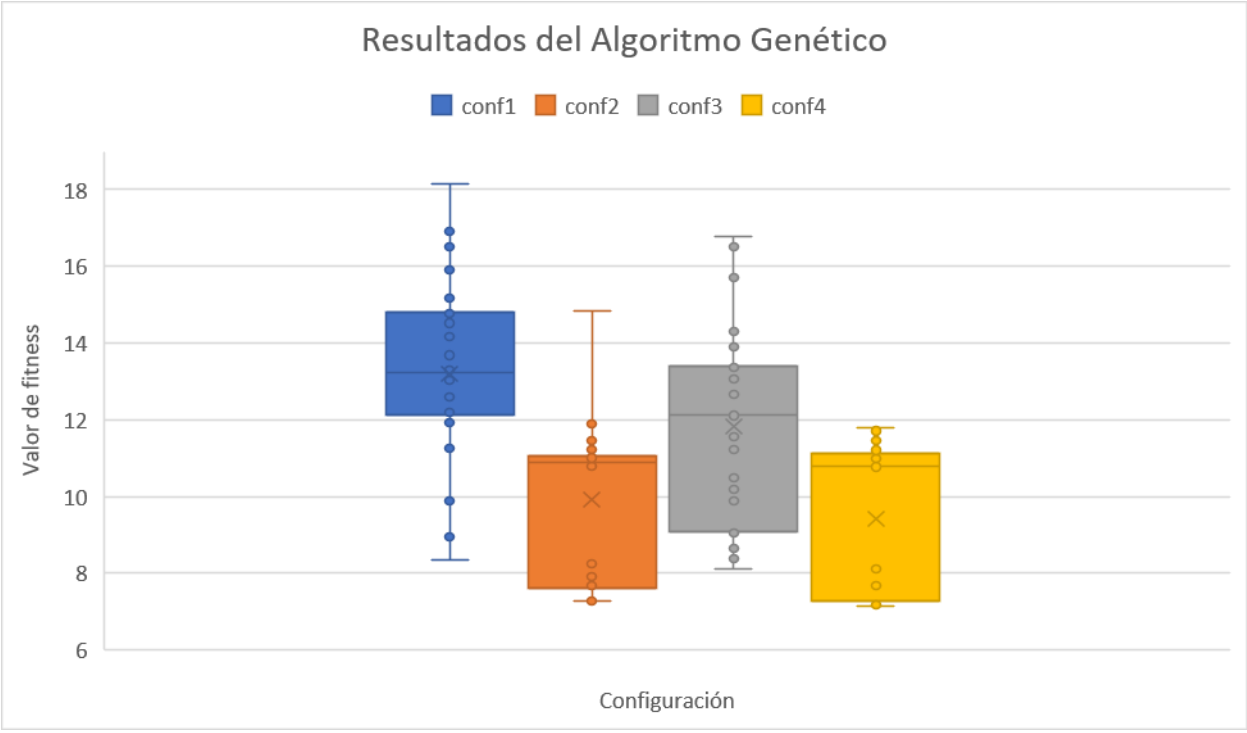



Figura 6.1: Boxplots asociados a los resultados del algoritmo genético

Con el objetivo de poder comparar si las diferencias entre los resultados obtenidos por las diferentes configuraciones son estadísticamente significativas se ha aplicado el siguiente procedimiento.

En primera instancia, se realiza un test *Shapiro-Wilk* para comprobar si los valores de los resultados siguen una distribución normal (gaussiana). En caso afirmativo, se comprueba la homogeneidad de las varianzas con el test de *Levene*. Si existe homogeneidad en las varianzas, se lleva a cabo un test anova. En caso contrario, se realiza un test de *Welch*. Para distribuciones no gaussianas, se lleva a cabo el test no paramétrico de *Kruskal-Wallis*. Para todas las pruebas, se considera un nivel de significación $\alpha = 0,05$.

Tabla 6.5: Resultados del procedimiento de comparación estadístico entre configuraciones del algoritmo genético

| | conf1 | conf2 | conf3 | conf4 |
|---|---|---|---|---|
| conf1 | - | **Mejor: conf2 (7,890086e-07)** | **Mejor: conf3 (0,03750262)** | **Mejor: conf4 (6,799497e-08)** |
| conf2 | | - | **Mejor: conf2 (0,001301259)** | Sin diferencias estadísticas significativas (0,2938453) |
| conf3 | | | - | **Mejor: conf4 (0,0001285504)** |
| conf4 | | | | - |

En la Tabla 6.5 se muestran los resultados del procedimiento de comparación estadístico entre configuraciones del algoritmo genético. En concreto, para cada par de configuraciones, se muestra, entre paréntesis, el p-valor obtenido siguiendo dicho procedimiento de comparación estadístico. En aquellos casos donde el p-valor obtenido es menor que el nivel de significación, esto es, donde existen diferencias estadísticas significativas, se indica qué configuración de las dos comparadas es mejor. Una configuración es mejor que otra cuando, en primer lugar, existen diferencias estadísticamente significativas entre ambas y, al mismo tiempo, dicha configuración proporciona una media y mediana de los resultados inferior a la media y mediana obtenida por la otra configuración.

Tal y como se puede observar, los resultados confirman, estadísticamente, las conclusiones extraídas con anterioridad. Las configuraciones 2 y 4 no presentan diferencias estadísticas significativas. Sin embargo, si comparamos ambas por separado con el resto de configuraciones sí que se encuentran diferencias estadísticas significativas, siendo estas dos configuraciones estadísticamente superiores a las configuraciones 1 y 3. Lo anterior confirma que las configuraciones que proporcionan mejores resultados son aquellas que hacen uso de la mutación polinomial y que variar la probabilidad de cruce no parece tener un impacto significativo sobre las soluciones obtenidas.

## 6.3. Análisis cualitativo de soluciones

En este apartado se va a realizar un análisis cualitativo de soluciones diferentes de entre las obtenidas en el proceso de experimentación. La idea es comparar dos soluciones cuyos valores de fitness obtenidos difieran bastante, de manera que consideremos una solución “buena” y otra “mala” durante el análisis.

En la discusión planteada en (16) acerca de los parámetros que dominan la calidad de las soluciones, se determina tras un análisis del dominio que los parámetros que más impacto tienen en el modelo son $x13$, $x14$ y $x26$. Por el contrario, los parámetros que están siempre entre los cinco con el impacto más bajo en el modelo son $x24$ y $x5$. Además, las conclusiones extraídas con respecto a el impacto que producen estos parámetros en el modelo tienen sentido, porque se indica que un mayor número de pacientes infectados que ingresen al hospital ($x14$), implica un mayor impacto para la gestión de los recursos. Esto aplicaría también con $x19$, $x2$ y $x5$.

En la Tabla 6.6 podemos ver los parámetros de las soluciones escogidas para la comparación, así como también sus correspondientes valores de fitness obtenidos, donde se puede observar que hay bastantes parámetros en los cuales coinciden sus valores o la diferencia es mínima para ambas soluciones, como ocurre en $x12$, $x19$, $x20$, $x21$, $x24$, $x27$. Esto es un posible indicador de que dichos parámetros no estén generando un impacto notable en la calidad de las soluciones.

Por otro lado, el resto de parámetros tiene diferencias notables, pero los que más destacan son aquellos que están en los límites opuestos de los rangos recomendados o cerca de estos, ya que podrían indicar “dominancia” en lo que a la calidad de la solución respecta. Los parámetros que cumplen esto último son $x3$, $x5$, $x6$, $x7$, $x9$, $x11$, $x13$, $x14$, $x16$, $x17$, $x23$, $x25$.

Tabla 6.6: Tabla comparativa de soluciones

| Parámetros | Solución buena Fitness: 7.158 | Solución mala Fitness: 143.324 |
|---|---|---|
| $x1$: AmntDaysInfectedToHospital | 6 | 10 |
| $x2$: AmntDaysNormalToHealthy | 7 | 11 |
| $x3$: AmntDaysNormalToIntensive | 7 | 3 |
| $x4$: AmntDaysNormalToVentilation | 3 | 5 |
| $x5$: AmntDaysNormalToDeath | 3 | 6 |
| $x6$: AmntDaysIntensiveToAftercare | 5 | 8 |
| $x7$: AmntDaysIntensiveToVentilation | 5 | 3 |
| $x8$: AmntDaysIntensiveToDeath | 7 | 5 |
| $x9$: AmntDaysVentilationToIntensiveAfter | 25 | 35 |
| $x10$: AmntDaysVentilationToDeath | 25 | 22 |
| $x11$: AmntDaysIntensiveAfterToAftercare | 2 | 5 |
| $x12$: AmntDaysIntensiveAfterToDeath | 7 | 7 |
| $x13$: GammaShapeParameter | 0,25 | 1,764 |
| $x14$: FactorPatientsInfectedToHospital | 0,05 | 0,138 |
| $x15$: FactorPatientsHospitalToIntensive | 0.07 | 0,084 |
| $x16$: FactorPatientsHospitalToVentilation | 0,005 | 0,019 |
| $x17$: FactorPatientsNormalToIntensive | 0,07 | 0,124 |
| $x18$: FactorPatientsNormalToVentilation | 0,0001 | 0,001 |
| $x19$: FactorPatientsNormalToDeath | 0,08 | 0,083 |
| $x20$: FactorPatientsIntensiveToVentilation | 0,25 | 0,285 |
| $x21$: FactorPatientsIntensiveToDeath | 0,08 | 0,0873 |
| $x22$: FactorPatientsVentilationToIntensiveAfter | 0,5 | 0,635 |
| $x23$: FactorPatientsIntensiveAfterToDeath | 0,000001 | 0,008 |
| $x24$: AmntDaysAftercareToHealthy | 2 | 2 |
| $x25$: RiskFactorA | 0,000001 | 1,053 |
| $x26$: RiskFactorB | 0,000001 | 0,004 |
| $x27$: RiskMale | 2 | 2 |
| $x28$: AmntDaysIntensiveAfterToHealthy | 5 | 4 |
| $x29$: FactorPatientsIntensiveAfterToHealthy | 0,5 | 0,654 |

Por último, si se comparan las observaciones obtenidas con lo indicado en (16), se puede observar como coinciden en que uno de los parámetros que parece tener menos impacto en el modelo es $x25$ y que por el contrario, los que más impacto parecen tener serían $x13$ y $x14$.

# Capítulo 7

# Conclusiones y líneas futuras

Tras haber concluido con el desarrollo del proyecto, es necesario determinar si se ha conseguido o no cumplir con los objetivos planteados, cómo se han llevado a cabo, la calidad de los resultados obtenidos y determinar también las posibles líneas futuras de desarrollo.

En primer lugar, se tenía como objetivo llevar a cabo una investigación y estado del arte del problema, por lo que se realizó una búsqueda bibliográfica siguiendo la metodología PRISMA. Asimismo, se realizó una investigación sobre la computación evolutiva para ver si un enfoque haciendo usa de la misma podría llegar a aportar resultados significativos. Los resultados obtenidos en el estado del arte no fueron los deseados debido a que es un problema relativamente reciente y no se dispone apenas de recursos sobre el mismo más allá de los publicados por los encargados de mantener el software asociado al problema.

Tras la investigación y el estudio del estado del arte, la definición de las tecnologías era el siguiente objetivo planteado, por lo que se decidieron en base a las facilidades que aportarían en tiempo de desarrollo y la adaptación de las mismas con respecto al problema, escogiendo por lo tanto el lenguaje del que hace uso el simulador y también el de la librería de computación evolutiva utilizada. El grado de satisfacción con respecto de las tecnologías elegidas es alto, puesto que han permitido cumplir con los objetivos estipulados sin añadir complejidad al desarrollo.

Otros de los objetivos planteados era el desarrollo de un algoritmo de búsqueda aleatoria y un algoritmo genético que permitieran resolver el problema y tratar de encontrar soluciones lo más cercanas a la óptima posibles. Tal y como se ha podido observar en los resultados obtenidos, se ha conseguido cumplir con este objetivo debido a que la implementación de ambos algoritmos ha sido realizada con éxito, los resultados obtenidos haciendo uso de los mismos son, con diferencia, mejores que los obtenidos al realizar evaluaciones simples. Sin embargo, no se tienen referencias de soluciones consideradas como óptimas para poder establecer si las obtenidas son óptimas o no.

Por último, se había planteado llevar a cabo un análisis de los algoritmos implementados a través de la evaluación experimental, de manera que se comprobara el rendimiento de las implementaciones realizadas. Este objetivo ha sido completado satisfactoriamente tras haber analizado las diferentes configuraciones planteadas para los algoritmos implementados, donde se comprobó que el rendimiento ofrecido por el algoritmo genético era estadísticamente superior que el proporcionado por la búsqueda aleatoria, donde las diferencias entre un algoritmo de búsqueda informado y otro no informado quedaron plasmadas.

Por lo tanto, los objetivos planteados se han conseguido alcanzar con éxito. Sin embargo, se plantean las siguientes mejoras con respecto al proyecto:

- **Mejora de la documentación:** Debido al limitado tiempo de desarrollo, la documentación del proyecto no ha podido ser completada tal y como se hubiera deseado, por lo que sería deseable tener una documentación clara y completa tanto para la librería como también para las implementaciones realizadas.
- **Inclusión de pruebas:** Debido al limitado tiempo de desarrollo, las pruebas incluidas fueron en su mayoría para las adaptaciones realizadas en la librería, sin embargo, se deberían de plantear pruebas también para las implementaciones realizadas.
- **Paralelización de las simulaciones**: Uno de los problemas principales durante el desarrollo de este trabajo fueron los tiempos de espera de las ejecuciones, por lo que la paralelización de las simulaciones podría reducir notablemente los tiempos del simulador, aunque la cantidad de procesos en paralelo que se puedan lanzar dependería en gran medida de las características de la máquina donde se vayan a realizar las pruebas.

Por último, se plantean distintas líneas de trabajo que permitan mejorar los resultados obtenidos:

- **Realización de Análisis de Componentes Principales (ACP)**: Con el objetivo de estudiar en profundidad el impacto de cada parámetro en el simulador sería interesante llevar a cabo un ACP sobre los mismos, ya que esto permitiría averiguar la sensibilidad que tienen con respecto a las simulaciones y su influencia en la calidad de las soluciones.
- **Construcción de un modelo surrogado**: Debido a la dimensionalidad del problema, el elevado coste computacional de las simulaciones y los tiempos de ejecución de las mismas, es muy complicado llegar a obtener soluciones óptimas en tiempos asequibles. La construcción de un modelo surrogado permitiría reducir el número de evaluaciones realizadas pero obteniendo resultados de la misma calidad o mejor.

# Capítulo 8

## Summary and Conclusions

After having concluded the development of the project, it is necessary to determine whether the objetives have been achieved or not, how they have been carried out, the quality of the results obtained and also to determine possible future lines of development.

Firstly, the objective was to carry out an investigation and state of the art of the problem, so a literature search was carried out following the PRISMA methodology. Likewise, a research on evolutionary computation was carried out to see if an approach using it could provide significant results. The results obtained in the state of the art were not the desired ones because it is a relatively recent problem and there are hardly any resources on it beyond those published by the maintainers of the software associated with the problem.

After the research and the study of the state of the art, the definition of the technologies was the next objetive, so they were decided on the basis of the facilities they would provide in development time and their adaptation to the problem, thus choosing the language used by the simulator and also the evolutionary computing library used. The degree of satisfaction with respect to the chosen technologies is high, since they have allowed to fulfill the stipulated objetives without adding complexity to the development.

Another objective was to develop a random search algorithm and a genetic algorithm to solve the problem and try to find solutions as close to the optimum as possible. As can be seen in the results obtained, this objective has been achieved because the implementation of both algorithms has been carried out successfully, the results obtained by using them are, by far, better than those obtained by performing simple evaluations. However, there are no references of solutions considered as optimal to be able to establish whether those obtained are optimal or not.

Finally, it had been planned to carry out an analysis of the implemented algorithms through experimental evaluation, in order to verify the performance of the implementations carried out. This objective has been successfully completed after having analyzed the different configurations proposed for the implemented algorithms, where it was found that the performance offered by the genetic algorithm was statistically superior than that provided by the random search, where the differences between an informed and an uninformed search algorithm were clearly observed.

Therefore, the stated objectives have been successfully achieved. However, the following improvements are raised with respect to the project:

- **Improved documentation:** Due to the limited development time, the project documentation could not be completed as desired, so it would be desirable to have a clear and complete documentation for both the library and the implementations.
- **Inclusion of tests:** Due to the limited development time, the tests included were mostly for the adaptations made to the library, however, tests should also be considered for the implementations made.
- **Parallelization of the simulations**: One of the main problems during the development of this work was the waiting times of the executions, so the parallelization of the simulations could significantly reduce the simulator times, although the number of parallel processes that could be launched would depend largely on the characteristics of the machine where the tests are going to be carried out.

Finally, different lines of work are proposed to improve the results obtained:

- **Principal Component Analysis (PCA)**: In order to study in depth the impact of each parameter in the simulator it would be interesting to carry out a PCA on them, as this would allow to find out the sensitivity they have with respect to the simulations and their influence on the quality of the solutions.
- **Construction of a surrogated model**: Due to the dimensionality of the problem, the high computational cost of the simulations and their execution times, it is very difficult to obtain optimal solutions in affordable times. The construction of a surrogate model would allow to reduce the number of evaluations performed but obtaining results of the same or better quality.

# Capítulo 9

## Presupuesto

Se propone el siguiente presupuesto teniendo únicamente en cuenta un desglose de los costes de recursos humanos. Esto se debe a que no ha habido costes tecnológicos durante este desarrollo.

| Tipos | Duración | Coste |
|---|---|---|
| Estudio del estado del arte | Total de 80 horas | 25€/h |
| Trabajo de desarrollo | Total de 170 horas | 30€/h |
| Experimentación y análisis | Total de 64 horas | 25€/h |

Tabla 9.1: Resumen de tipos

El coste total del proyecto es de 8.700€.

# Bibliografía


[1] “GECCO 2021 Industrial Challenge: Call for Participation.” [Online]. Available: https://www.th-koeln.de/informatik-und-ingenieurwissenschaften/gecco-2021-industrial-challenge-call-for-participation_82086.php

[2] “Caja negra (sistemas),” Aug. 2021, page Version ID: 137656214. [Online]. Available: https://es.wikipedia.org/w/index.php?title=Caja_negra_(sistemas)&oldid=137656214

[3] “GECCO 2021 | Competitions.” [Online]. Available: http://gecco-2021.sigevo.org/Competitions

[4] “GECCO 2021 | HomePage.” [Online]. Available: http://gecco-2021.sigevo.org/HomePage

[5] “bartzbeielstein / babsim.hospital.” [Online]. Available: http://owos.gm.fh-koeln.de:8055/bartz/babsim.hospital

[6] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*, 2nd ed. Springer Publishing Company, Incorporated, 2015.

[7] C. Darwin, *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life.* London: John Murray, 1859.

[8] “GeneticsJS/GeneticsJS,” Jan. 2022, original-date: 2020-12-18T13:00:40Z. [Online]. Available: https://github.com/GeneticsJS/GeneticsJS

[9] “JavaScript With Syntax For Types.” [Online]. Available: https://www.typescriptlang.org/

[10] “Visual Studio Code - Code Editing. Redefined.” [Online]. Available: https://code.visualstudio.com/

[11] Node.js, “Node.js.” [Online]. Available: https://nodejs.org/es/

[12] L. Henry, H. Wickham, m. H. i. b. o. M. xxhashlite), Y. C. A. o. t. e. x. library), and RStudio, “rlang: Functions for Base Types and Core R and ’Tidyverse’ Features,” Jun. 2022. [Online]. Available: https://CRAN.R-project.org/package=rlang

[13] A. Liberati, D. G. Altman, J. Tetzlaff, C. Mulrow, P. C. Gøtzsche, J. P. A. Ioannidis, M. Clarke, P. J. Devereaux, J. Kleijnen, and D. Moher, “The PRISMA statement for reporting systematic reviews and meta-analyses of studies that evaluate healthcare interventions: explanation and elaboration,” *BMJ*, vol. 339, p. b2700, Jul. 2009, publisher: British Medical Journal Publishing Group Section: Research Methods & Reporting. [Online]. Available: https://www.bmj.com/content/339/bmj.b2700

[14] T. Bartz-Beielstein and M. Zaefferer, “Model-based methods for continuous and discrete global optimization,” *Applied Soft Computing*, vol. 55, pp. 154–167, Jun. 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494617300546

[15] T. Bartz-Beielstein, M. Zaefferer, F. Rehbach, M. Rebolledo, J. Stork (0000-0002-7471-3498), and C. Lasarczyk, “SPOT: Sequential Parameter Optimization Toolbox,” Jun. 2022. [Online]. Available: https://CRAN.R-project.org/package=SPOT

[16] T. Bartz-Beielstein, M. Dröscher, A. Gür, A. Hinterleitner, O. Mersmann, D. Peeva, L. Reese, N. Rehbach, F. Rehbach, A. Sen, A. Subbotin, and M. Zaefferer, “Resource Planning for Hospitals Under Special Consideration of the COVID-19 Pandemic: Optimization and Sensitivity Analysis,” May 2021, arXiv:2105.07420 [cs, math]. [Online]. Available: http://arxiv.org/abs/2105.07420

[17] T. Bartz-Beielstein, F. Rehbach, E. Bartz, O. Mersmann, and M. Zaefferer, “babsim.hospital: Bartz & Bartz Simulation Hospital,” May 2022. [Online]. Available: https://CRAN.R-project.org/package=babsim.hospital

[18] J. Kacprzyk and W. Pedrycz, *Springer handbook of computational intelligence*. Springer, 2015.

[19] “yargs.” [Online]. Available: https://www.npmjs.com/package/yargs