UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Diseño del Patrón de Carga del Núcleo de Reactores PWR Asistido por Inteligencia Artificial

Trabajo Fin de Grado

Grado en Ciencia de Datos

AUTOR/A: Jambrina Fernández, Nicolás

Tutor/a: García Díaz, Juan Carlos

CURSO ACADÉMICO: 2021/2022

Resumen

La energía es un bien cada vez mas preciado en la sociedad actual, donde observamos un aumento constante en sus precios y en deterioro de la naturaleza cada día mayor. Este trabajo es una propuesta para mejorar la obtención de energía a través de la organización del patrón de carga de un reactor nuclear pretendiendo optimizar al máximo la energía generada. Esto se consigue a través de la maximización de la k-efectiva, variable que nos muestra la cantidad de neutrones medios de una fisión que producen otra fisión. Por motivos de seguridad este parámetro se suele ajustar a uno. Esta es una de las diferentes reglas de seguridad que un reactor ha de seguir para que su funcionamiento sea optimo.

En la literatura actual se han propuesto diferentes estrategias para lograr este objetivo. Nosotros hemos aglomerado varias de estas metodologías. En primer lugar, se pretende en primer lugar aplicar las redes neuronales con el objetivo de agilizar el cálculo de las constantes nucleares y una vez estas redes han sido generadas, diseñar y optimizar un algoritmo genético el cual nos permita optimizar el patrón de carga. De esta manera, con un patrón de carga dado a través de cruces y mutaciones de las matrices se ha conseguido obtener la maximización de la k-efectiva.

Palabras clave: Patrón de carga, k-efectiva, algoritmo genético, redes neuronales, potencia radial, potencia axial, potencia tridimensional.

Abstract

Nowadays energy is a precious good and the tendency is growing, each day the price is higher, and we can see the trace that the consume of energy makes in our world. This project is a proposal of improvement, we pretend to make an improvement developing a method that allows us to optimize the energy generated by a nuclear reactor. In order to achieve this purpose, the idea is to maximize the k-effective of the reactor distributing appropriately the load pattern. The k-effective is one of the most important of our variables, because it gives a coefficient of neutrons of a fission that generates another fission, generating a chain of effects. As a security measure, this value is set to one. This among other security rules are rules that a reactor must follow so the process is optimal and safe.

In the actual world there are a lot of strategies to achieve this objective. Our proposal is an agglomeration of this methodologies. First of all, our idea is to apply neuronal network in order to calculate the different nuclear constants. With this information, we are going to plan and optimize a genetic algorithm which optimize the load pattern. This way, the resulting loading pattern will be the one with the better k-effective.

Keywords: Loading pattern, k-effective, genetic algorithm, neural networks, radial potence, axial potence, tridimensional potence.

Tabla de contenidos

Contenido

Índice de Figuras

Índice de Táblas

1. Introducción

1.1 Motivación

Desde los años 40 del siglo XX la energía nuclear se convirtió en una de las principales fuentes de energía del mundo. Según la *World Nuclear Association* [1] el 10% de la energía generada en el mundo proviene de la energía nuclear, generada gracias a los 440 reactores nucleares situados en torno al mundo. La energía proveniente de estas centrales se origina en la fisión nuclear. La fisión nuclear es el proceso resultante de impactar un neutrón contra un átomo pesado, esto provoca que el átomo se divida en átomos más ligeros emitiendo neutrones, que a su vez impactan en otros más pesados, provocando así una reacción en cadena la cual genera una gran cantidad de energía, siendo un millón de veces mayor la energía desprendida a la de quemar un bloque de carbón con la misma masa.

Esto convierte a este sistema en un gran generador de energía, pero también tiene un gran inconveniente y es que necesita de un control y de una seguridad superior a la de otros métodos de obtención de energía pues los fallos en una central pueden originar catástrofes como nos enseña la historia. Y en esto es en lo que la ciencia de datos puede ayudar.

Nos encontramos ante la tarea de automatizar el sistema de gestión de carga de un reactor nuclear con el objetivo de maximizar la energía obtenida de él. Además, la entrada de sistemas de control informáticos puede devolver la visión de seguridad a este tipo de energía y con ello nuevas inversiones.

Lo que llamó mi atención de este trabajo fin de grado es su actualidad, pues en un mundo donde el cambio climático es cada vez mas palpable y situaciones actuales como la guerra ruso-ucraniana y el aumento de costes de la energía a raíz, es cada vez más importante optimizar al máximos nuestros medios de obtención de la misma. Es por esto y por la complejidad del tema por lo que decidí abordar esta temática como proyecto de fin de grado.

1.2 Objetivos

El objetivo principal de este trabajo como se ha mencionado anteriormente es la maximización de la energía obtenida de un reactor nuclear a lo largo de uno de sus ciclos de funcionamiento. Para el cumplimiento de esta propuesta se proponen una serie de objetivos específicos:

- Generación de datos. Generar la cantidad necesaria de patrones de carga para poder realizar un estudio preciso
- Implementación de las redes feedforward y convolucionales para el cálculo de las variables necesarias para la optimización del reactor nuclear.
- Implementación de un algoritmo genético que nos permita optimizar el valor energético.
- Evaluación y selección del mejor resultado generado por el algoritmo genético.

Además de esta serie de objetivos primarios también se considera importante establecer un conjunto de objetivos personales para el desarrollo del trabajo de fin de grado:

- Aumento de los conocimientos de Linux para el trabajo en servidores.
- Mejora y ampliación de los conocimientos adquiridos durante el grado sobre redes neuronales y algoritmos genéticos
- Adquisición de conocimiento relacionado con el mundo de la energía nuclear.

1.3 Impacto esperado

Este TFG pretende ser un trabajo de tremenda actualidad. Es evidente que hoy en día hay una necesidad de producción de grandes cantidades de energía en todos los países. Esto que ya era evidente a raíz del cambio climático y como teníamos que ajustar nuestras energías para ser más productivas y menos lesivas con el medio ambiente. Pero en el mundo actual nos hemos dado cuenta, además, de la dependencia energética que tenemos hacia países como Rusia y su petróleo, de una manera semejante que en los años 70 con la crisis del petróleo. Estos hechos son pruebas de que debemos optimizar nuestros métodos de obtención de energía para que seamos menos dañinos al medio ambiente y a la vez podamos obtener el máximo posible de energía de los medios que disponemos.

Pero más allá de esta gran tarea de impacto mundial, este trabajo pretende tener un impacto personal. En primer lugar, se espera una mejora a la hora de trabajar en solitario en trabajos de esta magnitud, una mejora a la hora de planificar el trabajo y alcanzar sus objetivos, pues a lo largo de esta carrera yo y mis compañeros nos hemos enfrentado a trabajos de esta magnitud como grupo, pero nunca a un trabajo de esta magnitud de una manera individual. Por esto, se cree que este trabajo tendrá un impacto en mi organización a la hora de trabajar y en el desarrollo de los conocimientos adquiridos en la universidad.

También cabe destacar que el lenguaje de programación escogido para este proyecto es Python. Esta elección se debe a que se ha considerado que esta decisión aumenta el impacto de este trabajo. En primer lugar, por la actualidad de este lenguaje que recibe mejoras constantes en sus librerías y es un leguaje utilizado por empresas de primer orden. En segundo lugar, por la disponibilidad de librerías como tensorflow o pytorch, de las que dispone para el desarrollo e implementación de redes neuronales y finalmente, por su escalabilidad, ya que Python puede ser fácilmente integrado junto con pySpark para el tratamiento de grandes cantidades de datos, lo que podría llegar a ser necesario para este trabajo.

1.4 Metodología

La metodología llevada a cabo en este proyecto, en primer logar ha sido la creación de un organigrama en Trello, con el objetivo de poder planificar las diferentes tareas que irán surgiendo a lo largo del trabajo de fin de grado. El organigrama se divide en una serie de grupos de trabajo en primer lugar Tareas a programar y Tareas ya programadas, en estas dos columnas se situarán primero en Tareas a programar, aquellas tareas que necesiten ser programadas en Python y una vez esta este terminada pasara a las Tareas ya programadas. El segundo grupo de tareas serán Apartados a redactar y Apartados redactados. En este grupo se encontrarán aquellos apartados de la memoria que faltan por redactar y a medida que vayan siendo completados irán pasando a la pila de Apartados redactados.

En cuanto a la metodología empleada en el ámbito global del TFG, será explicada en los apartados 3 y posteriores donde detalladamente se explicarán los diferentes pasos seguidos, y de las posibles guías que se podrían haber seguido en su lugar. También se hablará de los conocimientos necesarios para su desarrollo. Pero pasaremos a hacer un resumen estructurado de los pasos seguidos:

- Generación de los datos usando Python y PARCS.
- Extracción de los datos
- Creación y evaluación de las redes neuronales
- Creación y aplicación de un algoritmo genético

Durante todo el TFG se ha seguido una mentalidad y una metodología puramente científica, donde se ha intentado demostrar todo con datos y respaldarlo con evidencias.

1.5 Estructura

En los próximos capítulos se verán los siguientes temas. En primer lugar, el estado del arte donde se expondrán los diferentes puntos de vista relacionados con esta temática y finalmente se expondrá la propuesta realizada para el trabajo de fin de grado. En segundo lugar, encontramos la explicación detallada del problema. En este apartado se explicará de forma técnica cual es el problema que se nos presenta y cómo se va a intentar afrontar el mismo de una manera minuciosa. Seguimos con otro apartado donde se hablará de los datos, cómo son, cómo se han generado, restricciones que tienen, todo esto mostrando ejemplos de los mismos y posteriormente se muestra un pequeño análisis de los mismos.

El último de los apartados detallara de una forma minuciosas los métodos escogidos para el TFG. Primero, se explicarán las redes neuronales y los métodos utilizados para medir su error, para optimizarlas, sus funciones de activación… una vez echo esto se pasará a explicar con minucioso detalle los algoritmos genéticos y todas los apartados que forman al mismo. Con el algoritmo genético también se explicará su aplicación y evaluación. Tras esto solo quedarían las conclusiones del trabajo, donde también se explicarán futuras ampliaciones y el apéndice donde se explicará que impacto tiene este TFG con relación a los objetivos de desarrollo sostenible.

2 Estado del arte

La optimización del patrón de carga de un reactor nuclear es un tema de enorme actualidad debido al beneficio energético que produciría la optimización de los reactores nucleares. Uno de los *papers* de relevancia que muestran la aplicación de las redes neuronales en este campo es el *artículo Application of deep neural networks for high-dimensional large BWR core neutronics* de *Rabie Abu Saleem*[2]. En este artículo se nos presenta una manera de aplicar las redes neuronales en los cálculos de los valores de potencia del reactor nuclear, intentando optimizar el tiempo de calculo necesario en PARCS. En este trabajo, los autores proponen la realización de tres redes neuronales: una para calcular los aspectos de potencia, otra calcular la k-efectiva y finalmente otra red neuronal encargada de la estimación del ciclo en el que se encuentra el reactor. Los inputs corresponden a vectores de una longitud de 196 casillas, cada cual corresponde a una de las posiciones del reactor y contiene al combustible situado en dicha casilla. Este *paper* nos demuestra la posibilidad que hay de aplicar modelos de redes neuronales profundas en los problemas de alta dimensionalidad presentes en el mundo de los reactores nucleares. También deja el trabajo abierto a posibles optimizaciones de las redes neuronales propuestas utilizando métodos como los algoritmos genéticos. Este trabajo en concreto ha sido una fuerte inspiración a la hora de desarrollar este TFG.

Otro de los artículos de referencia en el desarrollo de este TFG es el *Novel genetic algorithm for loading pattern optimization based on core physics heuristics*[3]. En este trabajo se intenta optimizar el patrón de carga del reactor teniendo como parámetro a maximizar la k-efectiva del mismo. En este trabajo se proponen como método de representación de los cromosomas, un vector en el cual según el número del combustible, iremos introduciendo su posición de tal manera que estaría ordenado primero las posiciones del combustible uno, luego las del dos, las del tres… se explican métodos de mutación y de cruce previamente utilizados en trabajos relacionados con la materia, como podría por ejemplo ser el uso de cruces geométricos a la hora de realizar el cruce, usando una superficie rectangular a intercambiar entre los dos padres. Los resultados de este *paper* nos demuestran que una aproximación geométrica del problema nos trae beneficios como una identificación mas fácil de duplicados a la hora de hacer los cruces y en general da mejores resultados que una aproximación no geométrica del problema. También se remarca en los resultados de este trabajo que la varianza en los datos disminuye la probabilidad de converger en un mínimo lo cal en vez de en el mínimo global. Por otro lado, también se llega a la conclusión que cuando esta varianza decae, la población se homogeneiza y el tamaño de esta pierde el sentido. Otra de las conclusiones de este artículo es que el mantenimiento de la varianza en la población a través de los elementos de mutación mejora la calidad de los resultados, ya que una mayor varianza en la población nos permite encontrar unos mejores patrones de carga, pues el área explorable es mayor.

Otro de los artículos sobre la temática es el Nuclear fuel loading pattern optimisation using a neural network[4]. El objetivo de este artículo es entrenar una red neuronal con los resultados de ejecución de una serie de patrones de carga, entrenar una red neuronal de tal manera que esta genere nuevas configuraciones, que posteriormente serán evaluadas. El patrón utilizado en este proceso de optimización es la minimización del pico de potencia máximo, pues como explican esto es un factor de vital importancia en la seguridad del reactor. Debido a la simetría que guarda el reactor solo se utiliza un porcentaje de la matriz de entrada ya que el resto de esta es simétrica. Se recalca a lo largo del *paper* también el uso de las redes neuronales en los patrones de carga como un modo de agilizar el cálculo realizado por códigos nucleares como PARCS de los parámetros del reactor. Una de las conclusiones a las que se llega en este *paper* es que, a pesar de conseguir un mínimo, no se puede asegurar con este método que ese mínimo sea el óptimo global.

Otro de los trabajos de referencia es *Using a multi-state recurrent neural network to optimize loading patterns in BWRs*[5]. Se vuelve a hablar de la complejidad de tratar los problemas de los patrones de carga en particular de los reactores de tipo BWR debido a la complejidad de los mismos, ya que poseen un gran número posible de combinaciones. Como en el anterior *paper* se habla de la importancia de buscar la maximización de la k-efectiva y la minimización factor de pico de potencia. Para este proyecto se ha utilizado como red neuronal un modelo MSRNN, que consiste en una generalización de la red neuronal binaria de Hopfield, la cual contiene N x N neuronas las cuales están completamente interconectadas. Según explican, este tipo de red neuronal es una red recurrente ya que los datos de salida sirven como *feedback* al *input*, generando un proceso que se repite hasta satisfacer el criterio establecido de parada. Llamamos estado de la neurona a su salida y el criterio de parada es dado por una función de energía la cual decrece su valor al cambiar de estado las neuronas. Este tipo de redes neuronales se ha propuesto en más trabajos de la temática. La red neuronal se estructura de tal manera que para neurona le asignamos una posición del reactor. En este *paper* también se describe el algoritmo de optimización diseñado el cual sigue los siguientes pasos: se genera aleatoriamente un estado inicial valido para el MSRNN, se elige una neurona y su simétrica en caso de existir. Para el resto de las neuronas se modifican en caso de ser posible, se obtiene un nuevo estado y el valor energético es actualizado, se repiten los paso del 2 al 4 hasta alcanzar la convergencia. Una de las diferencias respecto con los modelos de redes neuronales ya definidos y este en particular es que este tipo de redes neuronales no necesita prefijar valores con valores aleatorios. También se proponen funciones de energía diferentes con el objetivo de aumentar los grados de libertad de la red neuronal. Como aspecto a mejorar usar más parámetros del reactor como objetivos de la optimización además de los mencionados.

También cabe destacar el trabajo realizado en le *paper Optimisation of used nuclear fuel canister loafing using a neural network and genetic algorithm*[6]. En este paper se proponen utilizar tanto redes neuronales como algoritmos genéticos en el proceso de optimización del reactor. En primer lugar, se usan las redes neuronales como un método de aceleración del cálculo de la k-efectiva. Para la red neuronal se utilizan como entrada las diferentes posiciones del reactor, como ya hemos visto en la mayoría de los *papers* y como nosotros haremos mas adelante. Con el objetivo de reducir la dimensionalidad del problema se propone la aplicación de un PCA. Por otro lado, el algoritmo genético se utiliza con el objetivo de cumplir cuatro tareas. La k-efectiva y el calor del reactor tiene que estar dentro de los estándares de seguridad y ambos tienen que tener una distribución lo más posiblemente homogénea.

2.1 Propuesta

Con todos estos trabajos expuestos pasamos a analizar en profundidad el trabajo realizado en este TFG. El objetivo es conseguir un patrón de carga donde hallemos una k-efectiva lo máxima posible, ya que una maximización de esta variable significa una maximización de la cantidad de energía producida por el reactor. Para ello, en primer lugar se generarán 25000 reactores diferentes siguiendo unas normas de seguridad y se ejecutaran usando el código nuclear PARCS.

Seguidamente, se aplicará lo propuesto en los diferentes *papers* y entrenaremos una serie de redes neuronales con el fin de optimizar el cálculo de las variables del reactor con respecto al tiempo que le cuesta calcularlas al código nuclear PARCS. Se analizará en primer lugar las redes feedforward propuestas en el trabajo de *Saalem* analizando la estructura de estas expuesta en la tabla uno del mismo *paper*, se intentara una mejora de estos valores a través de una prueba ajustando los valores propuestos, posteriormente pasaremos a intentar una mejora de estos valores aplicando redes convolucionales y optimizando los parámetros de las mismas redes en busca de una mejora de ajuste a la hora de calcular la k-efectiva y las diferentes potencias extraídas de los archivos de PARCS.

Tras obtener los valores claves del reactor, se propone el uso de un algoritmo genético teniendo en cuenta los valores geométricos de la matriz como se propone en el *paper* mencionado en el estado del arte *Novel genetic algorithm for loading pattern optimization based on core physics heuristics*, para optimizar lo máximo posible el patrón de carga del reactor con dicho algoritmo genético, obteniendo una mayor k-efectiva posible, dentro de los parámetros estipulados como seguros. Con esta idea en mente pasamos a explicar el trabajo realizado.

3 Análisis del Problema

Hoy en día las centrales nucleares utilizan principalmente dos tipos de reactores nucleares, el reactor PWR y los reactores BWR. El objetivo de estudio de este trabajo es un reactor PWR. Un reactor PWR es un reactor nuclear con una estructura de 17 x 17 vainas, donde cada en cada una de esas vainas contiene un tipo de combustible. Los combustibles se suelen distribuir por la cantidad de uranio enriquecido que tienen y suelen estar distribuido en tres grupos cada cual representa un tercio del reactor. Esto se debe a que el uranio cuanto mayor tiempo pasa en el reactor nuclear, menor es la concentración de uranio enriquecido que posee, ya que el combustible (enriquecimiento) ha ido consumiéndose a través del proceso. Los combustibles pues se distribuyen entre las nuevas vainas de uranio, aquellas que ya llevan medio ciclo de uso y las que ya han completado el ciclo entero y por consiguiente las debemos cambiar.

El objetivo de este TFG es desarrollar un sistema el cual dado un reactor encuentre la distribución óptima de este mismo teniendo en cuenta los valores de la k-efectiva, ya que nuestro objetivo con el reactor es obtener la mayor energía posible. Para conseguir este objetivo se nos ha otorgado una matriz de patrón de carga real de una central nuclear. Dada esta matriz vamos a generar 25000 reactores diferentes para poder seguir con el estudio. Para la generación de estas matrices hay que seguir una serie de pasos establecidos para que el proceso nuclear sea seguro.

1. En primer lugar, el reactor nuclear tiene que guardar una simetría a un cuarto. Esto quiere decir que en una matriz de 17 x 17, la posición (3,4) tiene que se equivalente a la (3, 14), a la (15, 4) y a la (15,14).
2. En segundo lugar, el combustible situado en una posición no puede estar repetido en las posiciones adyacentes, es decir, si tenemos el combustible uno en la posición (i, j) los combustibles de la posición (i-1, j), (i+1, j), (i, j-1) e (i, j+1) tienen que ser diferentes a uno.

Con estas restricciones en cuenta vamos a generar las matrices y a ejecutarlas usando PARCS, un simulador tridimensional de reactores nucleares el cual resuelve ecuaciones de difusión multigrupo de neutrones tanto de manera estacionaria como a lo largo del tiempo, si se le es proporcionada las constantes de difusión[7]. Tras realizar la ejecución de los diferentes casos el siguiente paso será la extracción de los datos. En primer lugar, nos interesa el valor de la k-efectiva, que corresponde al número de electrones medio de una fisión que provoca otra fisión. Este parámetro puede estar entre tres valores:[8]

1. $K<1$: La reacción en cadena no se puede mantener y el proceso acaba parándose

2. K=1: las fisiones generan otra fisión como media. Este es el valor al cual operan las centrales nucleares por seguridad.
3. K>1: este valor de la k es denominado super crítico. En este estado las reacciones de fisión empiezan a aumentar exponencialmente, ya que por cada fisión nuclear es esperado que sucedan k fisiones.

Los siguientes valores que nos interesan son los de la potencia pico, tanto en el plano radial, como el axial y en el plano tridimensional. Estos valores corresponden al punto de mayor densidad de potencia en su respectivo plano dividido por la densidad de potencia media en el reactor. Estos valores se suelen estimarse para comprobar que las vainas de uranio no se funden[9]. Una vez extraído la información de los datos se realizará un análisis de las variables para observar las distribuciones de los datos.

Tras la generación y la extracción de los datos, pasaremos a la creación y despliegue de modelos. En primer lugar, como se ha mencionado en el estado del arte, según demuestran los diferentes artículos citados el uso de códigos como PARCS, exige una gran cantidad de recursos computacionales y de tiempo. Por eso, el primer paso propuesto es de aplicar redes neuronales para el cálculo de los valores previamente mencionados, k-efectiva y las potencias axial, radial y tridimensional. Como punto de partida se tomarán las redes feedforward propuestas en el *paper* de *Saleem*. Una vez tengamos estas redes desplegadas, pasaremos a intentar la mejora de resultados tuneando los valores. Las medidas de error que se observarán con este propósito es el MAE y RMSE, los cuales nos dan el error absoluto medio y la raíz del cuadrado del error medio. La principal diferencia reside en que el segundo de los errores es mas sensible a casos extremos de error. También se utilizará para el análisis del error el MAPE, medida que nos devuelve el valor porcentual de error.

Se utilizará otra aproximación a este problema intentando mejorar los resultados obtenidos por estas redes feedforward, aplicando redes convolucionales. Estas redes son habitualmente aplicadas a reconocimiento de imágenes, una representación matricial, y al disponer de una matriz de combustibles, se plantea el uso de estas redes neuronales en la resolución del problema debido a la afinidad con el problema en el que comúnmente se aplica este tipo de soluciones.

Optimizada la obtención de los datos, pasamos a la optimización del valor de k-efectiva. Para esto vamos a seguir la limitación de esta seguida en las centrales nucleares y es que esta sea igual a uno. Por ello, lo que vamos a hacer es maximizar su valor, siempre y cuando este se mantenga por debajo de su valor límite. Con este objetivo vamos a desarrollar un algoritmo genético cuya función fitness u objetivo, que va a calcular el fitness de cada una de las matrices, y la propuesta es que cada vez que este fitness supere nuestro limite, devuelva un cero. Para la representación de los datos se propone utilizar un vector tal y como se introducen los datos a las redes neuronales, donde cada posición es el valor

del combustible en dicha posición. Los pasos que debemos definir son las diferentes estrategias en cada una de las fases del algoritmo genético. En primer lugar, para la selección de padres, se propone analizar el impacto de la estrategia de selección *best n of m*, selección por torneo y finalmente la selección de ranking lineal. Las tres son estrategias que pueden reportar buenos resultados, aunque estrategias como la selección de ranking lineal pueden provocar convergencias tempranas de los datos. También hay que destacar que las tres estrategias se caracterizan por ser elitistas y por eso se proponen métodos como el *deterministic crowding* para intentar mantener la diversidad de la población. Se propone el *deterministic crowding* frente al *fitness sharing* ya que esta segunda alternativa, requiere mucho más tuneado de hiperparámetros y su aplicación es mucho mas compleja que la del *deterministic crowding*.

La mutación es el paso en el cual modificamos al hijo con el objetivo de expandir nuestras áreas de búsqueda introduciendo una mayor diversidad. Se propone un algoritmo que seleccione un número de celdas y cambie el combustible que encontramos en estas, siempre y cuando sea posible teniendo en cuenta las restricciones establecidas anteriormente.

El cruce de padres que se propone es el intercambio de una fila entre ambos padres, también teniendo en cuenta la simetría de ambos padres, por lo que al cambiar una de estas filas, hay que hacer lo propio con su simétrica, de la misma forma que en la mutación la variación de un valor supone en realidad la variación de cuatro valores. También se propuso el intercambio únicamente de una serie de celdas, pero al cambiar una fila entera en vez de únicamente una celda aumentamos más nuestro área de búsqueda ya que el hijo resultante del cruce tiene una diferencia mayor a la del padre.

El último de los elementos a estudiar es el de el reemplazamiento de la población. En este inicialmente se propuso la aplicación del reemplazamiento elitista y reemplazamiento del peor, pero como se explicará en el apartado en el cual expliquemos el algoritmo genético, el uso del método de remplazamiento del peor esta demostrado que empeora los resultados cuando hay posibilidad de haber casos repetidos y dado que no hemos puesto dicha restricción, no tiene sentido la aplicación de este método. Con esto definido ya podemos poner en marcha nuestro algoritmo genético.

Resumiendo, la aplicación de este trabajo nos permite aumentar la eficacia en tiempo de computación en el cálculo de los diferentes parámetros nucleares gracias a la aplicación de redes neuronales. También se consigue una maximización energética, gracias a la maximización de la k-efectiva producida por el algoritmo genético.

4 Preparación y Comprensión de los datos

4.1 Generación de los datos

Para el desarrollo de este TFG, los datos iniciales eran dos archivos de entrada para el programa PARCS. PARCS es un código usado en el cálculo nuclear el cual, a una configuración de reactor dada y a una serie de parámetros establecidos en los archivos necesarios de entrada, calcula diferentes variables a lo largo de un ciclo de funcionamiento de un reactor nuclear. La Comisión nuclear reguladora de los Estados Unidos (USNRC), define a PARCS como un código que resuelve la dependencia temporal de la difusión de dos grupos de neutrones en una geometría cartesiana tridimensional, usando un método de nodos para obtener la distribución de la transición del flujo de neutrones[10]. Los dos archivos de entrada necesarios para el funcionamiento del programa son el ya mencionado anteriormente *ss_pwr_paths* y el *paths_s1c1*. En el primero se definen valores como la potencia de funcionamiento del reactor, los patrones de carga del reactor, la distribución de las varillas de seguridad, la cantidad de boro en el reactor, entre otras. En el segundo, se introduce información sobre la hidráulica interna del reactor.

Teniendo originalmente solo un rector, el primer paso será la generación de diferentes casos. Para ello modificaremos los patrones de carga del reactor, podemos ver en la figura 1 un ejemplo de cómo es el patrón de carga. El reactor es un reactor de 17x17 y observamos la presencia de varios números diferentes: los ceros, estos son puntos donde no hay ningún material insertado; los 22 son reflectores que se sitúan entrono al núcleo del reactor y dentro encontramos podemos observar un rango de números entre el uno y el ocho, sin tener en cuenta al cinco, que corresponden a los diferentes elementos introducidos en el reactor, estos elementos han sido previamente definidos en el archivo de input de PARCS (*ss_pwr_paths*). El patrón ha de ser igual en ambos archivos para que PARCS pueda procesarlos correctamente, ya que ambos hacen referencia al mismo reactor.

Para la generación del patrón de carga de los reactores tenemos que tener en cuenta algunas restricciones: en primer lugar un elemento en la posición (i, j), siendo i el número de fila y j el de columna, no puede estar también presente en ninguna de las siguientes posiciones (i,j+1), (i+1,j),(i-1,j),(i,j-1); también el reactor tiene que guardar simetría, por lo que nosotros solo tenemos que generar un cuarto de forma aleatoria, ya que el resto del reactor ha de guardar la

simetría con esta porción, es decir, simetría en cada uno de los cuartos; y finalmente el elemento 22 no hay que modificarlo, pues es el material reflector que se sitúa en torno al reactor. Esta parte es la encargada de retener los neutrones que tienden a escapar del reactor, con la finalidad de aumentar la eficiencia de este, los elementos siete y ocho también permanecen sin alterar.

Una vez las restricciones han sido establecidas, se aplicará un programa en Python para generar la población necesaria para el futuro entrenamiento de la red neuronal. Inicialmente se han generado un total de 25000 casos. Cada caso debe tener un patrón de carga único, para ello el programa de generación de las matrices, comprueba que la matriz no esté en la colección de matrices ya generadas. Dicho programa en primer lugar extraerá la matriz del primer reactor y aleatoriamente, cumpliendo las restricciones, irá generando números aleatorios para cada una de las posiciones. Una vez que todas la posiciones han sido alteradas, en primer lugar se comprueba que esa matriz no ha sido generada con anterioridad, para ello se mira una lista con todos los reactores generados y usando funciones de la librería *numpy* de Python se comprueba que no esté repetida dicha matriz en la lista; se generaran ambos archivos, manteniendo la estructura original y cambiando la matriz y el nombre del archivo, teniendo así cada conjunto de archivos diferenciado por un número, *ss_pwr_paths_00001*, ..., *ss_pwr_paths_25000*, y siendo cada conjunto de archivos almacenado en un directorio núcleo_00001, de tal forma que a la hora de ejecutarlos con PARCS quede ordenado por directorios.

```
GEOM
      geo_dim         17 17 28  2  2   !nasyx,nasyy,nz
      rad_conf
      0  0  0  0  0  0 22 22 22 22 22  0  0  0  0  0  0
      0  0  0  0 22 22 22  7  7  7 22 22 22  0  0  0  0
      0  0  0 22 22  7  8  6  2  6  8  7 22 22  0  0  0
      0  0 22 22  7  6  3  2  4  2  3  6  7 22 22  0  0
      0 22 22  7  2  3  2  3  2  3  2  3  2  7 22 22  0
      0 22  7  6  3  2  3  1  3  1  3  2  3  6  7 22  0
     22 22  8  3  2  3  2  3  2  3  2  3  2  3  8 22 22
     22  7  6  2  3  1  3  2  3  2  3  1  3  2  6  7 22
     22  7  2  3  2  3  2  3  2  3  2  3  2  3  2  7 22
     22  7  6  2  3  1  3  2  3  2  3  1  3  2  6  7 22
     22 22  8  3  2  3  2  3  2  3  2  3  2  3  8 22 22
      0 22  7  6  3  2  3  1  3  1  3  2  3  6  7 22  0
      0 22 22  7  2  3  2  3  2  3  2  3  2  7 22 22  0
      0  0 22 22  7  6  3  2  4  2  3  6  7 22 22  0  0
      0  0  0 22 22  7  8  6  2  6  8  7 22 22  0  0  0
      0  0  0  0 22 22 22  7  7  7 22 22 22  0  0  0  0
      0  0  0  0  0  0 22 22 22 22 22  0  0  0  0  0  0
```

Fig 1. Patrón de carga de un reactor

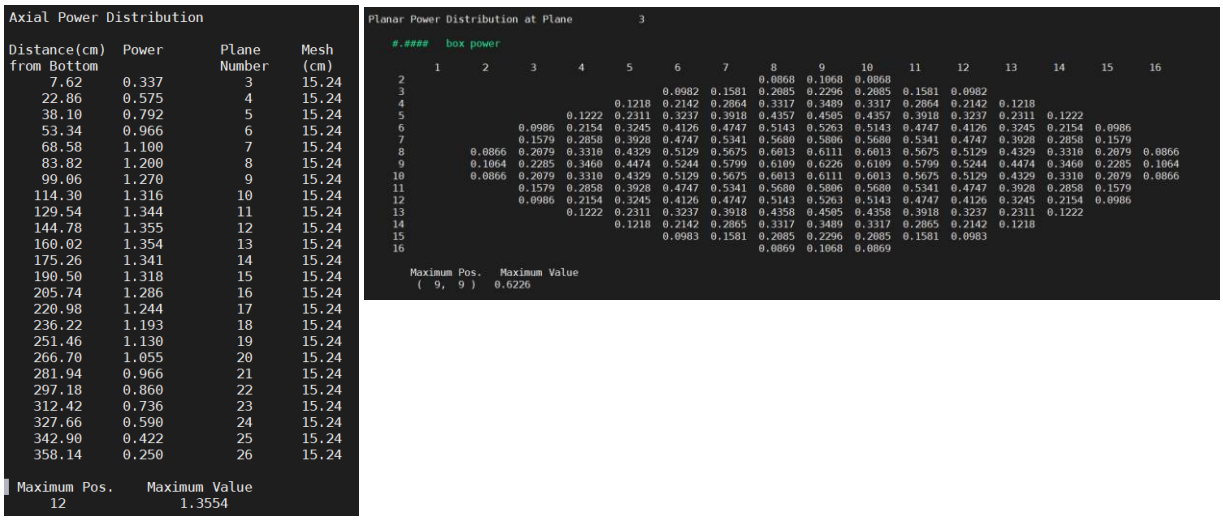
Tras generar los casos pasaremos a ejecutar los diferentes patrones de carga usando PARCS y así poder obtener los valores que necesitaremos pasar a la red neuronal. Primero se ejecutará un PARCS con la potencia del reactor reducida y sin tener en cuenta las varillas del reactor, con finalidad de testear el programa, y posteriormente con las varillas activadas y la potencia del reactor al cien por cien. El tiempo de generación de todos los casos a máxima potencia del reactor

es de 180719.619 segundos. En la Figura 2 podemos ver un extracto del archivo *ss_pwr_paths_25000.out*, el cual es la salida generada al ejecutar el archivo *inp* de mismo nombre. En esta imagen podemos observar información importante como es la K-efectiva, factor de multiplicación de los neutrones y la distribución de energía en el plano transversal, siendo cada celda la potencia media de cada una de las vainas en su plano axial.

```
 K-Effective:     1.136889
 Boron Conc:    1044.00
 Core Power Level: 1.000000E+00

Assembly Power Distribution

   #.####   box power

       1       2       3       4       5       6       7       8       9      10      11      12      13      14      15      16
   2                                                       0.2827  0.3459  0.2827
   3                                               0.3180  0.5037  0.6559  0.7191  0.6559  0.5037  0.3180
   4                                       0.3913  0.6715  0.8832  1.0093  1.0588  1.0093  0.8832  0.6715  0.3913
   5                               0.3927  0.7217  0.9865  1.1729  1.2910  1.3293  1.2910  1.1729  0.9865  0.7217  0.3927
   6                       0.3191  0.6757  0.9888  1.2302  1.3881  1.4899  1.5155  1.4899  1.3881  1.2302  0.9888  0.6757  0.3191
   7                       0.5031  0.8805  1.1779  1.3883  1.5382  1.6147  1.6478  1.6147  1.5382  1.3883  1.1779  0.8805  0.5031
   8               0.2823  0.6547  1.0099  1.2820  1.4869  1.6137  1.6945  1.7114  1.6945  1.6137  1.4869  1.2820  1.0099  0.6547  0.2823
   9               0.3450  0.7167  1.0503  1.3224  1.5116  1.6463  1.7111  1.7407  1.7111  1.6463  1.5116  1.3224  1.0503  0.7167  0.3450
  10               0.2822  0.6547  1.0099  1.2820  1.4869  1.6137  1.6946  1.7114  1.6946  1.6137  1.4869  1.2820  1.0099  0.6547  0.2822
  11                       0.5032  0.8805  1.1779  1.3884  1.5382  1.6147  1.6478  1.6147  1.5382  1.3884  1.1779  0.8805  0.5032
  12                       0.3191  0.6757  0.9888  1.2302  1.3881  1.4899  1.5155  1.4899  1.3881  1.2302  0.9888  0.6757  0.3191
  13                               0.3927  0.7217  0.9866  1.1729  1.2910  1.3293  1.2910  1.1729  0.9866  0.7217  0.3927
  14                                       0.3914  0.6715  0.8832  1.0094  1.0589  1.0094  0.8832  0.6715  0.3914
  15                                               0.3182  0.5037  0.6559  0.7192  0.6559  0.5037  0.3182
  16                                                       0.2829  0.3459  0.2829

     Maximum Pos.    Maximum Value
      (  9,  9 )    1.7407
```

Fig 2. K-efectiva y potencia radial del archivo ss_pwr_paths_25000.out

Con los datos generados, procedemos a la extracción de valores. PARCS, tras ejecutar un archivo *inp,* devuelve una serie de archivos cada uno con una información diferente*, dep, hst, out*... Para este análisis nos centraremos en el *out*. Los datos que extraeremos serán en primer lugar las diferentes matrices para cada caso, ya que en un futuro la matriz de patrón de carga será nuestra entrada para la red neuronal. Otra información interesante será la k-efectiva, el número medio de neutrones de una fisión que provocan otra fisión. Otro de los datos extraídos será la potencia radial máxima, la cual se extrae de la figura 3, los valores que observamos es la media de la potencia de cada una de las vainas. También se extraerá la potencia axial media, que sería la potencia media en cada una de las capas, como podemos observar en la figura 3, empieza a contar en el plano tres y acaba en el 26, esto se debe a que las dos primeras capas y las dos últimas se reservan para reflectores, por eso no tienen potencia.



Axial Power Distribution

Distance(cm) from Bottom	Power	Plane Number	Mesh (cm)
7.62	0.337	3	15.24
22.86	0.575	4	15.24
38.10	0.792	5	15.24
53.34	0.966	6	15.24
68.58	1.100	7	15.24
83.82	1.200	8	15.24
99.06	1.270	9	15.24
114.30	1.316	10	15.24
129.54	1.344	11	15.24
144.78	1.355	12	15.24
160.02	1.354	13	15.24
175.26	1.341	14	15.24
190.50	1.318	15	15.24
205.74	1.286	16	15.24
220.98	1.244	17	15.24
236.22	1.193	18	15.24
251.46	1.130	19	15.24
266.70	1.055	20	15.24
281.94	0.966	21	15.24
297.18	0.860	22	15.24
312.42	0.736	23	15.24
327.66	0.590	24	15.24
342.90	0.422	25	15.24
358.14	0.250	26	15.24

Maximum Pos.	Maximum Value
12	1.3554

Planar Power Distribution at Plane 3

#.#### box power

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2								0.0868	0.1068	0.0868						
3						0.0982	0.1581	0.2085	0.2296	0.2085	0.1581	0.0982				
4					0.1218	0.2142	0.2864	0.3317	0.3489	0.3317	0.2864	0.2142	0.1218			
5				0.1222	0.2311	0.3237	0.3918	0.4357	0.4505	0.4357	0.3918	0.3237	0.2311	0.1222		
6			0.0986	0.2154	0.3245	0.4126	0.4747	0.5143	0.5263	0.5143	0.4747	0.4126	0.3245	0.2154	0.0986	
7			0.1579	0.2858	0.3928	0.4747	0.5341	0.5680	0.5806	0.5680	0.5341	0.4747	0.3928	0.2858	0.1579	
8		0.0866	0.2079	0.3310	0.4329	0.5129	0.5675	0.6013	0.6111	0.6013	0.5675	0.5129	0.4329	0.3310	0.2079	0.0866
9		0.1064	0.2285	0.3460	0.4474	0.5244	0.5799	0.6109	0.6226	0.6109	0.5799	0.5244	0.4474	0.3460	0.2285	0.1064
10		0.0866	0.2079	0.3310	0.4329	0.5129	0.5675	0.6013	0.6111	0.6013	0.5675	0.5129	0.4329	0.3310	0.2079	0.0866
11			0.1579	0.2858	0.3928	0.4747	0.5341	0.5680	0.5806	0.5680	0.5341	0.4747	0.3928	0.2858	0.1579	
12			0.0986	0.2154	0.3245	0.4126	0.4747	0.5143	0.5263	0.5143	0.4747	0.4126	0.3245	0.2154	0.0986	
13				0.1222	0.2311	0.3237	0.3918	0.4358	0.4505	0.4358	0.3918	0.3237	0.2311	0.1222		
14					0.1218	0.2142	0.2865	0.3317	0.3489	0.3317	0.2865	0.2142	0.1218			
15						0.0983	0.1581	0.2085	0.2296	0.2085	0.1581	0.0983				
16								0.0869	0.1068	0.0869						

Maximum Pos.	Maximum Value
(9, 9)	0.6226

Figura 3. Distribución de la potencia axial a la izquierda y de la potencia radial por plano a la derecha del archivo ss_pwr_paths_25000.out

El último de los campos extraídos para el análisis es la máxima potencia, pero esta vez para cada plano, para obtener este valor se ha usado un programa de Python que almacenaba el mayor encontrado hasta el momento mientras avanzaba capa por capa. En la figura tres podemos encontrar un ejemplo de la primera de las capas en el reactor 25000. Como ya se ha explicado, las capas uno, dos, 27 y 28 se reservan para reflectores.

4.2 Análisis de los datos

Con los datos ya extraídos en un archivo de texto, se ha realizado un pequeño análisis de estos en Python. De este análisis podemos observar, en primer lugar, que todas las máximas de valores se encuentran en el centro del reactor como observamos en la figura 4. Podemos ver en estas figuras como para todos los casos los valores se mantienen en el mismo punto. En el caso en el que observamos la posición radial tanto en planos o en la media, observamos que el valor máximo se encuentra en el punto (9,9), que es el punto medio de la matriz. En el caso axial, también observamos que el valor máximo se encuentra en el punto medio, en el nivel 12. Así que el punto medio del reactor es el que más potencia concentra.

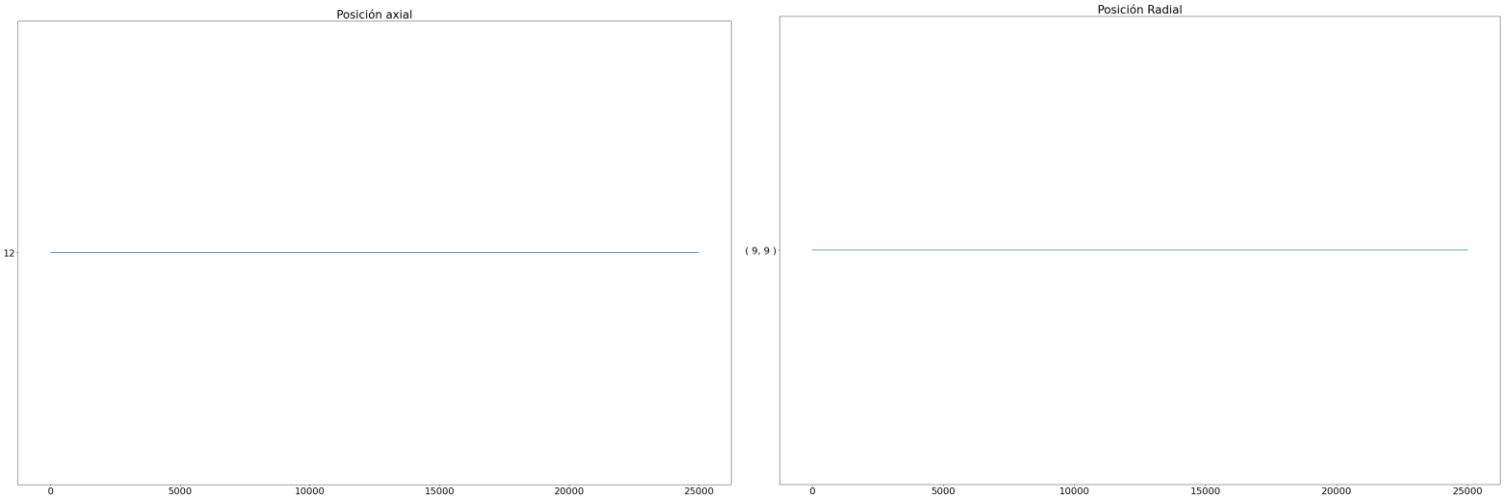


Fig 4. Distribución de la posición de la potencia axial y radial.

También se han realizado un análisis de las distribuciones de todas las variables mencionadas anteriormente. Podemos observar en el análisis que todas las variables siguen una distribución muy parecida. Esto lo podemos observar en la tabla 1 donde se ve como para todas las variables son del orden de 10^{-3}. Para observar esto también podemos ver los gráficos de la imagen 5, donde también se observa como las distribuciones son muy parecidas para todas las variables.

	K-effective	Radial_potencia	Axial_pot	Tres_D_pot
count	25000.000000	25000.000000	25000.000000	25000.000000
mean	1.137111	1.737108	1.354631	2.369725
std	0.000215	0.005503	0.000941	0.007086
min	1.136210	1.716200	1.351300	2.339200
25%	1.136970	1.733700	1.354000	2.365100
50%	1.137112	1.737500	1.354600	2.369900
75%	1.137254	1.740900	1.355200	2.374600
max	1.137932	1.757600	1.358700	2.398200

Tabla 1. Datos estadísticos de las variables.

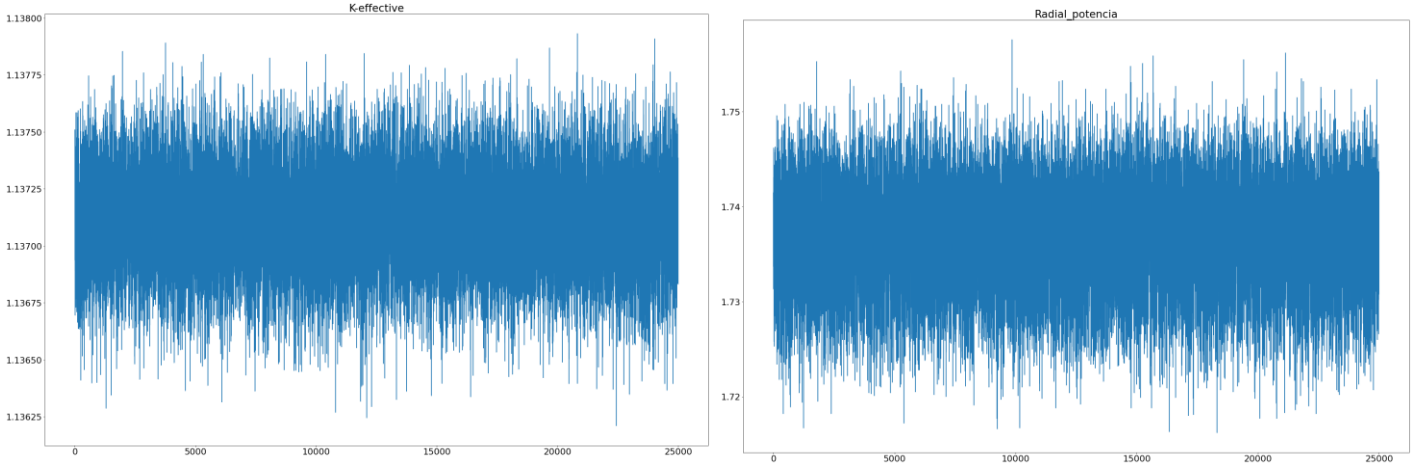


Fig5. Gráficos de distribución de las variables

5 Conocimiento extraído y evaluación de modelos

5.1 Redes neuronales

5.1.1 Redes feedforward

Tras realizar la generación de los diferentes reactores y recopilación de los datos, se continuará con la aplicación de los mismos. La idea principal es aplicar redes neuronales para dada una matriz poder predecir los datos que resultarán de dicho patrón de carga. La estructura seguida en esta parte del desarrollo será la propuesta por el *paper* "*Application of deep neural networks for high-dimensional large BWR core neutronics*"[2] en este *paper* se utilizaban tres redes neuronales:

- La primera de ellas se utiliza para el cálculo de los *Power Peaking Factors*, es decir para calcular lo que sería el máximo axial, radial y en cada una de las capas. Los parámetros propuestos en ese *paper* para esta primera red serían 5 capas ocultas; con 66, 59, 51, 46 y 41 nodos cada capa respectivamente; el activador utilizado en las diferentes capas es ReLu, siendo su función:

$$f(x) = \max(0, x)$$

  En las redes neuronales los activadores se utilizan para generar los pesos de las neuronas en cada iteración, con el uso de ReLu obligamos a que es esos pesos sean positivos. Los principales beneficios de ReLu frente a otras aproximaciones es que por un lado reduce la probabilidad de desaparecer del gradiente y por otro que es computacionalmente menos costoso que otros métodos como sigmoid[11]. Finalmente, el optimizador propuesto para la red neuronal es Adam. Un optimizador en las redes neuronales es un algoritmo que se usa para cambiar atributos de tus redes neuronales como pueden ser los pesos de los nodos para reducir las pérdidas. Hay muchos tipos diferentes de optimizadores que se pueden utilizar en redes neuronales, Adagrad (que cambia el learning rate en cada iteración), AdaDelta (no decae el *learning rate*) o el Adam que es el propuesto en este paper. Este método de optimización es una mezcla de otros métodos, el Adagrad, ya mencionado antes y el *Root Mean Square Propagation* o RMSProp. Las ventajas de Adam es que el método es rápido y converge rápidamente y que soluciona problemas como que el *learning rate* vaya desapareciendo y la varianza.[12]

$$V_t = \beta V_{t-1} + (1 - \beta)\nabla f(\theta_t)$$

$$S_t = \rho S_{t-1} + (1-\rho)(\nabla f(\theta_t) * \nabla f(\theta_t))$$

$$V_{t,corr} = \frac{V_t}{1-\beta^t}$$

$$S_{t,corr} = \frac{S_t}{1-\rho^t}$$

$$\theta_{i,t+1} = \theta_{i,t} - \alpha \frac{V_{i,t,corr}}{\epsilon + \sqrt{S_{i,t,corr}}}$$

Finalmente, el *batch size* y el número de *epochs* utilizados serán 64 y 100 respectivamente. Estos parámetros definen en primer lugar, el *batch size* el número de muestras para trabajar antes de actualizar los parámetros internos del modelo como los pesos, mientras que los *epochs* definen cuantas veces se va a repetir el proceso de aprendizaje sobre los datos.

- La segunda de las redes neuronales propuestas se encarga del cálculo de la K-efectiva. Para esta red neuronal los parámetros propuestos son: tras campas ocultas, las cuales tienen 67, 50 y 45 nodos respectivamente; como método de activación se vuelve a utilizar el ReLu; como optimizador se utiliza como en el anterior caso el Adam. Por otro lado, para el *batch size* y para los *epochs* los valores utilizados serán 64 y 100 igual que para la anterior red neuronal.

- La tercera de las redes neuronales se corresponde a la que calcula la duración del ciclo. Para esta red neuronal se proponen 4 capas ocultas de 50, 45, 45 y 40 nodos respectivamente; los valores asignados al *batch size* y a los *epochs* son los mismos valores asignados a las anteriores redes neuronales.

De estas redes neuronales del *paper* de Saleem, para este trabajo se ha decidido replicar al primera y la segunda, es decir la que nos permite predecir las potencias máximas del reactor en los diferentes perfiles y aquella red neuronal que nos permite el cálculo de la K-efectiva.

Para realizar la comprobación de la exactitud de los resultados se ha utilizado la métrica MAPE (*Mean Absolute Percentage Error*), MAE (*Mean Absolute Error*) y el RMSE (*Root Mean Square Error).* El MAPE es una medida la cual nos describe el tamaño del error ocurrido a través de porcentajes.

$$MAPE = \frac{1}{N}\sum_{j=1}^{n} \left| \frac{u_j - \hat{u}_j}{u_j} \right|$$

El MAE es una métrica que mide la magnitud media del error en una predicción, no se tiene en cuenta la dirección ya que se aplica un valor absoluto. El cálculo es la media de la diferencia en valor absoluto de los datos predichos menos los valores reales.

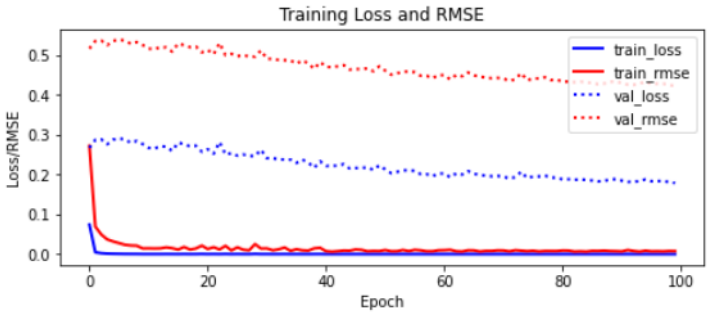
$$MAE = \frac{1}{n}\sum_{j=1}^{n}|u_j - \hat{u}_j|$$

Por otro lado, el RMSE tiene implicaciones similares a MAE, pero en este en vez de aplicar valores absolutos, en primer lugar, la diferencia se eleva al cuadrado y al total se le aplica una raíz cuadrada esto provoca que a los errores grandes les da un mayor peso, lo que le hace la métrica indicada para detectar cuando hay errores grandes y evitarlos.

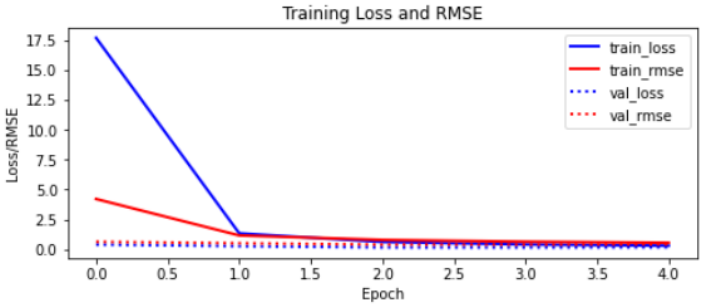
$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(u_j - \hat{u}_j)^2}$$

Con estos parámetros establecidos y usando los valores sugeridos por el *paper* de Saleem, se han programado ambas redes neuronales usando la librería de python keras. La entrada de ambas redes neuronales sería un vector donde cada uno de los valores es una posición de la matriz, la cual no se introduce directamente si no que se eliminan las posiciones a 0, 22, 7 y 8, ya que estos valores no afectan a los resultados. Mientras que la salida correspondería en la primera de las redes neuronales a un vector de tres valores, correspondientes a la potencia radial, axial y en el plano 3D. Por otro lado, en la segunda de las redes neuronales la salida sería un único valor de la K-efectiva. He de recalcar que en la capa de salida el método de activación utilizado para ambas redes es lineal puesto que ambas redes devuelven valores numéricos.

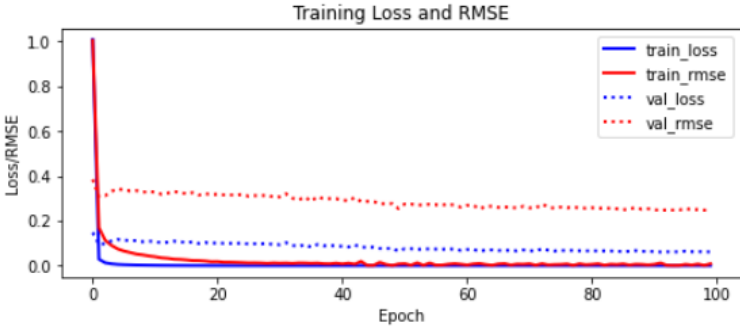
Para la primera de las redes neuronales encontramos que hay un ligero sobreajuste ya que podemos observar en el gráfico de perdida como hay una diferencia entre el error en el training frente al error de la validación del modelo. Esto puede venir causado por el gran número de *epochs* utilizados. En la segunda de las imágenes podemos observar los resultados de la misma red neuronal cambiando el número de *epoch* a 5 y el *batch size* a 1024, en este caso vemos que el valor del error es muy similar en ambos casos, vemos como no se produce el mismo caso de sobreajuste que ocurría en la base de datos original.



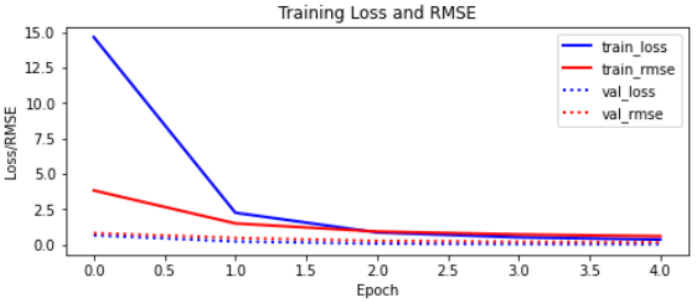
predición
mape: 22.15724828024905
rmse: 0.42313131546267974
mae: 0.40783597343686623



predición
mape: 22.606929618911742
rmse: 0.4110034552746237
mae: 0.3946487555054329



predición
mape: 21.802089562800525
rmse: 0.2479144214668719
mae: 0.2479141415991343



predición
mape: 13.205890517676938
rmse: 0.1801804508689389
mae: 0.1501657440693299

Fig.6. Graficos de perdida redes feedforward. Los dos superiores para potencias los dos inferiores k-efectiva

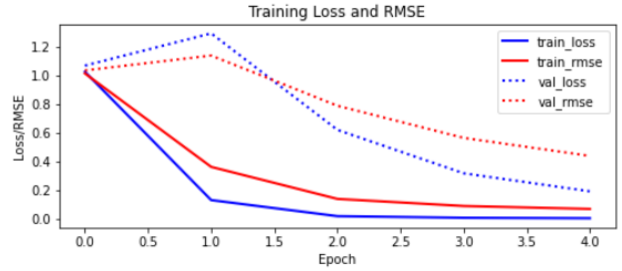
En la red neuronal encargada del cálculo del cálculo de la k-efectiva, los resultados son similares a los obtenidos en la enterior red neuronal. En primer lugar con los valores propuestos en el paper de Saleem nos encontramos, con una tendencia a sobrejuste y un estacionamiento en la mejora de los resultados en las 20 primeras epochs. Con objetivo de mejorar dichos resultados se han probado diversas configuraciones de redes neuronales. Finalmente, con una red de dos capas de 135 y 100 nodos respectivamente, y 5 epochs y 1024 como batch size. Con estos nuevos parametros observamos como ha habido una mejora tanto en el tema de sobreajuste como en una reducción de error. Recalcar que la diferencia entre mae y rmse viene dada porque el rmse da más peso a casos extremos.

5.1.2 Redes neuronales convolucionales

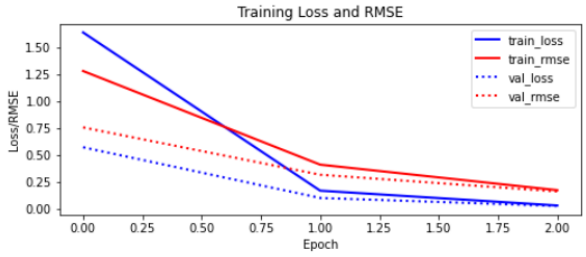
También se han propuesto una serie de redes neuronales convolucionales para aproximar el problema. Estas redes neuronales son utilizadas para procesar datos que tienen forma de cuadricula, como en nuestro caso una matriz. La

característica principal de este tipo de red es el uso de una operación matemática llamada convolución en una de sus capas.

Para el primero de los casos, se ha utilizado un *filter size* de 32 para la capa convolucional y posterior mente se han seguido los pasos habituales en las redes convolucionales. Se ha aplicado una capa de activación ReLu, este tipo de operación se realiza para introducir ya que los valores suelen ser lineales, así se rompe la linealidad. Posteriormente se ha usado una *Pooling Layer*, la cual extrae un resumen de cada uno de los sectores, con esta operación se reduce el tamaño, lo que disminuye el coste computacional del problema. En nuestro caso se ha usado *MaxPool*, el cual busca el valor máximo en el área acotada y lo guarda, de tal manera que si tenemos una matriz [[4,5], [6,9]] y queremos usar *MaxPooling* para comprimirlo a un valor la matriz pasaría a ser 9. También se ha usado el *avergePool*, el cual en vez de coger el máximo calcula la media de los valores, siguiendo con el ejemplo anterior nos devolvería 6. Y generando cuatro redes iguales concatenadas. Para este caso los mejores resultados obtenidos ha sido con un MAPE de 8.498. Como podemos observar en la imagen X es una mejora respecto a las anteriores redes neuronales y también respecto a la red convolucional anterior la cual no generaba varias capas, podemos ver como la reducción del MAPE es significativa de 21.018 a 8.489.



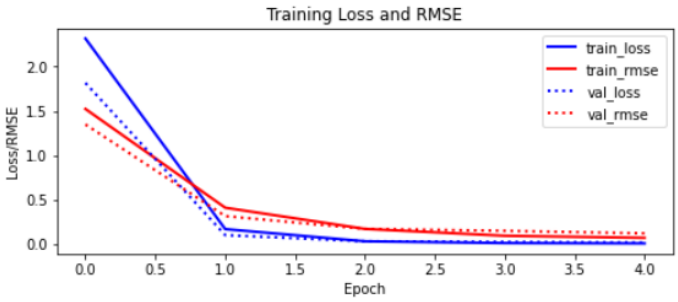
predición
mape: 21.018702499649184
rmse: 0.4381274649836371
mae: 0.3057832299306455



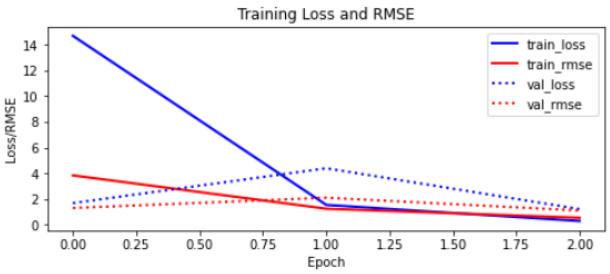
predición
mape: 8.498268291751945
rmse: 0.16012963818571363
mae: 0.1507239766524856

Fig8. Gráficos redes convolucionales para el cálculo de potencias

Para la red convolucional encargada del cálculo de la k-efectiva, se han utilizado dos capas de filtro una 64 y otra de 128, mientras que el resto de los parámetros es igual al de las redes convolucionales descritas para el cálculo de la potencia radial, axial y en el plano 3D. El número de *epochs* y de *batch_size* se conserva en todas las pruebas a 5 y 1025 respectivamente. El procedimiento para las redes neuronales es igual al realizado en el anterior caso, en primer lugar, se usa la red ya descrita y posteriormente se utiliza un procedimiento similar en el cual se utilizan consecutivamente 4 veces la red neuronal. Para el caso de la k-efectiva el mejor resultado es utilizando el primer modelo, donde el mejor resultado obtenido ha sido un MAPE de 8.44 mientras que como podemos observar en la imagen el resultado obtenido utilizando el segundo procedimiento es bastante peor que con el primero.




```
predición
mape:  8.440164565386317
rmse:  0.11831770938751017
mae:  0.0959740820806784
```




```
predición
mape:  96.18160738225285
rmse:  1.0987463908648296
mae:  1.0936923055124104
```

Fig9. Gráficos de perdida redes convolucionales para el cálculo de la k-efectiva

En la tabla 2 podemos observar como con la aplicación de las redes neuronales ha reducido el coste temporal de los cálculos realizados por PARCS significativamente. Como se observa es una reducción de dos segundos por caso

Matriz	Tiempo red neuronal	Tiempo PARCS
1	0.043 seg	2.066 seg
2	0.011 seg	2.089 seg
3	0.011 seg	2.082 seg
4	0.013 seg	2.065 seg
5	0.011 seg	2.075 seg
6	0.012 seg	2.068 seg
7	0.012 seg	2.085 seg
8	0.012 seg	2.093 seg
9	0.012 seg	2.064 seg
10	0.011 seg	2.093 seg

Tabla 2. Comparativa de tiempos

5.2 Algoritmos genéticos

Tras obtener estos resultados guardamos las redes neuronales utilizando *model.save*, esto nos permite cargar dichos modelos en un futuro para futuras pruebas.

Las redes neuronales nos permiten calcular los cuatro valores, potencia axial, potencia radial, potencia en 3D y la k-efectiva de una manera más rápida y cómoda que usando PARCS. Con esta redes neuronales ya entrenadas el siguiente paso a desarrollar será un algoritmo genético el cual nos permita optimizar las salidas de tal forma que podamos obtener el patrón de carga que nos maximice el valor de las potencias.

En primer lugar, definiremos qué es un algoritmo genético. Un algoritmo genético es un metaheurístico que se basa en la teoría de la evolución natural, lo que nos permite optimizar problemas, para poder encontrar la mejor solución a un problema[13]. El proceso que siguen los algoritmos genéticos sigue la siguiente pauta: se inicia una población, calculamos fitness, seleccionamos unos individuos de esa población para cruzarlos, una vez cruzados mutamos a los individuos resultantes y le calculamos el fitness. Si ese fitness es el mejor posible paramos si no seguimos hasta alcanzar el punto de pausa que hayamos decidido al modelar el problema. A continuación, vamos a explicar los diferentes pasos y las decisiones tomadas para la aplicación de nuestro algoritmo genético.

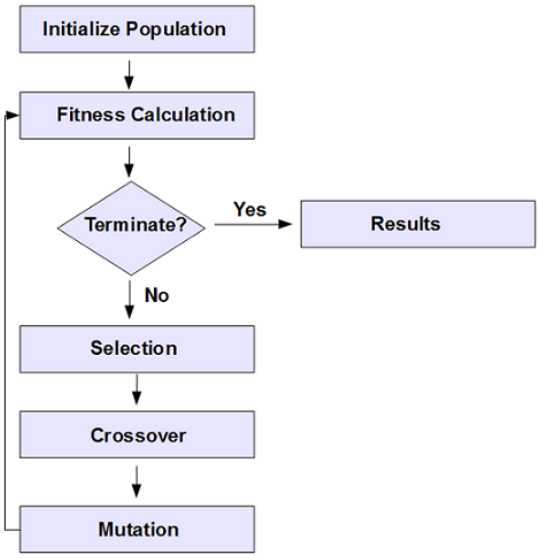


Fig10. Esquema algoritmo genético

5.2.1 Representación del problema y generación de cromosomas

En primer lugar, explicaremos el modo de representación escogido para los cromosomas. Tomando la representación utilizada para las matrices los cromosomas se representan en vectores, donde cada posición es la posición en la cual el combustible está situado en el patrón de carga y por otro lado el valor es el tipo de combustible utilizado. Con la representación de los cromosomas ya definida, el siguiente paso sería la generación de reactores.

Para generar un reactor utilizaremos una versión del generador de reactores que se utilizó para generar los datos. Este lo que hará es ir recorriendo los valores de la matriz que no se correspondan a los valores cero, 22, siete u ocho, y los ira cambiando atento a la norma de que ninguno de los valores de uno de los combustible se puede repetir en su casilla superior, inferior, izquierda y derecha, para esto se utilizará el mismo proceso que se sigue y explica en la generación de los datos y es analizar el (i+1, j), (i-1, j), (i, j+1) e (i, j-1). Con ese generador de reactores configurado lo serializamos dentro de un bucle para generar la población que necesitemos.

5.2.2 Función fitness

El cálculo del fitness es el siguiente paso dentro del algoritmo genético. Esta función nos devuelve el valor de cada uno de los individuos que estamos evaluando y es el valor que se busca optimizar con el algoritmo genético. Para el cálculo de la función fitness nos serviremos de las redes neuronales generadas y ya preparadas. Para generar la función tenemos en cuenta que es aquello que queremos maximizar en nuestro reactor. En nuestro caso es la k-efectiva del reactor. Para esto lo que hemos decidido es usar las redes neuronales para calcular la k-efectiva del individuo y una vez tenemos esta comprobamos el valor. Si esta es mayor que uno, nuestro valor limite ya que un valor mayor a este es considerado inseguro, devolvemos cero, ya que sería un valor que no sería válido y por lo tanto no es una solución posible. En caso de ser un valor inferior a uno lo devolvemos.

5.2.3 Métodos de selección

La selección de padres en un algoritmo genético define la zona de población por la cual nuestro algoritmo buscara las posibles soluciones al problema. Esto quiere decir que una selección de padres muy elitista podría tener problemas a la hora de encontrar la mejor solución, pues nuestro algoritmo tenderá a buscar

zonas donde a priori hay mejores soluciones, dejando aquellas donde el fitness no es óptimo sin investigar. Lo que podría provocar que la solución final sea una aproximación, pero no la mejor solución posible a nuestro problema. Por ejemplo, una selección proporcional al fitness puede tener este problema, ya que en este método nos dedicamos a dar probabilidades según su valor de fitness dando mayor probabilidad a aquellos casos en los cuales el valor de fitness es superior. Esto provoca que los valores con un alto fitness ocupen rápidamente la población y tengamos una convergencia prematura.

Se han propuesto tres métodos para el funcionamiento de esta algoritmo genético. En primer lugar, se ha programado un método sencillo el cual de una población escoge aleatoriamente m cromosomas y de los cuales los ordena según su fitness y serán seleccionados como padres los n cromosomas con el fitness más alto. En este modelo de selección de padres hay una probabilidad mayor de escoger fitness elevados a cromosomas con un menor valor de fitness.

La segunda de las metodologías escogidas es la selección por torneo. El fundamento de esta metodología consiste en realizar varios torneos entre una muestra de cromosomas escogidos aleatoriamente entre la población total. El ganador de cada torneo es escogido para pasar a la siguiente ronda, así sucesivamente hasta que conseguimos que solo quede un individuo posible[14]. Este tipo de selección es también elitista pues, aunque podría haber valores con fitness bajos en los torneos, la probabilidad de que los fitness seleccionados sean los grandes es mucho mas elevada. La aplicación de este método en Python se ha realizado con dos bucles anidados, uno externo para el número de individuos a devolver y otro interno para el numero de individuos que 'competirán' en el torneo, dentro de este bucle en cada iteración elegimos a un individuo aleatorio de la población para introducirlo en el torneo, cuando los tenemos seleccionados ordenamos la lista según fitness y nos quedamos con el cromosoma con mejor índice. Al terminar el torneo tenemos a n padres seleccionados. Para minimizar los efectos de estas estrategias elitistas se utilizan métodos de mantenimiento de la diversidad que explicaremos más adelante.

Finalmente, la tercera de las metodologías escogidas. Es la selección por ranking lineal. La base de este método es en vez de usar el fitness directamente del individuo, utilizamos un ranking dependiendo de la posición según su fitness, ya que, según este se le asigna una probabilidad que va incrementando linealmente a medida que el fitness aumenta. Este método tiene un hiperparámetro a ajustar *(s, 1<s≤2)*. Cuanto más próximo a uno se encuentre más se asemeja a una selección uniforme, mientras que, a más próximo al dos, mayor es la probabilidad acumulada en el individuo con un mayor fines. Aunque el máximo de probabilidad del mejor cromosoma, independientemente del tamaño de la muestra, es $\frac{2}{|P|}$ siendo P la población. Para esta aproximación generamos una lista de probabilidades para todos los individuos de tal manera que aquellos que tengan un mayor fitness tengan una mayor probabilidad de ser escogidos. Tras

generar estas listas generamos un bucle que se repita n veces siendo n el número de padres a escoger, y generamos un numero entre cero y uno, si ese numero es inferior a la probabilidad acumulada hasta el momento añadimos al padre y rompemos el bucle. La función utilizada para el cálculo de las diferentes probabilidades sería la si siguiente:

$$p_i = \frac{2-s}{|P|} + \frac{2i(s-1)}{|P|(|P|-1)}$$

5.2.4 Estrategias de cruce

Las estrategias de cruce son el siguiente paso en los algoritmos genéticos. El concepto de cruce entre dos individuos de un algoritmo genético pretende asemejar lo ocurrido en el cruce genético y lo ocurrido en la reproducción biológica. A través de dos padres buscamos sacar hijos que nos permitan mejorar los resultados obtenidos hasta el momento. Es decir, de los individuos seleccionados a través del torneo vamos a emparejarlos y vamos a dada una probabilidad cruzarlos, lo que quiere decir que se realiza un *random.choice,* lo que nos devolverá un valor aleatorio entre el cero y el uno. Si dicho valor se encuentra por debajo de una probabilidad establecida realizaremos el cruce. Hay muchas estrategias de cruce ya predefinidas como puede ser *cut and crossfill*. El procedimiento de esta estrategia es el siguiente: tenemos dos padres, y seleccionamos una posición de ambos vectores, la misma en los dos, y copiamos ese primer segmento en el hijo uno el del padre uno y el del padre dos en el hijo dos. Mientras vamos a ir añadiendo los elementos del sector segundo del padre dos al hijo uno hasta acabar los elementos del padre dos, si en este momento no está completo el hijo uno vamos a acabar de rellenarlo con los valores del sector uno del padre dos. Con el hijo uno completo, hacemos el mismo proceso, pero esta vez con el padre uno y con el hijo dos hasta completar a este. Hay operadores de cruce también más sencillos como podría ser cruce por un punto, donde tomamos tanto en el padre uno y dos un punto de corte, también el mismo en ambos vectores. Posteriormente establecemos el primer segmento de ambos como hijo uno e hijo dos respectivamente y el segmento dos del padre dos pasaría a ser el segmento dos del hijo uno y lo propio en el vector del hijo uno. Otro de los métodos clásicos de cruce es el cruce uniforme. En este método se eligen x números aleatorios de ambos vectores y se intercambian mutuamente para así generar al hijo uno y al hijo dos. Dependiendo el problema que intentamos resolver el método a utilizar será uno u otro, ya que no todos los métodos son utilizables en cualquier situación. Para nuestro problema se ha generado un sistema de cruce basado en el intercambio de filas entre matrices. Para ello simplemente el sistema utilizado es un selector aleatorio entre las filas de la dos a las ocho ya que las filas superiores a esas serian innecesarias pues guarda simetría con las anteriores. Por lo que elegimos una fila de esta e

intercambiamos dicha fila y su correspondiente simétrica. Lo que nos resulta en dos hijos fruto del cruce de ambos padres.

5.2.5 Estrategias de mutación

Las estrategias de mutación son una serie de operadores que se utilizan para mantener la diversidad en la población entre generaciones.[15] La idea se basa en la suposición de que, si un padre tiene un buen material genético, entonces si modificamos ligeramente al padre, el hijo tendrá buen material genético y nos permite explorar las áreas cercanas al plane con intención de mejorar el resultado del padre.

Existen diversos métodos de mutación dependiendo del problema que intentemos resolver, por ejemplo, para un problema donde la representación es binaria podemos realizar mutación por bit, cambiando celdas aleatorias de uno a cero o viceversa. Otro ejemplo podría ser en caso de tener un vector de números enteros podríamos cambiar celdas aleatorias del mismo por números enteros aleatorios. Hay otros métodos donde se suman a las celdas aleatorias un número racional y el resultado se redondea. Como podemos ver hay diversas formas de realizar mutaciones. El método aplicado para este algoritmo genético se basa en escoger celdas aleatorias y modificar sus valores respetando la simetría de la matriz como ya se ha explicado anteriormente, comprobando que el valor a cambiar es diferente de aquellos que se encuentran a su entorno y mantiene la simetría de la matriz. La única diferencia con el apartado de la generación de las matrices es que en esta solo escogemos un punto o un conjunto, pero no el total y solo tenemos que comprobar que esos puntos escogidos cumplan que no están entre los valores a no cambiar, es decir que no sean cero, 22, siete u ocho.

5.2.6 Gestión de la población

La gestión de población se realiza a través de métodos de reemplazamiento para determinar que hijos sustituyen a los padres. Existen dos aproximaciones a este problema:

1. El modelo generacional: donde la generación anterior es totalmente sustituida por los hijos.
2. El modelo de estado estable: se generan x individuos que van a sustituir a *x* individuos de la población previa

Para este trabajo se han propuesto en un principio dos métodos de reemplazamiento de la población. La primera es el reemplazamiento elitista el

cual se basa en la idea de reemplazar a aquel individuo que tenga un mayor fitness, así siempre sobrevivirá aquel individuo con un mayor valor de fitness.

El segundo de los métodos de reemplazamientos propuestos es el genitor o reemplazamiento de los peores, en el cual se reemplazan los peores individuos de la población, que nos proporciona normalmente una rápida mejora del fitness medio, aunque puede ocasionar una convergencia prematura. Pero hemos descartado esta opción, debido a como se menciona en un *paper* de Darrell Whitley[16], el uso de este tipo de metodología en problemas donde pueden existir duplicados podría afectar a los resultados, ya que como explica el aumento de duplicados, aunque nos aumenta la población disminuye la diversidad de esta lo que provoca un empeoramiento de los resultados. Por eso se ha descartado este método en la selección de padres y se ha decidido trabajar únicamente con el reemplazamiento elitista.

5.2.7 Mantenimiento de la diversidad

Para mantener la diversidad de la población también se ha utilizado un método de *crowding determinista*. En esta método con el objetivo de mantener la diversidad hacemos competir a padres con hijos. Esto hace que al enfrentar padres similares con hijos similares se mantenga un espacio de búsqueda homogéneamente distribuido. Esto se hace calculando la distancia entre padres e hijos *(p1, h1), (p1, h2), (p2, h2) y (p2, h1),* la distancia depende del dato que se utilice normalmente para valores binarios utilizamos la distancia de Hamming, mientras que para otros valores usaríamos la distancia Euclídea[17]. Resumiendo, el proceso realizado sería:

1. Aleatoriamente los padres son seleccionados
2. Cruzamos a los padres para producir a dos hijos
3. Mutamos a los hijos
4. Emparejamos a los padres y a los hijos de forma *(p1, h1), (p1, h2), (p2, h2) y (p2, h1)* buscando minimizar la suma de las distancias entre las parejas con tal que las parejas sean similares.
5. Realizamos un torneo entre padres e hijos para decidir cuales pasan de generación

Para programar este torneo se ha utilizado la función de numpy *argmax*[18], la cual es una función que nos devuelve para una matriz dada la posición del menor valor. Esto nos permite en una lista donde encontramos las combinaciones cual de ellas es la de menor valor y por lo tanto cual de ellas escogemos para que sobreviva la generación.

Finalmente quedaría explicar el método de parada definido en el algoritmo genético y este seria en primer lugar un bucle el cual para la ejecución al

alcanzar el número de iteraciones dadas por el usuario. El segundo de los métodos de parada del algoritmo es un contador el cual usa dos iteraciones. La primera esta encargada de almacenar cual es el mejor fitness obtenido, partiendo de cero, tras la primera iteración almacena el valor y si en algún momento es superado lo almacena. La segunda variable almacena cuantas veces seguidas ha aparecido el valor de la anterior variable, parando la ejecución en el caso de llegar a 10 ejecuciones.

Con esto termina la explicación de los diferentes métodos que son necesarios para hacer funcionar un algoritmo genético y pasaríamos a mostrar los resultados obtenidos a lo largo de la experimentación.

5.3 Experimentación

El primero de los pasos realizados en la experimentación ha sido el intento de búsqueda de la configuración optima del reactor. Para ello se ha realizado un bucle donde se ejecutará 10 veces consecutivas el algoritmo genético o hasta que encuentre cinco veces seguidas el mismo fitness considerando este como el fitness óptimo. Para cada una de las iteraciones generamos una población aleatoria de 1000 individuos y para cada una de las iteraciones, el número de iteraciones configurado para el algoritmo genético es de 100, como estrategia de selección de padres se ha escogido la selección por torneo con un número de padres igual a 50. La probabilidad de que haya cruce y de que haya mutación se han configurado a 0.8 y 0.2 respectivamente, mientras que los individuos en cada torneo a competir son ocho. Finalmente, el *crowding determinista* ha sido activado para mantener la diversidad de la población en cada iteración y se ha utilizado un método de reemplazamiento elitista. Con estos parámetros el algoritmo genético nos devuelve un individuo con un fitness igual a 0.9999, lo que es prácticamente nuestro valor objetivo 1. En el grafico 11 podemos observar la evolución del fitness en cada una de las iteraciones, viendo el valor máximo obtenido de fitness, el mínimo valor del *finess* en cada una de las iteraciones, que como podemos observar es cero debido a aquellos valores que sobrepasan el límite de uno en la k-efectiva. Finalmente, tenemos el valor del fitness medio en rojo, que como podemos observar va incrementando a medida que las iteraciones pasan y la zona de búsqueda del algoritmo se va acotando en zonas con fitness más prometedor.

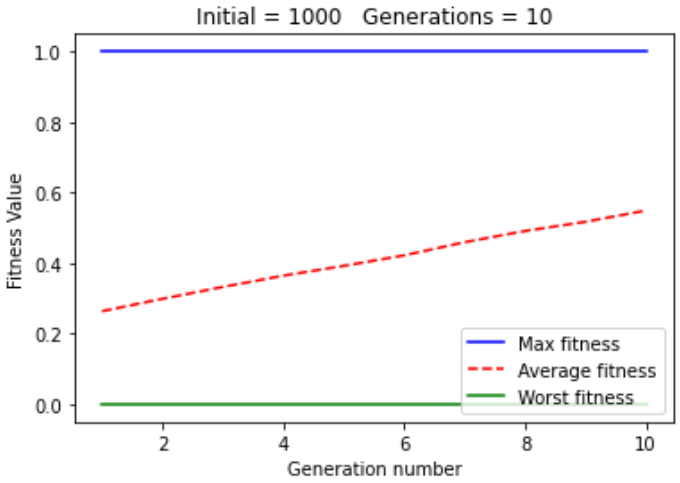


Fig11. Evolución del genético

Para la experimentación vamos a ir probando los diferentes métodos desarrollados para cada una de las fases del algoritmo genético. Para cada uno de estos se comprobarán sus resultados dependiendo los valores que se le otorguen al parámetros.

Para comprobar cómo se distribuyen los resultados de fitness obtenidos de cada uno de los experimentos realizados anteriormente, vamos a dibujar un diagrama de caja y bigotes por resultados de cada experimentación y comprobar si existen resultados extremos o atípicos que puedan afectar a los resultados obtenidos de las pruebas de significancia. Luego comprobaremos la normalidad de los datos para llevar a cabo un test t de muestras dependientes o pareadas, y de esta forma indicarnos si verdaderamente existen diferencias en cuanto a las medias y por tanto valores de fitness de cada configuración probada para asegurarnos de que verdaderamente existe un efecto en el algoritmo genético.

La idea general será copiar los resultados devueltos antes por cada experimentación, aquellos con los mejores fitness (en la última iteración) en cada repetición y comparar sus medias mediante el test de significancia mencionado. Cabe destacar que como en cada repetición se ha utilizado la misma población inicial para todas las configuraciones, los valores de fitness en cada una sí que están relacionados, existiendo cierta dependencia. Es por ello por lo que pensamos que el test t de muestras pareadas es el mejor para sacar conclusiones sobre las experimentaciones.

5.3.1 Análisis del número de individuos enfrentados por torneo

En este primer paso de la experimentación se ha estudiado cual es el número de individuos óptimos a enfrentar en cada torneo si queremos que nuestros resultados sean lo mejores posibles. Para ello se han analizado cuatro valores

dos, cuatro, ocho y 16 individuos a competir en cada torneo. Esto se ha realizado en un bucle de dos iteraciones donde se han generado cuatro algoritmos para cada uno de los valores, los cuales han mantenido constantes el restos de los valores, siendo 200 el número máximo de iteraciones, 100 el número de padres generados por cada una de las iteraciones y la probabilidad de cruce y la probabilidad de mutación han sido establecidas a 0.8 y 0.2 respectivamente. En la imagen 12 podemos observar los resultados proporcionados tras la ejecución de los algoritmos. En primer lugar, observamos que los cuatro alcanzan un fitness bastante cercano a uno, pero no a todos les cuesta el mismo número de iteraciones. Podemos ver como cuando el número esta en ocho la convergencia se consigue en promedio a 13.5 iteraciones mientras que con un numero de k igual a cuatro esta se alcanza en las 21 iteraciones, mientras que para una k igual a dos o 16 el promedio es igual, 18.5. También podemos ver como es la k igual a cuatro la que consigue un mejor ajuste mientras que con una k igual a ocho usamos menos generaciones, pero el resultado también es peor. Pero a la hora de realizar el test de la t para comprobar diferencias significativas entre los valores observamos que la única relación que tiene un p valor por debajo de 0.05 y por lo tanto podemos decir que la diferencia es significativa, ya que rechazamos la hipótesis nula, es la relación entre k igual a 8 y k igual a 16 cuyo p valor es igual a 0.02. Podemos decir por consiguiente que podemos utilizar cualquiera de los valores ya que no encontramos diferencias significativas entre los resultados otorgados por los diferentes casos, aunque si hay diferencia entre usar una k igual a ocho e igual a 16, no existe diferencia en usar cualquiera de los otros valores propuestos para el análisis. También encontramos en el gráfico de caja y bigote que no hay ningún dato anómalo.

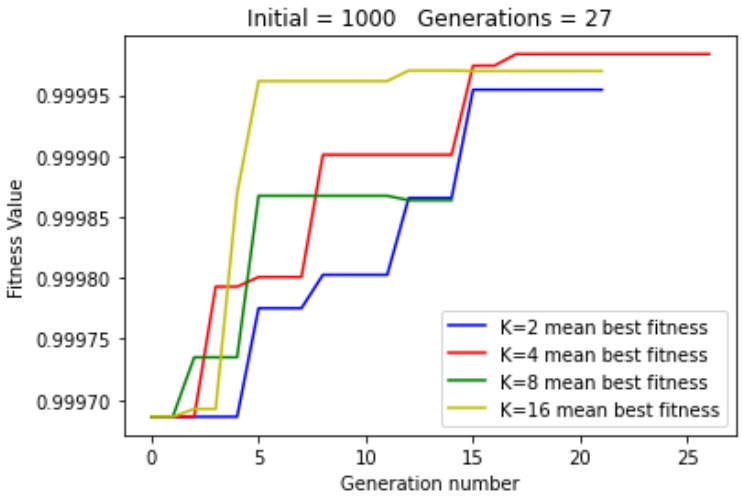


Fig12. Comparación individuos enfrentados en un torneo

5.3.2 Valoración del valor de m en la selección de padres best n of m.

Recordemos que este método se basaba en la selección de los n mejores individuos de una población m escogida aleatoriamente de la población total. Esta parte de la experimentación tiene como objetivo buscar cual es el valor de m que reporta un mejor valor de fitness. Para ello se han ejecutado cuatro algoritmos uno con la m igual a 100 lo que sería el 10% de la población total, otro con 200 lo que equivale al 20%, con un 500 es decir un 50% y finalmente un 100%. Los resultados tras dos iteraciones del problema nos demuestran como en el anterior de los casos que todos consiguen gran aproximación al valor objetivo, una k-efectiva de uno. El caso que consigue la convergencia en menor tiempo es con una m 500, mientras que el más tardío sería la m igual a 1000 que tardaría 20.5 iteraciones en alcanzar el punto de convergencia. Podemos observar en el gráfico 13 como es el m igual a cien el que consigue el mejor de los fitness teniendo muy próximo a la m igual a mil. También podemos observar que la m igual a cien alcanza la convergencia en 15.5 generaciones por lo que es la segunda más próxima. El posterior análisis a través de los gráficos de caja y bigote nos demuestra que no hay valores atípicos en ninguno de los casos. Finalmente, siguiendo el protocolo establecido se ha realizado el análisis de t de muestras pareadas. En este análisis podemos observar que en todos los resultados la p-valor está situada por encima de los 0.05 por lo que se aceptaría la hipótesis nula y podemos afirmar que ninguno de los resultados es significativamente diferente al resto y por ello cualquiera de los valores que otorguemos a la *m* nos reportará unos resultados finales estadísticamente similares.

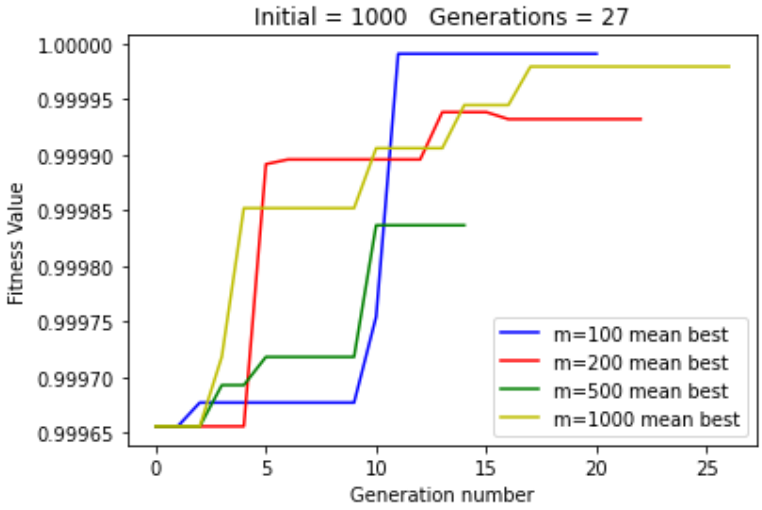


Fig13. Evolución de la m en best n of m

5.3.3 Valoración del parámetro s en la selección de padres con ranking lineal

Como se explicó en el apartado de selección de padres, la s es un hiperparámetro a ajustar en la selección con ranking lineal. Este hiperparámetro se debe ajustar con un valor entre dos y uno, cuanto mas cerca del uno se encuentre este parámetro mas parecida a una selección uniforme será y a más próximo al dos el individuo con mejor fitness acumulará una mayor probabilidad de ser seleccionado. Para esta etapa de la experimentación se ejecutarán cinco algoritmos genéticos. El primero de ellos tendrá una s configurada a uno, el siguiente a 1.25, otro con 1.5, 1.75 y finalmente con una s igual a dos.

Podemos observar en la experimentación como los casos con s igual a 1.5 y 1.75 son aquellos que tienen una convergencia mas rápida mientras que cuando la s es equivalente a uno es cuando mayor son las generaciones necesarias para alcanzar la convergencia. Podemos observar en la imagen X como todos los resultados tienen un fitness muy próximo con una diferencia de 0.0001 entre el menor y el mayor, siendo el que obtiene el mejor resultado aquel con un s equivalente a 1.25 y el que peor resultados devuelve sería aquel con un s de 1.75. También en el grafico se observa cómo hay una caída de todos los valores en la generación 10 y en la que se estancan todos los valores a excepción de la s a 1.25 que vuelve a aumentar. Una posible explicación es que, al no dar mas probabilidad a los casos de mayor fitness, lo que pasa al aumenta la s, podamos explorar zonas. Al igual que en el resto de los casos se ha observado el grafico de caja y bigote en busca de datos anómalos, pero no se observa ningún resultado anómalo. Con esto en mente el siguiente de los pasos ha sido analizar la diferencia significativa entre los diferentes casos usando la prueba de t. Al igual que pasaba en el anterior caso ninguna de las pruebas realizadas devuelve un p valor inferior a 0.05 por lo que aceptamos la hipótesis nula y podemos afirmar que no hay una diferencia estadística entre los diferentes valores propuestos en el análisis y por consiguiente cualquiera de los valores nos reportaría unos resultados estadísticamente parecidos.

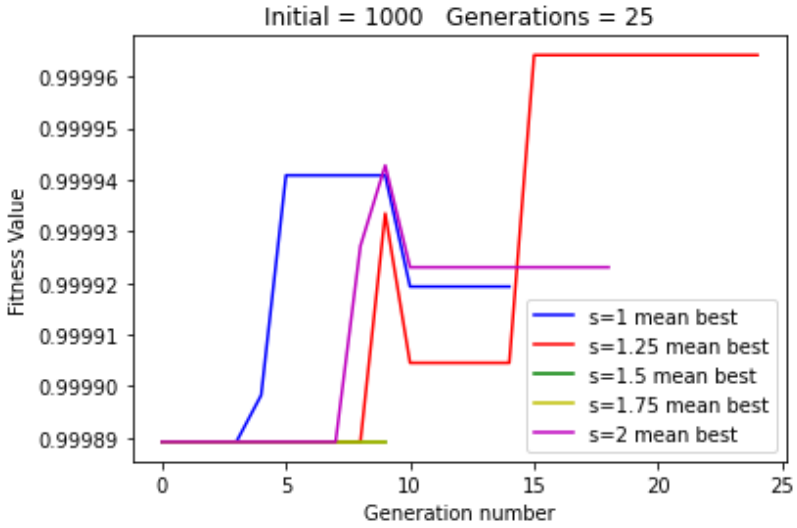


Fig14. Comparativa del valor de la s en la selección por ranking lineal

5.3.4 Comparación de las estrategias de selección de padres.

En este apartado vamos a realizar una comparación de los resultados ofrecido por cada una de las estrategias de selección de padres previamente explicadas usando en cada una de ellas el mejor valor obtenido en la experimentación. Esta experimentación es bastante sencilla pues solo tenemos que ejecutar consecutivamente los tres algoritmos. En esta situación las tres estrategias encuentran la convergencia en un número de iteraciones próximo, siendo 14 el menor y 15 el mayor. Se observa a priori que la mejor estrategia de cruce es la selección por torneo, seguido por la selección por ranking lineal. Como venia pasando en los anteriores pasos de la experimentación encontramos que no hay ningún valor anómalo en los resultados y posteriormente hemos realizado el análisis en busca de diferencias significativas. Los resultados de este test nos demuestran que los resultados son semejantes, ya que todas las combinaciones nos devuelven un valor de p mayor al valor de nuestro intervalo de confianza por lo que no hay diferencias significativas entre los resultados de las diferentes estrategias. Cabe destacar que la mayor diferencia de los resultados se encuentra entre aplicar la estrategia *best n of m* y el *lineal ranking*.

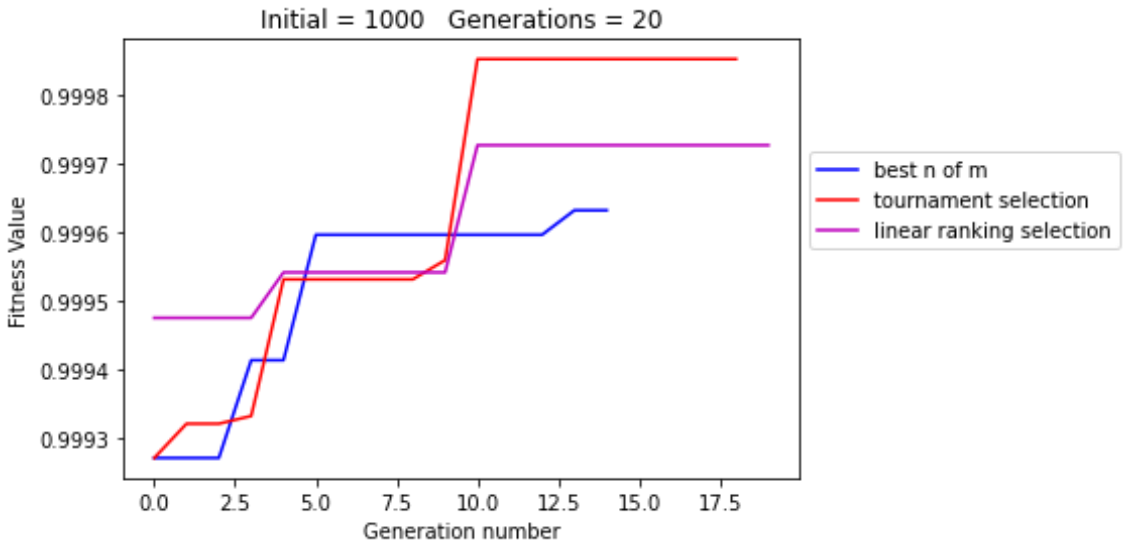


Fig15. Comparativa estrategias de selección de padres

5.3.5 Comparación de las probabilidades de mutación

El siguiente paso ha sido el análisis de la probabilidad de que una mutación ocurra, que recordemos es un hiperparámetro que se establece y que va a determinar posteriormente si en un hijo se produce o no la mutación. Para este análisis hemos probado cuatro casos, uno con una probabilidad de 0.2, otro con una probabilidad de 0.4, el siguiente sería de 0.6 y finalmente el último tendría una probabilidad de ocurrir de 0.8. En la figura 16 podemos observar como un probabilidad igual a 0.6 es la que mejores resultados devuelve, pero a la vez es esta la que mas generaciones necesita, mientras que la probabilidad equivalente a 0.8 es la que menos iteraciones tiene y es la que peor resultado nos devuelve mientras que las probabilidades a 0.2 y 0.4 se encuentran en medio. Finalmente, tras analizar esta grafica hemos realizado el usual test de similitud podemos observar que todos los valores de las p están por encima del 0.05 por lo que podemos decir que no hay diferencias significativas entre las probabilidades y el uso de cualquiera de ellas nos reportará un valor similar.

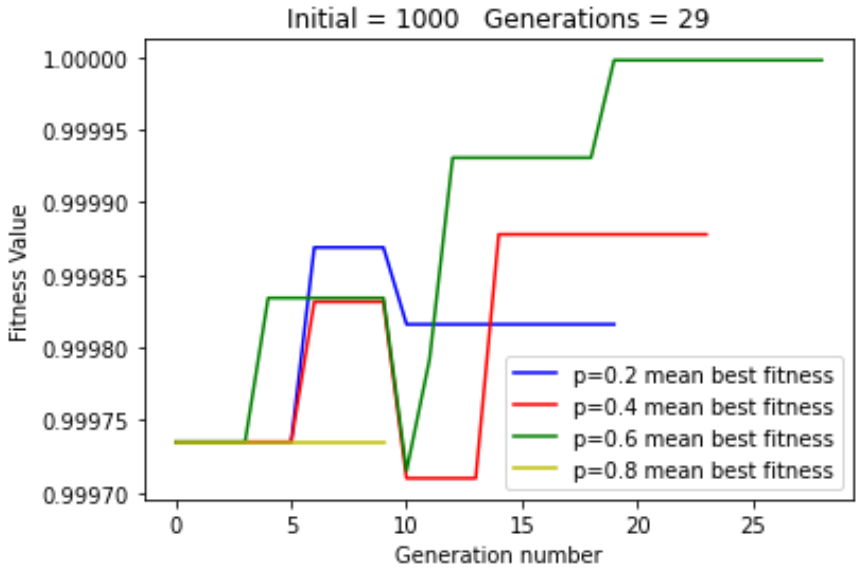


Fig16. Comparativa de las probabilidades de mutación

5.3.6 Comparación de las probabilidades de cruce

El siguiente de los pasos en la experimentación ha sido el de la probabilidad de cruce. Como ya se ha explicado anteriormente la probabilidad de cruce es un hiperparámetro el cual decide si un cruce sucede o no, esto se hace otorgando un valor aleatorio entre cero y uno y observando si este se encuentra por debajo del valor de probabilidad estipulado. Para este análisis, como en el anterior, se ha decidido usar cuatro valores, un primero equivalente a 0.2, otro igual a 0.4, otro a 0.6 y finalmente uno a 0.8. Podemos observar que las dos situaciones donde mejores resultados nos ofrece el algoritmo genético es cuando la probabilidad de cruce es 0.2 o 0.8, pero al realizar la prueba de la t entre todas las probabilidades descubrimos que no hay ninguna diferencia significativa en los resultados, ya que todas las p-valor son mayores a 0.05 y por lo tanto aprobamos la hipótesis nula. Por consiguiente, queda demostrado que el uso de cualquiera de las probabilidades propuestas no afecta a los resultados finales del problema.

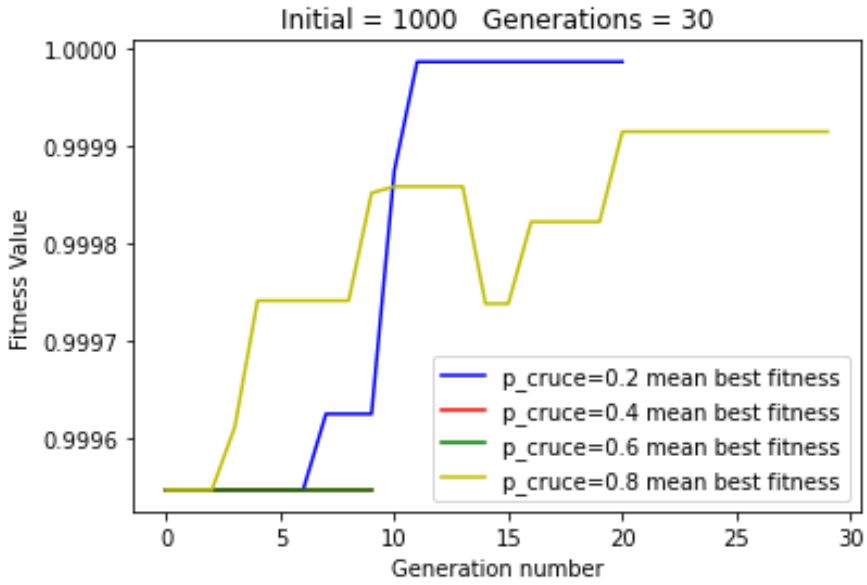


Fig17. Comparativas probabilidades de cruce

5.3.7 Análisis del efecto del *deterministic crowding*

Como se ha explicado anteriormente el *deterministic crowding* es un método de mantenimiento de la diversidad dentro de la población. El objetivo ahora es comprobar cual es el efecto del mismo en los resultados finales. Para ello hemo ejecutado con la misma población dos algoritmos genéticos con los mismos parámetros a excepción del *crowding* el cual en uno esta activado y en otro no. Los resultados nos muestran que ambos obtienen unos resultado muy parecidos. Podemos ver como en la grafica cuando no usa *deterministic crowding* sigue unos valores continuos y como aparentemente los resultados mejoran cuando se aplica el *deterministic crowding*. Podemos observar como este necesita mas generaciones para alcanzar la convergencia que el algoritmo genético cuando no las usa. Finalmente, los resultados del test de similitud nos muestra que da igual si se aplica o no el *deterministic crowding*, ya que los resultados no son estadísticamente diferentes, debido a que el p-valor de la prueba entre los dos casos es 0.1 y por lo tanto es mayor a 0.05.

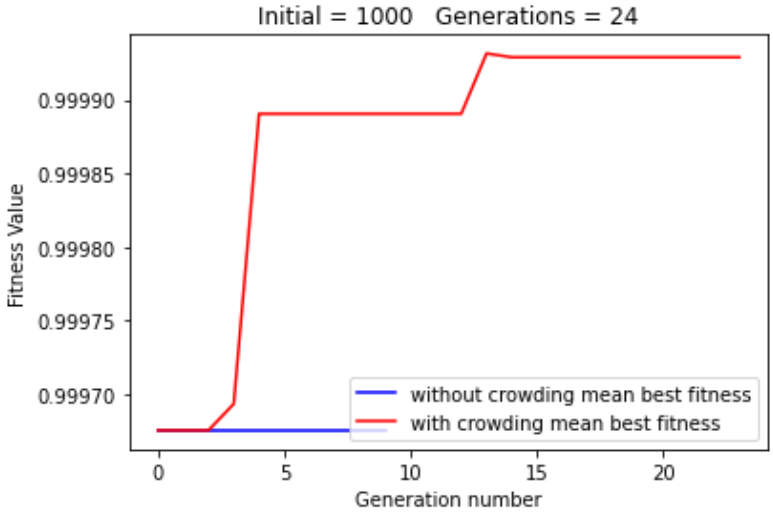


Fig18. Diferencia de resultados con deterministic crowding y sin él

5.3.8 Análisis del número de padres

Finalmente se ha realizado un análisis del número de padres que se escogen en los métodos de selección de padres, con el fin de optimizar nuestros resultados. Los resultados nos muestran como todos los valores dados a este parámetro nos reportan un gran ajuste, vemos como es aun así con el numero de padres a 50 cuando obtenemos un mejor resultado, pero es con parámetro a cien cuando alcanzamos una convergencia en un numero menor de generaciones. Tras esto analizamos los resultados del test de similitud y observamos que no hay una diferencia significativa entre los diferentes valores, pues como ha pasado en el resto de los casos se observa que los p valores son superiores a 0.05 y por lo tanto rechazamos la hipótesis nula. Con estos resultados concluimos que cualquiera de las elecciones en cuanto al numero de padres se refiere nos reporta unos resultados similares estadísticamente.

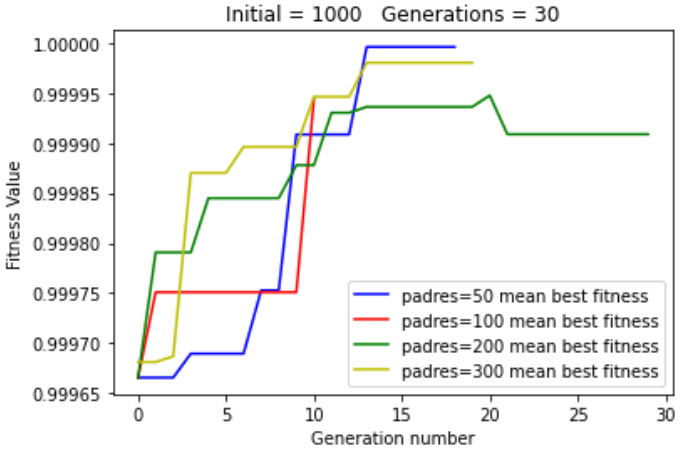


Fig19. Comparativa dependiendo del número de padres escogidos

6. Conclusiones

La tarea abordada en este trabajo ha sido de gran complejidad, la cual consiste en la optimización de la k-efectiva de un reactor nuclear de tipo PWR. Se han cumplido todos los pasos planteados a lo largo del TFG, se han creado dos redes convolucionales, las cuales nos han permitido calcular las variables necesarias y se ha desarrollado un algoritmo genético que nos permite optimizar la k-efectiva tal y como nos habíamos propuesto en un principio. Sin embargo, sería necesario un mayor desarrollo del algoritmo genético, tanto en las estrategias de mutación y cruce desarrolladas, las cuales podrían ser optimizadas o mejoradas por ejemplo aumentando el número de celdas mutadas o en el caso del cruce cambiando las filas por áreas, ámbitos que podrían se estudiados en futuros trabajos, se podrían también tener más en consideración esos aspectos geométricos de los que se ha hablado para que no se produzcan repeticiones en los datos. También el coste computacional de este no nos ha permitido desarrollar todas las pruebas posibles, por lo que con una mayor capacidad computacional u optimizando el código del algoritmo, podríamos realizar mas pruebas y con ello posiblemente mejorar los resultados finales.

Las dificultades de este TFG han sido diversas, en primer lugar, la dificultad del tema. La cantidad de diferentes variables físicas y la familiarización con los programas que las calculan como PARCS, ha requerido muchas horas de lectura sobre la materia. Otro problema al que nos hemos enfrentado, relacionados con PARCS es que la parte de cálculo hidráulico es una implementación novedosa y por ello ha habido que ir ajustando parámetros que han provocado que tengamos que regenerar los datos hasta en 5 ocasiones. Otro problema de la generación de datos es la gran cantidad de disco que ocupaban los datos, para lo cual fue necesario usar un servidor donde almacenarlos, ya que el espacio de disco necesario era superior al de mi ordenador.

A la hora de desarrollar las redes neuronales el principal problema planteado fue el del método de activación de la campa de salida, el cual en un principio fue la *softmax*, lo que nos resultaba en unos errores sin sentido, y esto se debe a que la función *softmax* es una función utilizada para ejercicios de clasificación, por lo que no tenía sentido aplicarlo en este caso. Finalmente, se opto por usar una función de activación lineal para la capa de salida.

La dificultades planteadas a la hora de desarrollar el algoritmo genético han sido varias. El primero de ellos ha sido el problema de la representación de los cromosomas, los cuales en primer lugar se pretendió aplicar la representación propuesta en el *paper Novel genetic algorithm for loading pattern optimization based on core physics heuristics*[2], pero la representación fue descartada y se aplicó una representación en forma de vector de la matriz del

patrón de carga, ya que para calcular en la función de fitness necesitaríamos reconstruir la matriz original y esta forma lo facilita, y porque el problema plantado en ese *paper* los datos de entrada son mas simples, por lo que no es exactamente aplicable a nuestro problema y los problemas que nuestra representación puede tener no son solucionados por la otra, si no que siguen presentes. Problemas como facilitar la mutación haciendo innecesario tener que observar las posiciones próximas constantemente. Otro de los problemas que nos hemos encontrado es el tema del coste computacional del algoritmo genético planteado y como se ha mencionado previamente esto quedaría atado a futuras mejoras del algoritmo. La función de cruce de padres tiene un error y es que no comprueba que la matriz resultante cumpla que todos los combustibles no tengan un combustible igual adyacente. Este es un error que se podría haber solucionado mirando todos los valores cambiados en un bucle y modificar aquellos que incumplen la norma. El problema de esta solución es que genera un coste computacional alto y por ello no se ha aplicado. También se podría haber modificado la forma de representación o que al ocurrir esto devolviera cero el fitness, aunque esto mantiene el problema de coste computacional, otra solución es cambiar el sistema de cruce, pero es un error que no se ha podido subsanar y queda abierto a posibles investigaciones futuras, ya sea con una forma que revise en un conste inferior o de otra manera.

El desarrollo de este TFG ha permitido expandir mi conocimiento en diseño y utilización de redes neuronales, sobre todo en su programación usando Kera para lo cual no tenía tanta formación. A través de buscar información, libros de consulta, lectura de artículos… mi conocimiento en este campo también ha aumentado considerablemente. El apartado de variables relacionadas con el ámbito de la energía nuclear como el significado de la k-efectiva o de la potencia máxima radial han sido conocimientos que a través de la realización de este trabajo he ido asimilando. También el uso de PARCS, un programa totalmente ajeno a mi formación y el haber usado un servidor ha hecho que mi control de herramientas de Linux y el manejo de la consola de comando haya mejorado considerablemente.

Finalmente, la aplicación del algoritmo genético en un mundo diferente al visto a lo largo de la carrera me ha permitido ganar soltura en el mundo de los algoritmos genéticos y en su aplicación práctica. Conceptos como los explicados en el *paper* anteriormente mencionado (*Novel genetic algorithm for loading pattern optimization based on core physics heuristics*) como puede ser el uso de la geometría en el diseño de las estrategias de cruce, aunque no hayan sido implementadas completamente. En definitiva, el TFG ha ayudado a potenciar los conocimientos obtenidos a lo largo de la carrera y a adquirir conocimientos nuevos procedentes del mundo de aplicación de este.

La energía es un bien que podemos observar en el mundo actual como cada vez es más importante y la aplicación de este trabajo nos demuestra como es posible

aumentar la eficacia de nuestros métodos de obtención de esta en este caso en el ámbito nuclear. Somos consientes de que es un trabajo de gran envergadura y por eso este TFG es simplemente una toma de contacto con este problema y deja las puertas abiertas a mejoras y a nuevos puntos de vista. Pero se ha demostrado como la aplicación de la ciencia de datos y de los métodos algorítmicos de optimización y las redes neuronales, pueden facilitar nuestra manera de obtener la energía y darnos una mayor eficiencia temporal a la hora del cálculo y de eficacia energética a la hora de su extracción.

Bibliografía

1. World Nuclear Association. (2022, Agosto), Nuclear Power in the world today. https://world-nuclear.org/information-library/current-and-future-generation/nuclear-power-in-the-world-today.aspx#:~:text=Around%2010%25%20of%20the%20world's,from%202657%20TWh%20in%202019
2. Rabie Abu Saleem, Majdi I. Radaideh , Tomasz Kozlowski. (2020, mayo 13). Application of deep neural networks for high-dimensional large BWR core neutronics
3. E. Israeli, E. Gilad, (2018, abril 7). Novel genetic algorithm for loading pattern optimization based on core physics heuristics
4. Eduardo Fernandes Faria, Claubia Pereira, (2002, agosto 30). Nuclear fuel loading pattern optimisation using a neural network
5. Juan Jose Ortiza, Ignacio Requena (2003, noviembre 7). Using a multi-state recurrent neural network to optimize loading patterns in BWRs
6. Virginie Solans, Dimitri Rochman, Christian Brazell, Alexander Vasiliev, Hakim Ferroukhi, Andreas Pautz, (2021, julio 4). Optimisation of used nuclear fuel canister loafing using a neural network and genetic algorithm
7. Ahmad Pirouzmand, Fatemeh Mohammadhasani (2016). PARCS code multi-group neutron diffusion constants generation using Monte Carlo method, Progress in Nuclear Energy, Volume 86, Pages 71-79, https://doi.org/10.1016/j.pnucene.2015.10.005.
8. Nuclear chain reaction. Wikipedia. https://en.wikipedia.org/wiki/Nuclear_chain_reaction#Effective_neutron_multiplication_factor
9. In Ho Bae, Man Gyun Na, Yoon Joon Lee, Goon Cherl Park. Calculation of the power peaking factor in a nuclear reactor using support vector regression models. Annals of Nuclear Energy, Volume 35, Issue 12, 2008, Pages 2200-2205, https://doi.org/10.1016/j.anucene.2008.09.004
10. USNRC, Computer codes, (2021, febrero 10). https://www.nrc.gov/about-nrc/regulatory/research/safetycodes.html
11. Stat Exchange, (2016, mayo 7). https://stats.stackexchange.com/questions/126238/what-are-the-advantages-of-relu-over-sigmoid-function-in-deep-neural-networks
12. Sanket Doshi, (2019, enero 13) Various Optimization Algorithms for Training Neural Network. Towards data science https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6
13. Algoritmos genéticos. Wikipedia. https://en.wikipedia.org/wiki/Genetic_algorithm

14. Tournament selection. Wikipedia. https://en.wikipedia.org/wiki/Tournament_selection
15. Mutación. Wikipedia. https://en.wikipedia.org/wiki/Mutation_(genetic_algorithm)
16. Whitley, Darrell. (2000). The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best. 89.
17. Sato, J. and Akashi, T. (2018), Deterministic crowding introducing the distribution of population for template matching. IEEJ Trans Elec Electron Eng, 13: 480-488. https://doi.org/10.1002/tee.22591
18. Numpy. https://numpy.org/doc/stable/reference/generated/numpy.argmax.html
19. Foro Nuclear. https://www.foronuclear.org/descubre-la-energia-nuclear/preguntas-y-respuestas/sobre-energia-nuclear-y-medio-ambiente/como-influye-la-energia-nuclear-en-el-medio-ambiente/
20. J. P. McBride and R. E. Moore and J. P. Witherspoon and R. E. Blanco (1978). Radiological Impact of Airborne Effluents of Coal and Nuclear Plants, Science, 202, 4372: 1045-1050, https://www.science.org/doi/abs/10.1126/science.202.4372.1045
21. Mike Mueller, (2021, marzo 24). Nuclear Power is the Most Reliable Energy Source and It's Not Even Close. Office of nuclear energy. https://www.energy.gov/ne/articles/nuclear-power-most-reliable-energy-source-and-its-not-even-close#:~:text=Nuclear%20Has%20The%20Highest%20Capacity%20Factor&text=This%20basically%20means%20nuclear%20power,than%20wind%20and%20solar%20plants.
22. Hannah Ritchie and Max Roser, (2020), Energy. https://ourworldindata.org/fossil-fuels
23. (2021, septiembre). Economics of Nuclear Power. World Nuclear Association. https://world-nuclear.org/information-library/economic-aspects/economics-of-nuclear-power.aspx

Apéndice 1: Objetivos y metas de desarrollo sostenible

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

Tabla 3. Objetivos y metas de desarrollo sostenible

La energía nuclear junto con las energías renovables es de las energías que menos emisión de CO2 y gases de efecto invernadero tiene a la atmósfera[19]. Esto es un hecho que nos demuestra que a pesar de los residuos que deja es una energía eficaz contra el efecto invernadero y aunque sus residuos contaminen en el proceso de extracción de energía las emisiones son mínimas. También cabe destacar un estudio de la revista *Science,* donde se demuestra que centrales como las térmicas de carbón producen una ceniza hasta 100 veces más contaminantes que las producidas por los reactores nucleares y no solo eso, sino que también se demuestra las personas que viven entorno a centrales de carbón presentan una radiación ingerida similar o superior a la de las centrales nucleares[20]. Es por esto por lo que nuestro trabajo al ser un intento de promover las energías nucleares y optimizarlas de manera que saquemos el mayor beneficio posible de ellas en primer lugar tiene un impacto claro en el bienestar y en la salud de la población, pues una menor emisión de gases de efecto invernadero junto con una gestión responsable de los desechos nucleares puede traer un impacto al clima mundial positivo y con ello a la salud y bienestar de la población. Y obviamente esto puede beneficiar a ecosistemas tantos terrestres como marítimos pues es una reducción de la contaminación global. Otro punto a favor de la energía nuclear es la cantidad de energía que produce una sola central respecto a otros tipos de energía como demuestra la oficina de energía nuclear de estados unidos[21] o podemos observar como la cantidad de energía generada a través de centrales nucleares es superior a la producida por energías renovables o fosiles[22], esto hace que la cantidad de centrales necesarias sea menor para producir la misma energía que con parques eólicos o centrales fósiles. Este echo demuestra que la contaminación visual es menor y por lo tanto beneficia al medio ambiente.

El coste de producción de la energía nuclear en la actualidad es menor que el coste de producción de otras centrales como las de carbón[23], si a esto le añadimos una propuesta innovativa como es la aplicación de la ciencia de datos en la optimización del proceso podemos optimizar el proceso optimizando los beneficios obtenidos de la energía y a la vez modernizar la industria.

Apéndice 2: Código

En el siguiente apéndice se va a realizar un resumen de los diferentes códigos utilizados y un resumen de su utlizacion:

- Reader.py:
  - Objetivo: generar las x nuevas matrices a partir de un archivo de entrada de PARCS ss_pwr_paths y paths_s1c1. Genera los archivos y los organiza en carpetas.
  - Entrada: ss_pwr_paths y el paths_s1c1.
- Generate_new.py:
  - Objetivo: modificar los archivos de PARCS, manteniendo las matrices generadas por el reader.py, y reorganizarlos en nuevos directorios.
  - Entrada: archivos con la matriz modificada por el reader.py y archivo con la modificación que se quiere aplicar a todos (por ejemplo, un valor cambiado a True)

En un principio sería suficiente el reader.py ya que, con él, si le das un ss_pwr_paths y un paths_s1c1, generará los archivos que se necesiten. El generate_new.py en caso de tener un nuevo archivo con nuevos parámetros modificados y querer las matrices ya generadas, este programa las transporta a un ejecutable con los nuevos valores.

- Out_info.py:
  - Objetivo: extraer la información para entrenar las redes neuronales: K-efectiva, potencia axial, radial y 3D.
  - Entrada: conjunto de archivos out.
- Rrnn.py y rrnn2.py:
  - Objetivo: de rrnn.py calcular la k-efectiva, de rrnn2.py calcular potencia axial, radial y 3D
  - Entrada: fichero de datos generado por out_info.py
- Genético.ipynb:
  - Objetivo: generación del algoritmo genético.
  - Entrada: fichero de datos generado por out_info.py y las redes neuronales guardadas en rrnn.py y rrnn2.py
  - Se puede acceder a este código en https://colab.research.google.com/drive/1uW0X10De6E4ZgEB9O591mncWNKsQzwMJ?usp=sharing este enlace.

El resto de los códigos pueden ser accedidos en el enlace https://drive.google.com/drive/folders/1L1ygHa_agv4qjSngxCchkoHJ6N8pcY6x?usp=sharing