UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE LA TELECOMUNICACIÓN

Trabajo de Fin de Grado

Algoritmos de Soft-Computing en el aprovechamiento de la energía mareomotriz por olas.

Autor: Manuel Montoya Catalá

Director: Sancho Salcedo Sanz

TRIBUNAL:

Presidente: .....

Vocal 1º: .....

Vocal 2º: .....

CALIFICACIÓN: .....

FECHA: .....

Agradecimientos

Quisiera dar las gracias primeramente a mi tutor, Sancho Salcedo, y a mi compañero, Álvaro Pastor, por su tiempo y dedicación por hacer posible este proyecto.
A mi familia por apoyarme y creer en mí, a mi padre Tato, por su ejemplo, a mi madre Natalia, por su cariño y a mis hermanos Héctor y Niki.
A mis amigos por soportarme, que no es tarea fácil, y en especial a Andrés, mi compañero de armas, a Hidalgo y a Gamino.
Y a todas aquellas personas que han formado parte de mi vida a lo largo de estos últimos años, a todas ellas, muchas gracias.

Índice de contenidos

Índice de Figuras

Índice de Cuadros

Acrónimos

AA	Algoritmo de Aprendizaje
AI	Inteligencia Artificial (Artificial Intelligence)
AIS	Sistema Inmune Artificial (Artificial Inmune Sistem)
ANN	Red Neuronal Artificial (Artificial Neural Network)
BP	Back-Propagation
EA	Algoritmo Evolutivo (Evolution Algorithm)
ELM	Extreme Learning Machine
ES	Búsqueda Exhaustiva (Exhaustive Search)
FFNN	Feed-Forward Neural Network
FP	Propiedades de las Características (Feature Properties)
FS	Selección de Características (Feature Selection)
GA	Algoritmo Genético (Genetic Algotihm)
MSE	Error Cuadrático Medio (Mean Square Error)
NDBC	National Data Bouy Center
RMSE	Root Mean Square Error
RNA	Red Neuronal Artificial
SA	Temple Simulado (Simulated Anneling)
SLFN	Single-hidden Layer Feedforward Neural Networks
SBE	Sequential Backward Elimination
SFS	Sequential Forward Selection
SS	Espacio de Soluciones (Search Space)
TA	Algoritmos de Umbral (Threshold Algorithm)
TEe	Error de Testing
TRe	Error de Training
VS	Vector de Selección
VTe	Vector de Testing
VTr	Vector de Training

Resumen

El uso de fuentes de energía renovable como la energía solar, eólica o hidráulica es remonta a la antigüedad, estas han sido utilizadas durante siglos antes de nuestro tiempo, por ejemplo, los molinos hacían uso de la energía eólica para moler el trigo. Sus aplicaciones cesaron durante la "Revolución Industrial", momento en el cual, dado el bajo precio del petróleo, fueron abandonadas. En los últimos años, debido al incremento del precio de los fueles fósiles y los problemas medioambientales creados por estos, se está volviendo al uso de energías renovables.

Las energías renovables son virtualmente inagotables, limpias y pueden utilizarse de una manera descentralizada. Uno de los principales problemas de este tipo de energía es su aleatoriedad y variabilidad de producción ya que dependen de variables meteorológicas difíciles de predecir. Entre ellas la energía mareomotriz posee la ventaja de ser relativamente concentrada, constante y predecible. A diferencia de otras energías como la solar, que solo puede utilizarse durante el día, la energía mareomotriz es aprovechable durante todo el día, por contrapartida, esta energía solo se encuentra disponible en el mar.

El estudio y predicción de la energía de olas disponible en un punto es esencial para su correcto aprovechamiento, viabilidad y diseño de sistemas capaces de transformar esa energía en energía útil. En este proyecto, se propone el uso de técnicas de Soft-Computing para la predicción de las características del oleaje en un punto, a partir de las características del oleaje de otros tantos, para un mismo instante de tiempo. Se utilizan redes neuronales para predecir la altura del oleaje a partir de los demás parámetros y técnicas metaheurísticas para encontrar el subconjunto de estos parámetros que propicia la mejor predicción.

Palabras clave: Energía mareomotriz, Redes neuronales, ELM, Metaheurística, Algoritmos Genéticos, Selección de características.

Abstract

The use of renewable energy sources, such as solar, wind and hydraulic energies, is very old, they have been used since many centuries before our time, by example, mills use wind energy to grind wheat. Its applications ceased during the "Industrial Revolution", at which time, due to the low price of petroleum, they were abandoned. During recent years, due to the increase in fossil fuel prices and the environmental problems caused by the use of conventional fuels, we are reverting back to renewable energy sources.

Renewable energies are virtually inexhaustible, clean and they can be used in a decentralized way. One mayor problem of this kind of energy is the randomness and variability of its production because of its dependence on meteorological variables, which are hard to predict. Among them, wave energy has the advantage of being relatively concentrated, constant and predictable. Unlike other renewal energies such as solar energy, which can only by used during the daylight, wave energy is usable throughout the whole day, but it's only available on the sea.

The study and prediction of the wave energy available in a certain area is essential for its proper viability, performance and design of systems able to transform that energy into usable energy.
In this project, Soft-Computing techniques are proposed for the prediction of wave characteristics in a sea area, using sea wave parameters from other places, in the same period of time. Neural networks are used to predict the height of the sea waves using other parameters as input; and metaheuristic search techniques are used to find the subset of these parameters that produces the best prediction.

Key-words: Wave energy, Neural networks, ELM, Metaheuristics, Genetic Algorithms, Feature Selection.

Capítulo 1
Introducción

1. INTRODUCCIÓN

La energía de las olas oceánicas es enorme, incluso la fracción de esta energía que es potencialmente explotable es muy grande comparada con el consumo actual de electricidad en el mundo. Se estima que la energía mundial explotable es de 2TWh año y que las aguas europeas son capaces de cubrir más del 50% del consumo total de potencia en el continente. Aunque la energía oceánica presenta un gran potencial energético, no todas las tecnologías disponibles tienen la misma viabilidad para aprovecharlo, el principal obstáculo para su utilización es la fuerte inversión inicial que se requiere.

Es por ello que es tan importante ser capaz de estimar la energía disponible en un sitio para la máxima rentabilización de las infraestructuras. La energía de ola disponible posee una gran componente aleatoria debida a su dependencia de las condiciones meteorológicas. En este proyecto se propone utilizar técnicas de Soft-Computing para predecir las características del oleaje en un punto, a partir de las características en otros tantos puntos y así poder estimar la energía de ola disponible. Para ello se requiere de mediciones del oleaje, que son realizadas por boyas oceánicas. Estas boyas miden los desplazamientos que producen las olas y calculan el espectro de las mismas, normalmente cada hora. Las mediciones del espectro de las olas pueden ser descargadas de las página web de la National Data Bouy Center, http://www.ndbc.noaa.gov/. En este estudio se utilizarán 5 boyas del mar caribe:

- 42056
- 42057
- 42058
- 42059
- 42060

La posición geográfica de las boyas se encuentra señalada en el siguiente mapa obtenido a partir de una imagen de la NDBC.

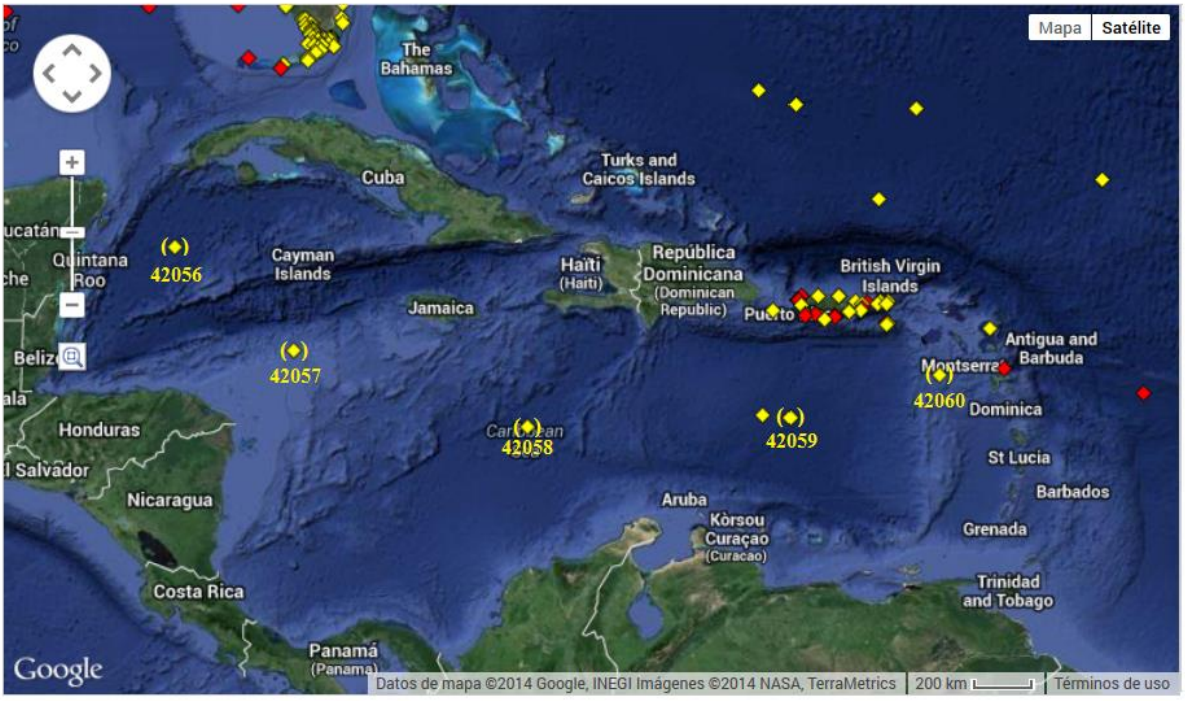


Fig. 1 Mapa con las boyas utilizadas en el estudio de este documento. Fuente [47].

Una ola oceánica está formada por la suma de muchas ondas senoidales puras. El espectro $S(f)$ de la ola nos da información de las ondas puras que la forman y es utilizado para estimar la energía de la misma. A partir del espectro de las olas descargado en la página web de la NDBC, se calcularán 15 parámetros del oleaje para cada una de las boyas. El primero de estos parámetros es la altura efectiva de la ola, siendo esta la magnitud que se pretende predecir, pues la energía de ola disponible está muy relacionada con ella. Durante este proyecto se realizará un estudio de la **boya 42058**, para la cual se intentará predecir, para un mismo instante de tiempo, la altura del oleaje en esa boya a partir de:

- Los 14 parámetros restantes de la boya.
- Los 60 parámetros correspondientes a las otras 4 boyas.

Si denominamos al conjunto de $'n'$ parámetros que se usarán para predecir la altura como el vector:

$$\bar{X} = [x_1, x_2, \dots, x_N]$$

Donde $x_i \in \bar{X}$ es un parámetro del oleaje y denominamos a la altura de la ola como H. Se hace necesario un sistema $f(\cdot)$ que sea capaz de hallar las relaciones entre los $'n'$ parámetros y la altura de tal manera que:

$$H = f(\bar{X})$$

Existen diversos sistemas que pueden realizar la función $f(\cdot)$, en este documento se ha utilizado una Red Neuronal Artificial entrenada con el algoritmo Extreme Learning Machine. Una RNA es un sistema que intenta emular el funcionamiento del cerebro humano, siendo capaz de relacionar un conjunto de entradas, con una o más salidas. Las RNA requieren un proceso de aprendizaje en el cual se le introducen un conjunto de vectores de entrada $\bar{X}$ y se le indica la salida deseada H. La RNA se autoajustará internamente para hallar las dependencias entre ellos y presumiblemente será capaz de calcular la salida H de nuevos vectores de entrada $\bar{X}'$ que no se le habían presentado antes.

Como es de esperar, el sistema no es perfecto, ante un nuevo vector de entrada $\bar{X}'$, lo más probable es que no consiga calcular exactamente el valor de H, sino que más bien, nos devuelve una estimación del mismo, denominado como $\widehat{H}$. Donde el error de la predicción suele darse como el cuadrado de la diferencia entre el estimador y el valor estimado:

$$Error = \left(H - \widehat{H}\right)^2$$

Para comprobar la capacidad de predicción de un determinado subconjunto de parámetro $S \in X$, lo que se hace es entrenar el sistema con un conjunto de vectores de N vectores de entrada $\{\bar{X}_i, H_i\}\ i = 0,1..N$ llamados vectotes de entrenamiento y después introducir un conjunto de vectores de prueba $\{\bar{X}_j, H_j\}\ j = 0,1..M$ llamados vectores de test y comprobar el error producido. El error del sistema será dado por el RMSE del conjunto de test:

$$Error_{RMSE} = \sqrt{\frac{1}{M}\sum_{j=1}^{M}\left(H - \widehat{H}\right)^2}$$

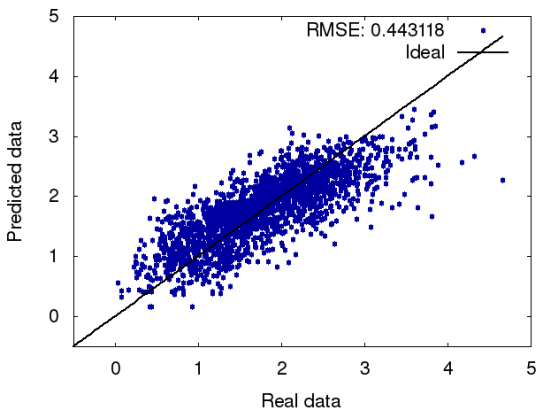


Fig. 2 Ejemplo de recta de regresión.

También se puede representar el error mediante una gráfica de regresión donde se compara el valor deseado con el predicho. Idealmente los puntos deberían formar una recta de pendiente 1. El objetivo de este proyecto es encontrar el menor subconjunto de características $S \in X$ para el que se obtenga el menor error RMSE.

Nos encontramos por tanto, ante un problema de **selección de características** en el cual se debe encontrar el subconjunto más pequeño de características que consiga la mejor predicción de la altura de ola y por tanto reduzca el RMSE al mínimo. Se puede representar el vector de características como un array de bits de selección, con un bit por característica, donde un '1' significa utilizar la característica asociada y un '0' no utilizarla. Así pues, en un problema con n parámetros tenemos un vector de selección de dimensiones:

$$S = \{0,1\}^n$$

Con un vector de selección de $'n'$ características, se pueden tener 2^n posibles subconjuntos diferentes, de entre los cuales tenemos que encontrar el mejor. Una opción para encontrar el mejor subconjunto de características, es probar cada uno de los posibles subconjuntos y elegir el mejor entre ellos, este proceso se denomina "búsqueda exahustiva". Para grandes tamaños de $'n'$ no podemos realizar esta búsqueda ya que el tiempo computacional necesario sería excesivo, para $n = 60$ se tienen más de 1.000.000.000.000.000.000 posibilidades!

Es por ello que es necesaria la utilización de algoritmos de búsqueda inteligentes que sean capaces de encontrar los mejores subconjuntos, sin tener que probar todos ellos. Estos algoritmos realizan lo que se llama, una búsqueda metaheurística, es decir una búsqueda "inteligente" sobre el conjunto de parámetros. Estos algoritmos suelen poseer un conjunto de soluciones, llamado población, que van siendo modificadas, ya sea entre sí o por separado, buscando subconjuntos prometedores de características con la esperanza de encontrar el subconjunto óptimo.

En este documento se utilizan los siguientes algoritmos metaheuristicos:

- SA (Temple Simulado): Basado en el recalentamiento de metales.
- GA (Algoritmos Genéticos): Basado en la teoría de la evolución de Darwin.
- AIS (Sistema Inmune Artificial): Basado en el funcionamiento del sistema inmune.

La mitad del esfuerzo de este proyecto está invertido en el desarrollo de un programa en C capaz de realizar todas las operaciones necesarias para llevar a cabo el estudio:

- Realizar la ELM de un subconjunto de características.
- Realizar estudios del algoritmo de la ELM.
- Implementar algoritmos de búsqueda metaheurísticos y deterministas, entre los que se encuentran ES, SA, GA, AIS, SFS y SBE. Además el programa realiza estudios de los parámetros por separado y de la afinidad entre cada par de ellos.
- Ser capaz de escribir los resultados en ficheros externos y generar gráficas.

Capítulo 2
Energía Mareomotriz

2. ENERGÍA MAREOMOTRIZ

Actualmente existe un interés creciente sobre el uso de energías eficientes, renovables y ecológicas, principalmente debido a su efecto decisivo en la economía y el cambio climático. Estos factores están produciendo grandes esfuerzos en la búsqueda y desarrollo de sistemas de energía renovables y sostenibles, tanto en generación, distribución y consumo. Entre las energías renovables como el viento o la energía solar, se encuentra la energía marina, que si bien está en minoría, posee un gran potencial de crecimiento y utilización sostenible.

La energía de las olas oceánicas es enorme, incluso la fracción de esta energía que es potencialmente explotable es muy grande comparada con el consumo actual de electricidad en el mundo. Se han realizado diversos estudios con el propósito de estimar el potencial mundial. Se estima que la energía mundial explotable es de 2TWh año y que las aguas europeas son capaces de cubrir más del 50% del consumo total de potencia en el continente.

Aunque la energía oceánica presenta un gran potencial energético, no todas las tecnologías disponibles tienen la misma viabilidad para aprovecharlo. Esto se debe principalmente a factores económicos ya que las energías renovables suelen requerir una fuerte inversión e capital. La energía marina se puede aprovechar de diferentes maneras, la más común y rentable es aprovechar la energía cinética de las olas, ya sea debida a la oscilación vertical que producen o su movimiento lineal de arrastre.

Los WECs (Wave Energy Converters) son los dispositivos que convierten la energía de olas marinas en energía eléctrica. Su coste y rendimiento son fundamentales para la viabilidad de este tipo de tecnología. Se estima que los aparatos instalados en la costa tengan un coste de la unidad energética generada entre 0,09€/kWh y 0,14€/kWh mientras que los que se encuentran fuera de la costa tienen un valor bastante mayor.

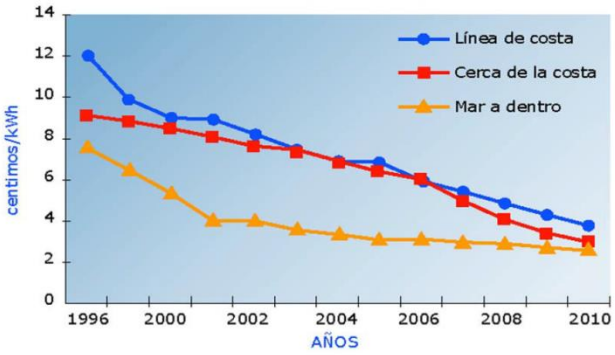


Fig. 3 Coste de la energía mareomotriz. Fuente [20]

Las ventajas más destacables de este tipo de energía.

- Se trata de un recurso renovable, sostenible y abundante.
- No produce CO_2 ni contamina el ambiente.
- Se trata de una energía muy concentrada, relativamente consistente y predecible.

Entre los posibles impactos negativos pueden señalarse los siguientes:

- Impacto visual y ruido. Cuando un área depende del turismo, la obstrucción visual es crítica.
- Destrucción de la vida marina y erosión de la costa. Algunos WEC concentran la energía de olas en un área antes de su captación originando un incremento de la erosión.
- Conflictos con la navegación. Una vez instalados, los sistemas de captación de la energía del oleaje podrían ser un peligroso obstáculo para cualquier embarcación que no pueda verlos o detectarlos por radar.
- Interferencia con la pesca comercial, deportiva y actividades recreativas.

2.1 Naturaleza de las olas

Una ola es cualquier tipo de oscilación periódica sobre la superficie de agua. Las olas de los océanos son originadas por diversas causas como pueden el viento, las fuerzas de atracción gravitacional de la Luna y el Sol o los maremotos. De entre todas ellas, el viento constituye el agente que genera las olas más comunes y de mayor densidad energética. Dado que la energía del viento es producida como consecuencia del desigual calentamiento que el Sol produce sobre la superficie terrestre, se dice que la energía de las olas es un derivado terciario de la energía solar.

Las olas son generadas cuando las moléculas del aire en movimiento interactúan con las moléculas del agua modificando la superficie del océano, dando lugar a pequeños rizos, conocidos como olas de capilaridad. Las olas de capilaridad dan lugar a una mayor superficie de contacto, incrementando la fricción entre agua y viento. Esto produce un crecimiento de la ola, que aumenta su superficie y por tanto la energía que el viento le transfiere. El tamaño de las olas generadas por un campo de viento depende de tres factores:

- La velocidad del viento.
- El tiempo durante el cual éste está soplando.
- La distancia sobre la cual la energía del viento se transfiere al océano.

Fig. 4 Ola oceánica. Fuente [21].

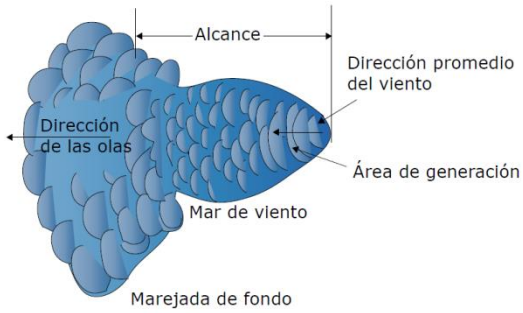


Fig. 5 Formación de las olas por viento. Fuente [20].

Las olas situadas dentro o cerca de las áreas donde fueron generadas se denominan *olas tormentosas*. Estas forman un mar irregular y pueden viajar hacia otras áreas para producir grandes olas. Una vez que las olas se alejan del área de generación, sus crestas son más lisas y menos caóticas, a este oleaje se le llama *marejada de fondo*. Estas olas se dispersan sobre la superficie oceánica con muy poca pérdida de energía, aunque pierden altura, principalmente por dispersión angular .

Las olas de los océanos están constituidas por moléculas de agua que se mueven formando círculos. En zonas profundas, en la superficie del agua, los movimientos son del mismo tamaño que la altura de la ola, pero estos movimientos disminuyen exponencialmente en tamaño al descender debajo de la superficie. En zonas menos profundas, el movimiento pasa de circulas a elíptico.

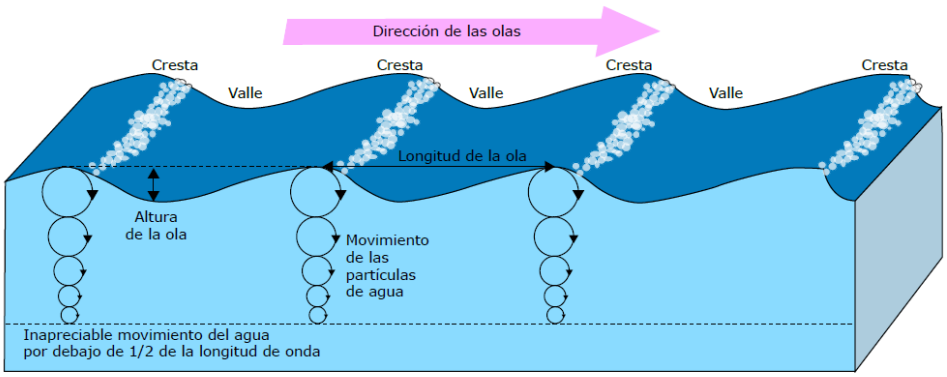


Fig. 6 Movimiento de las moléculas del agua dentro de una ola. Fuente [20].

Las olas senoidales puras están caracterizadas por su:

- Longitud de onda L: Es la distancia entre dos picos consecutivos.
- Altura de onda H. Es la diferencia de altura entre un pico y un valle.
- Periodo T: Es el tiempo que tarda un valle o un pico de la ola en recorrer su longitud de onda. La frecuencia f de la ola se define como el número de oscilaciones pico a pico (o valle a valle) de la superficie de la ola.

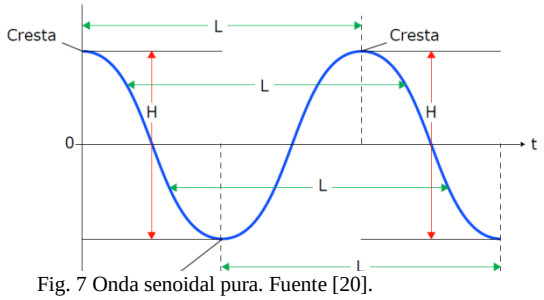


Fig. 7 Onda senoidal pura. Fuente [20].

El océano no se compone de ondas senoidales puras sino más bien por una superposición de muchas de ellas. La superficie del mar puede ser expresada como una suma de ondas senoidales puras de frecuencia y amplitud variable. Por tanto, un estado típico del mar se compone de una combinación de ondas de diferentes características, es decir, cada una con su propia velocidad, periodo, altura de onda, y dirección. El resultado de esta combinación se llama envolvente y es lo que se observa cuando se mira la superficie del mar. La envolvente de estas olas viaja a una velocidad distinta de la de las ondas individuales, y se la denomina velocidad de grupo v_g

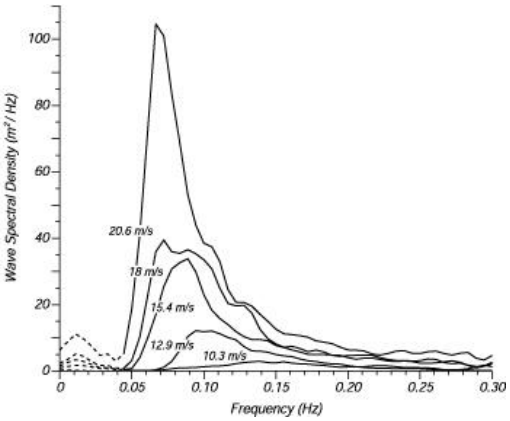


Fig. 8 Espectro de olas marinas. Fuente [23].

El conocimiento del espectro $S(f)$ de las olas es muy importante en el diseño de estructuras costeras y de agua profunda, como barcos o plataformas petroleras, cuya respuesta a las distintas condiciones de oleaje debe ser bien estudiada. El espectro $S(f)$ nos proporciona información de las diferentes ondas senoidales puras que componen las olas marinas y cuanta energía poseen. Arriba podemos observar el espectro de diversas olas en función de la velocidad del viento. Como se puede apreciar, a mayor velocidad del viento, mayor energía posee la ola, manteniéndose la frecuencia fundamental prácticamente constante.

La mayoría de la energía de ola en las aguas costeras proviene de aquellas generadas por el viento y las corrientes marinas. En definitiva, las olas oceánicas son movimientos de energía. La energía de las olas tiene 2 componentes, energía cinética y energía potencial.

- La energía cinética proviene del movimiento de las partículas individuales de agua que se mueven constantemente en una forma circular. Esta energía cinética puede ser utilizada en diferentes clases de aparatos de conversión de energía del oleaje (WEC).

- En su movimiento circular las moléculas individuales de agua son elevadas encima de la línea inmóvil de la superficie del agua obteniendo energía potencial dentro del campo gravitatorio terrestre.

2.2 Energía marina

Al planear el despliegue de un WEC en una localización determinada, es esencial caracterizar primero la cantidad de energía de ola disponible. La energía de ola (wave energy) de una región proviene de los vientos locales y lejanos que soplan sobre la superficie del océano, transfiriendo su energía a la ola. El 95% de la energía de la ola se encuentra entre la superficie del agua y un cuarto de longitud de onda por debajo, así pues, para calcular la energía de una ola, debemos centrarnos en la superficie de la misma. Usando la teoría de olas lineales [23], la elevación vertical de una ola en un punto $\vec{r}$ del espacio, en un instante t se expresa como:

$$\eta(\vec{r}, t)$$

Para una onda senoidal pura, esta altura puede expresarse como:

$$\eta(\vec{r}, t) = H \cdot \sin\left(\omega \cdot t + \vec{k} \cdot \vec{r}\right)$$
$$= H \cdot \sin\left(\frac{2\pi}{T} \cdot t + \frac{2\pi}{L}\,\overrightarrow{k_n} \cdot \vec{r}\right)$$

Siendo:

H: La altura de la ola.

T: El periodo de la ola.

$\vec{k}$: La dirección de propagación de la ola.

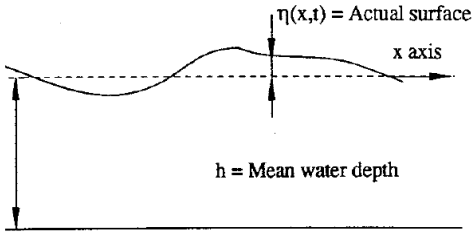


Fig. 9 Altura de ola. Fuente [23]

La altura $\eta(\vec{r}, t)$ del océano puede ser expresada como la combinación de los diferentes armónicos puros que forman la ola y que podemos obtener calculando el espectro $S(f)$ de la misma. Podemos hacer esta asumción cuando las componentes frecuenciales de la superficie no cambian significativamente en el tiempo y el espacio, es decir, son temporalmente estacionarias estadísticamente y espacialmente homogéneas. Con estas suposiciones podemos considerar la elevación $\eta(\vec{r}, t)$ como un proceso estocástico Gaussiano de media 0 con simetría estadística entre crestas y valles. La figura de la derecha muestra la gráfica de la distribución estadística Gaussiana.

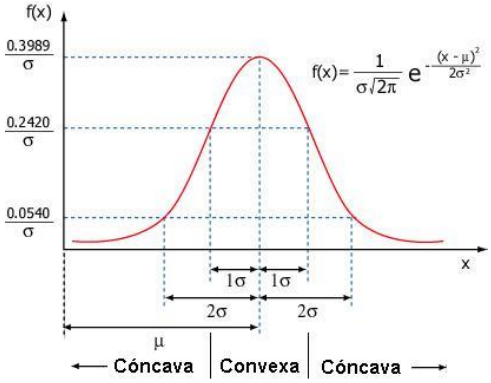


Fig. 10 Distribución Gaussiana. Fuente [23]

El "Estado del Mar" (Sea State) es un área de mar en concreto, en un tiempo determinado, en la cual las componentes estadísticas y espectrales de las olas no varían significativamente. La energía total de un Sea State es la suma de energías de sus diferentes fuentes, lo que da lugar a diferentes configuraciones de mar:

- **Wind sea**: Cuando las olas son causadas por la transferencia de energía entre el viento local y la superficie libre del agua.

- **Swell**: Cuando las olas han sido generadas por vientos soplantes de una zona lejana (tormentas) que se propagan hasta la región de observación.

Normalmente el estado del mar es la composición de estas 2 componentes puras, llevando a los llamados mares multinodo o mixtos.

El Estado de Mar resultante puede ser descrito como un espectro $S(f)$ que proporciona información sobre la energía transmitida por todos los sistemas de olas existentes. El primer indicador de la cantidad de energía disponible en un área dada es el "flujo de energía de onda" o "densidad de potencia por metro" de la cresta de la ola, P, que puede ser estimada a partir del espectro direccional $S(f,\theta)$ como:

$$P = \rho g \int_0^{2\pi} \int_o^{\infty} C_g(f,h) S(f,\theta) df \, d\theta$$

Donde:

- ρ: Es la densidad del agua, $1025 \left[\frac{Kg}{m^3}\right]$.
- g: Es la aceleración de la gravedad.
- $C_g(f,h)$: Es la velocidad de grupo de cada componente frecuencial f siendo h la profundidad del agua.
- $S(f,\theta)$: Es el espectro de onda direccional, que está relacionado con el espectro de onda como:

$$S(f) = \int_o^{2\pi} S(f,\theta) \, d\theta$$

El **momento espectral** de orden n, denominado m_n es definido como:

$$m_n = \int_0^{2\pi} \int_o^{\infty} f^n S(f,\theta) df \, d\theta = \int_o^{\infty} f^n S(f) df$$

Estos momentos son muy útiles ya que nos permiten calcular los 2 parámetros más importantes utilizados en aplicaciones de energía: H_s y T_e

- La altura significativa de ola H_s puede ser estimada por:

$$H_s = 4\sqrt{m_o}$$

- El periodo de energía de la ola T_e puede ser estimado como:

$$T_e = T_{-1,0} = \frac{m_{-1}}{m_0}$$

La densidad de potencia de ola por metro de la cresta P puede ser escrita como función de H_s y T_e:

$$P = \frac{\rho g^2}{4\pi} \int_o^{\infty} \frac{S(f)}{f} df = \frac{\rho g^2}{4\pi} m_{-1} = \frac{\rho g^2}{4\pi} H_s^2 T_e$$

Para una H_s expresada en metros y T_e en segundos se tiene que:

$$P = 0.49 \cdot H_s^2 T_e$$

Esta ecuación permite a los ingenieros estimar la cantidad de energía disponible cuando planean instalar un WEC en una posición dada.

2.3 Wave Energy Converters

Un Wave Energy Conveter (WEC) es un dispositivo capaz de transformar la energía de las olas en energía eléctrica para su aprovechamiento. Normalmente utilizan pistones para convertir la energía mecánica a eléctrica e independientemente de la forma que lo hagan, necesitan cables subacuáticos con los que transportar dicha energía desde el dispositivo a la central eléctrica. Existen miles de tipos de WECs, aunque la mayoría están en proceso de desarrollo y muy pocos han sido construidos y probados en los océanos. A continuación se verán diversos criterios para clasificar los WECs.

Según la profundidad y lugar donde se utilicen:

- **Shoreline**: Situados cerca de la costa y por tanto, de la central eléctrica. Producen menos energía ya que las olas suelen tener menos energía en la costa que mar adentro. Son los más fáciles de mantener y tienen la menor probabilidad de ser destruidas por condiciones climatológicas extremas.
- **Nearshore**: Situados en aguas relativamente poco profundas (normalmente 1/4 de la longitud de onda). Producen menos energía que los dispositivos offshore.
- **Offshore**: Situados en aguas profundas siendo las que más energía producen. Lo malo es que son más difíciles de construir y mantener.

Otra característica que puede ayudar a diferenciar los WECs es el tamaño de la parte mecánica que es movida por las olas y que captura su energía dando lugar a los tipos "large absorvers" y "point absorvers". Una de las formas más comunes de clasificación es hacerlo a partir de la posición del dispositivo respecto a la dirección predominante de la ola, así tenemos:

- **Paralelo (atenuattor):** Son WECs flotantes situados sobre la superficie del agua. Están formados por cilindros semiflotantes unidos por juntas hidráulicas. Aprovechan el movimiento relativo de los cilindros debido al oleaje para mover las juntas y generar energía.

- **Perpendicular (terminator):** Son WECs formados por un gran absorvedor situado perpendicular a la dirección de las olas que es movido hacia adelante y atrás por las mismas. Normalmente son de tipo *nearshore* sujetos al suelo a unos 10-15 metros de profundidad. Al moverse hacia adelante y atrás convierten la energía mecánica en eléctrica gracias a dos pistones.

- **Punto de absorción**: Son WECs formados por un pequeño dispositivo que aprovecha el incremento y descenso de altura que produce la ola sobre el nivel del mar. Pueden absorver energía de cualquier dirección. Los hay de 2 tipos:
    - **Flotantes:** Aprovechan el desnivel del mar.
    - **Sumergidos:** Aprovechan la diferencia de presión.

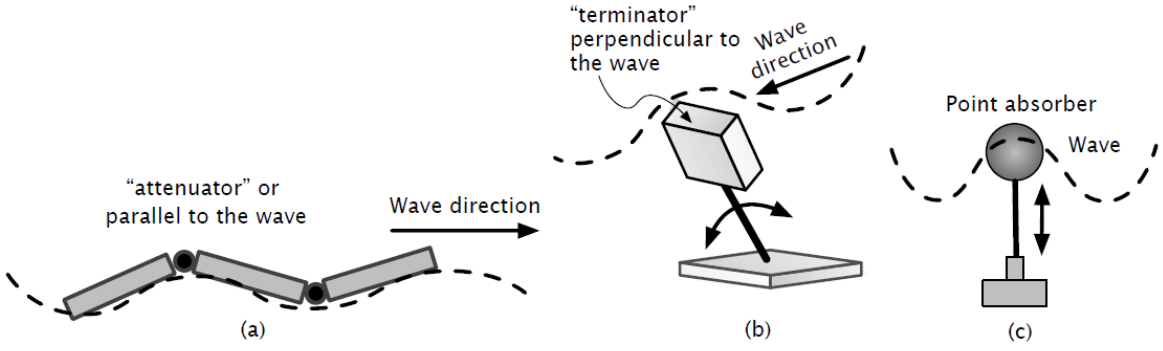


Fig. 11 (a) WEC paralelo, (b) WEC perpendicular, (c) WEC de punto de absorción. Fuente [21].

Existen otros tipos de WEC como pueden ser:

- **Oscillating water column:**

Una "columna de agua oscilante" es una estructura hueca parcialmente sumergida que se encuentra abierta al mar por debajo del nivel de este para que las olas hagan que suba y baje el agua dentro de la "Water column". Al subir el nivel del mar, el aire dentro de la columna hueca se comprime y pasa hacia la atmosfera por una turbina que será movida por este, generando electricidad.

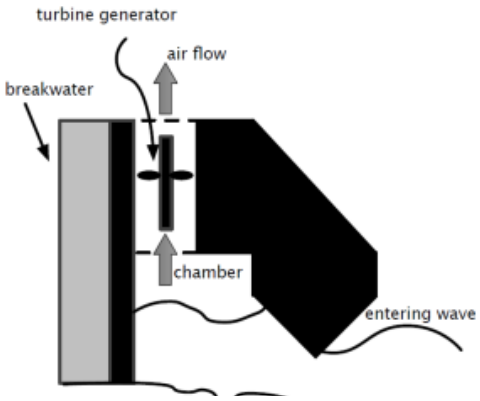


Fig. 12 Oscillating water column. Fuente [21].

- **Overtopping/Terminator device**

Este tipo de WEC captura el agua de las olas cuando estas entran en la reserva. El agua es devuelta al mar pasando por una turbina convencional "low-head" que genera energía. Este tipo de WECs suelen utilizar colectores para concentrar la energía de la ola.

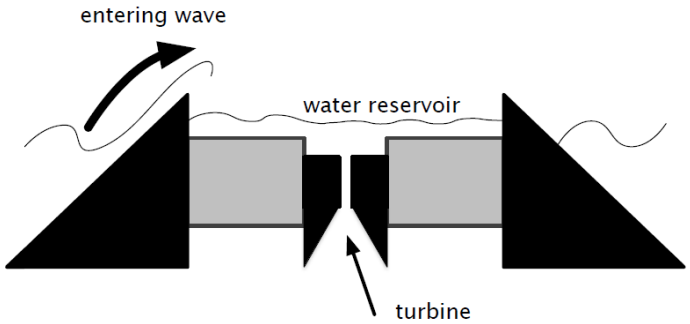


Fig. 13 Overtopping/Terminator device. Fuente [21].

- **Rotating mass-based WECs**

En este tipo de WECs, el movimiento oscilante que producen las olas hace rotar un giroscopio que se encuentra dentro del dispositivo. La parte rotatoria tiene un generador eléctrico dentro. Por ejemplo en el Salters Duck hay este tipo de WECs que poseen una eficiencia del 90%, poseen giroscopios y un generador eléctrico que transforma la rotación en energía. Su eje principal está situado perpendicular a la dirección predominante de las olas.

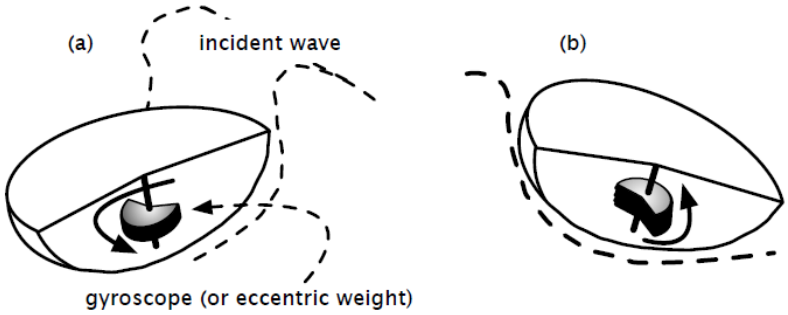


Fig. 14 Rotating mass-based WECs Fuente [21].

2.4 Parámetros del oleaje

Las boyas oceanográficas son el instrumento más usado en la medida del oleaje en un punto. Tienen uno o varios acelerómetros que miden las series temporales de aceleraciones que producen las olas en la boya. Estas series de aceleraciones se integran dos veces obteniendo así los desplazamientos del oleaje. Existen dos tipos básicos de boya para la medida del oleaje:

- **Boya escalar**: Tiene un único sensor que registra la elevación del oleaje $\eta(\vec{r},t)$. Mediante un análisis de Fourier se estima la densidad espectral de potencia del oleaje $S(f)$. La frecuencia de las olas varía en el rango entre $0{,}04\,Hz$ y $1\,Hz$ aproximadamente. La función $S(f)$ proporciona información sobre las condiciones en las que ese oleaje fue generado. $S(f)$ está relacionado con la distribución de energía para diferentes frecuencias o, análogamente, periodos de las olas.

Fig. 15 Boya escalar. Fuente [47].

- **Boya direccional**: A parte del acelerómetro vertical, estas boyas tiene más acelerómetros dispuestos sobre diferentes ejes. Pueden medir otras características de las olas, como sus pendientes en diferentes direcciones, desplazamientos horizontales… Pueden estimar el espectro direccional $S(f,\theta)$, que informa, no sólo de la distribución de energía para diferentes periodos de las olas, sino también de la distribución de energía por direcciones de propagación θ. Así, por ejemplo, con la función $S(f,\theta)$ se puede saber si las olas vienen de una dirección determinada, o si hay varios sistemas de olas superpuestos provenientes de diferentes lugares.

2.4.1 Parámetros escalares

Estos parámetros se obtienen a partir del espectro escalar del oleaje $S(f)$ y entre ellos se encuentran algunos de los que se utilizarán en el estudio de este proyecto.

- **Momentos espectrales**: Proporcionan información sobre diferentes características estadísticas y físicas del oleaje. El momento de orden n-ésimo se define como:

$$m_n = \int_o^\infty f^n S(f) df \qquad n = -1, 0, 1, 2, 3, 4$$

- **Altura significativa de ola** H_s: Es el parámetro relacionado con la altura de ola muy usado en el diseño de buques y construcciones marítimas (diques, plataformas petrolíferas, etc.) y en protección costera (playas, etc.).

$$H_s = 4\sqrt{m_o}$$

- **Periodo de pico** T_p: Es el tiempo en segundos que tarda un valle o un pico de la ola en recorrer su longitud de onda.

$$T_p = \frac{1}{f_p}$$

Donde f_p es la frecuencia de pico espectral (frecuencia donde el espectro $S(f)$ es máximo).

- **Periodo de energía T_e:** Es una estimación del periodo medio utilizado en cálculos de diseño de turbinas para la extracción de energía del oleaje.

$$T_e = \frac{m_{-1}}{m_0}$$

- **Periodo medio** Tm_{01}: Es un estimador del periodo medio utilizado en el estudio de olas gigantes y en la predicción de oleaje.

$$Tm_{01} = \frac{m_0}{m_1}$$

- **Periodo medio** Tm_{02}: Es un estimador del periodo medio utilizado en el diseño de estructuras marinas.

$$Tm_{02} = \sqrt{\frac{m_0}{m_2}}$$

- **Parámetro de anchura espectral** ν.

- **Factor de apuntamiento de Goda** Q_p.

- **Correlación entre alturas de ola consecutiva** γ. Este parámetro mide propiedades del agrupamiento del oleaje (propiedad que tienen las olas de propagarse en grupos de olas altas seguidos de olas más bajas). Este parámetro depende del llamado parámetro de agrupamiento de Kimura κ, que se calcula a partir del espectro $S(f)$ y de un estimador del periodo medio. Existen dos estimadores de κ, y por tanto de γ, según se utilice el periodo Tm_{01} o Tm_{02}.

Además de estos parámetros ya calculados, existen otros importantes que se derivan de los anteriores. El más relevante es el peralte de ola ϵ, que es la relación entre su altura y la longitud de onda. Normalmente se expresa en términos del periodo de la ola en vez de su longitud de onda.

$$\epsilon = \frac{2\pi H_s}{gT_c}$$

Donde $g = 9,81\, m/s^2$ es la aceleración de la gravedad en la superficie y T_c es un periodo característico del oleaje. Se suele usar T_p, Tm_{01} o Tm_{02}.
Tanto ν, como Q_p, son parámetros bastante inestables.

2.4.2 Parámetros direccionales

- **Dirección de pico $\boldsymbol{\theta_p}$**: Es la dirección de propagación de ola de máxima energía. Es decir, el valor de θ donde $S(f,\theta)$ es máximo.

- **Dirección media $\overline{\boldsymbol{\theta}}$:** Representa el valor medio de la dirección de propagación de todo el campo de oleaje. Se calcula utilizando $S(f,\theta)$ como función peso y teniendo en cuenta que θ es una variable angular.

A parte de estos parámetros, existen otros parámetros direccionales que informan sobre la dispersión direccional del oleaje, las características direccionales para cada frecuencia f, etc.

2.4.3 Parámetros de este proyecto

A partir de los datos de las boyas que ofrece la National Data Bouy Centre, se pueden obtener una serie de parámetros del oleaje que lo caracterizan. Los parámetros de los que disponemos se pueden dividir en 3 grupos, siendo el último grupo, el conjunto de parámetros utilizado en el estudio de este documento.

- **Parámetros temporales**

Representa el instante de tiempo en que se empieza la medida (que es a cada hora en punto) más un último número que indica si la medida es válida o no. Siguen el formato:

```
YY, MM, DD, HH, mm, iCal
```

La fecha está expresada como: YY/MM/DD mm:HH
Para el Índice de calidad: Si iCal = 1 significa que la medida es válida.

- **Espectro:**

Es el conjunto de valores Nf del espectro tomados por la boya.

$S(1), S(2), S(3), \dots, S(Nf)$:

- **Parámetros del oleaje**

Son los parámetros que derivamos a partir del espectro. Estos son:

H_s:	Altura significativa [m]
m_{-1}:	Momento espectral de orden -1
m_0:	Momento espectral de orden 0
m_1:	Momento espectral de orden 1
m_2:	Momento espectral de orden 2
T_p:	Periodo de pico [s]
Tm_{10}:	Periodo medio (estimador m-10, periodo de la energía) [s]
Tm_{01}:	Periodo medio (estimador m_{01}) [s]
Tm_{02}:	Periodo medio (estimador m_{02}) [s]
Q_p:	Factor de apuntamiento de Goda
nu:	Factor de ancho de banda de Longuet-Higgings
k_{01}:	Parámetro de Kimura usando Tm_{01} como estimador del periodo medio
$gamma_{01}$:	Correlación entre Alturas de ola consecutivas a partir del parámetro k_{01}
k_{02}:	Parámetro de Kimura usando Tm02como estimador del period0 medio
$gamma_{02}$:	Correlación entre Alturas de ola consecutivas a partir del parámetro kimura02

Estos son los 15 parámetros del oleaje que se utilizarán para investigar la relación entre en el oleaje en un punto y el oleaje en otros tantos.

H_s	m_{-1}	m_0	m_1	m_2	T_p	Tm_{10}	Tm_{01}	Tm_{02}	Q_p	nu	k_{01}	g_1	k_{02}	g_{02}

2.4.4 Obtención de los parámetros

A continuación se narrará el proceso que se ha seguido para obtener un archivo de datos en texto plano que contiene los 75 parámetros en formato double que corresponden a las 5 boyas del estudio. Cada boya aporta sus 15 parámetros anteriormente descritos.

El primer paso es dirigirse a la página web del NDBC, http://www.ndbc.noaa.gov/, introducir el número de la boya en el buscador y presionar el botón de "Go".

Fig. 16 Búsqueda de una boya. Fuente [47].

Esto nos llevará a la página web de la boya, donde se nos muestra información relativa a la misma, una foto y su localización. El enlace que nos interesa son los datos históricos, que se encuentra prácticamente al final de la misma, por lo que nos desplazamos hacia abajo y hacemos click en el enlace "Historical Data & Climatic Summaries".

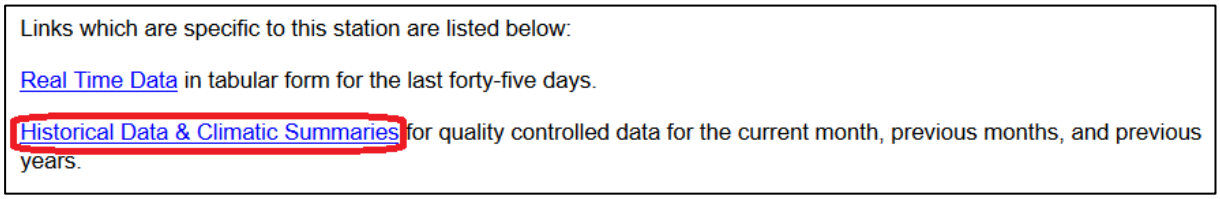


Fig. 17 Acceso a los datos históricos de una boya. Fuente [47].

De entre los datos disponibles para descargar a nosotros nos interesa la densidad espectral de potencia de la ola "Spectral wave density data". Hacemos click en uno de los años para descargarnos las mediciones.

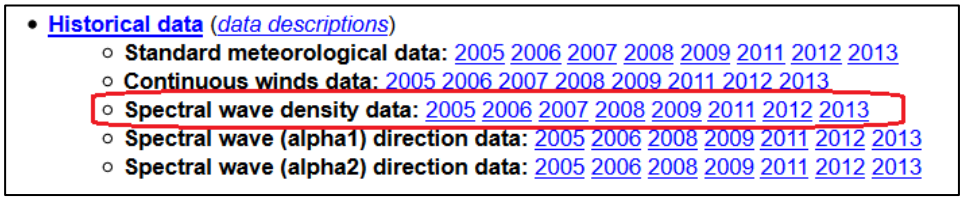


Fig. 18 Link de descarga de los datos de la boya. Fuente [47].

Repetimos este proceso, descargándonos los datos históricos de las 5 boyas para el mismo año. Una vez lo tengamos descargados debemos tratar los ficheros con MATLAB utilizando el script llamado "IntegratedParameters" desarrollado por el profesor Enrique Alexandre Cortizo, que obtiene los parámetros de la boya antes mencionados. Por ejemplo, para obtener los datos válidos sincronizados en el tiempo de las 5 boyas (75 parámetros) para el año 2009 en un archivo de texto plano, se ejecuta el siguiente script.

Tabla 1 Código MATLAB para obtener datos.

```
[f1, t1, S1, x1] = IntegratedParameters('42056w2009.txt',1);
[f2, t2, S2, x2] = IntegratedParameters('42057w2009.txt',1);
[f3, t3, S3, x3] = IntegratedParameters('42058w2009.txt',1);
[f4, t4, S4, x4] = IntegratedParameters('42059w2009.txt',1);
[f5, t5, S5, x5] = IntegratedParameters('42060w2009.txt',1);
x = [x1; x2; x3; x4; x5];
for i=1:75
nulos = find(x(i,:) == 999);
x(:,nulos) = [];
end
x = x.';
save ('datos2009.dat','-ascii', 'x');
```

Capítulo 3
Redes Neuronales

3. REDES NEURONALES

Una Red Neural Artificial (ANN Artificial Neural Network) es un paradigma de Inteligencia Artificial basado en el modo de funcionamiento del cerebro de los seres vivos [4]. El cerebro puede considerarse como un ordenador complejo, no-lineal y con procesamiento en paralelo que es capaz de realizar funciones como reconocimiento de patrones y control de motores mucho más rápido que cualquier sistema digital actual, pudiendo además aprender, memorizar y generalizar. Una ANN emula la estructura funcional del cerebro, la cual se explica a continuación.

El cerebro está formado por una red de millones de células llamadas Neuronas que se encuentran interconectadas entre sí. Las neuronas están principalmente formadas por:

- **Soma:** Es el cuerpo de la neurona, contiene sus orgánulos y núcleo, responsables del sistema que dirige el comportamiento de la neurona.
- **Dendritas**: Son prolongaciones cortas que se originan del soma neural. Su función es recibir impulsos de otras neuronas y enviarlas hasta el soma de la neurona.

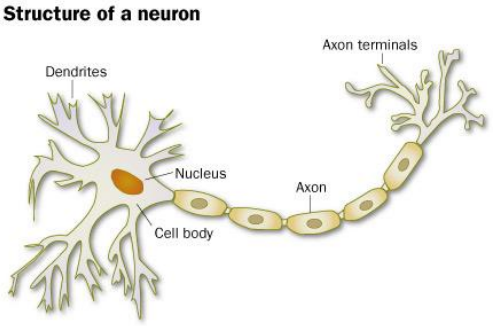


Fig. 19 Neurona biológica. Fuente [6].

- **Axón**: Es una prolongación única y larga. Su función es sacar el impulso desde el soma neuronal y conducirlo hasta otro lugar del sistema.

De la misma manera, una RNA está formada por una red neuronas artificiales interconectadas entre sí que en conjunto trabajan para resolver un problema, generalmente se utilizan para reconocimiento de patrones o clasificación de datos. Normalmente requieren un proceso de aprendizaje en el que las neuronas son entrenadas, modificando sus parámetros internos (soma) para después realizar la función deseada. Las neuronas artificiales poseen mmúltiples entradas (dendritas) y una sola salida (axón) a través de las cuales se intercambian información unas con otras y con el exterior.

Una Red Neuronal viene definida por:

- **Modelo neuronal**: El modelo de la neurona, unidad básica de procesamiento de información.
- **Arquitectura**: El conjunto de neuronas del sistema y la forma en la que estas se interconectan.
- **Algoritmo de aprendizaje**: Es el algoritmo utilizado para dar valor a los parámetros de las neuronas, los cuales definen la función del sistema. Su objetivo es conseguir una RNA entrenada que generalice bien ante nuevas entradas.

El algoritmo de aprendizaje está muy relacionado con la Arquitectura, estos se diseñan para una arquitectura en concreto y comúnmente solo funcionan para dicha arquitectura.

Como cualquier otro sistema, una Red Neuronal puede expresarse como una función no lineal que mapea un espacio de entrada R^I en uno de salida R^O.

$$f_{RN}: \mathbb{R}^I \rightarrow \mathbb{R}^O$$

Donde I y O son las dimensiones de los datos de entrada y de salida respectivamente.

Los diferentes tipos de Redes Neuronales que existen vienen definidos por especificaciones en las anteriores características. A continuación explicaremos más en detalle cada una de estas y después veremos los tipos de ANN más utilizados y con los que trabajaremos en este documento.

3.1 Modelo Neuronal

La neurona es la unidad básica de procesamiento de información de una RNA. Al igual que las neuronas naturales, la neurona artificial (también llamada perceptrón) está formada por un conjunto de conexiones con otras neuronas (dendritas), que son las entradas de información y por una salida (axón), que es función de dichas entradas y de ciertos parámetros de la neurona (soma).

El modelo de una neurona artificial está formado por:

- Un conjunto de ***N* entradas** (inputs) $\bar{X} = [x_1, x_2, x_3 \dots x_N]$ a través de las cuales la neurona obtiene la información a procesar. Cada entrada x_i a su vez tiene un **peso w_i** asociado que indica la importancia que tiene dicha entrada para la neurona. El valor de cada entrada se multiplica por su peso asociado. Cada neurona a su vez posee otro parámetro llamado **bias *b*** que es un valor de offset que se añade a la neurona. Puede considerarse como una entrada cuyo valor es siempre el mismo.

- Un **Combinador lineal** que realice la suma ponderadas de las entradas con los pesos.

$$u = \bar{X} \cdot \bar{W} + b = \sum_{i=1}^{N} w_i \cdot x_i + b$$

  Este es el tipo de combinador más utilizado, pero existen otros modelos.

- Una **función de transferencia f_A**: (Activation function or Transfer Function).
  Función que relaciona las entradas ponderadas con la salida. La salida queda definida como:

$$o = f_A(u) = f_A(\bar{X} \cdot \bar{W} + b) = f_A\left(\sum_{i=1}^{N} (w_i \cdot x_i) + b\right)$$

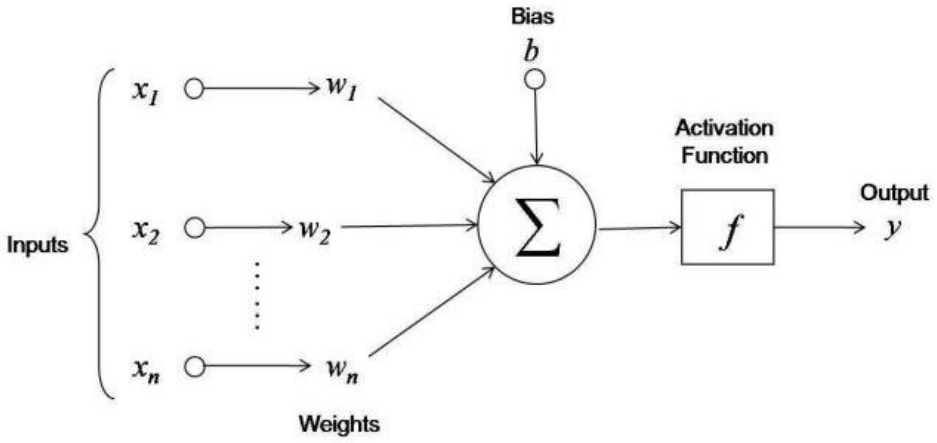


Fig. 20 Neurona Artificial. Fuente [5].

Así pues los parámetros que definen una neurona son:

- El vector de pesos: $\bar{W} = [w_1, w_2, w_3 \dots w_N]$.
- El bias b.
- La función de transferencia $f_A(u)$.

Matemáticamente podemos definir la Neurona Artificial como una función que mapea el espacio $\mathbb{R}^N$ a un valor de salida que suele estar limitado al rango $[0,1]$ *o* $[-1,1]$.

$$f_A: \mathbb{R}^N \rightarrow [0,1]$$

3.1.1 Función de Trasferencia

El comportamiento de una RNA depende en gran medida de la función de transferencia de sus neuronas, pudiendo ser esta diferente para cada una de ellas. La función de transferencia relaciona las entradas ponderadas de la Neurona con la salida de la misma. El bias suele utilizarse para situar el origen en algún punto específico de la función de transferencia. Suelen distinguirse 3 tipos de funciones de transferencia:

- **Lineal**: La salida es proporcional a la suma ponderada de las entradas.
- **Umbral:** La salida toma valores discretos en función de la suma ponderada de entradas
- **Sigmoide:** La salida es una función continua no lineal. Entre ellas podemos encontrar, la función seno, tangente hiperbólica, sigmoide o gaussiana.

Algunos ejemplos de funciones de transferencia son:

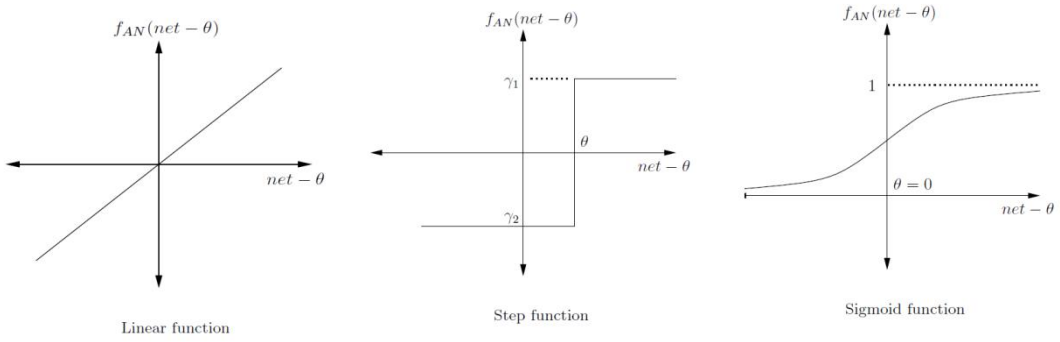


Fig. 21 Funciones de transferencia. Fuente [4].

Las neuronas artificiales con una función de transferencia de tipo sigmoide tienen un comportamiento más parecido a las neuronas reales, sin embargo, los 3 tipos son solo burdas aproximaciones. La práctica nos muestra que las funciones de activación con valores tanto positivos como negativos y centrados en el origen aprender patrones más fácilmente.

Hornik [13] demostró que si la función de activación es continua, limitada y no constante, entonces una RNA puede emular cualquier función continua si los valores de los pesos son ajustados correctamente. Leshno demostró que las redes de tipo feedforward con función de activación no lineal pueden aproximar funciones continuas.

El tipo de función de más utilizada es de tipo sigmoide con salida limitada en el rango [0,1] ó [-1,1], dentro de este tipo de funciones tenemos:

- **Función sigmoide**:

$$f_{sig}(v) = \frac{1}{1-e^{-\lambda \cdot v}} \qquad f_{sig} \in (0,1)$$

El parámetro λ controla la pendiente de la función y suele tomar valor 1.
Se suele utilizar esta función debido a las propiedades de su derivada:

$$\frac{\partial f_{sig}(v)}{\partial v} = \lambda \frac{e^{-\lambda \cdot v}}{(1-e^{-\lambda \cdot v})^2} = \lambda \left(\frac{1}{1-e^{-\lambda \cdot v}} - \frac{1}{(1-e^{-\lambda \cdot v})^2}\right) = \lambda \left(f_{sig}(v) - f_{sig}^2(v)\right)$$

- **Función tangente hiperbólica**:

$$f_{tanh}(v) = \frac{e^{\cdot v} - e^{-\cdot v}}{e^{\cdot v} + e^{-\cdot v}} = 2f_{sig}(2v) - 1 \qquad f_{tanh} \in (-1,1)$$

3.2 Arquitectura

La Arquitectura de una Red Neuronal especifica las interconexiones llevadas a cabo entre las diversas neuronas de la red. Estas conexiones marcan las dependencias e influencias entre neuronas y utilizan los pesos de entrada para indicar el grado de dichas influencias.

Como cualquier otro sistema, la RNA debe tener entradas y salidas, las entradas del sistema son las dendritas de algunas neuronas y las salidas son los axones de otras. Según el tipo de conexiones que poseen, existen 3 tipos de neuronas:

- Neuronas de entrada: Aquellas que tienen como entrada alguno de los inputs del sistema.
- Neuronas intermedias: Aquellas que sólo tienen como entrada la salida de otras neuronas.
- Neuronas de salida: Aquellas cuya salida es parte de la salida del sistema.

Una neurona puede ser de cualquier combinación de los tipos y para que sea un sistema, por lo menos deben existir neuronas de los tipos 1 y 3.

Las neuronas se suelen agrupar en conjuntos llamados **capas** (layers), estas capas están formadas por un conjunto de neuronas que siguen el mismo tipo de geometría de interconexión y modelo neuronal. Normalmente una RNA se encuentra dividida en capas donde cada una de ellas tiene una función específica dentro de la red.

Existen diferentes tipos de Arquitecturas en función de las diversas características que pueden tener las conexiones, según el flujo de las señales entrada por la red, los más importantes son:

- **Feed-Forward Networks (FFNN)**: Aquellas RNA donde las señales de entrada viajan en una sola dirección, desde la entrada hacia la salida. No poseen realimentación ni bucles, la salida de una neurona nunca es entrada de otra de neurona de la misma capa o anterior. El tipo de FFNN más extendida es la SLFN, que consiste básicamente en 3 capas, una de entrada, una de salida y otra intermedia.
- **Feed-Back Networks (FBNN):** Aquellas RNA donde las señales de entrada viajan en ambas direcciones, de entrada a salida y de salida a entrada mediante el uso de bucles entre neuronas. Este tipo de RNA son muy poderosas y dinámicas, su "estado" está continuamente cambiando hasta que alcanzan el punto de equilibrio. Cuando se cambia el input, el estado vuelve a cambiar y un nuevo equilibrio debe ser encontrado.

La Arquitectura de RNA más utilizada es la FFNN (Feed Forward Neural Network), en la cual el flujo de información es unidireccional y está formada por 3 tipos de capas:

- **Input Layer** (Capa de Entrada): Es la primera capa de la RNA está formada por neuronas que tienen como entrada los Input del sistema. Suele ser una capa imaginaria de acondicionamiento de la señal donde cada Neurona tiene como única entrada un parámetro y su función de transferencia es lineal.

- **Output Layer** (Capa de Salida): Es la última capa de la RNA está formada por neuronas que tienen como salida los Input del sistema. Posee tantas neuronas como salidas y su suelen realizar la función de acondicionamiento de señal de salida.

- **Hidden Layer** (Capa Oculta): Conjunto de capas entre la Capa de Salida y la de Entrada.

3.2.1 SLFN

Una SLFN (single-hidden layer feedforward neural networks) es una FFNN que posee una única capa intermedia oculta. Su estructura general se muestra en la siguiente figura.

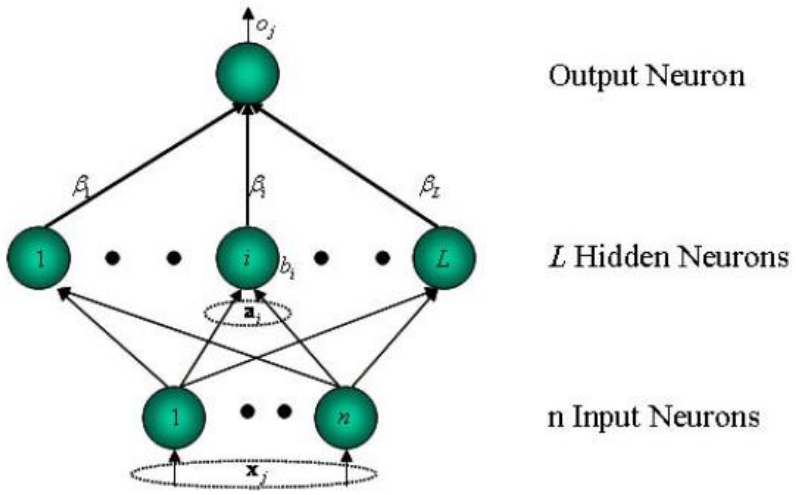


Fig. 22 SLFN. Fuente [7].

Es uno de los modelos más utilizados por su fácil entrenamiento y propiedades. Huang and Babri [13] demostraron que una SLFN con N neuronas ocultas como máximo, con una función de transferencia diferenciable no lineal puede aprender exactamente N observaciones distintas.

En el modelo más general, cada neurona de una capa la Capa Oculta tiene como entrada todas las neuronas de la Capa de Entrada y cada neurona de la Capa de Salida tiene como entrada cada neurona de la Capa Oculta. A continuación veremos las propiedades generales de las 3 capas.

- **Capa de Entrada:**

Generalmente cada una de estas Neuronas posee una única entrada x_i cuya única función es la de realizar un acondicionamiento lineal del parámetro de entrada mediante su peso w y offset b. La salida de una neurona de esta capa puede expresarse como:

$$o = x_i \cdot w + b$$

- **Capa de Salida:**

La Capa de Salida suele tener una única neurona, si el sistema posee varias salidas se suele crear una RNA por cada una de ellas, de esta forma los pesos de las neuronas pueden especializarse para cada output específico. Esta neurona tiene por entrada cada una de las Neuronas Ocultas y es la más importante del sistema ya que con su vector de pesos $\beta = [\beta_1, \beta_2, \dots, \beta_L]$ especifica en gran medida el comportamiento del mismo. Suele tener como función de transferencia una función lineal que puede ser utiliza junto con el bias b para el acondicionamiento de la señal de salida.

Sea $\beta = [\beta_1, \beta_2, \dots, \beta_L]$ el vector de pesos de entrada y sea o la salida de la neurona, y por tanto, la salida del sistema. Tenemos que para una SLFN con L Neuronas Ocultas, la salida para una entrada $\bar{X}$ se puede expresar como:

$$o = \beta_i \sum_{j=1}^{L} u_j = \beta_i \sum_{j=1}^{L} f_A\left(\bar{X} \cdot \overline{W_j} + b_j\right)$$

- **Capa Oculta:**

Esta capa está formada por un grupo de L Neuronas Ocultas que poseen las propiedades que el diseñador considere oportunas. Sea una SLFN con n parámetros de entrada y L neuronas ocultas, la salida de cada una de las Neuronas Ocultas 'j' con parámetros $\left(\overline{W_j} = [w_{j,1}, w_{j,2}, \dots, w_{j,N}], b_j, f_A(u)\right)$ para un vector de entrada $\bar{X} = [x_1, x_2, \dots, x_N]$ es:

$$u_j = f_A\left(\bar{X} \cdot \overline{W_j} + b_j\right) = f_A\left(\sum_{i=1}^{n} x_i \cdot w_{j,i} + b_j\right) \qquad i = 1, \dots, n$$

Dado un conjunto de N vectores de entrada, cada vector i-ésimo $\overline{X_\iota}$ tiene la forma:

$$\overline{X_\iota} = \left[x_{i,1}, x_{i,2}, \dots, x_{i,n}\right] \in R^n, \qquad i = \ 1, \dots, N$$

Sea $u_{i,j}$ la salida de la neurona oculta j para el vector de entrada i podemos definir la matriz H como:

$$H(W, X, b) = \begin{bmatrix} u_{1,1} & \dots & u_{1,\mathrm{L}} \\ \dots & \dots & \dots \\ u_{N,1} & \dots & u_{N,\mathrm{L}} \end{bmatrix}_{N x \mathrm{L}}$$

La matriz H es la **"Matriz de Salida de la Capa Oculta"** y representa la respuesta de cada una de las Neuronas Ocultas para cada uno de los vectores de entrada.

- Cada fila es la salida de todas las Neurona Oculta para un vector de entrenamiento 'i' dado.
- Cada columna es la salida de una Neurona Oculta 'j' para cada uno de los vectores de entrada.

- **Ecuaciones de la SLFN:**

Cada vector de entrada $\overline{X_\iota}$ tiene su vector de salida asociado $\overline{T_\iota}$, formando la pareja entrada-salida $\{\overline{X_\iota}, \overline{T_\iota}\}$. Si la SLFN tiene '$m$' Neuronas de Salida, entonces los vectores de resultados tienen la forma:

$$\overline{T_\iota} = \left[t_{i,1}, t_{i,2}, \dots, t_{i,m}\right] \in R^m, i = \ 1, \dots, N$$

El conjunto de estos vectores da lugar a la matriz de salida $\boldsymbol{T}_{Nxm}$.

Cada una de las neuronas de salida tendrá su propio vector de pesos de entrada $\overline{\beta_\iota}$:

$$\overline{\beta_\iota} = \left[\beta_1, \beta_{i,2}, \dots, \beta_{i,\mathrm{L}}\right] \in R^{\mathrm{L}}, i = \ 1, \dots, m$$

El conjunto de estos vectores da lugar a la matriz de pesos de salida $\boldsymbol{\beta}_{\mathrm{L}xm}$.

Idealmente, los parámetros $\{W, b, f_A(u)\}$ de la SLFN deberían ser tales que:

$$\boldsymbol{H}_{N x \mathrm{L}}\, \boldsymbol{\beta}_{\mathrm{L}xm} \ = \ \boldsymbol{T}_{Nxm}$$

Siendo dichas matrices:

$$H(\overline{w_1}, \dots, \overline{w_{\mathrm{L}}}, \quad b_1, \dots, b_{\mathrm{L}}, \overline{x_1}, \dots, \overline{x_N}) = \begin{bmatrix} f_A(\overline{w}_1 \cdot \overline{x_1} + b_1) & \dots & f_A(\overline{w}_{\mathrm{L}} \cdot \overline{x_1} + b_{\mathrm{L}}) \\ \dots & \dots & \dots \\ f_A(\overline{w}_1 \cdot \overline{x_{\mathrm{N}}} + b_1) & \dots & f_A(\overline{w}_{\mathrm{L}} \cdot \overline{x_{\mathrm{N}}} + b_{\mathrm{L}}) \end{bmatrix}_{N x \mathrm{L}}$$

$$\boldsymbol{\beta} = \begin{bmatrix} \beta_1^m \\ \dots \\ \beta_L^m \end{bmatrix}_{L x \mathrm{m}} \quad \text{y} \quad \boldsymbol{T} = \begin{bmatrix} T_1^m \\ \dots \\ T_N^m \end{bmatrix}_{N x \mathrm{m}}$$

3.3 Algoritmos de aprendizaje

Los algoritmos de aprendizaje son aquellos que se utilizan para dar valor a los parámetros de las neuronas (normalmente sus pesos y bias), definiendo así la función del sistema. Su objetivo es conseguir una RNA entrenada que generalice bien, comportándose correctamente ante entradas nuevas, es decir que el error entre la salida proporcionada por la red y el esperado sea el mínimo posible. El algoritmo de aprendizaje de la red depende en gran medida de la arquitectura de la misma, estos algoritmos se suelen diseñar para una arquitectura en concreto.

Todos los Algoritmos de Aprendizaje usados para RNA adaptativas pueden ser clasificados en 2 categorías principales:

- **Supervised learning:** Incorpora un supervisor externo, de manera que cada Neurona de Salida sabe la respuesta deseada para cada vector de entrada $\overline{X_\iota}$. Durante el proceso de aprendizaje puede requerir más información. El objetivo es determinar el conjunto de pesos y bias que minimizan el error de entrenamiento.
- **Unsupervised learning:** No incorpora un supervisor externo si no que se basa solamente en información local. La red por si misma auto-organiza los datos que se le presentan y detecta sus propiedades colectivas.

Se dice que una red aprende offline si la fase de aprendizaje y la de utilización son distintas y online si aprende y opera al mismo tiempo. Supervised learning es offline y unsupervised learning es online.

El objetivo final del aprendizaje es minimizar lo máximo posible el error de la salida de la red con respecto al valor deseado. Existen 2 tipos de error que podemos identificar:

- Error de entrenamiento (**Training Error TRe**): Es el error que presenta la red para el set de entrenamiento $\{\overline{X_\iota}, \overline{T_\iota}\}$. Es el error que se pretende minimizar durante el aprendizaje.
- Error de testeo (**Testing Error TEe**): Es el error que se obtiene cuando presentamos nuevas muestras a la RNA una vez está entrenada. Es el verdadero indicador de calidad.

Durante el entrenamiento se pretende minimizar el TRe con la esperanza de después, a la hora de utilizar la RNA se obtenga un buen TEe. Sin embargo puede darse el caso del sobreentrenamiento (overfitting) en el cual los parámetros de la red están demasiado adaptados al set de entrenamiento, generando un TRe muy bajo pero no capaz de generalizar bien ante nuevas entradas, produciendo un TEe muy alto, más que para una RNA con un TRe mayor.

La forma más común de expresar el error, ya sea de Training o de Testing es mediante el Error Cuadrático Medio (Mean Square Error **MSE**) que se define como:
Sea el vector $\overline{Y}^N$ de estimadores con N predicciones y sea $\overline{T}^N$ el vector con los valores a predecir, el MSE se define como:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(Y_i - T_i)^2$$

En este documento se utiliza el error Root Mean Square Error **RMSE**, la raíz cuadrada del MSE.
Para una RNA, el estimador es el vector de resultados de la misma, y el valor estimado su valor real. Para una red de K neuronas de salida y aplicando N vectores de entrenamiento, el error de entrenamiento es:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{1}{K}\sum_{k=1}^{K} e_k^2\big(o_k(i) - t_k(i)\big)^2\right)$$

3.3.1 Back-Propagation Algorithm

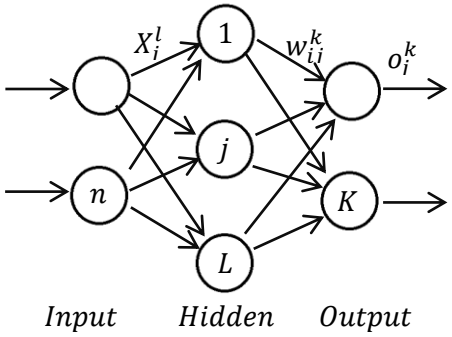
El algoritmo de aprendizaje clásico de las FFNN es un método basado en el gradiente descendente llamado Back-Porpagation (BP). Este se basa en actualizar los pesos de las neuronas iterativamente en base al gradiente descendiente del error de la red. Esto es posible gracias a que la función de transferencia utilizada es una función continua de los pesos, diferenciable en cualquier punto. En cada iteración del algoritmo, cada peso de la red se desplaza un paso (step) en la dirección de máximo decrecimiento del error total de la red E, cuya direccion es el opuesto del gradiente del mismo.

$$w_{ji} = w_{ji} + \Delta w_{ji}$$

El proceso iterativo termina cuando el error cuadrático medio es mejor que un umbral como 0.01.
El algoritmo BP aprende de la misma manera que lo hace una sola neurona, busca valores de los pesos que minimicen el error total de la red para un conjunto dado de vectores de training. Consiste en la repetida aplicación de las 2 siguientes fases:

- **Forward pass**: En este paso, introducimos a la RNA uno de los input $\overline{X_\iota}$ y se calcula el error de cada una de las neuronas de la capa de salida.
- **Backward pass**: En este paso, el error de la red es utilizado para actualizar el valor de los pesos. El error se propaga de salida a entrada, capa por capa. Esto se hace calculando recursivamente el gradiente local para cada neurona.

A continuación se desarrollará el cálculo del gradiente del error de la red E respecto a cualquier peso w_{ji} de la misma. El desarrollo es válido para cualquier FFNN.



Notación:
X_j^l : Input del nodo j de la capa l
w_{ij}^l : Peso de entrada del nodo j de la capa l para su entrada i. La entrada será la salida del nodo i.
$f(v)$: Función de transferencia sigmoide.
b_j^l: Bias del nodo j de la capa l.
o_j^l: Salida del nodo j de la capa l.
t_j : Salida deseada del nodo j de la capa de salida.

Fig. 23 SLFN

El error de una neurona de salida k para el n-ésimo Vector de training $\overline{X_n}$es:

$$e_k(n) = o_k(n) - t_k(n)$$

Para una RNA con K neuronas de salida, el error de la red es el MSE del vector de salida:

$$E(n) = \frac{1}{2}\sum_{k=1}^{K} e_k^2(n) = \frac{1}{2}\sum_{k=1}^{K}\big(o_k(n) - t_k(n)\big)^2$$

El MSE total para un el set de training de N vectores es:

$$E_{AV} = \sum_{n=1}^{N} E(n)$$

Nuestro objetivo es minimizar este error.

Una vez calculado el error para un Vector de Training $'n'$, $E(n)$, al que por ahora denotaremos únicamente como E, tenemos que actualizar todos los pesos en función de cómo afectan a este error. Para saber cómo afecta cada peso w_{ij}^{l} al error E, realizamos el gradiente del mismo respecto al peso. Este cálculo difiere para las neuronas de salida y aquellas que forman parte de capas ocultas:

- **Gradiente de E respecto a un peso de la capa de salida w_{ik} :**

Utilizando la regla de la cadena se tiene que:

$$\frac{\partial E}{\partial w_{ik}} = \frac{\partial E}{\partial o_k} \cdot \frac{\partial o_k}{\partial w_{ik}} = \frac{\partial}{\partial o_k}\left(\frac{1}{2}\sum_{m=1}^{K}(o_m - t_m)^2\right)\frac{\partial o_k}{\partial w_{ik}} = (o_k - t_k)\frac{\partial o_k}{\partial w_{ik}}$$

El sumatorio desaparece dado que solo una de las neuronas de salida se ve afectada por el peso. Siendo la salida o_k una función sigmoide dependiente de los pesos, entradas y bias. Utilizando otra vez la regla de la cadena, y utilizando las propiedades vistas de la sigmoide tememos que:

$$\frac{\partial E}{\partial w_{ik}} = (o_k - t_k)\frac{\partial f_{sig}(v)}{\partial w_{ik}} = (o_k - t_k)\frac{\partial f_{sig}(v_k)}{\partial v} \cdot \frac{\partial v}{\partial w_{ik}} = (o_k - t_k) f_{sig}(v_k)\left(1 - f_{sig}(v_k)\right)\frac{\partial v_k}{\partial w_{ik}}$$

Dado que v es un sumatorio ponderado de cada entrada con su peso, tenemos finalmente que:

$$\frac{\partial E}{\partial w_{ik}} = (o_k - t_k) f_{sig}(v_k)\left(1 - f_{sig}(v_k)\right) x_{ik}$$

Siendo $f_{sig}(v_k) = o_k$ y x_{ik} la entrada i-esima de la neurona de la salida k, si esa entrada viene de otra neurona i de la capa anterior, la salida de dicha neurona será $o_i = x_{ik}$. Quedando la ecuación como:

$$\frac{\partial E}{\partial w_{ik}} = (o_k - t_k) o_k (1 - o_k) o_i$$

Denominamos al delta δ_k como:

$$\delta_k = (o_k - t_k) o_k (1 - o_k) \quad \rightarrow \frac{\partial E}{\partial w_{ik}} = \delta_k \cdot o_i$$

Por lo que la derivada del error respecto al peso i, de la neurona k de la capa de salida es:

$$\frac{\partial E}{\partial w_{ik}} = \delta_k \cdot o_i$$

Para las neuronas de la Capa Oculta, el procedimiento es parecido, pero la diferencia es que dado que la salida de la red no depende directamente de los pesos de la capa oculta, sino de la salida de sus neuronas, no se puede eliminar el sumatorio $\sum_{k=1}^{K}(o_k - t_k)$ porque un peso de una neurona de una capa oculta, afecta a todas las neuronas de salida k, afectando por tando al error total desde cada una de ellas. En el otro caso, un peso de una neurona de salida solo afectaba al error a partir de dicha neurona.

- **Gradiente de E respecto a un peso i de un nodo de la capa oculta j w_{ij} es:**

Utilizando el mismo procedimiento que en el apartado anterior tenemos que:

$$\frac{\partial E}{\partial w_{ij}} = \sum_{k=1}^{K} \left((o_k - t_k) f_{sig}(v_k) \left(1 - f_{sig}(v_k)\right) \frac{\partial v_k}{\partial \boldsymbol{w_{ij}}} \right)$$

Volvemos a utilizar la regla de la cadena nuevamente, usando como elemento intermedio la salida del nodo al que afecta el peso, es decir, la salida $o_j = f_{sig}(v_j) = f_{sig}(X_j \cdot W_j)$. Esto tiene la ventaja que la salida del nodo o_j es una entrada de la neurona de salida, por lo que la derivada de v_k respecto a o_j es simplemente el peso que asocia el nodo k, a esa entrada y que llamaremos $\boldsymbol{w_{ik}}$. Tenemos que

$$\frac{\partial v_k}{\partial \boldsymbol{w_{ij}}} = \frac{\partial v_k}{\partial o_j} \frac{\partial o_j}{\partial w_{ij}} = \boldsymbol{w_{ik}} \frac{\partial o_j}{\partial w_{ij}}$$

La segunda derivada $\frac{\partial o_j}{\partial w_{ij}}$ es igual que la anterior $\frac{\partial o_k}{\partial w_{ik}}$ por lo que:

$$\frac{\partial o_j}{\partial w_{ij}} = \left(o_j(1 - o_j) \cdot x_{ij}\right) = \left(o_j(1 - o_j) \cdot o_i\right)$$

Esta parte no depende de las neuronas de salida por lo que podemos sacarla fuera del sumatorio:

$$\frac{\partial E}{\partial w_{ij}} = o_i \cdot o_j(1 - o_j) \sum_{k=1}^{K} \left((o_k - t_k) o_k (1 - o_k) \cdot w_{ik} \right)$$

Reemplazando el valor de $\delta_k = (o_k - t_k) o_k (1 - o_k)$ tenemos que:

$$\frac{\partial E}{\partial w_{ij}} = o_i \cdot o_j(1 - o_j) \sum_{k=1}^{K} (\delta_k \cdot w_{ik})$$

Llamamos al delta δ_j como:

$$\delta_j = o_j(1 - o_j) \sum_{k=1}^{K} (\delta_k \cdot w_{ik})$$

Resultando que:

$$\frac{\partial E}{\partial w_{ij}} = \delta_j \cdot o_i \quad siendo \ \delta_j = o_j(1 - o_j) \sum_{k=1}^{K} (\delta_k \cdot w_{ik})$$

Esta ecuación puede utilizarse para cualquier capa intermedia j, siendo k, la siguiente capa a esta. Por lo que para calcular el error del peso de una capa, primero tenemos que calcular las deltas de las capas entre esta capa y la salida, empezando por la salida.

- **¿Cómo varía el error respecto a los bias?**

Podemos considerar el bias como una entrada más que no se multiplica por ninguna señal de entrada, por lo que, a diferencia de cualquier peso, cuya derivada será la entrada o_i tendremos que:

$$\frac{\partial v_j}{\partial b} = 1 \qquad \rightarrow \qquad \frac{\partial E}{\partial b_j} = \delta_j$$

Una vez realizado el desarrollo matemático, el BP consiste en los siguientes pasos:

1. Introducir un Vector de Training en la Red y transportarlo hasta la salida para obtener el output o_k de las neuronas de salida.
2. Para cada nodo de salida (capa k) calcular su delta:

$$\delta_k = (o_k - t_k)o_k(1 - o_k)$$

3. Para cada nodo intermedio (capa j y capa siguiente k) calcular su delta:

$$\delta_j = o_j(1 - o_j)\sum_{k=1}^{K}(\delta_k \cdot w_{ik})$$

4. Actualizar los pesos i y bias de todas las neuronas de cada capa j-esima como:

$$w_{ji} = w_{ji} + \Delta w_{ji} \quad y \quad b_j = b_j + \Delta b_j$$

Siendo:

$$\Delta w_{ji} = -\eta \cdot \delta_l \cdot o_{l-1} \quad y \quad \Delta b_j = -\eta \cdot \delta_l$$

El parámetro η se llama el Learning Rate y se utiliza para controlar la velocidad del aprendizaje.
Generalmente, los pesos iniciales son aleatoriamente elegidos con valores típicos entre -1.0 y 1.0.
El número de Vectores de training debe ser al menos 5-10 veces mayor que el número de pesos de la red. Otra regla es:

$$N > \frac{W}{1 - a}$$

Donde W, es el número total de pesos y a es la precisión de resultado que deseamos.

❖ **Pseudocódigo.**

Tabla 2 Pseudocódigo algoritmo BP

n = 1;
Inicializar los pesos w_{ji} aleatoriamente;
while (Condición de parada o n < max_iteraciones)
    for (Cada Training Vector $\{\overline{X_\iota}, \overline{T_\iota}\}$)
        Introducir el vector $\overline{X_\iota}$ y propagarlo hasta la salida para obtener el resultado $\overline{O_\iota}$
        Calcular el valor de la delta δ_j para cada neurona de la red de salida a entrada.
        Actualizar el valor de los pesos w_{ji} de cada neurona, empezando por la cada de salida::

$$w_{ji} = w_{ji} + \Delta w_{ji}$$

    end
    n = n+1;
end

Inconvenientes del BP:

- Cuando el Learning rate η es demasiado pequeño, el algoritmo puede converger demasiado despacio y cuando es demasiado grande, el algoritmo es inestable y diverge.
- Suele estancarse en mínimos locales.
- La RNA puede resultar sobre-entrenada para los datos que le hemos introducido dado que utiliza el error de training, obteniendo un peor rendimiento por lo que se requiere una condición de parada que tenga en cuenta esto.
- Requiere demasiado tiempo de cómputo para la mayoría de aplicaciones.

3.3.2 Extreme Learning Machine

Extreme Learning Machine (Huang et al., 2006) (ELM) es un algoritmo de aprendizaje para Single-hidden Layer Feedforward neural Networks (SLFNs) que elije aleatoriamente los pesos y bias de las Neuronas Ocultas de la red y determina analíticamente los pesos de salida. Este algoritmo suele proporcionar una buena generalización a una velocidad extremadamente rápida, miles de veces mayor que otros algoritmos de entrenamiento para la misma red.

Tradicionalmente se han utilizado algoritmos como el BP donde todos los parámetros de las neuronas debían ser tuneados. Los algoritmos basados en el gradiente tienen el problema de requerir tiempos de entrenamiento demasiado grandes y quedarse atrapados en óptimos locales.

Ha sido demostrado que una SLFN con L Neuronas Ocultas con valores aleatorios de pesos de entrada y bias puede aprender N observaciones distintas. Se puede demostrar que no es necesario tunear dichos valores de la red para dar un buen resultado generalizado y conseguir un error de entrenamiento 0 [12]. Los algoritmos usuales de aprendizaje de la SLFN requerían encontrar valores específicos para todos los pesos y bias de las neuronas lo que convierte el entrenamiento en algoritmo de optimización de estos valores complejo y costoso. El error de entrenamiento para una SLFN queda definido como:

$$Training_{error} = \|H\beta - T\|$$

Gracias al **Teorema 2.1**, al contrario de lo que se pueda pensar, sabemos que podemos dar valores aleatorios a $\boldsymbol{W}$ y $\overline{b}$, quedando definida la matriz $\boldsymbol{H}$ y aún así encontrar una óptima solución del sistema. El problema se reduce a calcular el vector de pesos $\boldsymbol{\beta}$ pudiendo resolverse como sistema lineal, reduciendo el entrenamiento a encontrar la solución de mínimos cuadrados (LSQS) $\hat{\beta}$ del sistema $H_{Nx\mathrm{L}}\,\beta_{\mathrm{L}xm} = T_{Nxm}$.

$$Encontrar\ \hat{\beta}\ tal\ que\ \left\|H\hat{\beta} - T\right\| = \min_{\beta}\|H\beta - T\|$$

Cuando el número de neuronas L es igual al número de vectores de entrenamiento N, la matriz H es una matriz cuadrada e invertible y existe una solución $\hat{\beta}$ tal que:

$$\mathrm{H}\beta = \mathrm{T}$$

Sin embargo L tiende a ser mucho menor que N, bajo estas circunstancias es posible que no exista una solución $\{\overline{W_i}, b_i, \beta_i\}(i = 1, \dots, L)\ tal\ que\ \mathrm{H}\beta = \mathrm{T}$. En estos casos, la solución de mínimos cuadrados se obtiene resolviendo el siguiente sistema lineal.

$$\hat{\beta} = H^{\dagger}T$$

Donde la matriz $H^{\dagger}$ es la matriz inversa de Penrouse-Moore de la matriz H y posee las siguientes propiedades útiles a nuestro problema:

- La solución $\hat{\beta} = H^{\dagger}T$ es una de las soluciones de mínimos cuadrados del sistema $\mathrm{H}\beta = \mathrm{T}$ lo que se traduce al mínimo error de training.

- Genera la solución de mínimos cuadrados $\hat{\beta}$ con la menor norma.

Así pues Huang et al. (2006) [13], definen el algoritmo de la ELM como sigue:

Dado el conjunto de entrenamiento:

$$N = \{(x_i, t_i) \mid x_i \in R^n, t_i \in R^m, i = 1, \ldots, N\}$$

y una SLFN con función de transferencia $f_A(x)$ y un número L de Neuronas Ocultas:

1) Asignar valores aleatorios a la matriz de pesos $\boldsymbol{W}$ y el vector de bias $\bar{b}$
2) Calcular la matriz de salida de Capa Oculta $\boldsymbol{H}$.
3) Calcular los pesos de salida β, usando la ecuación $\beta = H^{\dagger}T$,

Especificaciones:

- Normalizar pesos, bias y entradas.
- Funciones de transferencia recomendadas:
  Puede ser utiliza cualquier función infinitamente diferenciable, normalmente se suele utilizar la función sigmoide pero también se puede probar el seno, coseno, hiperbólica, tangente, exponencial… La diferencia de rendimiento y precisión de estas funciones no es muy significativa y varía según el problema.
- Cálculo de la matriz inversa se Penrouse-Moore $H^{\dagger}$:
  Existen diversos métodos para calcular $H^{\dagger}$, como pueden ser la ortogonalización, métodos iterativos o Singular Value Decomposition (SVD). En este documento se utilizará el algoritmo "geninv" desarrollado por Pierre Courrieu [10].

La ELM se basa en los siguientes teoremas:

- **Teorema 1:**

Dada una SLFN estándar con N Neuronas Ocultas y función de activación $g: R \to R$ que es infinitamente diferenciable en cualquier intervalo, con N instancias arbitrarias (x_i, t_i) donde $x_i \in R^n$ y $t_i \in R^m$. Para cualquier valor aleatorio de w_i y b_i de cualquier intervalo de R^n y R respectivamente, de acuerdo a una función estadística con distribución contínua, existe un número de Neuronas Ocultas $L < N$ que con probabilidad 1, la matriz H de la SLFN es invertible y $\|H\beta - T\| = 0$.

- **Teorema 2:**

Dado cualquier número positivo $\varepsilon > 0$ tan pequetan pequeño como queramos y dada una SLFN estándar con N Neuronas Ocultas y función de activación $g: R \to R$ que es infinitamente diferenciable en cualquier intervalo, con N instancias arbitrarias (x_i, t_i) donde $x_i \in R^n$ y $t_i \in R^m$. Para cualquier valor aleatorio de w_i y b_i de cualquier intervalo de R^n y R respectivamente, de acuerdo a una función estadística con distribución constante, entonces con probabilidad 1 se da que existe un vector de pesos β tal que $\|H\beta - T\| < 0$.

Capítulo 4
Optimización Discreta

4. OPTIMIZACIÓN DISCRETA

Un problema de optimización discreta DOP (Discrete Optimization Problem) es un problema de minimización o maximización de una función f que actúa sobre un conjunto de valores discretos S.
Un DOP puede ser expresado como una tupla (S, f) siendo:

- S: Es el conjunto de todas las finitas o infitinas contables soluciones del problema. Este espacio de soluciones se le denomina **Search Space**.
- f: Es la **función de coste** (cost function) que mapea cada elemento del conjunto S a un número real que indica la calidad de la solución. También se le denomina función objetivo.

$$f: S \rightarrow \mathbb{R}$$

Nuestro objetivo a la hora de resolver un DOP es encontrar una solución válida $x_{opt} \in S$ que sea óptimo global, es decir, que sea la mejor de entre las posibles soluciones.

$$f(x_{opt}) \leq f(x) \ \forall \ x \in S \qquad minimización$$
$$f(x_{opt}) \geq f(x) \ \forall \ x \in S \qquad maximización$$

Existen diversas formas de atacar un DOP y el modelo matemático con el que expresemos el problema influye significativamente en la forma de resolverlo. Un mismo problema puede ser expresado de diferentes formas para ser utilizado por diferentes algoritmos.

Matemáticamente, la optimización es definida como la búsqueda de un conjunto de parámetros llamados "variables de decisión" ($\bar{x} = \{x_1, x_2, x_3, \ldots x_n\}$) que minimice o maximice una cantidad ordinal T (coste o puntuación) asignada por una función objetivo (de coste) f bajo una serie de restricciones ($g = \{g_1, g_2, g_3, \ldots g_m\}$).

Para resolver un problema de optimización normalmente viajamos por el espacio de búsqueda S de solución en solución hasta encontrar el óptimo global x_{opt}. Para ello, partimos de una solución x_i y nos desplazamos a otra solución cercana a ella x_j modificándola mediante una función de vecindad $\mathcal{V}(x_i)$ que devuelve un vecino de x_i.

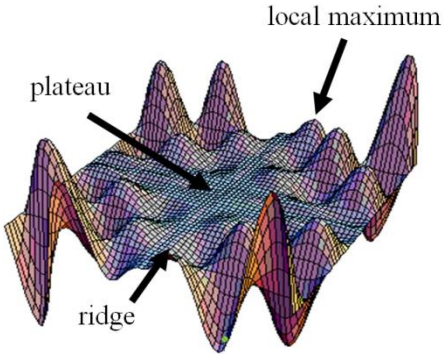


Fig. 24 Ejemplo Espacio de Búsqueda. Fuente [24].

La figura de la derecha representa un espacio de búsqueda bidimensional donde cada punto $x \in S$ tiene asociado su función objetivo $f(x)$, representada por la altura.

La función de vecindad $\mathcal{V}(x)$ determina el conjunto de soluciones al que podemos movernos desde una solución x_i dada. La vecindad de x_i es el subconjunto de soluciones $S_i \subseteq S$ que cumplen la relación $x_j = \mathcal{V}(x_i)$. En función de como sea la vecindad de una solución tenemos:

- **Óptimo local**: Si todas las soluciones de su vecindad son peores.
- **Plano**: Si todas las soluciones de su vecindad son iguales que ella.

A la hora de realizar la búsqueda tenemos que implementar mecanismos que nos permitan escapar de mínimos locales por lo que no debemos movernos únicamente a soluciones mejores. Si se puede, también se suelen explotar propiedades del Search Space como por ejemplo, si es continuo, movernos en la dirección del gradiente de la función de costo con respecto a sus variables de decisión $\nabla f(x)$. Desgraciadamente este no el caso del Search Space del problema de este documento.

4.1 Búsqueda local

Un **Algoritmo de búsqueda local** es aquel que a partir de una solución del problema x_i, realiza cambios en ella generando otra solución en su vecindad x_j, con objeto de encontrar mejores soluciones. El algoritmo elije adoptar esa nueva solución como solución actual o no a partir de la diferencia de la función de costo $\Delta f = f(x_j) - f(x_i)$ y otros parámetros de cada algoritmo en concreto. Una búsqueda local es la que basa su estrategia en el estudio de soluciones del **vecindario** o **entorno** de la solución que realiza el recorrido.

Elementos de la búsqueda local:

- **Representación de la solución:**
  Es la forma en la que expresamos las posibles soluciones del sistema. Normalmente se suelen representar como un conjunto de números reales o de bits.

- **Función de Evaluación (costo u objetivo):** $f(x)$
  Es la función que actúa sobre una solución y nos devuelve la calidad de la misma.

- **Función de Vecindad:** $\mathcal{V}(x)$
  Función que actúa sobre una solución dada y la modifica dentro del espacio de soluciones factibles para producir otra en el espacio cercano a la primera. Sea una solución x_i dada, podemos encontrar una solución x_j cercana a ella realizando la operación:

  $$x_j = \mathcal{V}(x_i)$$

  Normalmente esta función introduce pequeños cambios aleatorios en x_i que generan otra solución factible. Si la solución se representa como un vector de bits, esta función podría ser invertir uno de ellos.

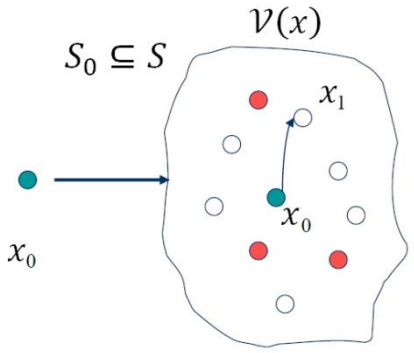


Fig. 25 Función de vecindad.

- **Función de aceptación:** $\mathcal{A}(\,)$
  Es la función que decide si aceptamos o no la nueva solución x_j como solución actual. Esta es la función que principalmente diferencia unos algoritmos de búsqueda local de otros y normalmente se basa en algún tipo de heurístico.

- **Condición de parada:**
  Es la situación que propicia el fin de la búsqueda, dando a la solución actual como la mejor que el algoritmo puede encontrar con los parámetros dados. Algunas condiciones son realizar un número dado de iteraciones o alcanzar un valor de función objetivo dado.

❖ **Pseudocódigo:**

Tabla 3 Pseudocódigo de Búsqueda Local

```
I = Solución Inicial
while (condición de parada)
        J = 𝒱(I)
        if (𝒜( ))
                I = J
        end
end
```

4.2 Heurística y Metaheurística

En un problema de optimización, aparte de las condiciones que deben cumplir las soluciones factibles del problema, se busca aquella solución que sea óptima, según algún criterio de comparación entre ellas. Dado que la mayoría de las veces el espacio de búsqueda es demasiado grande para realizar una búsqueda exhaustiva, es necesario realizar algoritmos que exploren el espacio de soluciones siguiendo algún tipo de lógica que los direccione hacia buenos subespacios de esta. Estos algoritmos se llaman heurísticos.

Podemos definir un algoritmo heurístico como:
Un procedimiento para el que se tiene un alto grado de confianza de que encuentra soluciones de alta calidad con un coste computacional razonable, aunque no se garantice su optimalidad o su factibilidad, e incluso, en algunos casos, no se llegue a establecer lo cerca que se está de dicha situación.

En AI, la palabra heurístico es un calificativo asociado a los procedimientos que, empleando conocimiento acerca de un problema y de las técnicas aplicables, tratan de aportar soluciones (o acercarse a ellas) usando una cantidad de recursos (generalmente tiempo) razonable.

Una solución heurística de un problema es aquella proporcionada por un método heurístico, es decir, aquella solución sobre la que se tiene cierta confianza de que es factible y óptima, o de que alcanza un alto grado de optimalidad y/o factibilidad.

Las heurísticas utilizadas para resolver un problema de optimización pueden ser más generales o específicas que otras.

- Los métodos heurísticos **específicos** están diseñados específicamente para cada problema, utilizando información sobre la estructura del mismo y el ***análisis*** teórico del modelo.
- Las heurísticas más **generales** en cambio son aplicables a casi cualquier problema.

Los procedimientos específicos bien diseñados suelen tener un rendimiento significativamente más alto que las heurísticas generales, pero estas presentan otro tipo de ventajas, como la sencillez, adaptabilidad, robustez...

Las **metaheurísticas** son estrategias generales de diseño de procedimientos heurísticos para la resolución de problemas con un alto rendimiento. Es decir, los algoritmos metaheurísticos son algoritmos heurísticos que presentan una inteligencia superior

De una u otra forma, todas las metaheurísticas se pueden concebir como estrategias aplicadas a **procesos de búsqueda**, donde todas las situaciones intermedias en el proceso de resolución del problema se interpretan como elementos de un espacio de búsqueda, que se van modificando a medida que se aplican las distintas operaciones diseñadas para llegar a la resolución definitiva.

Por ello, y porque los procesos de búsqueda heurística constituyen el paradigma central de las metaheurísticas, es frecuente interpretar que el término metaheurística es aplicable esencialmente a los procedimientos de búsqueda sobre un espacio de soluciones alternativas.

Existen principalmente 4 tipos de metaheurísticas:

- **Metaheurísticas de relajación:**

Se refieren a procedimientos de resolución de problemas que utilizan relajaciones del modelo original, es decir, modificaciones del modelo que hacen al problema más fácil de resolver, cuya resolución facilita la resolución del problema original. Una **relajación** de un problema es un modelo simplificado obtenido al eliminar, debilitar o modificar restricciones (u objetivos) del problema real. Otras estrategias modifican la función objetivo para obtener de forma más rápida, valoraciones aproximadas (por exceso o por defecto) de la calidad de la solución que orientan la búsqueda. En el caso de este proyecto, se llevará a cabo una relajación **reduciendo el número de neuronas** de las búsquedas iniciales.

- **Metaheurísticas constructivas:**

Se orientan a los procedimientos que tratan de la obtención de una solución a partir del análisis y selección paulatina de las componentes que la forman. Aportan soluciones del problema por medio de un procedimiento que **incorpora iterativamente elementos** a una estructura, inicialmente vacía, que representa a la solución. Las metaheurísticas constructivas establecen estrategias para seleccionar las componentes con las que se construye una buena solución del problema.
Entre las metaheurísticas primitivas en este contexto se encuentra la popular estrategia voraz o **greedy**, que implica la elección que da mejores resultados inmediatos, sin tener en cuenta una perspectiva más amplia.

- **Metaheurísticas de búsqueda:**

Guían los procedimientos que usan transformaciones o movimientos para recorrer el espacio de soluciones alternativas y explotar las estructuras de entornos asociadas. Establecen estrategias para recorrer el espacio de soluciones del problema **transformando** de forma iterativa **soluciones** de partida. Se suele asumir que las **búsquedas locales** sólo modifican la solución que realiza el recorrido mediante una **mejora en su propio entorno**. El principal inconveniente de estas búsquedas locales es que se quedan atrapadas en un óptimo local, una solución que no puede ser mejorada por un análisis local. Las búsquedas **Globales** son aquellas búsquedas que aceptan moverse a **soluciones peores** que la actual, escapando así de óptimos locales de baja calidad. Existen 3 formas fundamentales en las que estos algoritmos escapan de óptimos locales:

1. Volver a iniciar la búsqueda desde otra solución de arranque,
2. Modificar la estructura de entornos que se está aplicando.
3. Permitir movimientos o transformaciones de la solución de búsqueda que no sean de mejora, escapando así de los óptimos locales.

- **Metaheurísticas evolutivas:**

Están enfocadas a los procedimientos basados en conjuntos de soluciones que evolucionan sobre el espacio de soluciones. El aspecto fundamental de las heurísticas evolutivas consiste en la **interacción** entre los miembros de la población frente a las búsqueda que se guían por la información de soluciones individuales (Metaheurísticas de búsqueda).

4.3 Algoritmos de búsqueda estadísticos

Los algoritmos de optimización estocásticos son aquellos que utilizan la aleatoriedad para producir comportamientos no deterministas. La mayoría de algoritmos de los campos de Inteligencia Artificial y Metaheurística incorporar elementos aleatorios y pertenecen a este campo. Estos algoritmos no suelen ser aleatorios en comportamiento, si no que utilizan dicha **aleatoriedad para viajar por el espacio de soluciones** del problema, centrándose en zonas de interés y evadiendo zonas menos interesantes. Debe tenerse especial cuidado en asegurar la generación aleatoria de soluciones que no favorezca a ninguna parte específica del Search Space.

Los siguientes algoritmos estocásticos que describiremos son algoritmos de optimización global y metaheuristicos que utilizan algún tipo de procedimiento de búsqueda local. Se diferencian de algoritmos como el Temple Simulado o los Algoritmos Genéticos en que no poseen una analogía física o biológica asociada. Normalmente estos algoritmos únicamente requieren la capacidad de generar soluciones factibles del problema y de poder variar una solución dada para generar otra en su vecindad. Suelen tener también una condición de parada como un número máximo de iteraciones o una solución lo suficientemente buena.

4.3.1 Random Search

La búsqueda aleatoria es el algoritmo estocástico más simple. Consiste en simplemente generar soluciones aleatorias y quedarnos con la mejor. La búsqueda es totalmente ciega y sin memoria.

❖ **Pseudocódigo:**

Tabla 4 Pseudocódigo de Random Search

```
Candidate  = RandomSol()
Best = Candidate
while (Stop Cond)
        Candidate = RandomSol()
        if ( Cost (Candidate) < Cost (Best) )
                Best = Candidate
        end
end
return Best
```

4.3.2 Hill Climbing

Esta técnica se basa en generar una solución aleatoria inicial y seleccionar soluciones en su vecindad, si alguna de las soluciones en su vecindad es mejor, nos movemos a esa solución. El principal problema de este algoritmo es que se queda atrapado en óptimos locales.

❖ **Pseudocódigo:**

Tabla 5 Pseudocódigo de Hill Climbing

```
Candidate  = RandomSol()
Best = Candidate
while (Stop Cond)
        Candidate = Neighbor ( Best )
        if ( Cost (Candidate) < Cost (Best) )
                Best = Candidate
        end
end
return Best
```

4.4 Selección de características

La selección de características (Feature Selection FS) es un proceso de optimización discreta en el cual poseemos una serie de N características y una variable T relacionada con estas, siendo nuestro objetivo poder predecir el valor de T asociado a cada subconjunto de características (predictor) mediante algún Algoritmo de Aprendizaje (AA), también llamado clasificador.

Dado un conjunto de N características y una variable objetivo T, que podemos relacionar con cualquier subconjunto S de características mediante el Algoritmo de Aprendizaje $\mathcal{L}(S)$. El objetivo de la FS es encontrar un subconjunto S de N, $S \in N$, lo más reducido posible que consiga el mejor valor de T para el Algoritmo de Aprendizaje dado.

$$Encotrar\ S' \quad tal\ que \quad \min_{S} \|T - \mathcal{L}(S)\|$$

Un FSA (Feature Selection Algorithm) es un algoritmo de selección de características y está formado principalmente por:

- El conjunto de las N de características.
- La variable T objetivo.
- Algoritmo de Aprendizaje (AA). En nuestro caso la ELM.
- Técnicas de análisis sobre las características.

Podemos modelar un problema de selección de características como un DOP donde el espacio de búsqueda S es el conjunto de características a elegir. El espacio se soluciones puede ser modelado como un array de bits de selección con un bit por característica, donde un '1' significa utilizar la característica asociada y un '0' no utilizarla. Así pues en un problema con N parámetros tenemos un espacio:

$$S = \{0,1\}^N$$

Es espacio de soluciones tiene una dimensión de 2^N, por lo que para grandes tamaños de N, no podemos hacer búsqueda exhaustiva y necesitaremos realizar una búsqueda heurística.

A la hora de realizar un Algoritmo Metaheurístico para una selección de parámetros, existen una serie de preguntas que debemos hacernos [39]:

- ¿Pueden Input Redundantes ayudarse entre sí?
  - Inputs con prácticamente la misma información pueden ser útiles en conjunto debido a que pueden ayudar a reducir el ruido que poseen.

- ¿Qué relación existe entre la correlación de las variables y su capacidad de ayudarse?
  - Las variables perfectamente correladas son totalmente redundantes y no añaden más información al sistema.
  - Una correlación (o anti-correlación) muy alta no significa falta de complementariedad.

- ¿Puede una variable que es inútil por sí sola, ser útil con otras variables?
  - Una variable que es completamente inútil por si sola puede mejorar el error cuando es utilizada en conjunto con otras.
  - 2 o más variables que son inútiles por si solas pueden ser útiles juntas.

Importancia de la selección de características:

- Puede mejorar el rendimiento del algoritmo de clasificación.
- El algoritmo de clasificación puede no ser dimensionado para todo el conjunto de características por motivos de espacio o tiempo.
- Mejor entendimiento del dominio de características.
- Es más barato y ocupa menos espacio recoger datos para menos señales.

Existen 3 tipos de FSA en función de la relación entre la selección de características y el Algoritmo de Aprendizaje:

- **Filtros**: No utilizan el Algoritmo de Aprendizaje para la selección de características si no que se basan en datos estadísticos de las mismas. Se suele utilizar para una selección inicial cuando el espacio de búsqueda es demasiado grande.
- **Wrappers**: Utiliza el AA tratándolo como una caja negra.
- **Embebed methods**: El FSA interactúa con el AA tuneando sus variables internas.

Este documento se centra en los FSA de tipo Wrapper que utilizan una RNA entrenada mediante una ELM como Algoritmo de Aprendizaje. Se utilizan algoritmos de búsqueda metaheuristicos como Algoritmos Genéticos, Temple Simulado o Sistema Inmune Artificial para guiarla. También es posible realizar un filtrado inicial en base a las propiedades individuales y por parejas de cada característica, sin embargo filtrar las variables más relevantes suele llevar a predictores subóptimos debido a que estas variables pueden ser redundantes y que las malas variables individuales podrían complementarse bien con otras.

4.4.1 Filtros:

El filtrado consiste en una selección inicial de características en base a información que no provenga del Algoritmo de Aprendizaje, sino directamente de la relación entre las propias características (redundancia) y la salida a predecir. Los filtros son métodos de preprocesado de las características que intentan obtener información de las mismas para hacer una reducción inicial del conjunto cuando este es demasiado grande. No nos basamos en un Algoritmo de Aprendizaje, en vez de eso, seleccionamos las características en base a sus propiedades estadísticas y otras métricas como por ejemplo:

- Correlación
- Coeficiente de Pearse
- Ganancia de información.
- PCA

Ventajas:

- Escalable a vectores de entrada de muchas dimensiones.
- Computacionalmente rápido, simple e independiente del algoritmo de aprendizaje.
- Solo necesita ser utilizado una vez.

Contras:

- Ignoran la influencia de las características con el Algoritmo de Aprendizaje
- Normalmente son solo de una variable o de pocas variables, cada característica es evaluada por separado, ignorando por tanto las dependencias entre ellas lo que puede llevar a una peor predicción de la variable objetivo comparado con otras técnicas.
- Suelen producir overfitting.

4.4.2 Wrappers

La metodología Wrapper consiste en considerar el Algoritmo de Aprendizaje como una caja negra perfecta y usar su habilidad de predicción para asociar a cada subconjunto $S \in N$ una capacidad de predicción, realizando la selección en cada características en base a las características particulares de cada conjunto, sin obtener ni diferenciar información entre diferentes subconjuntos.

Si el espacio de soluciones es pequeño podemos utilizar búsqueda exhaustiva pero si no, el problema es NP-Hard lo que lo hace intratable. En tal caso tenemos métodos tanto deterministas como aleatorios para encontrar subconjuntos válidos.

- **Métodos estadísticos y metaheurísticos**: Son los algoritmos del tipo Hill-Climbing, Temple Simulado o Algoritmo Genéticos que se verán en aparados posteriores.
- **Métodos deterministas**: Las estrategias de búsqueda Greedy suelen ser computacionalmente rápidas y robustas contra el overfitting pero llevan a subconjuntos subóptimos.

Existen 2 métodos deterministas principales:

- **Forward Selection**: Empezamos con un subconjunto vacío y vamos añadiendo parámetros progresivamente formando grupos más grandes hasta que el añadir cualquiera de los parámetros restantes no produzca mejora.

- **Backward Elimination**: Consiste en empezar con el conjunto de todas las variables e ir eliminando parámetros uno a uno, empezando por aquellos que incrementen el error o solo lo decrementen un poco.

A continuación se describirán más en detalle 3 algoritmos deterministas.

1. **Forward selection (SFS)**:

Empezando con un conjunto vacío $C_0 = \{0\}$, se va añadiendo secuencialmente el parámetro x_i que resulte en la mejor función objetivo $\mathcal{L}(C_k + x_i)$ cuando se combina con el subconjunto ya formado. Realizar hasta el final, quedándonos con el mejor resultado. Para realizar este algoritmo, a cada iteración hay que probar todos los parámetros restantes con el subconjunto C_k actual.

❖ **Pseudocódigo:**

Tabla 6 Pseudocódigo de SFS

```
I = S{0};   Conjunto vacío.
while ( sizeof(I) < N )
        for (Cada característica x_i ∉ I )
                Calcular ℒ (I + x_i) y obtener el mejor x_i*
        end
        I = I + x_i*
end
```

Es el algoritmo determinista más simple, su principal desventaja de este algoritmo es que es incapaz de remover una característica que se vuelve obsoleta cuando añadimos otra. Funciona mejor con conjuntos pequeños. Posibles mejoras de este algoritmo son:

- Empezar el algoritmo con una población inicial aleatoria. Esto nos permitirá explorar más regiones del espacio de búsqueda.
- Cada cierto número de iteraciones, realizar una búsqueda para eliminar variables obsoletas.
- No aceptar siempre la mejor variable sino la primera que supere un cierto umbral aleatorio.

2. Sequential backward elimination (SBE)

Este algoritmo trabaja de forma opuesta al anterior, empezando con un conjunto lleno $C_0 = \{0\}$, ir removiendo secuencialmente el parámetro x_i que resulte en la mejor función objetivo $J(C_k - x_i)$ cuando se combina con el subconjunto ya formado. Realizar hasta el final, quedándonos con el mejor resultado. Para realizar este algoritmo, a cada iteración hay que probar todos los parámetros restantes con el subconjunto C_k actual.

❖ **Pseudocódigo:**

Tabla 7 Pseudocódigo de SBE

```
I = S{N};   Conjunto con todos los parámetros.
while ( sizeof(I) > 0 )
        for (Cada característica x_i ∈ I )
                Calcular ℒ (I − x_i) y obtener el mejor x_i*
        end
        I = I − x_i*
end
```

Funciona mejor con conjuntos grandes de características. Su principal desventaja es no poder realizar la reevaluación de una característica ya eliminada. Posibles mejoras de este algoritmo son:

- Empezar el algoritmo con una población inicial aleatoria. Esto nos permitirá explorar más regiones del espacio de búsqueda.
- Cada cierto número de iteraciones, realizar una búsqueda para añadir variables que ahora sí puedan ser de utilidad.
- No aceptar siempre la mejor variable sino la primera que supere un cierto umbral aleatorio.

3. Beam search and smart beam search (SBS):

Aunque el factor de dispersión de las características contiene información útil acerca de lo buenas o malas que son, es posible que características con bajos valores de I sean importantes para el AA. Es por ello que se diseñó un esquema de selección de características más genético llamado Beam Search. Este algoritmo funciona como sigue:

1. Calcular la capacidad de predicción de cada caraceterística por separado.
2. Seleccionar las mejores K características basándonos en algún algoritmo de selección, normalmente se seleccionan las K mejores.
3. Añadir una nueva característica a cada una de estas K características
4. Calcular la capacidad de predicción de cada una de las parejas y elegir las K mejores.
5. Repetir pasos 3 y 4 hasta que el criterio de parada. (Tamaño de los predictores = N)
6. Los mejores predictores de la búsqueda son el resultado del algoritmo.

❖ **Pseudocódigo:**

Tabla 8 Pseudocódigo de SBS

```
I = S{0};  Conjunto N Vectores de Selección vacíos
while ( Condición de parada )
        for (Cada VS de I)
                for (Cada característica x_n ∉ VS_i )
                        Calcular ℒ (I + x_n) y obtener el mejor x_n*
                end
        end
        VS = VS + x_n*
        I = Mejores K VS del conjunto
end
```

4.4.3 Taxonomía de Feature Selection:

La taxonomía de un algoritmo define donde encaja un determinado método dentro de un campo, como por ejemplo, el de la Inteligencia Computacional. La taxonomía proporciona un contexto para determinar las relaciones entre los distintos algoritmos. A continuación se exponen la taxonomía de los algoritmos de FS (Selección de Características).

Tabla 9 Taxonomía de FS

MODELO DE BUSQUEDA		**VENTAJAS**	**DESVENTAJAS**	**EJEMPLOS**
FILTRO	1 VARIABLE	Rápido, Escalable, Independiente del AP	Ignora dependencias entre las características y con el AP	X^2, Distancia Euclídea, t-test, Ganancia de Información.
	MULTI-VARIABLE	Modela dependencias entre características, Independiente del AP, Computacionalmente menos complejo que Wrapper	Más lento que las de 1 sola variable, menos escalable e ignora interactuación con el AP	Correlación (CFS), Markov Blancket Filter (MBF).
WRAPPER	DETERMINISTA	Simple, Interactúa con el AP. Modela dependencias entre características. Menos coste computacional que métodos aleatorios	Riesgo de overfitting. Mas tendencia a quedarse en óptimos locales que los aleatorios. Depende del AP	Secuential Forward Selection(SFS), Secuential Backward Elimination(SBE), búsqueda exahustiva (ES)
	ALEATORIO	Menos tendencia a quedarse en óptimos locales. Modela dependencia entre las características.	Computacionalmente lento y dependiente del AP. Más riego de overfitting que el determinista	Temple simulado, algoritmos evolutivos, hill-climbing, AIN.
EMBEBBED		Interactúa con el AP. Menos coste computacional que los wrapper. Modela dependencia de características.	Depende del AP	Árboles de decisión. Weighed naive Bayes, FS using the weight vector of SVM.

Este documento se centra principalmente en los algoritmos de tipo Wrapper utilizando técnicas metaheurísticas como son los Algoritmos Genéticos y el Temple Simulado. Como Algoritmo de Aprendizaje utiliza una Red Neuronal Artificial entrenada con el algoritmo Extreme Learning Machine.

Capítulo 5
Temple Simulado

5. TEMPLE SIMULADO

El "Temple Simulado" (Simulated Annealing) es un algoritmo metaheurístico y estocástico de búsqueda local utilizado en problemas de optimización discreta que poseen un gran Search Space y múltiples óptimos locales. Está inspirado en el proceso físico de enfriamiento controlado, (templado de metales) en el cual un metal se calienta a altas temperaturas y se enfría progresivamente de manera controlada. Si el enfriamiento es adecuado, los átomos que lo forman estarán dispuestos en una estructura regular con poca energía cinética, lo cual hace que aumenten las propiedades deseadas del material como la dureza o temperatura de fundición.

Fig. 26 Hefesto forjando una espada. Fuente [15].

Este proceso funciona a nivel atómico de la siguiente manera:
Cada átomo perteneciente el metal posee una cantidad movimiento aleatorio proporcional a la energía térmica que posee, cuanto más caliente esté el metal, más rango de movimiento tendrán sus átomos. Cada posición que puede poseer un átomo dentro de la red, lleva asociada una energía E, siendo deseable que esta sea la mínima posible. Así pues nuestro objetivo es encontrar la configuración de átomos que consiga la menor energía de la red, para ello los átomos deben poder viajar por la red para encontrarla, cuanto mayor sea su energía térmica a más regiones podrán llegar y más configuraciones podrán tener.

Para conseguir esta configuración de mínima energía, se calienta el metal a altas temperaturas, permitiendo a sus átomos establecer cualquier configuración. Después descendemos la temperatura muy lentamente por etapas, al hacerlo, los átomos perderán energía y ya sólo podrán moverse en una vecindad muy reducida llegando al equilibrio en un mínimo de energía local. Al final del proceso, los átomos forman una estructura cristalina altamente regular siendo mínima la energía del sistema y por tanto el material alcanza así una máxima resistencia.

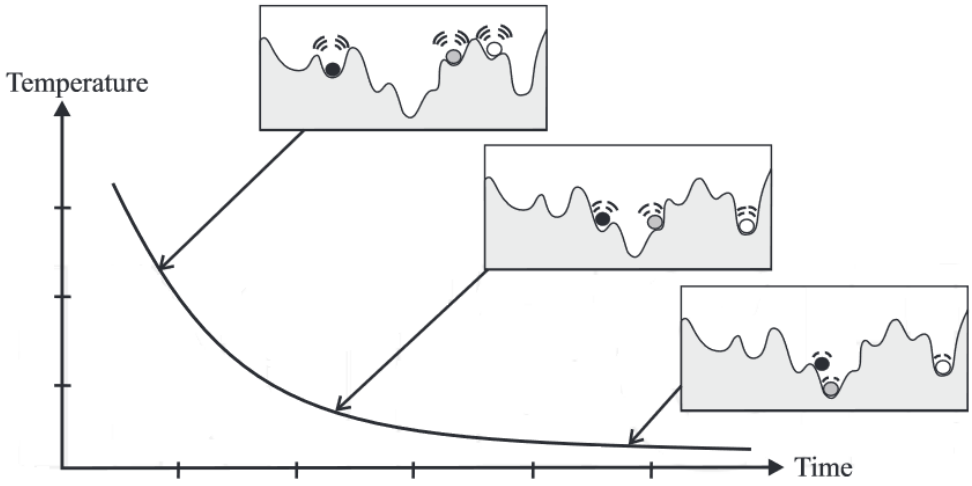


Fig. 27 Movimiento aleatorio de los átomos con la temperatura. Fuente [16].

Este proceso puede aplicarse a los DOP, donde la configuración de los átomos representa cada una de las posibles soluciones del problema S y su energía asociada, la función objetivo f. El algoritmo consiste en comenzar con una configuración x_i e ir viajando por su vecindad (realizando cambios en la solución) aceptando o no el cambio si resulta en una mejora de la función objetivo o si la temperatura es lo suficientemente alta.

5.1 Algoritmos de Umbral

El algoritmo SA pertenece al grupo de Algoritmos de Umbral (Threshold Algorithm - TA) cuyo funcionamiento básico es, dada una solución factible $x_i \in S$, viajar por la vecindad de su espacio de soluciones S_i, moviéndonos de la solución x_i a otra x_j en función de la diferencia de la función objetivo $\Delta f = f(x_j) - f(x_i)$ y de un parámetro t_k del algoritmo. Siendo k el número de iteraciones realizadas del algoritmo hasta el momento.

El **criterio de aceptación** es la función $\mathcal{A}(\Delta f, t_k)$ que indica si se pasa o no al siguiente estado en función de la diferencia de costo Δf y del parámetro t_k .

$$\mathcal{A}: \mathbb{R}^2 \rightarrow [0,1]$$

Los distintos algoritmos TA se diferencian principalmente en su criterio de aceptación $\mathcal{A}$ y en la manera en la que se elige el valor t_k. Existen 2 tipos de TA en su función de su $\mathcal{A}$:

- **Threshold Accepting**:

La función $\mathcal{A}$ compara Δf con t_k , sólo si Δf es menor que t_k, se acepta la nueva solución.

- Si $\Delta f = f(x_j) - f(x_i) \leq t_k \rightarrow$ Acepto x_j como solución.

En este caso, todas las soluciones que disminuyen el costo $f()$ son aceptadas, y las que lo incrementan (empeoran) son aceptadas en forma limitada. El valor inicial de t_k suele ser positivo y debemos decrementarlo con cada iteración para poder viajar en la dirección del optimo local.
La forma en la que lo modifiquemos t_k en cada iteración da lugar a diferentes algoritmos como:

- LSI (Local Search Improvement): $t_k = 0$
- GDA (Great deluge algorithm): $t_0 = N > 0; \quad t_{k+1} = t_k - v$
- RRT (Record-to-Record Travel): $t_k = f(x_{Best}) - dv; \quad dv \in R$

- **Simulated Annealing**:

La función $\mathcal{A}$ acepta la nueva solución x_j siempre que sea mejor que la anterior x_i, si no lo es, existe una cierta probabilidad de aceptarla en función de Δf y t_k. La probabilidad de movernos a la nueva solución x_j siendo esta peor que la actual x_i es:

$$P_{\mathcal{A}} = e^{-\frac{\Delta f}{t_k}}$$

Para ello, en cada iteración se genera un número aleatorio $\mathcal{R} \in (0,1)$, si $\mathcal{R} < P_{\mathcal{A}}$ entonces aceptamos la nueva solución.

- Si $\Delta f < 0$ entonces acepto j
- Si $\Delta f > 0$ entonces acepto j con probabilidad $e^{-\frac{\Delta f}{t_k}}$

El valor de t_k se va decrementando con cada iteración, disminuyendo así la probabilidad de pasar a una solución x_j peor y poder así alcanzar el mínimo local. El método más común es decrementarlo multiplicando por un factor $k < 1$, de tal forma que la probabilidad de pasar de un estado a otro disminuye de forma geométrica en $e^{-\frac{1}{k}}$ con cada iteración.

5.2 Simulated Annealing Algorithm

El algoritmo SA consiste en una serie de iteraciones (enfriamientos) en las cuales generamos soluciones en la vecindad de nuestra solución actual y decidimos si movernos o no a ellas en función de la mejoría que supongan y de la temperatura. A cada una de las iteraciones les corresponde una temperatura menor de la que tenía la etapa anterior, es por ello que hace falta un criterio de cambio de la temperatura ("cuanto tiempo" se espera en cada etapa para dar lugar a que el sistema alcance su "equilibrio térmico").

Para utilizar SA en un problema, primero tenemos que expresar dicho problema en términos del algoritmo, sea un DOP expresado con la tupla (S, f), identificando los elementos del problema de búsqueda con los del problema físico tenemos que:

- **Estado de los electrones**: Solución del problema $x \in S$.
- **Energía**: Función sobre la calidad de la solución $f(x)$.

Para poder operar con las soluciones debemos de **codificarlas** (representarlas). Normalmente se codifican como vectores de bits o números. Su codificación influirá en la manera en la que viajamos por el espacio de soluciones S.

El **operador de variación V** es la función que modifica una solución dada x_i para producir otra solución x_j de su vecindad. Si tenemos un vector de bits, para pasar de una solución a otra podríamos invertir ciertos bits de la solución.

Las consideraciones que deben tenerse en cuenta del algoritmo en sí son:

- Temperatura inicial.
- Solución inicial.
- Criterio de aceptación: $\mathcal{A}(\Delta f\ , T)$
- Estrategia de enfriamiento.
- Criterio de Parada.

- **Temperatura inicial (T_0)**

La temperatura inicial T_0 debe ser una temperatura que permita casi (o todo) movimiento, es decir que la probabilidad de pasar del estado x_i al x_j sea muy alta, sin importar la diferencia de energía ΔE. Esto se hace para que la configuración inicial pueda explorar cualquier parte del espacio S y se garantice conectividad total entre ellas:

$$T_0 \ \ tal\ que\ \ P_{\mathcal{A}} \cong 1 \ \ \forall\, \Delta \mathrm{E}$$

En general se toma un valor T_0 que se cree suficientemente alto y se observa la primera etapa para verificar que el sistema tenga un grado de libertad y en función de esta observación se ajusta. Otra opción es la de realizar un estudio previo de los cambios de temperatura del espacio de soluciones S, generando diversas soluciones x_i y moviéndonos por S, viendo los distintos valores de $abs(\Delta E)$ y en función de este estudio dar un valor a T_0.

- **Solución inicial (i_0)**

La solución factible inicial i_0 suele ser una solución aleatoria tomada del espacio S. Esto se hace así para que todas las regiones de S puedan ser exploradas y tengan la misma prioridad inicial. Si ya tenemos una solución buena y queremos viajar por sus alrededores para mejorarla encontrando un máximo local cercano podemos usarla como solución inicial y utilizar una temperatura inicial baja.

- **Estrategia de enfriamiento**

La estrategia de enfriamiento define la forma en la que varía la temperatura en cada iteración, se puede demostrar matemáticamente que si la temperatura t_k desciende lo suficientemente despacio, la probabilidad de encontrar el óptimo global es cercana a 1, sin embargo tal velocidad podría implicar realizar más iteraciones que la búsqueda exhaustiva.

Se suele utilizar una estrategia de enfriamiento geométrico donde:

$$t_{k+1} = t_k \cdot K$$

Donde:

$K < 1$: Factor de enfriamiento geométrico.

A medida que disminuye la temperatura, será más difícil pasar de una solución a otra peor, llegando un momento en el que encontramos un óptimo local y se llega al equilibrio.

- **Criterio de parada**

El criterio de parada indica la condición que, una vez satisfecha, propicia el término del algoritmo. Dicho criterio debería garantizar que la configuración final esté cerca del óptimo global o que, por lo menos, sea una buena solución del sistema. Se suelen utilizar como criterios de parada:

- Número máximo de enfriamientos.
- Número de enfriamientos en los que la solución no cambie (Sistema congelado).

❖ Pseudocódigo:

Tabla 10 Pseudocódigo de SA

$i = i_0$; Solución inicial
$T = T_0$; Temperatura inicial
K ; Constante de enfriamiento
while (Condición de Parada)
    $j = Variación\ (i)$
    if $(c(j) - c(i) < 0)$
        $i = j$
    else
        $r = Número\ Aleatorio$
        if $(r < e^{\left(\frac{c(i)-c(j)}{T}\right)})$
            $i = j$
        end
        $k = k + 1$
    end
    $T = T \cdot K$
end

Existen variaciones del algoritmo que implican recalentamiento. Si vemos que el sistema está congelado, es decir, que nos hemos quedado en un mínimo local y no nos podemos mover de la posición actual podemos subir la temperatura para escapar de ese mínimo. Una buena práctica es también guardar en todo momento la mejor solución encontrada hasta el momento, cosa que hace el algoritmo implementado en el programa del proyecto.

Capítulo 6
Algoritmos Evolutivos

6. ALGORITMOS EVOLUTIVOS

Evolutionary computation (EC) hace referencia al conjunto de sistemas de resolución de problemas basados en ordenadores que usan modelos computacionales de procesos evolutivos como pueden ser la selección natural, supervivencia y reproducción como componentes fundamentales de dicho sistema. La evolución puede ser definida como un proceso de optimización cuyo objetivo es mejorar la habilidad de un organismo (o sistema) para sobrevivir en entornos cambiantes y competitivos.

Los algoritmos evolutivos se basan en la **teoría Darwiniana de la evolución** que puede ser resumida como: *En un mundo con recursos limitados y poblaciones estables, cada organismo compite con el resto por la supervivencia. Aquellos individuos con las "mejores" características (aquellas que se adaptan mejor al ambiente actual) tienen más probabilidades de sobrevivir, pasando dichas características a sus descendientes. Dichas características son heredadas por los demás organismos durante las siguientes generaciones y se vuelven dominantes en la población. Durante la reproducción, la descendencia puede sufrir mutaciones aleatorias que cambian sus características. Si estas características son buenas para el organismo, entonces tendrá mayor probabilidad de sobrevivir y pasársela a su descendencia.*

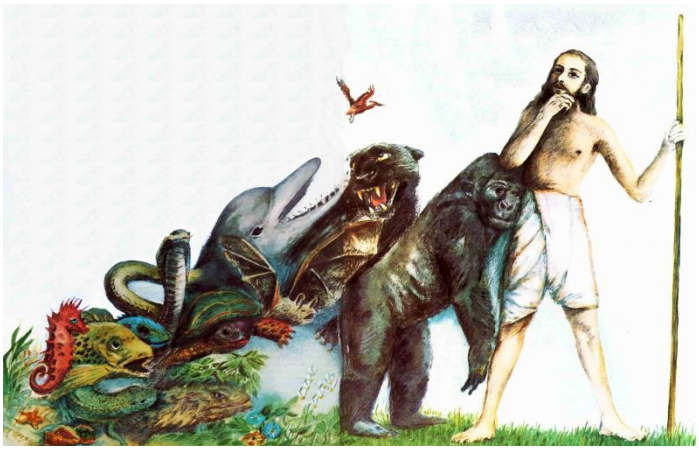


Fig. 28 Evolución biológica. Fuente [

Las características de dichos individuos se encuentran en los llamados **cromosomas** que son un conjunto de cadenas de ADN llamadas **genes**. Cada gen es responsable de una característica concreta del individuo y los posibles valores que puede tomar un gen se llaman **alelos**. El genotipo de un individuo es el conjunto de genes de este. En los algoritmos evolutivos cada uno de los individuos viene definido con su conjunto de genes, que directamente responsable de la capacidad de supervivencia del organismo.


Fig. 29 Cadena de ADN.

Para poder utilizar un EA (Algoritmo Evolutivo), primero debemos establecer una relación entre este y nuestro problema. Sea un DOP definido por la tupla (f, S):

- Cada individuo de la población representa una posible solución del sistema $x_i \in S$. Normalmente cada solución está compuesta por una serie de variables. Cada variable de la solución es codificada en un gen, formando en conjunto el cromosoma, que representa al individuo.
- La capacidad de supervivencia (**fitness**) de cada individuo viene determinada por sus genes, por lo que el fitness es un operador relacionado con la función objetivo f.

Así pues, en un EA, un número de criaturas artificiales viajan por el Search Space del problema. Las criaturas compiten continuamente con ellas mismas para descubrir zonas optimas del Search Space.
Se espera que a lo largo del tiempo las criaturas más exitosas evolucionen hasta encontrar la solución óptima del problema. Las criaturas artificiales del EA, llamadas individuos, están normalmente representadas por vectores de longitud fija.

6.1 Algoritmos genéticos

Un **algoritmo genético** (GA) es aquel algoritmo que modela la evolución genética, es un tipo de algoritmo de búsqueda estocástico y meta heurístico basados en poblaciones (soluciones). Un GA consiste en la creación de una **Población inicial**, generalmente aleatoria y hacerla evolucionar mediante operaciones de cruce y mutación hasta que se cumpla una determinada condición. En cada generación, se evalúa la función de **fitness** de cada individuo y de acuerdo a esta se **seleccionan** los organismos que se reproducirán y pasarán a la siguiente generación. No todos los organismos que pasen a la siguiente generación tienen que reproducirse. Durante la reproducción ocurren 2 fenómenos, el **cruce (crossover)** de los genotipos y la **mutación** de los mismos que dará lugar a nuevos organismos.

- El cruce permite combinar la información de dos padres.
- La mutación puede introducir nuevos alelos.

Podemos expresar el Algoritmo genético en Pseudocódigo como:

Tabla 11 Pseudocódigo de GA

Sea $t = 0$ el contador de generaciones
Crear e inicializar una Población inicial $C(0)$, formada por M organismos $\boldsymbol{x}_i(t)\ \ i = 1, \dots, M$
while (*Condición de parada)* **do**
    Calcular el fitness, $\boldsymbol{f}(\boldsymbol{x}_i(\boldsymbol{t}))$, de cada organismo.
    Seleccionar los organismos que pasan a la siguiente generación
    Realizar la reproducción (cruce y mutación) de los organismos seleccionados
    $t = t + 1$; Avanzar a la siguiente generación
end

Representando el algoritmo como diagrama de flujo tenemos:

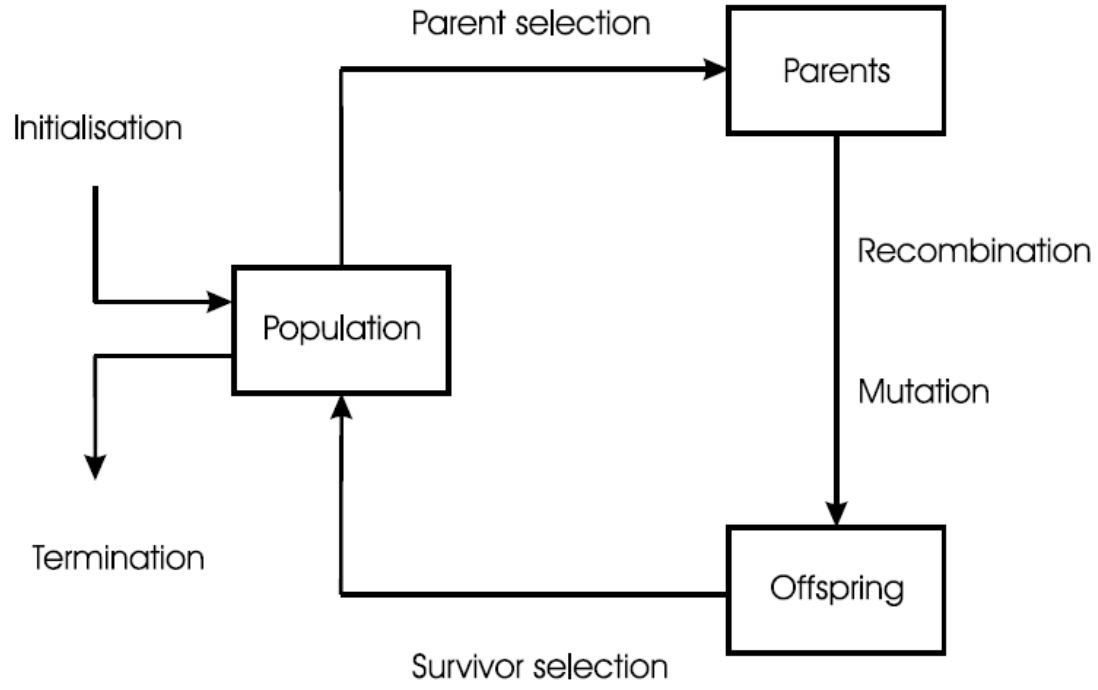


Fig. 30 Diagrama de flujo de un Algoritmo Genético. Fuente [34].

Existen diversas variantes sobre este algoritmo, por ejemplo ciertas implementaciones realizan la selección después de la reproducción u otras utilizan 2 selecciones, una pre-reproducción y otra post-reproducción. Dentro de cada una de las etapas del mismo, puede haber algoritmos en los cuales los padres, al reproducirse mueran, u otras en las que los organismos mueran tras cierto tiempo.

El primer tema que debemos abordar antes de usar el GA es hallar las relaciones entre nuestro problema OP y los elementos del GA:

- Codificación de los organismos.
- Función de fitness para poder operar con ello.

Una vez tenemos representado nuestro OP en forma de GA, tenemos que definir las características del algoritmo, entre las cuales se encuentran:

- Población Inicial.
- Mecanismo de Selección.
- Operadores de Cruce.
- Operadores de Mutación.
- Condición de Parada.

Estas propiedades son las que diferencian los distintos GA entre sí y deberemos definir cada una de ellas en función de la representación del problema.

El problema de selección de características que atañe a este documento, codifica los individuos como vectores binarios donde cada bit representa uno de los posibles parámetros a utilizar. Si el bit está a 1, utilizamos el parámetro, si está a 0 no lo incluimos. Así pues, cada característica es un gen del individuo , siendo el cromosoma del organismo, el conjunto de todos estos genes. El diagrama siguiente se ajusta al algoritmo diseñado para realizar la selección de características.

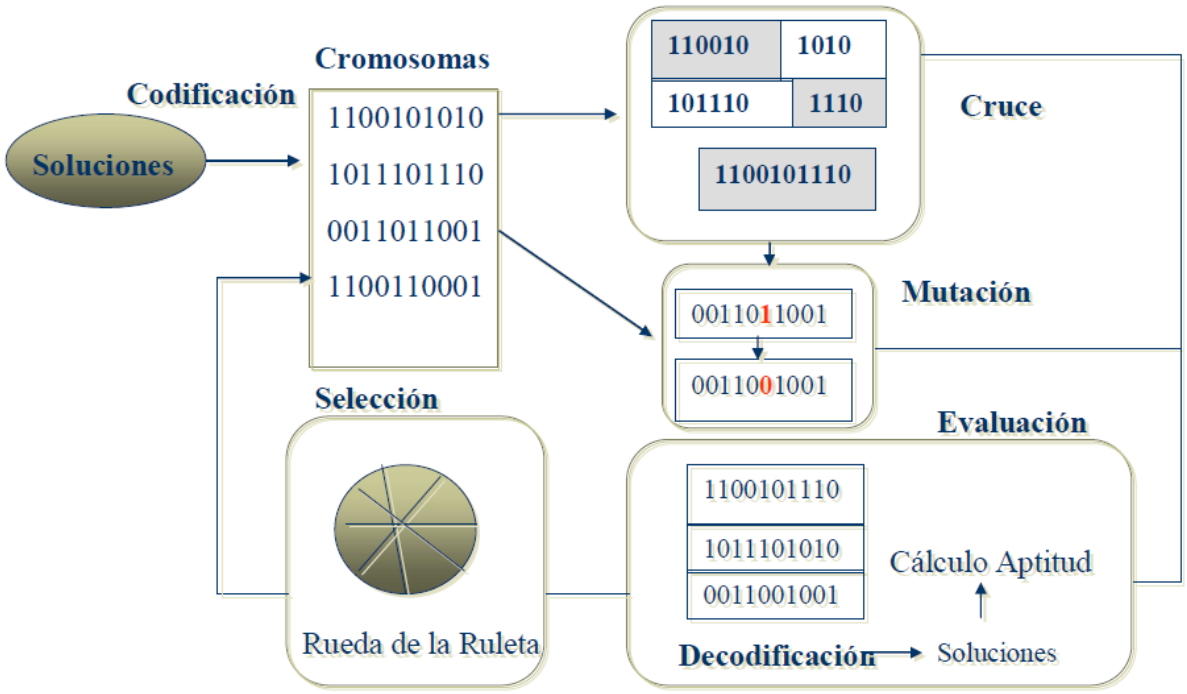


Fig. 31 Diagrama de flujo del Algoritmo Genético del documento. Fuente [33].

6.2 Representación del Problema

Antes de poder utilizar un GA para resolver nuestro problema, debemos poder representarlo en términos del GA. Esto consiste principalmente en definir la codificación de las soluciones a individuos y de la función de fitness del sistema. La codificación puede afectar al cruce y a la mutación.

1. **Codificación de los individuos:**

La codificación es la forma en la que expresamos una solución del problema, $x_i \in S$, como un array de valores, que pueden ser binarios, naturales, reales… o combinación de ellos. Podemos decir que la codificación es la transformación del fenotipo al genotipo, siendo:

- Fenotipo: Solución real del problema que tratamos. También llamado "**candidate solution**".
- Genotipo: Solución del EA asociada a la solución real. Llamado individuo u organismo.

Así pues, cada solución del sistema puede ser expresada como un organismo Γ^N formado por un conjunto de N genes, siendo cada gen una característica del organismo. Podemos expresar matemáticamente un organismo como:

$$x_i \xleftrightarrow[Codificación]{} \Gamma_i^N = [\Gamma_1, \Gamma_2, \dots, \Gamma_N]$$

Donde Γ es el tipo de datos del individuo N-Dimensional. Normalmente se utiliza una cadena de bits para representar a cada uno de los individuos.

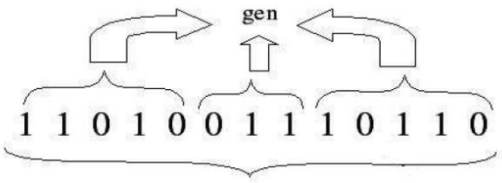


Fig. 32 Relación gen-solución. Fuente [32].

2. **Función de Fitness**

La función objetivo f de nuestro DOP se domina **función de Evaluación** en el contexto de los GA, la función de evaluación actúa sobre una solución x_i dándonos su costo.

$$f: x \to \mathbb{R}$$

Los GA son algoritmos de **maximización de la capacidad de supervivencia del individuo**. La función de Fitness $\mathcal{F}_{it}$ evalúa la habilidad que tiene cada individuo de la población para sobrevivir. Mapea un individuo a un valor real.

$$\mathcal{F}_{it}: \Gamma^N \to \mathbb{R}$$

Normalmente la función se fitness suele ser una función que actúa sobre la función de Evaluación y otros parámetros que deseemos como por ejemplo el número de generaciones que han pasado.

$$\mathcal{F}_{it}: f: x \to \mathbb{R}$$

Si nuestro problema es de minimización necesitamos convertirlo a maximización, para ello lo más común es restar al $\mathcal{F}_{it}$ máximo de la generación ($\mathcal{F}_{max}(x)$) el $\mathcal{F}_{it}$ de cada individuo:

$$\mathcal{F}_{it}(x) = \mathcal{F}_{max}(x) - \mathcal{F}_{it}(x)$$

Las operaciones normales que se suelen realizar sobre la función de costo son:

- Normalización de la población.
- Restar a la función de costo el valor final deseado.

6.3 Población inicial

Dado que los EA están basados en Poblaciones, cada EA mantiene una población de organismos ("candidate solutions") y por tanto, el primer paso de los EA es producir la generación inicial. El método estándar es generar una población inicial aleatoria, el objetivo de esto es asegurar que esta sea una representación uniforme de todo el Search Space para que el mayor número de zonas de este puedan ser exploradas. En ciertos problemas puede resultar difícil dar valores iniciales aleatorios a los genes y generar soluciones validas por lo que requieren un algoritmo previo de búsqueda local.

El número de elementos de la población influye en:

- Complejidad computacional por generación: Mas organismos → Mas operaciones
- Exploración del Search Space: A mayor número de organismos, mayor número de zonas del Search Space podremos explorar. Cuando la población es pequeña podemos aumentar la tasa de mutación para explorar más Search Space.

Goldberg (1989) realizó un estudio teórico, obteniendo como conclusión que el tamaño óptimo de la población para vectores de longitud N, con codificación binaria, crece exponencialmente con el tamaño del vector.

Si disponemos ya de un conjunto de soluciones consideradas buenas, podemos querer ponerlas como población inicial para buscar en un espacio cercano ellas y mejorarlas.

6.4 Algoritmos de selección

La selección es el proceso por el cual elegimos aquellos organismos que pasarán a la siguiente generación, ya sea para reproducirse o no. De acuerdo con la Teoría de Darwin, los organismos más adaptados tienen más posibilidades de pasar a la siguiente generación y reproducirse. De la misma manera, el proceso de selección debería hacer que aquellos organismos con mayor función de Fitness, tengan más probabilidad de pasar a la siguiente generación, asegurando así una población con buenos individuos. La selección depende de la función de fitness y de los parámetros de cada algoritmo.

Existen algoritmos que realizan 2 procesos de selección en cada generación:

- **Selección de reproductores**: Al inicio del algoritmo se eligen aquellos individuos que se reproducirán, normalmente realizado en base al fitness de cada individuo.
- **Selección de supervivientes**: Elegimos de entre padres e hijos quienes sobrevivirán, normalmente realizado en base al fitness y el edad de cada individuo.

Un algoritmo de selección da a cada individuo una probabilidad distinta de formar parte de la siguiente generación en función de su valor de fitness. La forma en que relacione esta probabilidad con el fitness nos da distintos tipos de algoritmos que modifican las propiedades de la selección.

- Presión selectiva: Es la velocidad con la que la mejor solución ocupará toda la población.
- Biodiversidad: Es la diversidad de genes que posee la población.

Algunas clasificaciones de algoritmos de selección son:

- Métodos de selección dinámicos: La probabilidades de selección varían de entre generaciones.
  - Por ejemplo la selección proporcional al fitness.
- Métodos de selección estáticos: La probabilidades de selección permanecen constantes
  - Por ejemplo la selección basada en rangos.
- Preservativo: Si se asegura que todos los individuos tienen asignada una probabilidad de selección distinta de cero. En caso contrario se acostumbra a denominarlo extintivo.

6.4.1 Roulette Wheel Selection

Este método es también llamado el Método Proporcional, ya que asigna a cada individuo una probabilidad de ser seleccionado proporcional a su Fitness. Sea una población de N individuos, la probabilidad de que el individuo x_i sea elegido es:

$$P_i = \mathcal{F}_{it}(x_i) \; / \sum_{j=1}^{N} \mathcal{F}_{it}(x_j)$$

Esta probabilidad se consigue visualizando los organismos como segmentos de una ruleta. Cada organismo ocupará un segmento de la ruleta con tamaño proporcional a su fitness. Cuanto mayor sea el fitness, mayor será su segmento y mayor será la probabilidad de ser elegido. Para seleccionar un organismo, elegimos un punto al azar de la ruleta, y el organismo que represente al segmento donde caiga el punto será el elegido.

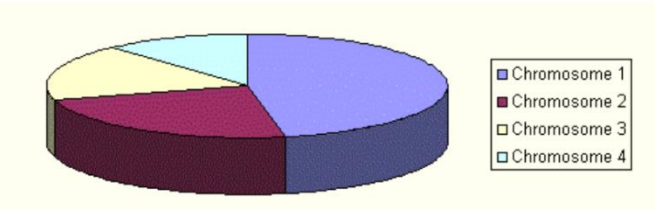


Fig. 33 Roulette Wheel Selection. Fuente [45].

❖ **Pseudocódigo:**

Tabla 12 Pseudocódigo de RWS

$x_s = x_0$; Inicialmente se establece como organismo seleccionado el primero de ellos.
Calcular el fitness de cada individuo $\mathcal{F}_{it}(x_i)$ y normalizarlo con la suma de fitness obteniendo P_i
R = *U*(0, 1); sum = 0; Seleccionamos valor aleatorio R e inicializamos la búsqueda a 0.
while (*sum* < *R)* **do**
$x_s = x_{s+1}$; Avanzar al siguiente individuo
sum = *sum* + P_i ; Avanzar en la ruleta
end
return x_s; Devolvemos el individuos seleccionado esta vez por el algoritmo.

Es un método muy sencillo, donde los mejores individuos tienen mayor probabilidad de supervivencia, sin embargo posee diversos defectos:

- Ineficiente a medida que aumenta el tamaño de la población (su complejidad es $O(n2)$).
- El mejor individuo podría no ser elegido y el peor ser elegido 1 o más veces.
- Tiene una alta presión selectiva dado que a la larga, solo los organismos con mayor fitness serán seleccionados.

6.4.2 Rank Selection

La selección por Ranking utiliza la posición relativa de los organismos dentro de la población para determinar probabilidad de selección y no la el valor de fitness de los mismos. La probabilidad de selección a partir de su posición en el ranking dependerá del algoritmo de ranking utilizado. El método más básico consiste en asignar a cada individuo una probabilidad proporcional a su posición en el ranking, de esta manera, el peor individuo tendrá valor 1, el segundo peor 2…

Sea $\mathcal{P}(x_1)$ la posición del organismo desde la cola, siendo $\mathcal{P}(x_{worst})$ = 1 y $\mathcal{P}(x_{best}) = N$, la probabilidad de elegir el organismo x_i es:

$$P_i = \mathcal{P}(x_i) \; / \sum_{j=1}^{N} j \;\; = \;\; \mathcal{P}(x_i) \;\; / \;\; n(n+1)$$

Mejoras:

- La probabilidad de selección no depende del fitness absoluto sino de la posición relativa del individuo. Esto consigue disminuir la presión selectiva ya que si por ejemplo un organismo muy bueno apareciese con un 90% de la ruleta, el algoritmo convergería.

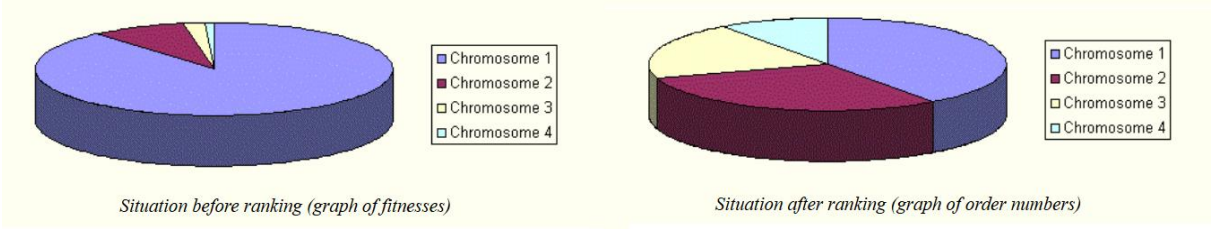


Fig. 34 Rank Selection. Fuente [45].

- Presión selectiva variable. Cambiando la forma en la que afecta el Rango en la probabilidad de selección.

Inconvenientes:

- Mayor costo computacional ya que se debe reordenar la población

La implementación de este algoritmo es igual que la Roulette Wheel Selection solo que ahora el perímetro de la ruleta es

$$L = n(n+1)$$

6.4.3 Tournament Selection

La selección por torneo consiste en elegir aleatoriamente k organismos de entre los N posibles y elegir de entre ellos el mejor o mejores.
La probabilidad de elegir el organismo x_i depende de:

- El fitness del organismo.
- El número de luchadores k.
- Si los contrincantes se eligen con/sin reemplazo
- Si el mejor siempre gana o depende de una probabilidad *p.*

El valor de k manda la presión selectiva, si k es grande, mayor será la probabilidad de escoger aleatoriamente los mejores individuos y por tanto, la siguiente generación estará llena de ellos. Si k es pequeño y todos los contricantes seleccionados en la iteración son malos, un individuo pobre pasará a la siguiente generación.

❖ **Pseudocódigo:**

Tabla 13 Pseudocódigo de Tournament Selection

```
Y = {∅}
Calcular el fitness de cada individuo F_it(x_i)
for (i = 0; i < k; i++)
        Y += Random_organism;
end
Ordenar organismos de Y en orden descendiente de fitness.
x_s = Y[0]
return x_s;
```

6.4.4 Variantes

Existen una serie de procedimientos que podemos aplicar a los algoritmos de selección vistos para aumentar su Biodiversidad, disminuir la presión selectiva, asegurar la supervivencia de los mejores… Algunos de estos procedimientos son:

- **Elitismo:** Es el proceso mediante el cual se asegura que los organismos con más fitness de una generación pasen a la siguiente. Estos organismos son copiados sin mutar en la nueva generación. Cuantos más organismos elitistas haya mayor será la presión selectiva pero aseguraremos no perder a los mejores organismos de cada generación

- **Hall of Fame**: Consiste en tener una segunda población especial llamada "Hall of Fame" donde se van copiando los mejores individuos de cada generación y sus individuos nunca se modifican. Puede ser utilizada como padre para futuras reproducciones y para mantener siempre los mejores organismos. Aumenta la presión selectiva.

- **Borrar el Peor:** Consiste en eliminar, antes de la selección, los peores organismos de la generación, asegurando que sus genes no pasen a la futura población. Aumenta la presión selectiva haciendo que la población converja antes.

- **Modificar el fitness de cada individuo:**
  Podemos modificar la función de fitness de cada individuo, de tal manera que individuos que individuos muy parecidos devalúen su fitness con objeto de que la población gane en biodiversidad.

- **Modelo de selección del valor esperado:**
  Dado que los algoritmos de selección que hemos visto funcionan con probabilidades, es posible que en una generación ocurran irregularidades como que el mejor organismo no pase o que el peor pase varias veces. Si queremos limitar el número de veces que puede ser seleccionado un organismo o asegurar un número de copias del mismo podemos hacerlo mediante contadores. Se procede de la manera siguiente:

  Para cada individuo x_i , se introduce un contador, inicializado generalmente como:

  $$g(x_i) = \left\lceil \frac{\mathcal{F}_{it}(x_i)}{g_t} \right\rceil$$

  Donde g_t denota la media del fitness en la generación t.

  Cada vez que el individuo x_i es seleccionado, dicho contador decrece en una unidad. Si el contador de un individuo tiene un valor negativo, deja de poder ser seleccionado en esa generación.

- **Gran probabilidad de mutación:**
  Aquellas implementación de un GA que posean una gran presión selectiva, pueden reducirla aumentando la probabilidad de mutación o generando individuos aleatorios periódicamente para introducir nuevos genes a la población.

6.5 Cruce

El cruce (Crossover) es el proceso por el cual los organismos de una generación mezclan sus genes para crear nuevos organismos hijo (offspring). Al generar el organismo hijo, los padres pueden desaparecer o no en función de la implementación. No todos los organismos que pasan el proceso de selección tienen por qué reproducirse, de entre los organismos seleccionados podemos seleccionar aquellos a reproducirse, formando lo que se llama la piscina de reproducción (**mating pool)**.

La reproducción permite que diversos organismos mezclen sus genes con la esperanza de obtener un organismo mejor sin añadir nuevos alelos a la población, es decir no cambia le valor de los genes, sólo hace recombinaciones de los ya existentes. Existen diversos tipos de cruce en función del número de padres que se vean envueltos:

- **Asexual**: Cuando el hijo es generado por un solo padre.
- **Sexual**: Cuando el hijo es generado como combinación de 2 padres.
- **Multi-recombinación**: Cuando el hijo es generado como combinación de más de 2 padres.

El cruce más común es el de tipo sexual, entre 2 organismos y será del que hablemos a partir de ahora.

- La **máscara de reproducción** Γ_{Cru}^{N}:

Es un valor binario de longitud M que indica los genes que el hijo hereda de cada padre. Los bits a 0 representan los genes que hereda de un padre y los bits a 1 los que hereda del otro. Sean 2 cromosomas padres de longitud N:

$$\Gamma_a^N = [\Gamma_{a,1}, \Gamma_{a,2}, \dots, \Gamma_{a,N}] \;\; y \; \Gamma_b^N = [\Gamma_{b,1}, \Gamma_{b,2}, \dots, \Gamma_{b,N}]$$

El hijo Γ_h^M con máscara de reproducción Γ_{Cru}^N se obtiene como:

$$\Gamma_h^M = (\Gamma_a^N \;\&\; \Gamma_{Mut}^N) \mid (\Gamma_a^N \;\&\; \sim\Gamma_{Mut}^N)$$

Es **operador de cruce** es la forma en la que generamos la máscara de reproducción, seleccionando los genes de cada padre que formarán parte del hijo. Existen diversos operadores de cruce:

- **One-point crossover:** Se selecciona un punto de corte $'i'$ en la máscara de reproducción, los bits a un lado del corte están a 0 y los que están al otro a 1.
- **Two-point crossover:** Se seleccionan dos puntos de corte $'i', 'j'$, los bits entre medias de los cortes están a 0 y los de los extremos a 1.
- **Uniform crossover:** Los bits de la máscara se generan de forma aleatoria siguiendo una distribución uniforme.

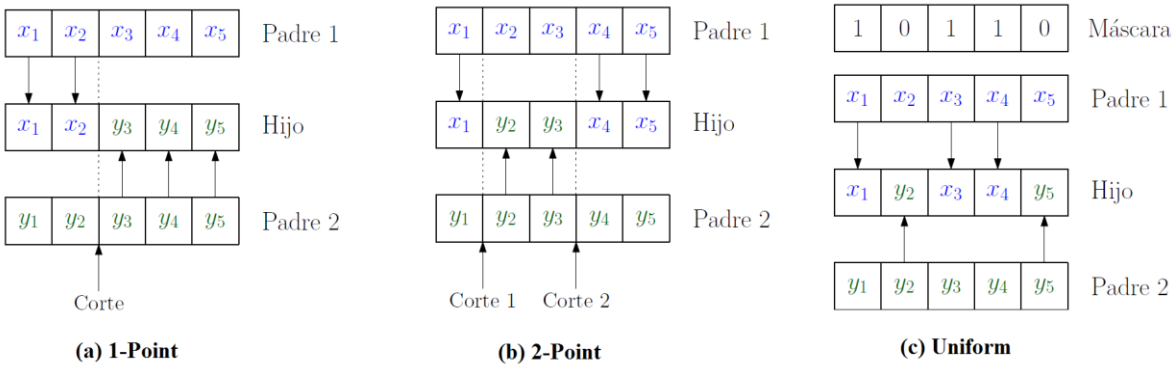


Fig. 35 Algoritmos de cruce. Fuente: Álvaro Pastor Sánchez.

6.6 Mutación

La mutación es el proceso por el cual los genes de un determinado organismo pueden ser modificados, reemplazándose por un alelo. Durante la reproducción se pueden producir fallos que lleven a mutaciones. La mutación debería ser mayor en organismos débiles con la esperanza de que dichos cambios produzcan un mejor organismo.

Aunque se pueden seleccionar los individuos directamente de la población actual y mutarlos antes de introducirlos en la nueva población, la mutación se suele utilizar de manera conjunta con el operador de cruce, modificando solo los organismos hijos.

- **La máscara de mutación Γ_{mut}^{N}:**

Es un valor binario de longitud N que indica con un '1' los genes del individuo que serán mutados. Para generar esta máscara, se suele utilizar un array binario vacío al que se invierten unos cuantos bits convirtiéndolos a 1. Cuanto mayor sea el número de 1s más agresiva será la mutación y más lugares del Search Space podremos ver. Normalmente esto se indica mediante una probabilidad de mutación.

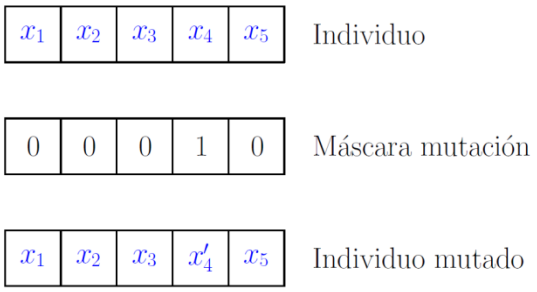


Fig. 36 Máscara de mutación. Fuente: Álvaro Pastor Sánchez.

6.7 Condición de Parada

El algoritmo evolutivo va transformando la población de organismos de generación en generación hasta que se produzca un cierto evento que propicie la finalización del mismo. La condición de parada es el evento que lleva a finalizar el EA, dando a la generación actual como la mejor solución encontrada. La condición más sencilla es implementar un número dado de generaciones fijo pero existen otras como pueden ser:

- **Terminar si la población no mejora durante varias generaciones consecutivas**. Para ello tenemos que controlar el fitness medio de la población el fitness del mejor individuo.

- **Terminar cuando no hay cambios en la población**. Si, durante varias generaciones consecutivas, el cambio promedio en información genotípica es demasiado pequeño, la EA puede ser detenida.

- **Terminar cuando se ha encontrado una solución aceptable.** Establecemos un valor objetivo X que debe cumplir la mejor solución.

Si tras la condición de parada no hemos obtenido un organismo que satisfaga nuestros requerimientos siempre podemos relanzar el algoritmo, ya sea con la última generación o con otra generación aleatoria, modificando algunos valores como por ejemplo la probabilidad de mutación para buscar espacios más amplios del Espacio de Soluciones.

6.8 Ventajas e inconvenientes de los algoritmos evolutivos

Por último, se van a listar una serie de ventajas e inconvenientes que presentan este tipo de algoritmos. Algunas de las ventajas de los algoritmos evolutivos son:

- **Paralelismo**.
  El cálculo del fitness de un individuo es independiente del resto de la población y por lo tanto puede realizarse de manera simultánea para varios individuos, lo cual permite reducir el tiempo de cómputo, ya que como se ha comentado el cálculo del fitness es lo que más incrementa el tiempo total.

- **Tipos de funciones.**
  Es posible optimizar funciones discontinuas y/o ruidosas, ya que no es necesario realizar un análisis matemático de la función a optimizar y por lo tanto es más viable.

- **Tamaño del espacio de búsqueda**.
  Por el mismo motivo, los algoritmos evolutivos permiten optimizar funciones con un gran número de variables, lo cual implica espacios de búsqueda muy grandes (si se desea optimizar una función real de N variables, el espacio de búsqueda es $\mathbb{R}^N$).

- **Soluciones << innovadoras >>**.
  Al no conocer nada a priori del problema planteado (la inicialización de los individuos se hace de manera aleatoria), es posible encontrar soluciones innovadoras que no se tengan en cuenta en otro tipo de métodos

Por otro lado, también es necesario tener en cuenta los inconvenientes de utilizar este tipo de algoritmos, como pueden ser:

- **Representación del problema**.
  Es necesario realizar una codificación que permita que el algoritmo se desarrolle de una forma correcta, lo cual no siempre es sencillo o incluso viable.

- **Obtención de la función**.
  De la misma forma, si la función no tiene una expresión analítica, puede ser complicada su obtención (como ejemplo, en este proyecto la función a optimizar es el resultado de una simulación de Montecarlo).

- **Convergencia prematura.**
  Puede darse el caso de que el algoritmo diseñado tienda rápidamente hacia soluciones sub-óptimas, debiéndose por ejemplo a que haya demasiada presión evolutiva y no converja al resultado óptimo global.

- **Solución analítica**.
  Es preferible no utilizar este tipo de algoritmos en problemas que puedan resolverse de una manera analítica, ya que este tipo de métodos puede ofrecer una solución más exacta matemáticamente

Capítulo 7

Sistema Inmune Artificial

7. SISTEMA INMUNE ARTIFICIAL

El sistema inmune biológico es un sistema complejo, robusto y adaptativo que defiende al cuerpo de patógenos externos. Es capaz de reconocer todas las células (o moléculas) dentro del cuerpo como propias (self-cells) o externas (non-self). Para ello el sistema dispone de un grupo de agentes distribuido que tiene la inteligencia necesaria para llevar a cabo acciones desde un punto de vista tanto local como global ya que utiliza una red de químicos como medio de comunicación.

La arquitectura del sistema inmune está formada por varias capas de defensa, cuando un patógeno entra en el organismo el sistema inmune responde con 2 subsistemas. El sistema inmune innato es un mecanismo invariante que detecta y destruye ciertos organismos invasores, mientras que el sistema inmune adaptativo responde a los invasores ya conocidos y genera una respuesta hacia ellos que puede perdurar en el cuerpo durante largos periodos de tiempo. Así pues el sistema inmune posee memoria sobre los patógenos que ya se ha encontrado y como eliminarlos.

Los sistemas inmunes artificiales (AIS) son una técnica de inteligencia artificial basada en el sistema inmune biológico. Dado que los AIS es un campo joven y en evolución, no existe un algoritmo fijo con el que trabajar por lo que las implementación variar según la fuente. En este documento se verán las principales metáforas que inspiran a los algoritmos y se presenta un sencillo algoritmo desarrollado por el autor de este proyecto que adapta las propiedades del sistema inmune al problema.

6.9 Resumen del sistema inmune

Antes de entrar en cómo funciona el sistema inmune, veamos las 2 células más importantes que posee, estas son las células blancas, llamadas *B-cells* y *T-cells*, ambas son generadas en la médula ósea y sirven para detectar los agentes invasores, antígenos para producir una respuesta. La diferencia es que las *T-cells* pasan por la *glándula timo* antes de pasar al sistema inmune, donde serán sometidas a un proceso de prueba para comprobar que no atacan a células propias del organismo.

Hay 3 tipos de *T-cells*:

- *Helper T-cells*: Aquellas que activan las células B.
- *Killer T-cells*: Se enganchan a los invasores externos e inyectan químicos venenosos provocando su destrucción.
- *Suppressor T-cells*: Inhiben la reacción de otras células del sistema inmune para prevenir reacciones alérgicas y enfermedades del sistema inmune.

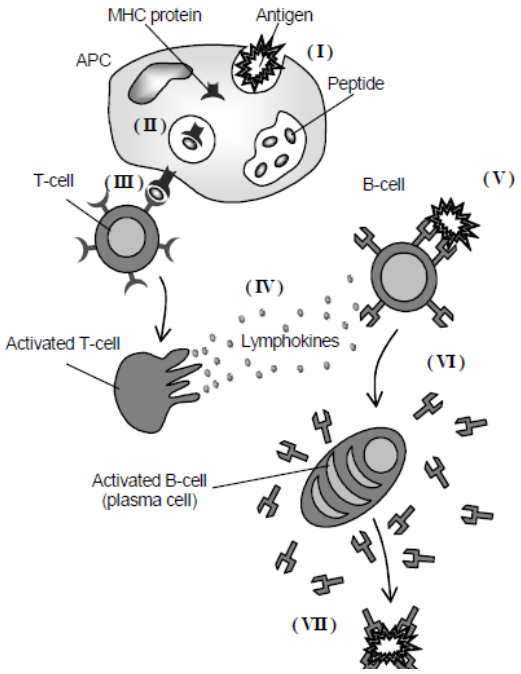


Fig. 37 Respuesta autoinmune. Fuente [43].

Las células B producen y secretan anticuerpos que se pegan al antígeno. Cada célula B solo puede generar un tipo concreto de anticuerpo. Si el anticuerpo se pega a la superficie del antígeno, es una señal para que las células *Killer T-Cells* la destruyan.

Así pues, cuando un antígeno (ente invasivo) entra en el organismo, las "*Helper T-cells*" lo detectan y activan la *B-cell* correspondiente para que genere el anticuerpo específico que se pegue al antígeno. Las *Killer T-cells* detectarán esta unión y destruirán al invasor.

7.1 Tipos de algoritmos AIS

Existen múltiples algoritmos inspirados en alguna de las propiedades de los sistemas inmunes, pero muchas de estas características no tienen aplicación directa en nuestro problema en particular. Los dos algoritmos más comunes son los que se explican a continuación:

- **Clonal Selection Algorithm.**

El principio de selección de clones describe la característica básica del sistema inmune como respuesta a un antígeno. Establece la idea de que solo aquellas células que reconozcan el antígeno proliferan y son seleccionadas para clonarse en el siguiente ciclo.

- Las nuevas células son copias de los padres sujetas a fuertes mutaciones.
- Las células que mejor reaccionen con el antígeno, proliferan y el resto mueren.

Cuando un anticuerpo reacciona muy bien con un determinado antígeno, su célula B correspondiente es estimulada para que genere más anticuerpos. Así pues. este algoritmo posee una población de células B que se corresponden con soluciones del problema. Estas son copiadas y modificadas en cada iteración para generar anticuerpos, que son soluciones en la vecindad de la célula B padre. Al final de cada iteración, las mejores células se convierten en células B que volverán a ser clonadas en el próximo ciclo del algoritmo.

- **Pseudocódigo**

Tabla 14 Pseudocódigo de CSA

Sea t = 0 el contador de ciclos
Crear e inicializar una Población inicial $B\ (0)$, formada por N B-Cells $\boldsymbol{x}_i(t)\ \ i = 1, \dots, M$
while (*Condición de parada)* **do**
    Producir la generación de anticuerpos $A(t)$ clonando y mutando $B\ (t)$
    Calcular la capacidad de defensa, $\boldsymbol{f}(\boldsymbol{x_i}(\boldsymbol{t}))$, de cada célula de $A(t)$
    Seleccionar las células que se convertirán en la población $B\ (t+1)$
    $t\ =\ t\ +\ 1$; Avanzar al siguiente ciclo
end

- **Negative Selection Algorithm.**

El propósito de la Negative Selection de un sistema inmunológico es proporcionar cierta tolerancia a las células propias (self-cells). Tiene que ver con la capacidad del sistema para detectar un antígeno desconocido y a la vez no reaccionar a sus propias células. Cuando las células B son creadas, estas son expuestas a un proceso de censura llamado selección negativa en el cual aquellas células que reaccionen a células propias son destruidas y solo aquellas que no lo hagan son liberadas al organismo para que realicen su función.

Esta propiedad puede ser utilizada en un algoritmo creando una población de soluciones muy malas y cada vez que se genere una solución comprobar si se parece a dicha población. Si lo hace, la solución es descartada.

7.2 Beato's Algorithm

Se ha desarrollado un algoritmo basado de en AIS llamado "Beato's Algorithm" que aplica los conceptos del sistema inmune a la resolución de un problema de selección de características. Se trata de un algoritmo metahuerístico y estocástico de búsqueda global que emula la clonación y mutación de las células B para producir anticuerpos que sean más efectivos contra un invasor.

El algoritmo posee 3 poblaciones de células, donde cada célula representa una posible solución del problema, estas poblaciones se relacionan entre sí por medio de mutaciones y actualizaciones. Es un algoritmo iterativo donde cada etapa representa un ciclo de clonación del sistema inmune. Las células B representan aquellas soluciones que están siendo mejoradas y poseen un cierto dominio que es el mínimo grado de diferencia que debe haber entre 2 células B para considerarlas diferentes.

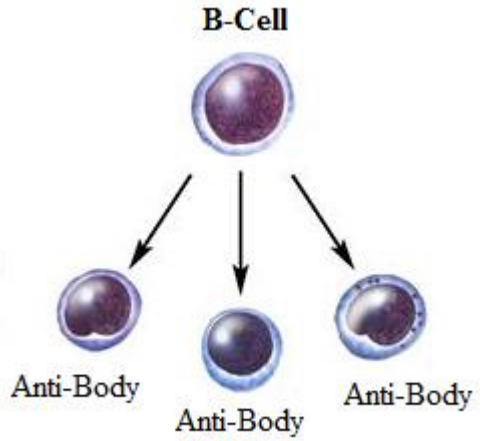


Fig. 38 Clonación de células B. Fuente [44].

Sea un población inicial de células B, en cada ciclo del algoritmo, las células B se clonan y mutan generando anticuerpos. La mutación se realiza dentro del dominio de las células B. También se genera cierto número de anticuerpos aleatorios.

- Las células B evolucionan en cada ciclo de la siguiente manera:
  - Si algún anticuerpo es mejor que su célula B madre, la célula B es reemplazada por dicho anticuerpo.
  - Si 2 células B se encuentran dentro del mismo dominio, la peor de ellas muere.
  - Si una célula B no mejora durante cierto número de ciclos, la célula B desaparece y se convierte en una célula M que ya no se clonará más.
  - En cada ciclo, si caben más células B en el organismo, se añade un célula B, la mejor de entre los anticuerpos generados aleatoriamente.

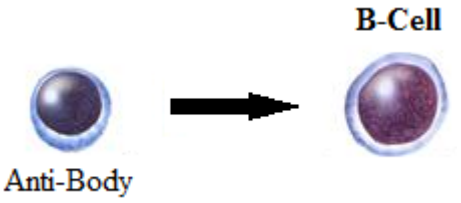


Fig. 39 Conversión de anticuerpo a célula B. Fuente [44].

La **Beato's Cell** es la mejor de las células M y es considerada la solución del algoritmo.

Fig. 40 Conversión de célula B a célula M. Fuente [44].

Así pues tenemos 3 tipos de poblaciones:

- Células B: Son soluciones del problema que están siendo mejoradas.
- Anticuerpos: Posibles mejoras de las células B.
- Células M: Mejores soluciones.

Como condiciones de parada podemos establecer:

- Número máximo de ciclos.
- Número máximo de células M.

Capítulo 8
Programa

8. PROGRAMA

Para realizar la Selección de Características se ha desarrollado un programa en C que utiliza la ELM como Algoritmo de Aprendizaje e implementa diversos algoritmos de búsqueda. Inicialmente se desarrolló un programa básico capaz de emular una RNA, entrenarla usando la ELM y testearla, devolviendo el error RMSE de testing. Para ello debemos especificar sus características básicas:

- Vectores de Training.
- Vectores de Testing.
- Número de Neuronas.
- Función de Activación.

El programa entrena la ELM con los vectores de Training, la prueba con los de Testing y devuelve el error RMSE del testeo. Se puede indicar al programa que haga la ELM únicamente de un subconjunto de los parámetros de entrada mediante un array binario, de tantos bits como parámetros tenga el problema, donde un "1" significa utilizar el parámetro en la ELM y un "0" no utilizarlo. Este array binario constituye una posible solución del problema, es decir, un subconjunto de parámetros (predictor) e internamente está formado por variables unsigned int. En este documento se denominará a estos arrays como "**Vector de Selección**".

Este programa se redujo a una única función get_ELM_RMSE() que es utilizada por un sistema de hilos capaz de lanzar varias ELM concurrentemente con el ahorro temporal que ello conlleva. Por último se han desarrollado algoritmos de búsqueda estadística y metaheurística del espacio de soluciones con el fin de realizar una selección de características, encontrando el Vector de Selección de menor número parámetros que más influye en la salida.

Por tanto, el objetivo principal del programa final es realizar una Selección de Características (Feature Selection) de entre los parámetros que cargamos en el programa. Los algoritmos de búsqueda que implementa son ES, SA, GA y AIS. Además puede realizar estudios sobre la ELM, y su dependencia con el número de neuronas, número de VTr, función e transferencia... Permite llevar a cabo también un filtrado de parámetros, analizando cada uno de ellos por separado y la relación entre cada par de ellos, incluyendo además búsqueda determinista mejora de tipo SFS y SBE.

Características del programa:

- El programa está escrito enteramente en lenguaje C sobre entorno Linux, siendo portable a cualquier entorno Unix. El código está totalmente comentado y escrito en lengua inglesa.
- Sigue un diseño estructurado y modular donde las diferentes características del programa se encuentran en diferentes ficheros.

- Utiliza la librería de hilos POSIX para poder realizar operaciones concurrentemente.
- Utiliza la librería GSL - GNU Scientific Library para la realización de la mayoría de operaciones matemáticas complejas (Algebra lineal principalmente). Para poder compilar y utilizar el programa es necesario tener instalada esta librería.

- Se ha utilizado el editor "Geany" para la generación del código si bien cualquier otro vale.
- El programa se compila y enlaza usando el software "gcc" mediante el uso de un Makefile.
- Posee una interfaz basada en ficheros de configuración que indicamos al programa mediante la línea de comandos.
- Los Vectores de Training y Testing se leen como variables "double" de ficheros de texto.
- El programa realiza diversas representaciones graficas haciendo uso del software "gnuplot".

8.1 Estructura de ficheros

El programa posee un diseño modular donde cada una de sus funcionalidades se encuentra en un fichero independiente de código ".c", con su correspondiente archivo de cabeceras ".h". Como es natural, en el archivo de cabeceras se encuentra la declaración de las funciones y tipos de datos (estructuras, uniones…) y en los archivos de código se encuentra la definición de las funciones. En el directorio principal del programa se encuentran los siguientes ficheros y carpetas referentes al código del programa:

- **main.c**: Archivo de código principal del programa, desde este llamamos a las funciones que cargan la configuración y los vectores de training y testing, los normalizan, convierten a vectores GSL y llaman a la operación especificada.
- **ELMheader:** Archivo de cabeceras principal que incluye al resto de cabeceras y aporta constantes globales del programa.
- **Makefile**: Archivo de configuración de compilación y enlazado.
- **code**: Carpeta con los archivos de código de cada uno de los ficheros del programa.
- **headers**: Carpeta con las cabeceras asociadas a los ficheros de código.

A parte de estos ficheros, encontraremos el ejecutable **ELM** cuando compilemos el programa, además de 4 carpetas más que no forman parte del código en sí sino que son parte del input y output del programa:

- **bin:** Carpeta el código objeto compilado de los ficheros de código ".o".
- **config:** Carpeta donde se encuentran los ficheros de configuración del programa. Estos archivos contienen toda la información que necesita el programa para su ejecución. Existe un archivo de configuración principal, con las características básicas de la ELM y después un archivo de configuración para cada tipo de operación que se puede realizar.
- **input_data:** Carpeta donde se encuentran los ficheros con los vectores de training y testing.
- **Results:** Carpeta donde se encuentran los resultados del programa, cada operación tiene su propia carpeta dentro de esta y comparten otras tantas.

Ficheros de código y cabeceras:

A continuación se despliega el conjunto de ficheros de código ".c" que forman el programa, estos están ubicados dentro de la carpeta "codes" y cada uno de ellos posee un archivo de cabeceras ".h" asociado en la carpeta "headers" que contiene la definición de tipos de datos y declaración de funciones asociadas.

- **ELM.c:** Contiene las funciones necesarias para realizar la ELM y obtener el RMSE de testing.
- **threads.c**: Contiene las funciones necesarias para gestionar (crear, utilizar y destruir) los hilos que realizarán las diversas ELM del programa.
- **ELMs.c:** Contiene las funciones necesarias para realizar diversas ELM con las opciones deseadas y para hacer un estudio de la aleatoriedad de la ELM.
- **Graph.c**: Contiene las funciones necesarias para la realización estudios gráficos de los tiempos y RMSE de la ELM en función del número de neuronas, vectores de training…
- **ES.c**: Contiene las funciones necesarias para la realización de una Búsqueda Exahustiva (Exahustive Search) dentro del espacio de soluciones especificado.
- **SA.c**: Contiene las funciones necesarias para la realización del Temple Simulado (Simulated Anneling) para realizar una búsqueda metaheurística dentro del espacio de soluciones.
- **GA.c**: Contiene las funciones necesarias para la realización del Algoritmo Genético (Genetic Algorithm) para realizar una búsqueda metaheurística.

- **AIS.c**: Contiene las funciones necesarias para la realización del algoritmo de tipo Sistema Inmunológico Artificial (Artificial Immune System).
- **FP.c**: Contiene las funciones necesarias para la realizar un estudio parámetro por parámetro y la afinidad de cada pareja de parámetros respecto a la ELM (Algoritmo de Aprendizaje). Este estudio podría ser utilizado para realizar un filtrado inicial de características o para alimentar otro tipo de algoritmo de búsqueda. Además realiza los algoritmos de búsqueda deterministas SFS y SBE.

- **gsl_func.c**: Contiene funciones para realizar operaciones con vectores y matrices GSL que no aparecen en la librería GSL estándar.
- **bitvector_func.c**: Contiene las funciones para tratamiento de los vectores de bits que constituyen los Vectores de Selección. Estos vectores son un array de variables unsigned int.
- **process_param.c**: Contiene las funciones necesarias para cargar y analizar los distintos ficheros de configuración del programa. Las funciones que leen la configuración de los ficheros, lo hacen palabra a palabra por lo que estas se pueden modificar pero no se pueden quitar ni añadir, excepto en aquellas partes donde explícitamente se especifique como por ejemplo aquellas en las que podemos indicar un número variable de Vectores de Selección.
- **output_func.c**: Contiene las funciones necesarias para la representación de gráficas a partir de vectores y vectores GSL con el programa "gnuplot" de Linux.
- **general_func.c:** Contiene funciones generales sueltas del programa como pueden ser las funciones de carga de vectores de training y testing o la conversión de vectores a GSL.
- **aux_func.c**: Contiene funciones para el tratamiento de arrays y matrices de datos:
  - Funciones de ordenamiento.
  - Funciones de max y min.
  - Funciones de inicialización.
  - Funciones de normalización.

8.2 Compilación y Enlazado

El código fuente debe ser primero compilado y enlazado para generar el programa y poder ser ejecutado. Para este efecto disponemos de un fichero **Makefile**, fichero que contiene las directrices necesarias para generar el ejecutable. El fichero tiene 2 opciones principales:

- **all:** Compila y enlaza todos los módulos para generar el ejecutable.
- **clean**: Elimina los archivos intermedios y residuos de la compilación.

Para generar el ejecutable simplemente ejecutamos el archivo Makefile, para ello abrimos una terminal (Alt + Ctr + t), nos movemos al directorio principal del código y usamos el comando "make":

Tabla 15 Compilación del programa

```
$ cd Desktop/ELM
$ make
```

Al ejecutarlo, se verán en la terminal los diferentes comandos de compilado y enlazado junto con los posibles Warnings y Errores del código. Si todo se compiló correctamente aparecerá el mensaje "********** SUCESS **********"

La ejecución del Makefile generará el archivo ejecutable ELM en el mismo directorio. Además se producirán los archivos binarios de cada archivo de código ".c" en la carpeta bin.

Dado que el programa hace uso de la librería GSL que no está incluida de serie en la distribución Ubuntu estándar, necesitamos instalarla. Tenemos 2 opciones:

1. **Shell de comandos:**

Tecleamos en una terminal el siguiente comando:

Tabla 16 Instalación librería GSL

```
$ sudo apt-get install libgsl0-dev libgsl0ldbl
```

2. **Ubuntu software center:**

Abrimos el "Ubuntu software center" e instalamos las librerías libgsl0-dev y libgsl0ldbl.

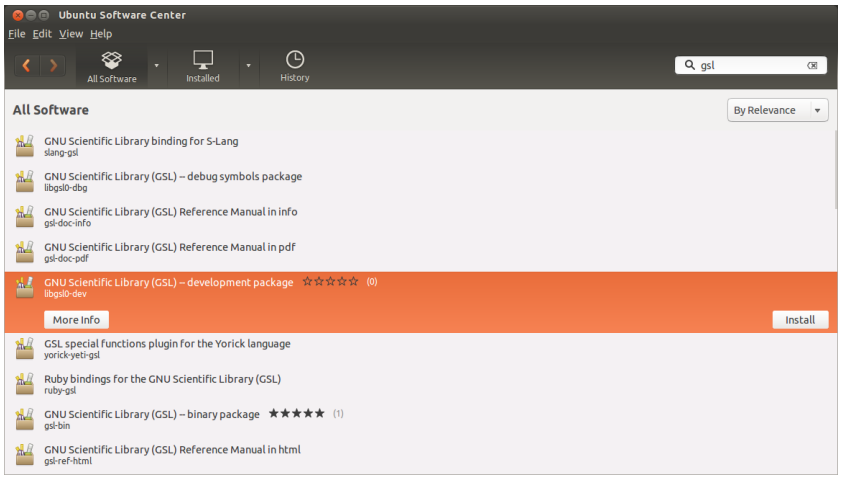


Fig. 41 Instación de la librería GSL mediante Ubuntu Software Center.

8.3 Depuración

Como herramienta de depuración se ha utilizado el programa valgrind. Esta herramienta nos asegura que no haya:

- Lagunas de memoria.
- Utilización de valores no inicializados.
- Buffer Overflows.

Sin embargo no se puede asegurar que el software esté 100% depurado y puede haber fallos según la plataforma donde se utilice. Este programa además suele dar un fallo de laguna de memoria referente a los hilos de la forma:

Tabla 17 Laguna de memoria de los hilos

```
==826==    by 0x400AE1F: _dl_new_object (dl-object.c:77)
==826==    by 0x4006447: _dl_map_object_from_fd (dl-load.c:1051)
==826==    by 0x40083AF: _dl_map_object (dl-load.c:2568)
==826==    by 0x4012D6C: dl_open_worker (dl-open.c:225)
==826==    by 0x400ECCE: _dl_catch_error (dl-error.c:178)
==826==    by 0x441BE20: do_dlopen (dl-libc.c:89)
==826==    by 0x42DED4B: start_thread (pthread_create.c:308)
==826==    by 0x43E2BAD: clone (clone.S:130)

==826== LEAK SUMMARY:
==826==    definitely lost: 0 bytes in 0 blocks
==826==    indirectly lost: 0 bytes in 0 blocks
==826==      possibly lost: 0 bytes in 0 blocks
==826==    still reachable: 954 bytes in 4 blocks
==826==         suppressed: 0 bytes in 0 blocks
```

Esto realmente no es una laguna de memoria y no podemos evitar el Warning.
Dado que el programa ha sido diseñado para problemas de tamaño medio, ciertos buffer internos han sido dimensionados teniendo en cuenta

8.4 Interfaz de uso

El programa requiere de 3 parámetros para su funcionamiento, que son facilitados al mismo por medio de la Shell de comandos en la forma:

./ELM ./main_config -Operation ./Operation_config

- **./main_config**: Es la dirección (PATH) del archivo de configuración principal del programa.
- **-Operation**: Es el tipo de operación que realizará el programa
- **./Operation_config**: Es la dirección (PATH) del archivo de configuración de la operación.

8.4.1 Archivo de configuración principal del programa

Este fichero contiene la información necesaria para poder cargar los Vectores de Training y Testing, los parámetros de la ELM y el número de hilos que utilizara el programa. Un ejemplo de este archivo de configuración podría ser:

```
Train_data_file         ./input_data/trainolas.dat
Train_result_file       ./input_data/trainresul.txt
Test_data_file          ./input_data/testola.dat
Test_result_file        ./input_data/testresul.txt

Train_vector_number     6000
Test_vector_number      1300
Input_vector_size       14
Output_vector_size      1

Number_of_ELM           1
Number_of_Neurons       20
Activation_func         0
N_threads               4
```

Fig. 42 Ejemplo de archivo de configuración prinpal del programa.

Como se puede ver, los datos que tenemos que especificar en este archivo son:

- **Dirección de los ficheros donde se encuentran los Vectores de Training y Testing.**
  Podemos especificar ficheros diferentes para Vectores de Training y Testing, de esta manera sabemos exactamente con qué vectores entrenamos la ELM. Para cada grupo de vectores tenemos que especificar 2 ficheros
  - Archivo con los vectores de input.
  - Archivo con los vectores de output asociados a los de input.

  Si queremos realizar una ELM donde los Vectores de Training y Testing se cojan aleatoriamente de un solo grupo de ficheros, basta con poner los ficheros de Testing a "NONE". Si no indicamos archivo de vectores de output, se considera que el output es el primero de los inputs.

  El programa supone que los datos de los ficheros son valores double en texto plano separados por espacios en blanco. En la mayoría de los casos se deberá **modificar la función de carga de Vectores para adaptarla a las necesidades de cada problema.**

- **Número de Vectores de Training y Testing a cargar.**
  Un mayor número de vectores de training puede suponer un mayor y normalmente mejor entrenamiento de la ELM pero se debe evitar el "overfitting". El tiempo computacional requerido para realizar la ELM crece linealmente con el número de vectores utilizado.

- **Número de parámetros del Input y Output.**
  Está demostrado que una RNA con varios output funciona peor que realizar una RN por cada uno de los output. Nuestro programa solo es capaz de realizar la ELM para un solo output, pero dado que la mayor parte del tiempo se gasta en realizar la inversa de Penrouse Moore que no se puede compartir para diferentes output, no afecta en gran medida al propósito del mismo.

- **Número de ELMs.**
  Dada la naturaleza aleatoria de los pesos y bias de la ELM, el resultado RMSE puede diferir bastante entre 2 ejecuciones aun teniendo los mismos Vectores de Training y Testing y número de Neuronas Ocultas. Este efecto se puede paliar realizando varias ELM con las mismas características y calculando el RMSE medio de estas. Este es el número de ELMs que se realizarán cada vez que se pida hacer una ELM en cualquier parte del programa.

- **Número de neuronas.**
  El número de Neuronas Ocultas es un factor muy importante en la ELM, impone la capacidad de aprendizaje de la red y el tiempo computacional requerido para realizarla.

- **Función de activación:**
  Es la función de activación que se utilizará en las Neuronas Ocultas, tenemos varias opciones:
  - 0 Sigmoide 1 Seno
  - 2 Coseno 3 Tangente hiperbólica

- **Número de hilos.**
  El programa utiliza un sistema de hilos POSIX para poder realizar varias ELMs en paralelo. Debemos elegir un número de hilos no mucho mayor al número de procesadores que tenga nuestra máquina ya que la ganancia de tiempo no subirá significativamente y se perderá tiempo en el cambio de un hilo a otro. Aunque tengamos un solo procesador es mejor utilizar más de un hilo debido a que si uno de ellos está ocioso realizando operaciones E/S, el otro puede utilizar la CPU.

8.4.2 Operación a Realizar

Una vez cargado el fichero de configuración principal de la ELM, el programa tiene todos los datos para realizar una ELM con todos los parámetros de entrada y salida lo que es útil para realizar una única ELM deseada. Si quisiésemos realizar varias ELM siguiendo una lógica dada en un intento por realizar un estudio realizar una selección de los mejores parámetros, esta opción es pobre ya que:

- Cada vez que lanzásemos el programa, este tendría que cargar todos los vectores otra vez.
- Si queremos seleccionar un subconjunto de parámetros. Tendríamos que externamente modificar con otro programa los Vectores, añadiendo o quitando los parámetros deseados.

Es por ello que el programa desarrolla una serie de mecanismos para que no tenga que ser relanzado para cada ELM, el programa permite:

- **Seleccionar internamente los parámetros a utilizar de entre los totales cargados:**
  Para ello disponemos de los llamados "Vectores de Selección", estos vectores consisten en un array binario formado por variables "unsigned int". Cada uno de los bits del vector representa un parámetro, si el bit está a 1 incluimos el parámetro, y si está a 0 no.

- **Realizar diversos algoritmos de búsqueda ELM y estudios de la ELM.**
  El algoritmo permite varias opciones que automáticamente utilizaran diversos "Vectores de Selección" de manera lógica para buscar en el espacio de soluciones o mostrar propiedades de la ELM y los parámetros.

Cada operación tiene asociado 4 elementos clave:

- **Fichero de configuración**.
  Es el archivo donde se encuentran los parámetros necesarios para llevar a cabo la operación, debe especificarse a través de la línea de comandos.
- **Estructura de datos**
  Estructura de programación C donde se almacenar los parámetros de configuración leídos del archivo y el resto de variables necesarias asociadas a la operación.
- **Funcion asociada**
  Funcion de programación C que realiza la operación pasándole la estructura.
- **Carpeta de resultados**
  Carpeta dentro del directorio ./Results con los datos específicos de la operación.

Las operaciones a realizar son:

- **ELMs**
  Realiza la ELM de Vectores de Selección especificados. Podemos indicar las distintas propiedades que tenga la ELM así como operaciones especiales como mostrar los tiempos de cada operación, que generé una gráfica con la regresión o que compruebe que inversa de Penrouse-Moore.
  Además si se especifica, es capaz de hacer un estudio estadístico básico sobre la aleatoriedad de la ELM debido a la distribución aleatoria de pesos y bias.

- **Graph**
  Genera gráficas para mostrar cómo influye el número de Neuronas Ocultas y número de Vectores de Training al RMSE de la ELM y el tiempo de ejecución. También permite ver cómo influye el número de parámetros utilizados.

- **ES (Exhaustive Search)**
  Realiza una búsqueda exhaustiva dentro del espacio de soluciones (Vectores de Selección) especificado. Cabe destacar que se pueden indicar los parámetros de los que se hará la búsqueda, los que están siempre a 1 y los que están siempre a 0. Así como el número mínimo y máximo de bits que puede tener la solución.

- **SA (Simulated Anneling)**
  Realiza una búsqueda local y estocástica usando como metaheurística el algoritmo del Temple Simulado. Como peculiaridad, esta implementación utiliza varias soluciones independientes y concurrentes para aprovechar la programación por hilos.

- **GA (Genetic Algorithm)**
  Implementa un Algoritmo Genético para realizar la selección de características.

- **AIS (Artificial Immune System)**
  Implementa un Algoritmo de Sistema Inmune para realizar la selección de características.

- **FP (Feature Properties)**
  Realiza un estudio individual y por parejas de la capacidad de predicción de cada característica. Además permite realizar filtrado de características y algoritmo semi-deterministas mejorados de tipo greedy como SFS y SBE para llevar a cabo una selección inicial de características.

8.5 Archivo principal main.c

Como en cualquier programa en C, el archivo de código principal es aquel que contiene la función **main()** y contiene la lógica de control del programa, manejando el flujo de operación del mismo, para simplificarlo, lo que se ha pretendido realizar todas las operaciones llamando a funciones, limitando el código del main.c a una inicialización de variables y una máquina de estados.

Las operaciones que realiza este archivo son:

1. Declarar las variables necesarias para la ejecución del programa.
   Estas variables son principalmente:
   - Arrays de tipo "double" donde guardar los Vectores de Testing y Training. Cada uno de estos vectores tiene su vector GSL asociado para su transformación.
   - Estructuras de datos de las operaciones.
   - Variables temporales para generar números aleatorios
   - Variables globales:
     i. n: Número de parámetros totales cargados.
     ii. n_var_chosen: Numero de variables unsigned int necesarias para cada "Vector de Selección".
     iii. ELM_p: Estructura de datos ELM.
     iv. Thread_p: Estructura de datos de los hilos.

2. Comprobar que se hayan dado los 3 parámetros de la Shell correctamente.

3. Cargar la configuración principal de la ELM.

4. Comprobar la operación que ha sido solicitada y cargar su fichero de configuración a consecuencia.

5. Cargar los Vectores de Training y Testing en los arrays "double" de acuerdo con la función de carga especificada. Se carga todos los valores de cada parámetro en un array diferente. Dependiendo del tipo de carga, los vectores de permutarán unos con otros para eliminar posibles dependencias.

6. Tratar los Vectores de Training y Testing, lo que incluye:
   - Normalización entre -1 y 1.
   - Transformación de los arrays de variables "double" a Vectores GSL.

7. Inicializar la estructura ELM y la estructura de Hilos.
   Para ello damos valores a los campos de la variable "ELM_p" y llamamos a las funciones que reservan memoria e inicializan los hilos a partir de la estructura ELM.

8. Realizar la operación solicitada.
   Esto consiste en una simple máquina de estados donde, en función de la operación seleccionada, llamamos a una única función que se encargará del resto de la lógica.

9. Liberar la memoria principal del programa.

8.6 ELM

El corazón del programa es la realización de la ELM de un conjunto de características seleccionadas entre las totales y obtener el error de testing RMSE resultante. Cuando digamos "Realizar la ELM" nos referimos a obtener el RMSE de testing. Para realizar la operación es necesario un conjunto de información mínima, dicha información se encuentra en una estructura de tipo ELM_params. A partir de esta estructura se crearán las matrices de vectores de Training y Testing (con los parámetros seleccionados) y se procederá a realizar la ELM. Los parámetros de la estructura son:

1. **n**: Número total de parámetros de entrada cargados.
2. **n_chosen**: Numero de parámetros de entrada seleccionados de entre los totales.
3. **chosen_vector**: "Vector de Selección" con los parámetros elegidos de entre los totales.
4. **n_ELMs:** Número de veces que se realizará la ELM, devolviendo el error RMSE medio.

5. **activation_f:** Función de transferencia de las Neuronas Ocultas de la ELM.
6. **Nh**: Número de Neuronas Ocultas utilizadas en la ELM.

7. **FLAGS**: Flags que indican operaciones adicionales a realizar por la función. Cada bit de este campo, cuando puesto a 1, activa una funcionalidad extra:
   - Bit 0: Muestra los tiempos que se tarda en computar las diferentes operaciones de la ELM.
   - Bit 1: Comprueba el error de Inversa de Penrouse-Moore a través de su propiedad
     $$H \cdot H^{+} \cdot H = H$$
     Esta operación gasta mucho tiempo y memoria, sobretodo memoria así que hay que tener cuidado con las dimensiones de la matriz H pedida.
   - Bit 2: Crea una carpeta con datos y gráfica de regresión de la ELM.
     - Data: Contiene los valores de configuración de la ELM como pueden ser la hora de realización, parámetros de entrada, RMSE, Numero de vectores de training y testing, número de neuronas.
     - Regression: Archivo de texto que contiene 2 vectores de valores double.
       - Primer Vector: Contiene los output reales de los vectores de test.
       - Segundo Vector: Contiene los output predichos por la ELM
     - Plot.png: Contiene la representación de Regresión.

8. **x_train_param[n][Ntrain]:** Array de "n" vectores GSL donde cada vector contiene las muestras correspondientes a un parámetro de entrada. Cada uno de los "n" vectores tiene tantas muestras como número de VTr. Un VTr se obtiene seleccionando el mismo elemento "m" de cada uno de estos vectores. Haciendo uso del VS "chosen_vector", se seleccionarán aquellos vectores correspondientes a los parámetros de entrada que han sido seleccionados.
9. **x_test_param[n][Ntest]:** Lo mismo que "x_train_param" pero para los vectores de test.

10. **y_train[Ntrain]:** Vector GSL con el output referente a cada VTr.
11. **y_test[Ntest]:** Vector GSL con el output referente a cada VTe.

12. **t_max y t_min**: Valores de normalización del output. Dado que hemos normalizado tanto los vectores de input como de output, para calcular la RMSE real del problema dado, necesitaremos desnormalizar los valores salida.

13. **RMSE**: Error de testing de la ELM, si **n_ELMs** es mayor que 1, será la media de las ELM hechas.

La función get_ELM_RMSE(ELM_params *p) opera con estos parámetros para calcular la RMSE de la ELM. Los pasos que sigue para obtenerla son:

i. Obtener la matriz ***X*** de los vectores de entrada, utilizando las características seleccionadas.
A partir del Vector de Selección "chosen_vector" y los vectores GSL de cada uno de los parámetros de entrada, **x_train_param[n][Ntrain] y x_test_param[n][Ntest]** creamos 2 matrices GSL con los vectores de Training y de Testing.

```
Xtrain = get_selected_input_matrix(x_train_param, chosen_vector, n_chosen);
Xtest  =  get_selected_input_matrix(x_test_param,  chosen_vector,  n_chosen);
```

Si por algún motivo el vector elegido es nulo, siendo 0 el número de inputs, dado que la ELM no se podría realizar con 0 inputs, elegimos el primero de los parámetros.

ii. Dar valores iniciales aleatorios [-1, 1] a los pesos W y bias de las Neuronas Ocultas.

```
set_WeightBias(&W, &b, Nh, n_chosen);
```

Para la generación de los números aleatorios se ha utilizado la función GSL gsl_rng_uniform() alimentada con una semilla temporal.

iii. Calcular la matriz de salida de la ELM *H* para la matriz con los vectores de training.

```
H = get_H_matrix(Xtrain,Nh,W,b,activation_f);
```

La función de transferencia de las Neuronas Ocultas viene dada por el parámetro activation_f.

iv. Calcular la matriz inversa de Penrouse Moore de H.

```
Hinv = get_Hinv_matrix2(H);
```

v. Comprobar el error de la inversa de Penrouse Moore.

```
error_Hinv = check_psudoinverse(H, Hinv);
```

Esto se hace para asegurar que el método que calcula la inversa es correcto.

vi. Calcular el vector de Betas.

```
beta = get_beta_vector(Hinv, y_train);
```

vii. Obtener la matriz *H* de los vectores de testing.

```
H = get_H_matrix(Xtest,Nh,W,b);
```

viii. Obtener el vector de output, desnormalizarlo y obtener el RMSE.

```
y_pred = test_ELM(H, beta);
desnormalize_array(y_pred->data, x_test_param[0]->size, t_min, t_max);
desnormalize_array(y_test_aux->data, x_test_param[0]->size, t_min, t_max);
ERMS = rmse_gsl_vector(y_pred,y_test_aux);
```

ix. Liberar la memoria reservada para la ELM.

8.6.1 Algoritmo usado para calcular la inversa de Penrouse-Moore

La operación que más tiempo consume en la realización de la ELM es el cálculo de la inversa de Penrouse Moore, existiendo diversos algoritmos para el cómputo de la misma. Por su facilidad y velocidad se ha elegido la función "geninv" [10] cuyo código en MATLAB se expone a continuación. Se ha exportado este código a C, usando la librería GSL para realizar las operaciones algebraicas.

```
function Y = geninv(G)
% Returns the Moore-Penrose inverse of the argument
    % Transpose if m < n
[m,n]=size(G); transpose=false;
if m<n
    transpose=true;
    A=G*G';
    n=m;
else
    A=G'*G;
end
    % Full rank Cholesky factorization of A
dA=diag(A); tol= min(dA(dA>0))*1e-9;
L=zeros(size(A));
r=0;
for k=1:n
    r=r+1;
    L(k:n,r)=A(k:n,k)-L(k:n,1:(r-1))*L(k,1:(r-1))';
    % Note: for r=1, the substracted vector is zero
    if L(k,r)>tol
        L(k,r)=sqrt(L(k,r));
        if k<n
            L((k+1):n,r)=L((k+1):n,r)/L(k,r);
        end
    else
        r=r-1;
    end
end
L=L(:,1:r);
    % Computation of the generalized inverse of G
M=inv(L'*L);
if transpose
    Y=G'*L*M*M*L';
else
    Y=L*M*M*L'*G';
end
```

Fig. 43 Algoritmo geninv en MATLAB.

8.6.2 Importante: Comprensión del "Vector de Selección"

Como se ha venido diciendo, un VS consiste en un array de variables unsigned int donde cada bit de las variables representa un Input de la ELM.

- **A nivel de funcionamiento interno**: Al leer los Inputs, estos se van guardando desde las posiciones más bajas del array hacia las más altas por lo que el primer Input se guardará en la posición VS[0][0] y por ejemplo si un unsigned int tiene 32 bits, el Input 54 será guardado en la posición VS[1][21].

- **A nivel de funcionamiento externo:** La forma que tiene el programa de recibir e indicar vectores de selección es mediante un array tipo char de '1s' y '0s' donde los primeros bits de la izquierda representan los primeros parámetros leídos.

  11010110001010111010101

  Si al indicar un vector no indicamos todos los input, el resto serán tomados como 0.

Para hacer la transformación entre array de unsigned int y array de char, al imprimir y recibir los array de char, siempre se empieza imprimiendo desde las posiciones más bajas del array de unsigned int.

8.7 Hilos

Con el fin de agilizar la ejecución del programa de cara a la realización de muchas ELM, se ha creado un sistema de hilos capaz de lanzar varias ELM concurrentemente. El sistema de hilos está pensado para actuar de la siguiente forma:

Dado un conjunto de "n_job" Vectores de Selección dispuestos consecutivamente en un array, de los cuales queremos calcular el RMSE de su ELM y guardarlo en un "Array de Resultados", llamamos a una función la cual simplemente creará los hilos y estos irán cogiendo "Vectores de Selección", los añadirán a una estructura ELM ya inicializada y realizarán la ELM, guardando el RMSE en el "Array de Resultados".

8.7.1 Implementación

Para la utilización de hilos son necesarios una serie de parámetros y variables que han sido agrupadas en la estructura de tipo Thread_params, esta estructura es **global** al programa ya que debe poder ser accedida por los diferentes hilos.

```
typedef struct {
    int n_threads;                  // Number of threads we can have simmultaneosly.
    int total_thread_jobs;          // Total number of thread jobs to do.
    int thread_job;                 // Current Job number taken by a thread so far.

    ELM_params * ELM_p_thread;      // Structures with the data for an ELM go of a theread.

    unsigned int **Thread_chosen_array; // Array with the chosen vectors for every ELM thread job.
    double *Thread_results;             // Array with the results of every job.

    pthread_mutex_t job_access;     // Mutex for the access to the "thread_job" variable.
    pthread_mutex_t output_access;  // Mutex for the access to the output files.
    pthread_t* threads_ID;          // Array with the threads ID's.

} Thread_params;
```

Fig. 44 Estructura de parámetros del sistema de hilos.

- **n_threads**: Es el número de hilos que serán creados para realizar los trabajos
- **total_thread_jobs**: Es el número total de trabajos (ELM) a realizar por el conjunto de hilos.
- **thread_job**: Es el último trabajo cogido por un hilo hasta el momento. Cuando thread_job = total_thread_jobs, los trabajos habrán acabado y los hilos se destruirán, continuando así el flujo del programa. Esta variable es accedida y alterada por hilos, incrementándose cada vez que un hilo coge un trabajo y por tanto debe ser protegida mediante un **mutex**.
- **ELM_p_thread[]**: Dado que para realizar una ELM es necesario darle una estructura con todos los datos necesarios. Estas estructuras nos servirán para anexarles el "Vector de Selección" del trabajo y así poder llamar a la función de la ELM. Por este motivo, todas la ELM de los hilos tienen los mismos parámetros de la ELM, exceptuando el VS.
- **Thread_chosen_array [total_thread_jobs]:** Array de "Vectores de Selección" sobre los que se realizará la ELM. Estos VS se copiarán en una estructura ELM_p_thread[x] para ser usados.
- **Thread_result [total_thread_jobs]:** Array de resultados (RMNE) de cada una de las ELM. El vector Thread_chosen_array[i] tienes asociado el resultado Thread_result [i]
- **job_access**: Mutex para el acceso a la variable "thread_job" accedida por los hilos.
- **Output_access**: Mutex para el acceso a escritura de ficheros de resultados.
- **threads_ID**: Array con los Identificadores de los Hilos para su manipulación.

8.7.2 Flujo de utilización

Para poder hacer uso de las funcionalidades de los hilos debemos crearlos, inicializarlos, usarlos y destruirlos. De acuerdo a esto se han codificado una serie de funciones que realizan estas operaciones.

1. **Reserva de memoria para los hilos:**

La función set_up_threads() indica al sistema cuántos hilos se van a crear y reserva memoria para su identificación y las estructuras ELM asociadas a estos. Esta función no crea los hilos aún, estos se crearán cuando haya trabajo que hacer.

2. **Inicialización de la estructura:**

La función init_threads() se encarga de inicializar los valores de las estructuras **ELM_p_thread** de los hilos, copiando en ellas las propiedades de la estructura ELM_p pasada como argumento.

3. **Realización de un trabajo:**

Para realizar la ELM mediante hilos de una serie de "Vectores de Selección" y guardarlo en un "Array de Resultados" simplemente tenemos que llamar a la función threads_WorkOut(). A esta función le indicamos:

- **Bitvectors**: Array con el conjunto de "Vectores de Selección" de los que realizar la ELM.
- **Results**: Array donde se quiere que se guarden los resultados (RMSE) de la ELM.
- **num_jobs**: El número de ELMs a realizar.
- **Thread_p**: Estructura de parámetros de hilos para realizar el trabajo.

Esta función copia los 3 primeros parámetros en sus homólogos dentro de la estructura de hilos, crea los hilos y espera a que acaben. Los hilos por si mismos empezarán a coger los trabajos, realizando las ELM solicitadas. Cuando no queden trabajos se destruirán. Cuando se destruyan todos los hilos, el programa principal podrá seguir avanzando. Cabe mencionar que al copiar los arrays, solo se copia la dirección, no el contenido del vector, los vectores de los hilos no tiene reservada memoria.
La función de hilos realiza la siguiente operación:

```
void* get_ELM_RMSE_thread(void *p){     // Thread function

    int doing_job;                          // Job that the thread will do at every iteration
    ELM_params *p_ELM = (ELM_params *)p;    // Give type to the pointer

    while (1) {
        pthread_mutex_lock (&(Thread_p.job_access));            // Mutex block
        if (Thread_p.thread_job < Thread_p.total_thread_jobs){  // Get any remaing job
            doing_job = Thread_p.thread_job;        // Use the job number as index for arrays
            Thread_p.thread_job++;
        } else {
            doing_job = -1;
        }
        pthread_mutex_unlock (&(Thread_p.job_access));  // Mutex unblock

        if (doing_job == -1){
            break;                                      // Exit the while(1)
        }
        p_ELM->chosen_vector = Thread_p.Thread_chosen_array[doing_job]; // Get the job
        Thread_p.Thread_results[doing_job] = get_ELM_RMSE(p_ELM);       //-------->Do the ELM job<----------
    }
    pthread_exit(NULL); // Exit thread
}
```

Fig. 45 Contenido de la función de un hilo.

4. **Destrucción:**

La función destroy_threads() libera la memoria reservada de los hilos.

8.8 ELMs

Esta es la opción más básica de todas y nos permite realizar la ELM de los "Vectores de Selección" especificados, obteniendo su RMSE. Podemos indicar cuántas ELM realizar de cada VS y los FLAGS de las mismas. Además, dado que la ELM tiene un carácter aleatorio debido a sus pesos y bias, esta opción nos permite calcular la distribución estadística del RMSE. Esto puede ser útil por ejemplo para determinar el número de veces que debemos lanzar la ELM, obteniendo el RMSE como la media de las iteraciones, para dar como válido y representativo el RMSE obtenido.

Esta opción se puede utilizar principalmente para:

- Realizar la ELM aislada de un conjunto de Vectores de Selección. Podemos indicar la obtención de la gráfica de regresión o no mediante los FLAGS.
- Realizar un estudio temporal de las distintas partes de las que forma parte la realización de la ELM indicando la opción con los FLAGS.
- Realizar un estudio estadístico de los posibles valores de RMSE para el primer Vector de Selección indicado.

Al terminar el programa se generarán los siguientes datos:

- La regresión de las ELM especificadas en la carpeta ./Results/RMSE. Sólo si se ha indicado el FLAG de obtener dicho resultado.
- Archivo con los datos de la distribución estadística de la ELM en la carpeta ./Results/Graph. Solo si ha indicado en el archivo de configuración.

Modo de empleo:

./ELM ./ELM_config_file -ELMs ./ELMs_config_file

8.8.1 Fichero de configuración

El fichero de configuración de esta operación nos permite elegir las diferentes variantes de las ELM a realizar, como pueden ser los parámetro de entrada seleccionados o el número de ELM a realizar. Sus características son leídas por la función get_ELMs_params() y guardadas en una estructura de tipo ELMs_data para ser luego utilizadas por la función ELMs_op().
Los parámetros que podemos configurar son:

- **Vectores de Selección**: Son los Vectores de Selección sobre los que realizaremos la ELM. Para indicarlos primero indicamos el número de ellos seguido de los mismos expresados como una cadena de unos y ceros donde los primeros bits corresponden a los primeros Inputs leídos.

- **Número de ELMs:** Número de ELMs a realizar de cada uno de los vectores.
- **FLAG:** FLAGS de las ELM a realizar.
- **Distribución estadística de la ELM:** Si queremos realizar un estudio estadístico de la ELM del primer VS especificado, indicamos este campo como "yes". De hacerlo tenemos que indicar otros 2 parámetros:
  - **Número de ELMs:** Número de ELMs a realizar para hacer la estadística.
  - **Divisiones de la estadística:** Después de realizar las ELM, se obtendrá el valor máximo, mínimo, la media y la varianza de los RMSE. A continuación realizará una gráfica en la que se muestra por rangos como se distribuyen los distintos valores de RMSE obtenidos. Esta variable es el número de intervalos entre el valor máximo y mínimo de la ELM en los que se dividirá el rango.

8.8.2 Ejemplo

Se va a utilizar el siguiente fichero de configuración:

```
Selection_Vectors       3
11111111111111
00001111111111
11110000000000
N_ELMs                  3
FLAGS                   4

Get_ELM_distribution    yes
Num_of_ELMs             5000
Num_of_divisions        50
```

Como se puede observar, se ha ordenado realizar 3 ELMs de 3 Vectores de Selección diferentes poniendo además el bit 2 del FLAG a 1 lo que indica que se obtendrá la gráfica de regresión de las ELMs realizadas.
También hemos indicado que se realice un estudio de la distribución estadística del primer Vector de Selección, realizando 5000 ELMs del mismo y agruparlas en 50 divisiones.

Fig. 46 Archivo de configuración de la operación ELMs.

Las diferentes gráficas de regresión formadas, se encuentran en la carpeta ./Results/RMSE, cada gráfica se encuentra en su determinada carpeta junto con los parámetros de la ELM utilizada, como el VS, el número de neuronas…

A continuación se muestran 4 gráficas de los resultados generados, las primeras 3 son las gráficas de regresión de los 3 VS y la última es la distribución de probabilidad del primer VS.

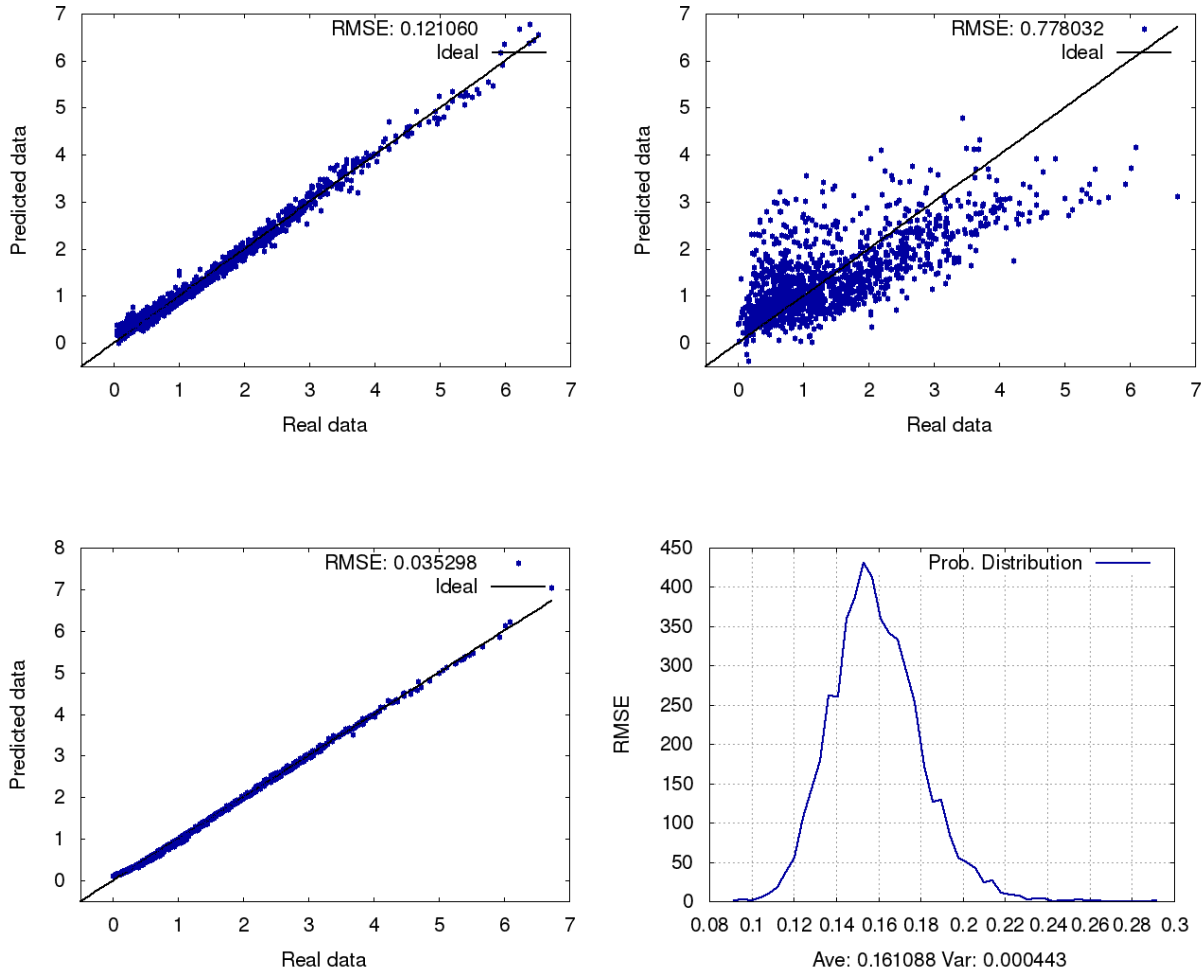


Fig. 47 Resultados operación ELMs

8.9 Graph

Esta opción nos permite realizar un estudio temporal y del RMSE de la ELM en función del número de Neuronas Ocultas, Vectores de Training, número de parámetros utilizados y función de transferencia. Podemos realizar estudios de 3 tipos:

1. **En función del número de Vectores de Training.**

Esta opción realiza la ELM para un VS dado en función del número de Neuronas Ocultas y el número de Vectores de Training utilizado. En el eje X tenemos situado el número de neuronas y dibuja un curva por cada número de Vectores de Training que deba realizar.
La información que podemos obtener con este tipo de estudio es:

- La relación entre el RMSE y el número de Neuronas Ocultas utilizado.
- La relación entre el RMSE y el número de Vectores de Training utilizado.
- La relación entre el tiempo y el número de Neuronas Ocultas utilizado.
- La relación entre el tiempo y el número de Vectores de Training utilizado.

La información que tenemos que especificar es:

- Para el eje X, Neuronas: Número de Neuronas inicial, final y espaciado entre ellos.
- Para el eje Y (Tiempo y RMSE): Número de Vectores de Training inicial, final y número de curvas a realizar.
- Vector de Selección.
- Número de ELMs a realizar para calcular la media del tiempo y RMSE.

2. **En función del número de parámetros.**

Esta opción realiza la ELM de un VS de '1s' contiguos de diferente longitud en función del número de Neuronas Ocultas. En el eje X tenemos situado el número de neuronas y dibuja un curva cada número de vectores de training que deba realizar.
La información que podemos obtener con este tipo de estudio es:

- La relación entre el tiempo y el número de Neuronas Ocultas utilizado.
- La relación entre el tiempo y el número de parámetros utilizado.

La información que tenemos que especificar es:

- Para el eje X, Neuronas: Número de Neuronas inicial, final y espaciado entre ellos.
- Para el eje Y (Tiempo y RMSE): Número de parámetros inicial, final y número de curvas a realizar.
- Número de ELMs a realizar para calcular la media del tiempo y RMSE.

3. **En función de la función de transferencia.**

Esta opción realiza la ELM para un VS dado en función del número de Neuronas Ocultas y el número las funciones de transferencia elegidas. En el eje X tenemos situado el número de neuronas y dibuja un curva por cada número función de transferencia que deba realizar.

La información que tenemos que especificar es:

- Para el eje X, Neuronas: Número de Neuronas inicial, final y espaciado entre ellos.
- Para el eje Y (Tiempo y RMSE): Función de transferencia Training inicial, final y número de curvas a realizar.
- Vector de Selección.
- Número de ELMs a realizar para calcular la media del tiempo y RMSE.

Modo de empleo:

```
./ELM ./ELM_config_file -Graph ./Graph_config_file
```

8.9.1 Fichero de configuración

El fichero de configuración de esta operación nos permite elegir las gráficas a utilizar y las características de las mismas. Este fichero es leído por la función get_Graph_params() y guarda las características en la estructura Graph_data para ser luego utilizadas por la función Graph_op(). En el fichero de configuración indicamos con un "yes" la realización de un tipo de gráfica y los parámetros a indicar para cada una son los expuestos en el apartado anterior.

8.9.2 Ejemplo

El siguiente fichero de configuración ordena la realización de los 2 primeros estudios:

```
Time_ERMS_Ntrain        yes
X-axis_Neurons: Init    1       End     100     StepSize        2
Y-axis_Ntrain:  Init    2000    End     6000    Num_curves      4
Param_array     00001111111111
N_average       10

Time_ERMS_Nparam        yes
X-axis_Neurons: Init    1       End     100     StepSize        2
Y-axis_Nparam:  Init    2       End     14      Num_curves      5
N_average       10

Time_ERMS_ActFunc       no
X-axis_Neurons: Init    200     End     300     StepSize        5
Y-axis_Nparam:  Init    0       End     3       Num_curves      4
Param_array     10000110111011
N_average       5
```

Fig. 48 Archivo de configuración de la operación Graph.

Las gráficas que generadas utilizando este archivo de configuración son:

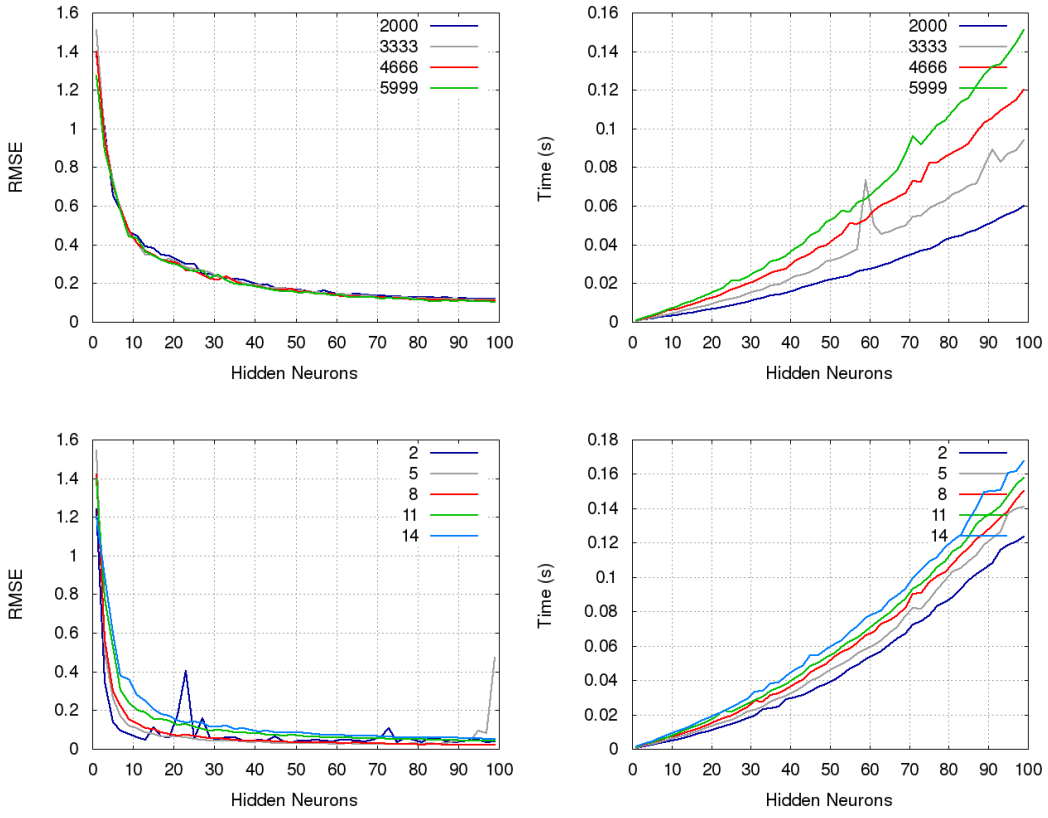


Fig. 49 Resultados operación Graph

8.10ES

Esta opción nos permite realizar una Búsqueda Exhaustiva (Exahustive Search) dentro del espacio de soluciones especificado (parámetros sobre los que realizar la búsqueda). Dentro del espacio de soluciones podemos indicar:

- Bits puestos siempre a 1.
- Bits puestos siempre a 0.
- Bits sobre los que realizar la búsqueda.
    - Número mínimo de bits.
    - Número máximo de bits.

Los bits puestos siempre a 0 son aquellos que no son especificados siempre a 1 ni de búsqueda. El número de bits de búsqueda está limitado a 32 bits → $4 \cdot 10^9$ ELM, aunque esto es fácilmente ampliable.

Al terminar el programa se generarán los siguientes datos:

- La regresión de las ELM de las mejores soluciones en la carpeta ./Results/RMSE.
- Archivo con la gráfica de los valores de RMSE obtenidos en la búsqueda. Se puede encontrar en la carpeta./Results/Graph. Solo si ha indicado en el archivo de configuración.

Modo de empleo:

./ELM ./ELM_config_file -ES ./ES_config_file

8.10.1Fichero de configuración

El fichero de configuración de esta operación nos permite elegir las diferentes variantes de la ES, sus características son leídas por la función get_ES_params() y guardadas en una estructura de tipo ES_data para ser luego utilizadas por la función ES_op().
Los datos a especificar son:

- **Número máximo de 1s**: Dentro del espacio de soluciones posibles dado el Vector de Búsqueda, solo aceptaremos aquellas con un número de 1s menor o igual a este número. Si se pone este valor a -1 no tendrá efecto.
- **Número mínimo de 1s**: Dentro del espacio de soluciones posibles dado el Vector de Búsqueda, solo aceptaremos aquellas con un número de 1s menor o igual a este número. Si se pone este valor a -1 no tendrá efecto.
- **Número de soluciones TOP**
  Cuando acabe la búsqueda exhaustiva, se ordenarán las soluciones de mejor a peor y se mostrará un número de mejores soluciones. Este es ese número.

- **Vector de Búsqueda:** Vector de Selección sobre el que se va a realizar la búsqueda exhaustiva, su número de 1s debe ser igual o menor a 32.
- **Vector Fijo:** Vector de selección cuyos 1s van a siempre fijos. Si alguno de los bits coincide con el del Vector de Búsqueda, se eliminan del Vector de Búsqueda.

- **Representación del SS**:
  Una vez finalizada la búsqueda exhaustiva se puede indicar al programa realice una gráfica con los valores RMSE calculados. Para ello se estribe la palabra “yes” en esta opción.

8.10.2 Ejemplo

El fichero de configuración utilizado tiene la forma:

```
N_1s_min                -1
N_1s_max                -1
N_top_solutions         20

Chosen_params           00001111111100
Fixed_params            11110000000000

Show_State_Space        yes
```

El VS utilizado tiene una componente fija VS_{fijo}: 11110000000000 una componente variable VS_{var}: 00001111111000. No hay número máximo ni mínimo de parámetros y se generará una gráfica con los resultados encontrados.

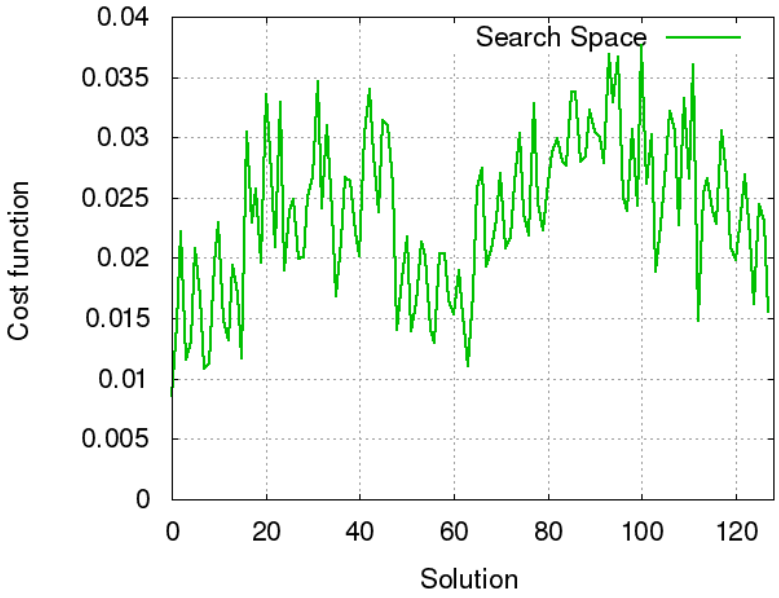


Fig. 50 Search Space de la operación ES.

Los mejores VS ordenamos de mejor a peor son:

00110011000000	RMSE: 0.006109
01110000000000	RMSE: 0.006156
01100000100000	RMSE: 0.006171
01010011000000	RMSE: 0.006264
01110000010000	RMSE: 0.006561

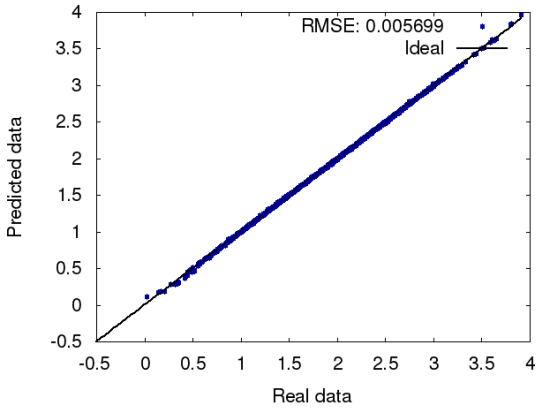


Fig. 51 Curva de regresión de la mejor predicción.

8.11SA

Esta opción realiza una búsqueda en el espacio de soluciones del tipo Temple Simulado (Simulated Anneling). Para aprovechar el sistema de hilos, de llevan a cabo varios SA en paralelo independientes sobre diferentes soluciones del problema. Generamos un conjunto inicial de lingotes (Vectores de Selección) que iremos mutando (Invirtiendo bits) y enfriando, cambiando así su energía (RMSE). Solo aceptaremos el nuevo lingote mutado si se da uno de estos 2 casos:

- Si es mejor que el anterior.
- Si es peor que el anterior, solo se acepta si la temperatura es lo suficientemente alta.

A medida que decrezca la temperatura, habrá menos posibilidades de mutar hacia lingotes peores y estos convergerán al mínimo local de su dominio.

Las ventajas de tener varios lingotes independientes en paralelo son:

- Ahorro de tiempo por carga de datos, normalización…
- Ahorro de tiempo debido a la utilización del sistema de hilos.
- Podemos establecer como condición de parada que la media de los lingotes no varíe significativamente.

Todos los lingotes comparten la misma temperatura y constante de decremento. Por cada GA en paralelo, se guardan 2 lingotes:

- Lingote actual.
- Mejor lingote hasta el momento para esa GA.

Como condiciones de parada podemos establecer:

- Número máximo de enfriamientos totales.
- Número máximo en enfriamientos sin cambio en los lingotes.

La generación inicial de lingotes puede ser de 2 tipos:

- Aleatoria.
- Especificada en al archivo de configuración.

Al terminar el programa se generarán los siguientes datos:

- La regresión de cada uno de los mejores lingotes en la carpeta ./Results/RMSE.
- Archivo con los datos de la SA que incluye la fecha, los parámetros del algoritmo y la generación final de mejores lingotes. Se halla en la carpeta ./Results/SA
- Gráfica “SA.png” de la evolución de los lingotes actuales y mejores lingotes. Se graba la energía media de los lingotes y mejores lingotes de cada generación.

Modo de empleo:

./ELM ./ELM_config_file -SA ./SA_config_file

8.11.1 Fichero de configuración

El fichero de configuración de esta operación nos permite elegir las diferentes variantes de la SA, sus características son leídas por la función get_SA_params() y guardadas en una estructura de tipo SA_data para ser luego utilizadas por la función SA_op().

Los parámetros que podemos configurar son:

- **Número de lingotes**: Número de lingotes independientes sobre los que se realizará la SA.
- **Número de enfriamientos:** Número máximo de enfriamientos a realizar.
- **Número máximo de mutaciones por enfriamiento.**
- **Temperatura inicial**: Es la temperatura inicial del SA que comparten todos los lingotes. Esta temperatura puede ser establecida de 2 formas:
  - Manual: Indicando un número positivo
  - Automática: Indicando un numero negativo. En este caso el programa realizara un viaje inicial por el espacio de búsqueda saltando de una solución a otra, guardando los cambios de energía. Establecerá como temperatura inicial, la media de estos desplazamientos, este valor suele ser más bajo que el normalmente deseado pero se puede utilizar como referencia.
- **Lingotes iniciales**:
  - Aleatorios: Si no indicamos ningún lingote inicial, escribiendo un 0 en el número de lingotes iniciales.
  - Específicos: Si indicamos 1 o más lingotes iniciales. Estos lingotes se repartirán a partes iguales entre los lingotes del algoritmo.

Según el valor que le demos a estos parámetros podremos darle un uso u otro al programa.

- Si no tenemos ninguna solución dada del problema: Corremos el algoritmo con muchos lingotes aleatorios y un número de mutaciones alto para buscar por todo el espacio de soluciones.
- Si tenemos un conjunto de soluciones y queremos intentar mejorarla, las establecemos como solución inicial y establecemos una temperatura baja y numero de mutaciones bajo para solo buscar en los alrededores cercanos de la solución.

8.11.2 Ejemplo

A continuación se muestran un par de archivos de configuración:

- El primero tiene población inicial aleatoria y temperatura automática.
- El segundo tiene población especificada y temperatura manual.

```
Number_of_ingots        100
Number_of_freezings     900
Number_of_Max_Mutations 15

Initial_Temperature     -1
Decreasing_konstant     0.995

Initial_Ingot           0
```

```
Number_of_ingots        30
Number_of_freezings     400
Number_of_Max_Mutations 15

Initial_Temperature     0.5
Decreasing_konstant     0.995

Initial_Ingot           3
01010011001010
01110010010000
11010001000010
```

Fig. 52 Archivos de configuración de la operación SA.

Los resultados de los anteriores ficheros de configuración son:

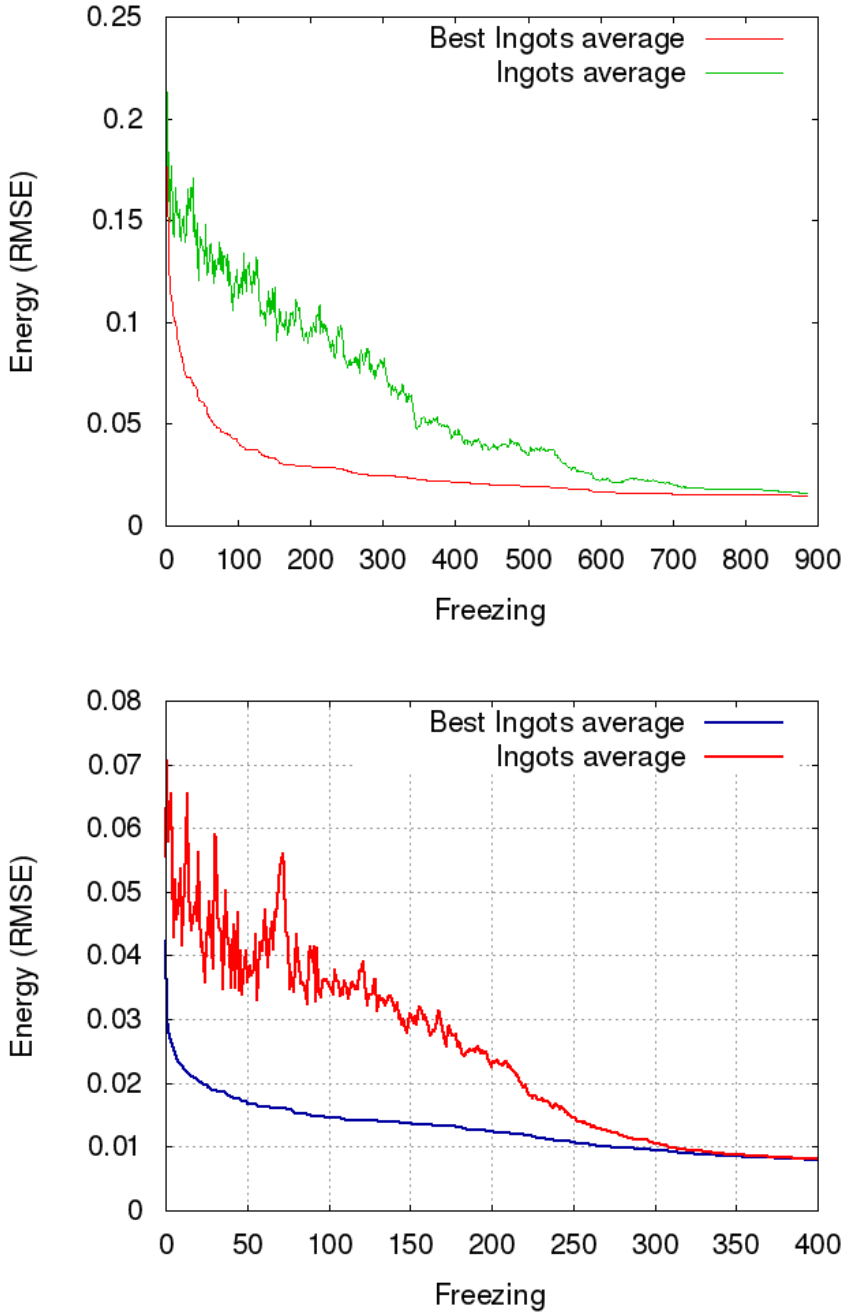


Fig. 53 Resultados operación SA

Se puede observar que al principio del algoritmo, cuando la temperatura es alta, las soluciones pueden mutar a zonas peores del espacio y que conforme esta desciende con el número de iteraciones, solo pueden moverse a soluciones mejores hasta converger con las mejores soluciones encontradas.

8.12GA

Esta opción realiza una búsqueda metaheurística de selección de características mediante un Algoritmo Genético GA. El programa crea una serie de organismos (Vectores de Selección) y los combina y muta durante varias generaciones. Los operadores implementados son:

- Mutación: Inversión de bit.
- Cruce: Sexual (2 padres) y utilizando un 2-point crossover a partir de 2 1-point crossover.

La Evaluación de cada organismo es el RMSE de su Vector de Selección asociado. Dado que el problema del que disponemos es de minimización, para transformarlo en uno de maximización, definimos para cada generación el Fitness del individuo x_i como:

$$f(x_i) = f(x_{max}) - f(x_i)$$

Además normalizamos el Fitness de todo individuo, quedando como:

$$f(x_i) = f(x_i) \ / \ \sum_{j=1}^{N} f(x_j)$$

En cada generación se llevan a cabo las siguientes operaciones en este orden:

- Selección.
- Cruce.
- Mutación.

El algoritmo permite elegir entre 3 algoritmos de selección:

- Roulette Wheel Selection.
- Rank Selection.
- Tournament Selection.

Como condiciones de parada podemos establecer:

- Número máximo de generaciones.
- Biodiversidad de la población.

Se puede llevar a cabo un control de Biodiversidad que consiste en remplazar organismos con demasiadas copias por organismos aleatorios.

La generación inicial de organismos puede ser de 2 tipos:

- Aleatoria.
- Especificada en el archivo de configuración.

Al terminar el programa se generarán los siguientes datos:

- La regresión de cada uno de los organismos elitistas en la carpeta ./Results/RMSE.
- Por cada generación se crea un archivo con los datos de la Población que incluye la fecha, los parámetros del algoritmo y la Población. Se halla en la carpeta ./Results/GA
- Gráfica "GA.png" de la evolución del RMSE de los individuos.

Modo de empleo:

./ELM ./ELM_config_file -GA ./GA_config_file

8.12.1 Fichero de configuración

El fichero de configuración de esta operación nos permite elegir las diferentes variantes del SA, sus características son leídas por la función get_GA_params() y guardadas en una estructura de tipo GA_data para ser luego utilizadas por la función GA_op() .

Los parámetros que podemos configurar son:

- **Número de organismos.**
- **Número de generaciones:** Número máximo de generaciones a realizar.
- **Probabilidad de mutación:** Tanto los hijos y como los padres tienen la posibilidad de mutar.
- **Probabilidad de cruce:** Tan solo un porcentaje de la población se reproduce, el resto pasa tal cual.
- **Elitismo:** Número de mejores organismos que pasan seguro a la siguiente generación.
- **Algoritmo de selección**: Se han realizado 3 algoritmos de selección diferentes a elegir, cada uno de ellos tiene asociado un número, el cual debe indicarse en la configuración:
  - Roulette Wheel Selection 0
  - Rank Selection 1
  - Tournament Selection 2

  Si elegimos este algoritmo, debemos especificar 2 parámetros más:
    - Número mínimo de luchadores
    - Número máximo de luchadores.
- **Población inicial**:
  - Aleatoria: Si no indicamos ningún organismo inicial, escribiendo un 0 en el número de organismos iniciales.
  - Específica: Si indicamos 1 o más organismos iniciales. Si especificamos menos organismos que la población, el resto serán organismos aleatorios.

8.12.2 Ejemplo

El fichero de configuración y resultado del mismo se muestran a continuación:

```
Number_of_organisms      50
Number_of_generations    40

Prob_of_Mutation         0.1
Prob_of_Crossover        0.85
Elitist_Proportion       0.03
Number_max_mutations     5

Selection_Algorithm      2
Number_min_fighters      2
Number_max_fighters      10

Initial_Organisms        0
```

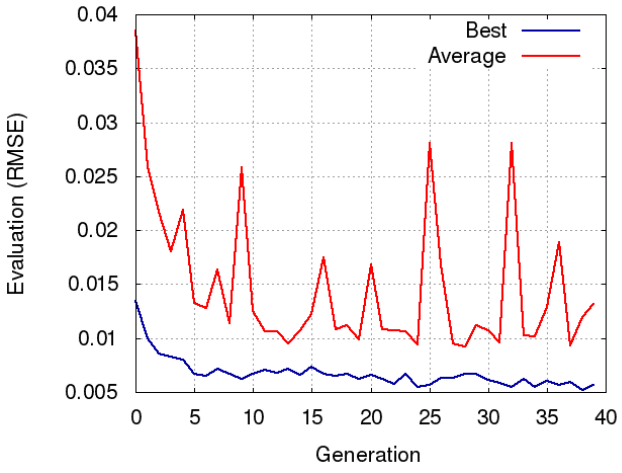


Fig. 54 Archivo de configuración y Resultados de la operación GA

8.13 AIS

Esta opción realiza una búsqueda de selección de características mediante el AIS. Implementa el algoritmo descrito en el capítulo sobre Sistemas Inmunes Artificiales. El algoritmo consiste en generar una población inicial aleatoria de células B. En cada generación:

- Las células B se clonan y mutan generando anticuerpos. La mutación se realiza dentro del dominio de las células B. También se genera cierto número de anticuerpos aleatorios.
- Las células B evolucionan en cada ciclo de la siguiente manera:
    - Si algún anticuerpo es mejor que su célula B madre, la célula B se transforma dicho anticuerpo.
    - Si 2 células B están dentro del mismo dominio, la peor muere.
    - Si una célula B no mejora durante cierto número de ciclos, la célula B desaparece y se convierte en una célula M.
    - En cada ciclo, si caben más células B, se añade un célula B, el mejor de los anticuerpos aleatorios.

Así pues tenemos 3 tipos de poblaciones:

- Células B: Son soluciones del problema que están siendo mejoradas.
- Anticuerpos: Posibles mejoras de las células B.
- Células M: Mejores soluciones.

Como condiciones de parada podemos establecer:

- Número máximo de ciclos.
- Número máximo de células M.

Al terminar el programa se generarán los siguientes datos:

- La regresión de cada una de las células M en la carpeta ./Results/RMSE.
- Archivo con los datos de la AIS que incluye la fecha, los parámetros del algoritmo y las células M. Se halla en la carpeta ./Results/AIS
- Gráfica "AIS.png" de la evolución de las células M.

Modo de empleo:

./ELM ./ELM_config_file -AIS ./AIS_config_file

8.13.1 Fichero de configuración

El fichero de configuración de esta operación nos permite elegir las diferentes variantes de la AIS, sus características son leídas por la función get_AIS_params() y guardadas en una estructura de tipo AIS_data para ser luego utilizadas por la función AIS_op().
Los parámetros que podemos configurar son:

- **Número de ciclos:** Número máximo de ciclos de clonación de las células B.
- **Número inicial de células B:** Número de células B iniciales.
- **Número Máximo de células B:** Número máximo de células B simultaneas.
- **Número de clones de células B:** En cada ciclo, las células B se clonan y mutan dentro de su dominio este número de veces dando lugar a Anticuerpos Ab
- **Número de anticuerpos aleatorios:** Numero de Ab aleatorios que se generaran en cada ciclo para generar nuevas células B, se escoge en cada ciclo el mejor Ab aleatorio.

- **Dominio de las células B:** Es el número máximo de mutaciones que puede sufrir una célula B al clonarse para producir anticuerpos. También es la distancia mínima que debe haber entre 2 células B, si no la hay, la mejor absorbe a la peor y se reinicia su tiempo de vida.
- **Tiempo de vida de las células B**: Es el número de ciclos que durará una célula B sin ser mejorada hasta ser convertida en una célula Máster M.
- **Número de células** M: Número máximo de células M. Cuando lleguemos a tener este número el algoritmo parará.

8.13.2 Ejemplo

El fichero de configuración posee la siguiente estructura y parámetros seleccionados:

```
Number_of_clonning_cycles       50
Init_B_cells                    5
Max_B_cells                     20
Number_of_B_clones              20
Number_of_random_Ab             5

B_domain                        3
B_living_time                   5

Max_Master_cells                100
```

Fig. 55 Archivo de configuración de la operación AIS

Los resultados obtenidos para la ejecución son:

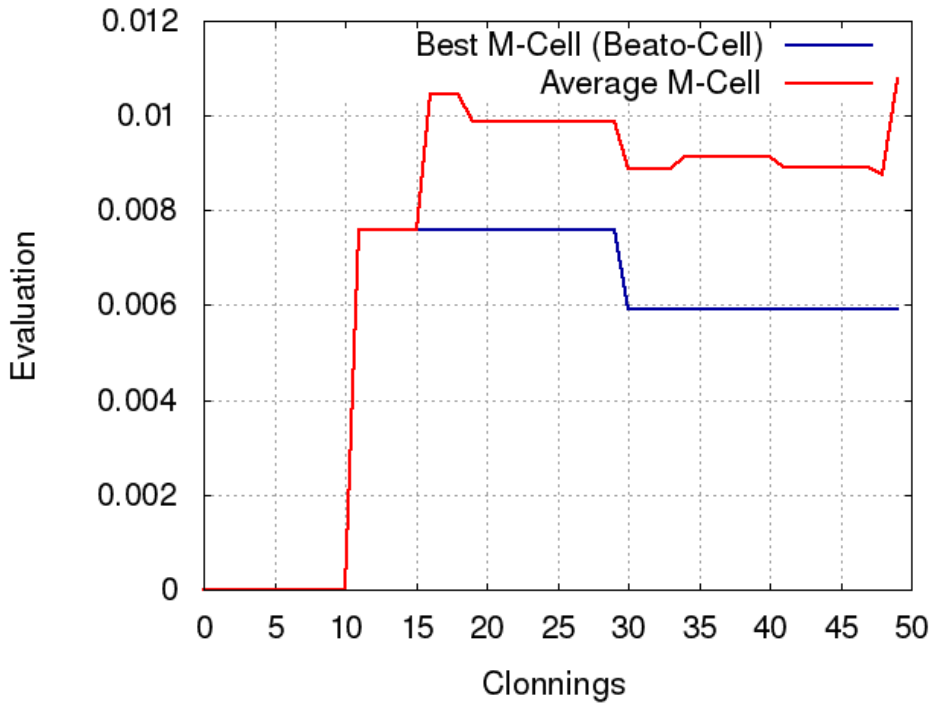


Fig. 56 Resultados operación AIS

8.14 FP

Esta opción nos permite principalmente saber el RMSE que produce la ELM de cada input por separado y la afinidad entre cada par de inputs. Así pues se realizan 4 operaciones principales:

- Realiza la ELM de cada input.
- Realiza la ELM de cada par de inputs.
- Realizar el algoritmo SFS.
- Realizar el algoritmo SBE.

Esta opción puede servir para realizar una búsqueda básica e intuitiva de los mejores parámetros para una posible primera selección y después trabajar sólo con ellos.
Definimos la afinidad entre 2 parámetros como la mejora del RMSE respecto al mejor de los 2 parámetros utilizados. Sean:

$$\mathcal{Aff}(x_i, x_j) = \min\left(RMSE(x_i), RMSE(x_j)\right) - RMSE\ (x_i, x_j)$$

Cuanto mayor sea la afinidad, mejor será la combinación de los parámetros.

Al terminar el programa se generarán los siguientes datos, si es que su operación correspondiente ha sido seleccionada en el fichero de configuración:

- Archivo con los datos de la FP que incluye la fecha, los parámetros de la ELM en carpeta ./Results/FP.
- Gráfica "FP.png" con el ERMS de cada input por separado y un archivo con la RMSE de cada parámetro ordenados en orden creciente.
- Gráficas "FPX.png" con la afinidad de cada parámetro X con el resto y un archivo con la afinidad de estos ordenada en orden creciente.
- Resultados del algoritmo SFS
- Resultados del algoritmo SBE

8.14.1 Fichero de configuración

El fichero de configuración de esta operación nos permite elegir las diferentes variantes de la FP, sus características son leídas por la función get_FP_params() y guardadas en una estructura de tipo FP_data para ser luego utilizadas por la función FP_op(). Los parámetros que podemos configurar son simplemente la realización o no de alguna de las opciones.

8.14.2 Ejemplo

A continuación se muestra un ejemplo del fichero de configuración y la gráfica del RMSE para cada uno de los parámetros por separado.

```
Get_Features_Evaluation          yes
Get_FeaturePairs_Fitness         no
DO_SFS                           no
DO_SBE                           yes
```

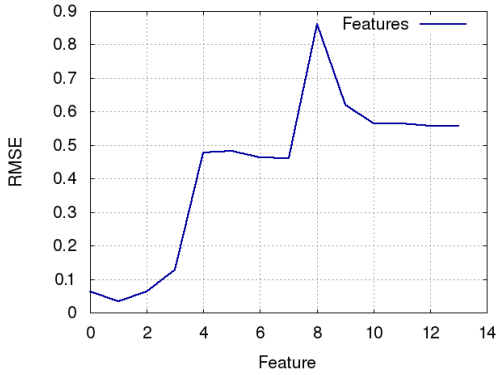


Fig. 57 Archivo de configuración y Resultados operación FP

Capítulo 9
Resultados

9. RESULTADOS

En la página web del National Data Bouy Centre http://www.ndbc.noaa.gov/ se puede encontrar el espectro de las olas marinas $S(f)$, medido por diversas boyas de todo el mundo. Se utilizará este espectro en el cálculo de diversos parámetros del oleaje que utilizaremos para intentar encontrar las relaciones entre la altura efectiva de ola H_s en un punto y las condiciones del oleaje en otros tantos.

Las boyas que se utilizarán en este estudio son 5 boyas pertenecientes al mar caribe:

- 42056
- 42057
- 42058
- 42059
- 42060

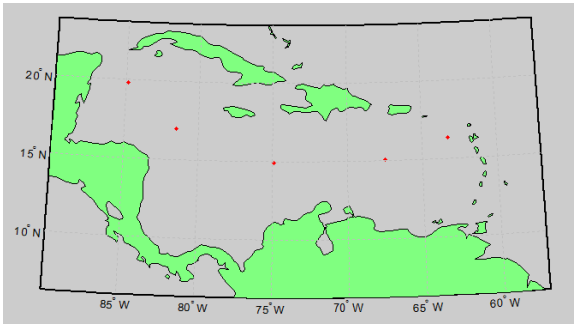


Fig. 58 Mapa de las boyas del caribe.

Disponemos de más de 17000 muestras sincronizadas y válidas para las 5 boyas de los años 2009, 2011, 2012 y 2013

Así pues nuestra tarea es hallar las dependencias entre la altura efectiva de la **boya 42058** y los parámetros del oleaje obtenidos por las otras boyas. Cada boya posee 15 parámetros:

H_s	m_{-1}	m_0	m_1	m_2	T_p	Tm_{10}	Tm_{01}	Tm_{02}	Q_p	nu	k_{01}	g_{01}	k_{02}	g_{02}

En este apartado de mostrarán y comentarán los resultados obtenidos al realizar la selección de características con cada tipo de algoritmo y técnica implementada en el programa desarrollado. Los resultados y gráficas presentados han sido obtenidos íntegramente a partir del programa.

9.1 Formulación del problema

Como se ha venido diciendo a lo largo del documento, este problema se puede expresar como un problema de optimización discreta, más concretamente un problema de selección de características (Feature Selection) que se formula de la siguiente manera:

Dado un conjunto de N características y una variable objetivo T, que podemos relacionar con cualquier subconjunto S de características mediante el algoritmo de aprendizaje $\mathcal{L}(S)$. El objetivo de la FS es encontrar un subconjunto S de N, $S \in N$, lo más reducido posible que consiga el mejor valor de T para el Algoritmo de Aprendizaje dado.

$$Encotrar\ S' \quad tal\ que \quad \min_S \|T - \mathcal{L}(S)\|$$

En nuestro problema:

- La variable T objetivo es la altura efectiva medida por una boya H_s.
- El conjunto de N características son aquellas que utilicemos para obtener H_s.
- El Algoritmo de Aprendizaje es una RNA entrenada con el algoritmo ELM.

Los subconjuntos S de características se representar como un array binario (Vector de Selección VS), donde cada parámetros se codifica como un bit. Si está a '1' incluimos esa característica y si está a '0' no. Para encontrar el mejor subconjunto S, se han implementado diversos algoritmos de búsqueda como pueden ser ES, GA, SA, AIS, SFS y SBE que se utilizarán de forma conjunta.

9.2 Una sola boya

En este apartado se intentará predecir la altura efectiva de ola H_s de la **boya 42058** a partir de los 14 parámetros restantes de la misma. A priori ya se sabe la relación casi determinista entre dicha altura y cada uno de los parámetros por lo que este apartado carece de utilizad práctica a nivel de predicción. Sin embargo, precisamente por este hecho y que el espacio de búsqueda es relativamente pequeño, nos permitirá determinar si los algoritmos desarrollados son correctos, tanto los algoritmos de búsqueda, como el comportamiento de la ELM y la capacidad de predicción de cada parámetro, pues se espera que esta capacidad aumente conforme lo haga su relación directa con la altura efectiva.

Así pues se procederá a llevar a cabo un estudio de las características de la ELM, los parámetros de la boya y los algoritmos de búsqueda implementados. El estudio de la ELM nos ayudará a determinar los valores óptimos de la ELM como el número de neuronas ocultas, el número de VTr y la función de transferencia. Sin embargo estos parámetros y su influencia en el RMSE pueden variar de problema a problema. También se debe tener en cuenta que en las sucesivas ejecuciones y estudios que se llevarán a cabo, los conjuntos de VTr y VTe cambiarán, ya que estos se permutan unos con otros, lo que se traduce a ligeras variaciones del RMSE.

9.2.1 ELM básica.

Para empezar, vamos a realizar la ELM de 2 Vectores de Selección para comprobar que el algoritmo funciona y es capaz de realizar una predicción de la salida. Dado que los primeros 4 parámetros son momentos muy relacionados con la altura efectiva H_s, se espera que estos posean una capacidad de predicción sobresaliente, es por ello que los 2 VS son:

- VS con las 14 características 11111111111111
- VS sólo con 10 últimas características 00001111111111

Los demás parámetros de la ELM elegidos son 15000 VTr, 2000 VTe, 30 neuronas ocultas y función de transferencia sigmoide.

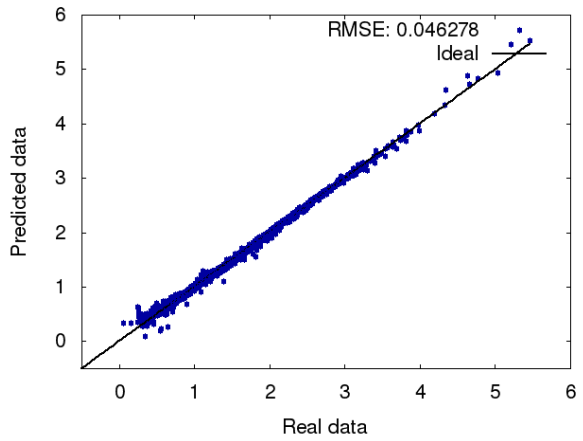


Fig. 59 Curva de regresión de la ELM.

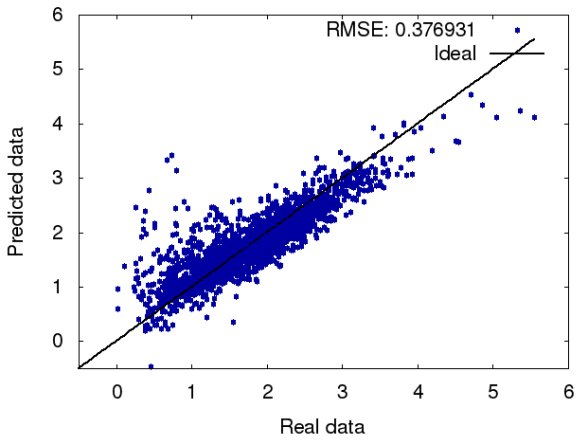


Fig. 60 Curva de regresión de la ELM.

Como se puede apreciar, la primera gráfica muestra un RMSE muy bajo, haciendo una predicción casi perfecta debido a la relación directa entre las entradas y la salida. En la segunda gráfica no utilizamos dichos parámetros por lo que el error es mayor, obteniendo una dispersión más pronunciada que no deja de ser una gran predicción. Se puede apreciar también que la altura de las olas suele rondar entre 1 y 3 metros de altura.

9.2.2 Estudio de la ELM

El algoritmo de la ELM posee diversos parámetros que debemos elegir y que influyen en la calidad de la solución y el tiempo computacional de ejecución.

- Número de Neuronas Ocultas
- Número de Vectores de Training
- Función de transferencia.

El valor óptimo de estos parámetros varía de problema a problema ya que depende de la estructura del mismo y del conjunto de VTr y VTe. Es por ello que no se podrán establecer unos valores únicos, solo se deducirán una serie de comportamientos que tomaremos como referencia. Primeramente vamos a realizar un estudio del tiempo y RMSE del problema en función del número de Neuronas y el Número de Vectores de Training. Se utiliza el VS 11111111111111 y la función de transferencia sigmoide. Además cada ELM se realiza 10 veces y se obtiene la media.

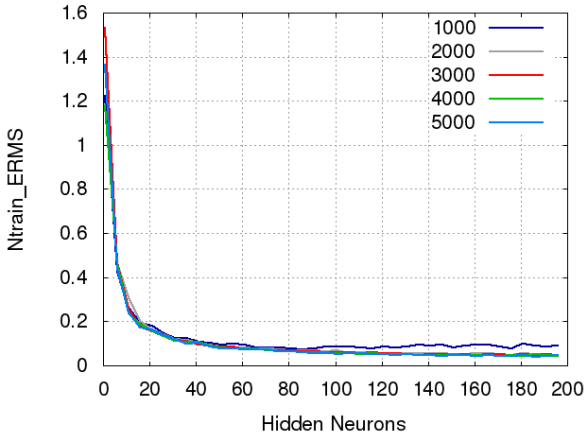


Fig. 62 RMSE de la ELM en función del número de VTr.

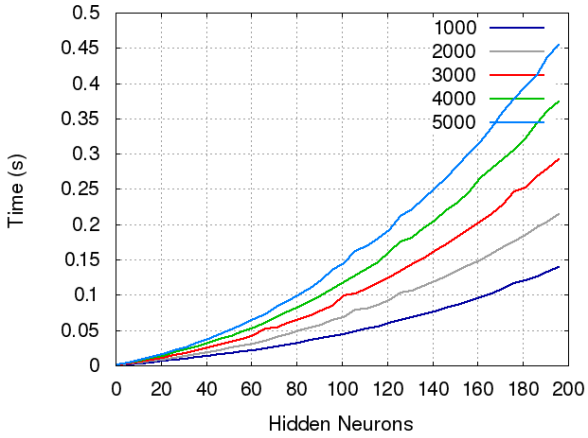


Fig. 61 Tiempo de la ELM en función del número de VTr.

Se pueden llevar a cabo las siguientes observaciones sobre las gráficas:

- El tiempo de ejecución crece $O(n^2)$ con el número de neuronas.
- El tiempo de ejecución crece $O(n)$ con el número de VTr.
- El RMSE decrece exponencialmente con el número de neuronas, estabilizándose en el rango 20-60 neuronas, partir de este valor, el RMSE se reduce muy lentamente.

También se puede apreciar que con 1000 VTr, cuando se aumenta el número de Neuronas Ocultas, llega un momento en el cual el error empieza a crecer. Para ver este efecto más en profundidad se va a representar la misma gráfica con un número menor de VTr.

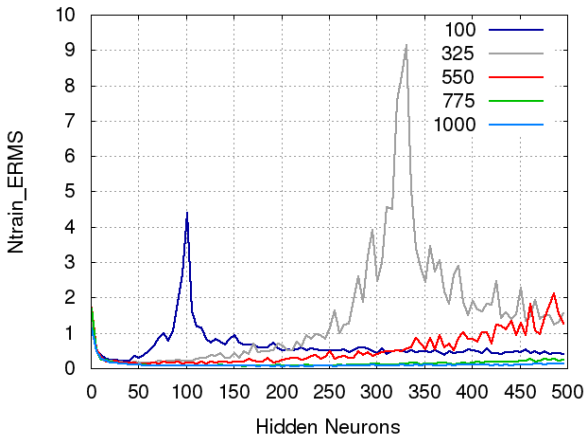


Fig. 63 RMSE de la ELM en función del número de VTr.

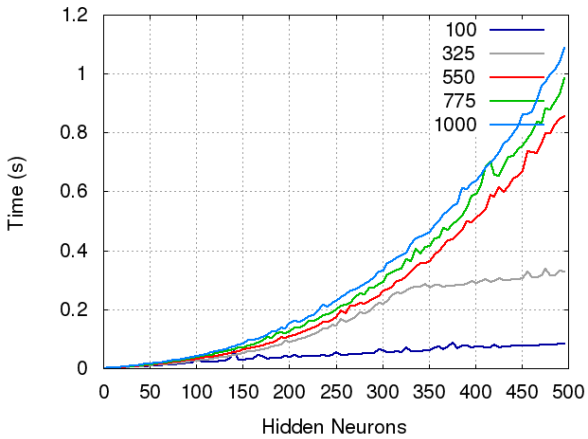


Fig. 64 Tiempo de la ELM en función del número de VTr.

Se observa que cuando el número de neuronas y el de VTr se igualan, el error llega a su máximo por lo que tenemos que evitar dicho punto. A este efecto se le llamará la **resonancia de la ELM**. También se puede apreciar que el tiempo computacional se satura en dicho punto dado que el algoritmo implementado para realizar la matriz inversa de Penrouse-Moore utiliza una matriz cuadrada de dimensiones min(VTr, neuronas).

Este comportamiento parece indicar que conforme se aumenta el número de neuronas, el RMSE disminuye hasta llegar a un número cercano al número de VTr, momento en el que empieza a incrementarse. Para un número pequeño de neuronas, 50, este efecto es insignificante y a priori da igual utilizar 1000 VTr que 5000 VTr siempre que estos sean significativos.

Llegados a este punto nos podemos hacer la siguiente pregunta ¿Si el número de VTr es lo suficientemente grande, el RMSE bajará ininterrumpidamente conforme aumente el número de neuronas? O la pregunta ¿Si utilizamos un número de neuronas lo suficientemente grande, 500, cuánta es la mejora que ejerce el número de neuronas? Para responder a estas preguntas, se va a realizar el mismo estudio pero utilizando mayores valores de VTr y de neuronas. Además esta vez se utilizará el VS 00001111111111 para comprobar que estas propiedades se mantienen para diferentes VS.

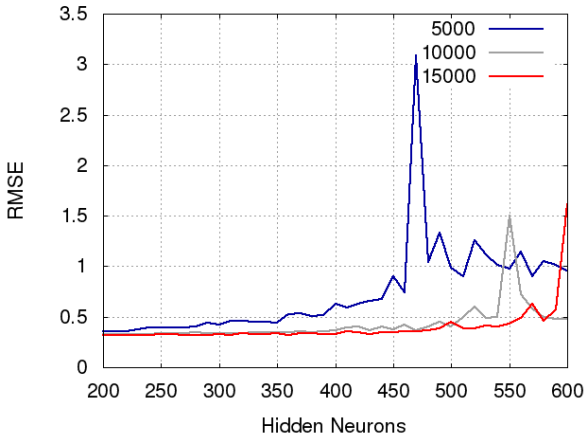


Fig. 66 RMSE de la ELM en función del número de VTr.

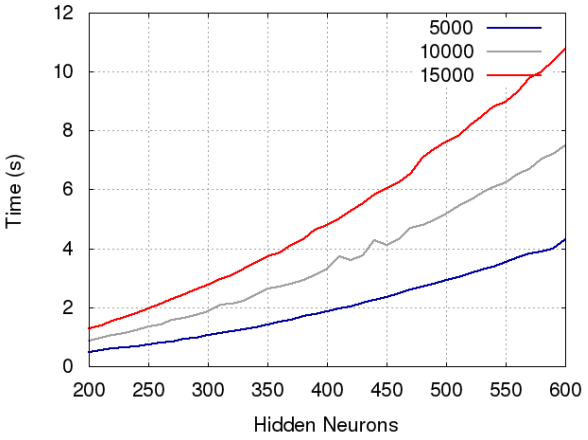


Fig. 65 Tiempo de la ELM en función del número de VTr.

De la gráfica se puede observar que existe un número de neuronas óptimo para el problema a partir del cual, si lo aumentamos, no mejora el RMSE, sin importar el número de VTr, aunque este número tiene que ser lo suficientemente grande como para que no produzca el efecto de resonancia. Por supuesto el punto a partir del cual el RMSE empieza a aumentar depende de los parámetros de entrada, para este problema, son 150 neuronas, aunque a partir de las 50 el RMSE no mejora significativamente.

- **Número de parámetros de la ELM:**

También resulta útil conocer la relación del tiempo de ejecución y el número de parámetros con el que alimentemos la ELM. La gráfica de la derecha muestra esta información para diferente número de parámetros de entrada. Como se puede deducir, el tiempo de ejecución es proporcional al número de parámetros, es decir, es $O(n)$.

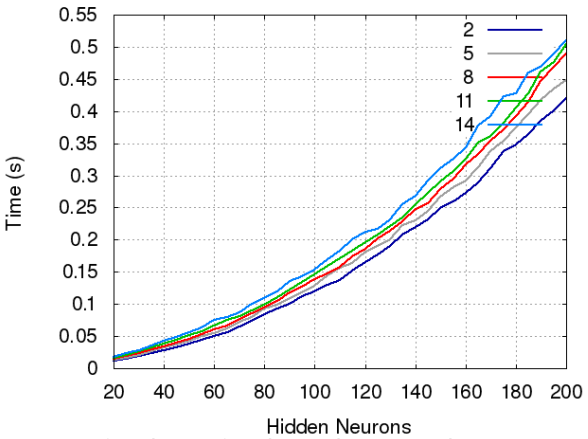


Fig. 67 Tiempo de la ELM en función de la función del número de parámetros

- **Función de transferencia de la ELM:**

La otra característica de la ELM que queda por determinar es la función de transferencia a utilizar en las Neuronas Ocultas. Esta función en principio afecta tanto al valor de la RMSE como al tiempo de computación. Las funciones de transferencia que se probarán son:

- 0 Sigmoide 1 Seno
- 2 Coseno 3 Tangente hiperbólica

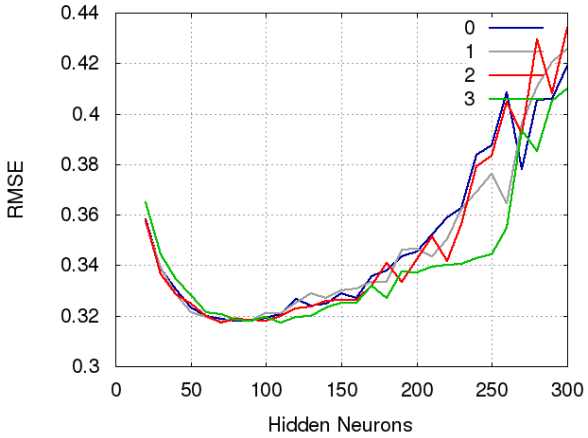


Fig. 69 RMSE de la ELM en función de la función de transferencia.

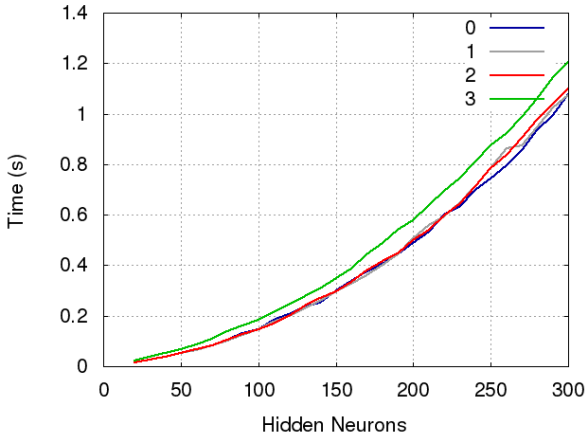


Fig. 68 Tiempo de la ELM en función de la función de transferencia.

Como se puede observar, el RMSE de las 4 funciones es prácticamente idéntico, al igual que sus tiempos de ejecución. La subida del RMSE a partir de las 100 neuronas se debe al efecto resonancia ya que sólo se han utilizado 3000 VTr. El RMSE no mejora añadiendo más VTr.

- **Aleatoriedad de la ELM:**

Dado que los pesos y bias de las neuronas ocultas se eligen aleatoriamente, existe una componente no determinista en el algoritmo de la ELM que hace que el resultado (RMSE) varíe de ejecución en ejecución. Para ver el grado y distribución de esta componente se ha optado por realizar un gran número de ELM del mismo VS y representar por rangos los valores de RMSE obtenidos.

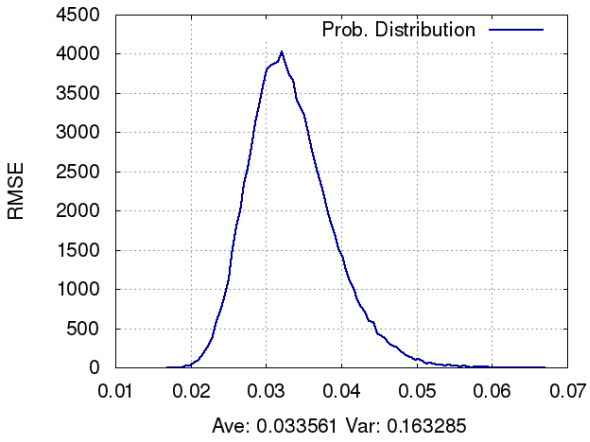


Fig. 71 Función de Distribución de la ELM.

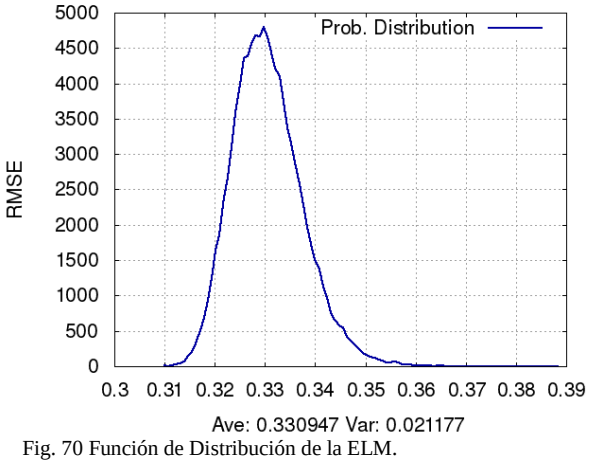


Fig. 70 Función de Distribución de la ELM.

Como podemos ver en las gráficas, el RMSE tiene una distribución estadística de tipo gaussiana o gamma con una varianza bastante alta. Otras pruebas realizadas demuestran que esta distribución, (su media y varianza), depende de muchos factores como la naturaleza de los parámetros de entrada, el número de neuronas o la función de transferencia.

Con toda información se deben elegir los parámetros de la ELM necesarios para producir un RMSE representativo en un tiempo aceptable. La solución de compromiso elegida es:

- 30 neuronas 4000 VTr 2000 VTe Función de transferencia sigmoide

9.2.3 Search Space del problema

Como se viene diciendo, el Search Space del problema, es el conjunto total de solución del mismo. En este problema, el SS es un espacio 14-dimensional binario que se podría representar mediante un hypercubo, lo cual no ofrece una visión clara del mismo.

En vez de eso, se va a representar a continuación el espacio de soluciones generado cuando tratamos el VS como un número binario codificado en gray y vamos incrementando en 1 su valor. La gráfica de la derecha es el conjunto de soluciones vecinas unas de otras cuando el operador de vecindad es sumar o restar 1 al VS codificado en gray. Se ha elegido esta codificación ya que 2 números separados por la unidad sólo se distancian en un 1 bit, lo que da más continuidad.

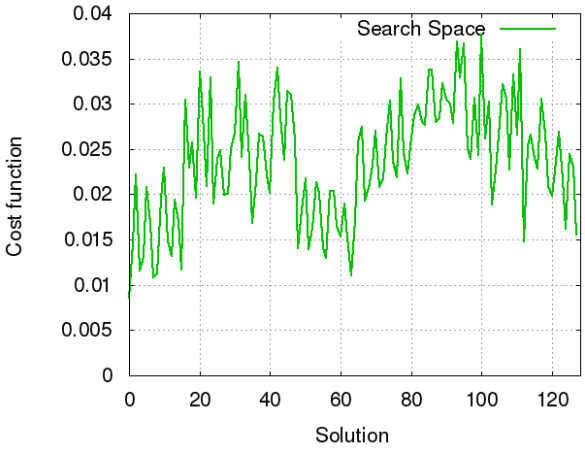


Fig. 72 Search Space del problema.

El VS utilizado tiene una componente fija VS_{fijo}: 11110000000000 para tener siempre los mejores parámetros y la componente variable VS_{var}: 00001111111000 codificada en gray.
La imagen muestra un Search Space abrupto con diversos óptimos locales para este operador de variación por lo que no se puede utilizar ningún **algoritmo de búsqueda basado en el gradiente**.

9.2.4 ES (Búsqueda Exahustiva)

Una Búsqueda Exahustiva consiste en probar todas las soluciones del problema para así poder encontrar la solución óptima. En un problema normal, el tamaño del Search Space es demasiado grande como para realizar este tipo de búsqueda pero sí que se podría realizar una ES de un subespacio del problema (ej. Para mejorar una solución ya encontrada).

Dado que en este apartado solo se dispone de 14 parámetros de entrada, se puede realizar la ES de todas las posibilidades para obtener así las mejores VS. Esto servirá de referencia para comprobar el correcto funcionamiento de los algoritmos de búsqueda desarrollados que serán probados en los próximos apartados.

Los mejores VS ordenamos de mejor a peor son:

00110011000000	RMSE: 0.006109
01110000000000	RMSE: 0.006156
01100000100000	RMSE: 0.006171
01010011000000	RMSE: 0.006264
01110000010000	RMSE: 0.006561
11010000000000	RMSE: 0.006616
01100000010000	RMSE: 0.006631
01010001000000	RMSE: 0.006653

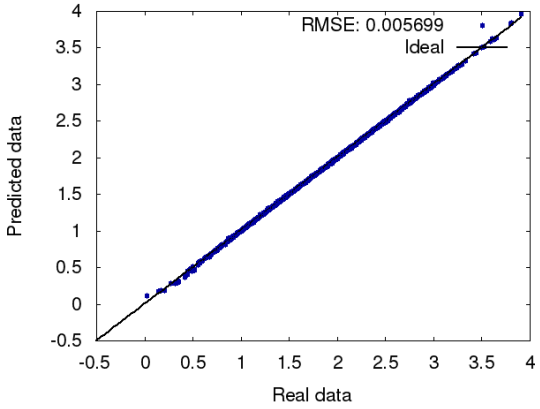


Fig. 73 Regresión de la mejor solución.

Como era de esperar la predicción es casi perfecta por la similitud entre los primeros 4 parámetros de entrada y la altura efectiva que intentamos predecir. Se puede destacar la complementariedad de

9.2.5 FP (Propiedades de las características)

Como primera aproximación del problema se va a realizar un estudio inicial de las 14 características una a una y por parejas para ver la relación de cada una con la salida y la complementariedad entre ellas. Estos datos pueden utilizarse para realizar un primer filtrado de características o para alimentar otros algoritmos de selección proporcionando información sobre la estructura del problema.

La gráfica de la derecha muestra el RMSE obtenido al realizar la ELM con cada uno de los parámetros individualmente. La gráfica nos muestra que los primeros 4 parámetros (0 – 3) son los que mayor capacidad de predicción individual poseen. Esto coincide con nuestras expectativas iniciales ya que los primeros 4 parámetros son los momentos del espectro, que están muy relacionados con la altura efectiva de ola.

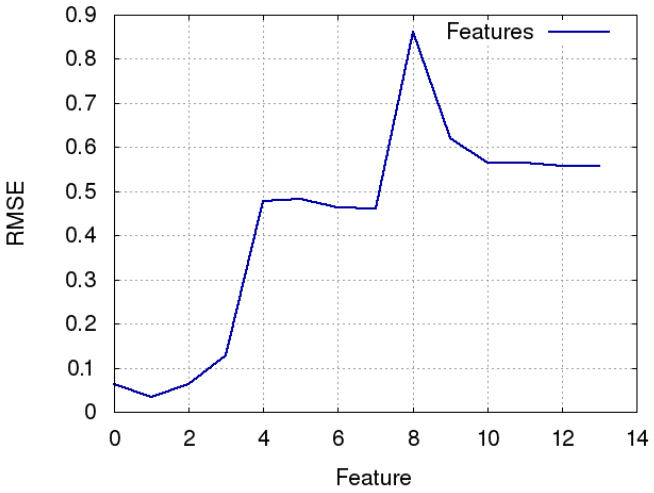


Fig. 74 RMSE de cada uno de los parámetros.

A partir de la gráfica se deduce que con que la solución contenga uno de los 3 primero parámetros, el RMSE será inferior a 0.1 lo cual es de por sí muy bajo.

A continuación de mostrará, para los parámetros 1 y 11, la gráfica de la afinidad entre dichos parámetros y el resto. La afinidad es una medida de lo bien o mal que se complementan las variables:

$$\mathcal{Aff}(x_i, x_j) = \min\left(RMSE(x_i), RMSE(x_j)\right) - RMSE\,(x_i, x_j)$$

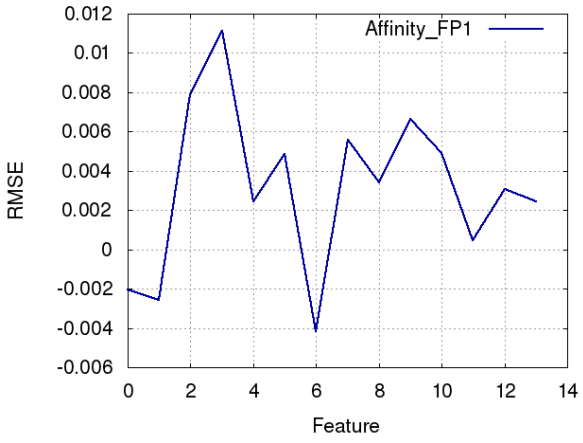


Fig. 76 Afinidad del parámetro 1.

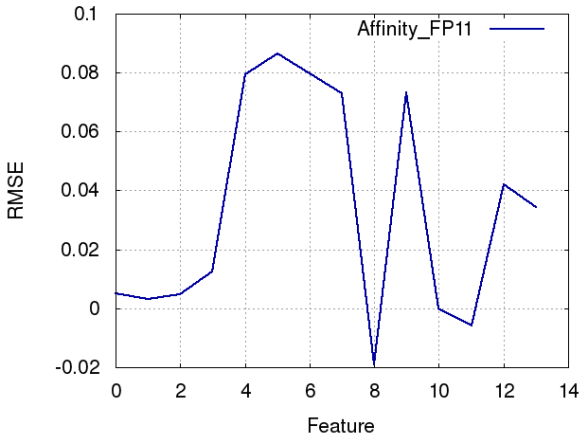


Fig. 75 Afinidad del parámetro 11.

El primer parámetro ofrece poco complementariedad ya que su valor ya es bastante bueno y el resto de parámetros no aporta mejor información, mientras que el RMSE del parámetro 11 se incrementa notablemente cuando se utiliza junto con otros parámetros.

Cabe destacar que aleatoriedad de la ELM se acentúa cuando se utilizan poco parámetros, lo que hace que estas gráficas varíen bastante de una implementación a otra. Un mayor estudio muestra que la afinidad entre los parámetros se incrementa conforme aumentamos el número de neuronas ocultas.

9.2.6 SA (Temple Simulado)

El primer del algoritmo de búsqueda metaheurístico para encontrar el mejor subconjunto que se va a utilizar es el Temple Simulado. A continuación ejecutaremos la opción SA del programa, utilizando una generación de lingotes iniciales aleatoria. El algoritmo ha sido lanzado con los siguientes parámetros:

- Número de lingotes en paralelo 50
- Número de enfriamientos 400
- Máximo número de mutaciones 5
- Temperatura inicial 0.6
- Constantes de enfriamiento 0.975

La figura siguiente muestra la ejecución del algoritmo, mostrando para cada iteración del mismo dos curvas:

- RMSE medio de los 50 de los mejores lingotes.
- RMSE medio de los 50 lingotes actuales.

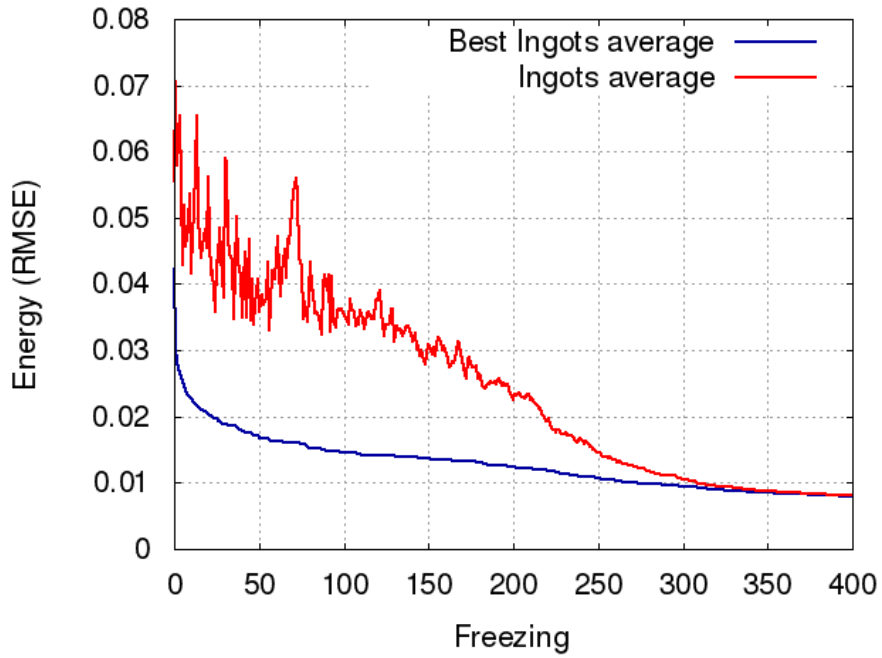


Fig. 77 Resultados algoritmo SA

Se puede observar que al principio del algoritmo, cuando la temperatura es alta, la soluciones pueden mutar a zonas peores del espacio y que conforme esta desciende con el número de iteraciones, solo pueden moverse a soluciones mejores hasta converger con las mejores soluciones encontradas.

De entre los mejores VS que ha encontrado el algoritmo se tiene:

01100000010000	01110000010000
01010000010000	00110000010000
01010000010000	00110100000000
11010000000000	01010000000000

Los parámetros 1-4 y el número 10 son los más utilizados y coinciden en gran medida con los mejores VS encontrados en la búsqueda exhaustiva. Se puede decir que el algoritmo implementado se comporta correctamente, si bien la constante de enfriamiento es demasiado grande, pero dado que el SS es pequeño, esta constante hace la gráfica más clara.

9.2.7 GA (Algoritmo Genético)

Ahora se utilizará un Algoritmo Genético para comprobar que encuentra los mejores Vectores se Selección. A continuación ejecutaremos la opción GA del programa, utilizando una generación de organismos iniciales aleatoria y el algoritmo de selección por torneo. Los parámetros del algoritmo utilizado son los siguientes:

- Número de organismos 50
- Número de generaciones 40
- Probabilidad de mutación 20%
- Probabilidad de cruce 85%
- Número mínimo de luchadores 2
- Número máximo de luchadores 6

La figura siguiente muestra la ejecución del algoritmo mostrando para cada iteración del mismo, las curvas correspondientes al RMSE medio de la población y el RMSE del mejor individuo respectivamente.

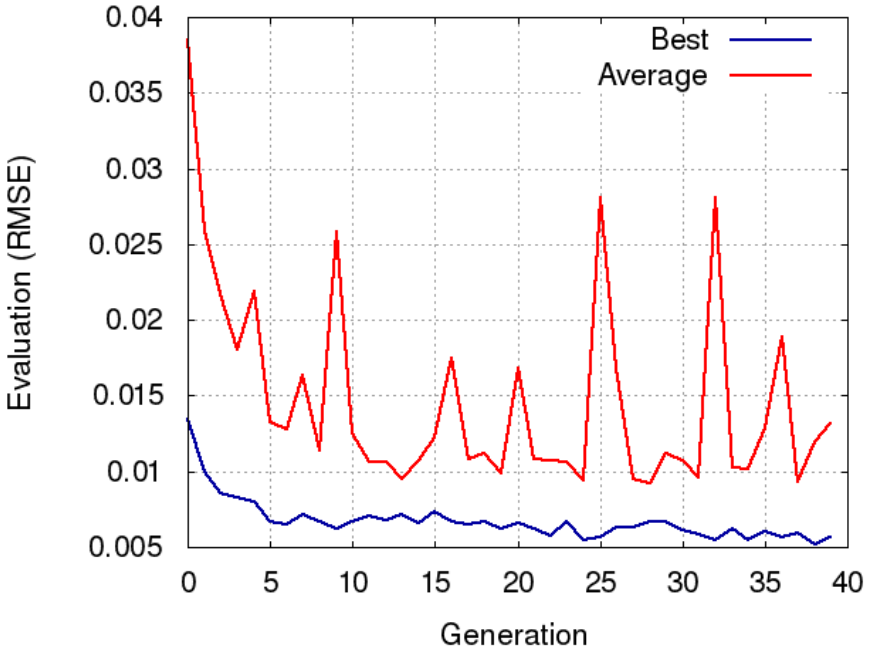


Fig. 78 Resultados algoritmo GA

Se puede observar que inicialmente el RMSE de la población es siempre mayor que el del mejor organismo lo que indica que este no se hace con el control de la población. Gracias a la mutación el RMSE medio de la población tiene altibajos, buscando en nuevas zonas del espacio que consiguen disminuir el RMSE continuamente.

De entre los mejores VS que el algoritmo posee en el Hall of Fame se tiene:

01010000010010	10110000010000
11110000000000	11010010000000
00110010000000	10110000000000
11110000000000	01110000000000

Los parámetros 1-4 y son los más utilizados y coinciden en gran medida con los mejores VS encontrados en la búsqueda exhaustiva. Se puede decir que el algoritmo implementado se comporta correctamente.

9.2.8 AIS (Sistema Inmune Artificial)

Por último se pondrá en práctica el algoritmo de tipo AIS desarrollado por el autor de este documento. Para ello se ejecutara la opción AIS del programa, utilizando una generación de células iniciales aleatoria. Los parámetros del algoritmo utilizado son los siguientes:

- Número de ciclos 50
- Número máximo de células B 10
- Número de clonaciones 10
- Numero de anticuerpos aleatorios 5
- Dominio de las células B 2

La figura siguiente muestra la ejecución del algoritmo mostrando para cada iteración del mismo, las curvas correspondientes al RMSE medio de la población de células M y el RMSE la mejor célula M. Notar que dado que durante los primeros ciclos, dado que no se han generado aún células M, la gráfica no muestra valores.

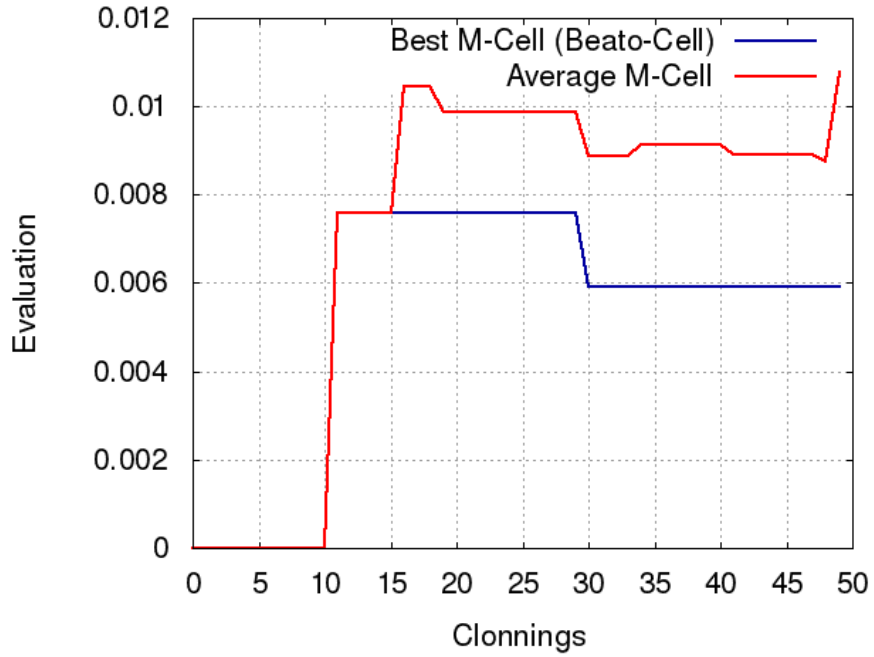


Fig. 79 Resultados algoritmo AIS

Se puede observar que el RMSE de la población de células M es bastante bajo desde el principio ya que estas son soluciones que han pasado por un proceso de búsqueda local de hill-climbing.

De entre los mejores VS que el algoritmo posee en su población de células M se tiene:

01010001000000	00110000011000
00110010000000	01110000000000
11110001010000	00110000010000
01010001000000	10110100010000

Los parámetros 1-4 y son los más utilizados y coinciden en gran medida con los mejores VS encontrados en la búsqueda exhaustiva. Se puede decir que el algoritmo implementado se comporta correctamente.

9.3 Múltiples Boyas

Una vez se ha comprobado que los algoritmos implementados funcionan correctamente y se conocen sus propiedades se va a proceder a resolver el principal problema de este documento. Este consiste en intentar predecir la altura efectiva de la boya 42058 a partir de los 15 parámetros de las otras 4 boyas. En total se posee un conjunto de 60 parámetros con la estructura:

$$\underbrace{111111111111111}_{42056}\ \underbrace{111111111111111}_{42057}\ \underbrace{111111111111111}_{42059}\ \underbrace{111111111111111}_{42060}$$

Primeramente se volverá a realizar un breve estudio de las propiedades de la ELM para este conjunto de parámetros y así poder decidir el número de neuronas ocultas y VTr.

En este problema se puede observar una disminución notable del RMSE con el número de neuronas ocultas, aún para valores superiores a 500 neuronas. Esto puede deberse a que un mayor número de parámetros requiere un mayor número de neuronas para establecer relaciones entre ellos de cara a obtener mejores resultados de salida. El número de VTr también juega un papel fundamental en este caso, pues el efecto de la resonancia impide obtener los mejores valores.

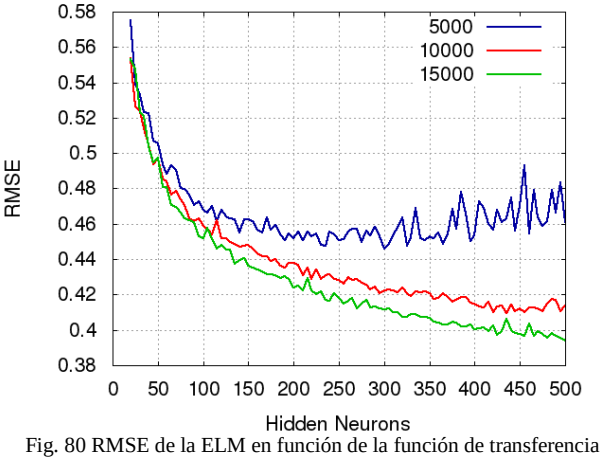


Fig. 80 RMSE de la ELM en función de la función de transferencia

En cuando a la distribución aleatoria del RMSE para este caso, aparentemente, para 100 neuronas y 15000 VTr, las propiedades son las mismas que el problema de una boya. La experiencia muestra que estas propiedades son muy dependientes del número de neuronas y el conjunto de parámetros utilizado. Por ejemplo, como se puede observar en la gráfica de arriba, conforme aumenta el número de neuronas, la varianza del RMSE aumenta.

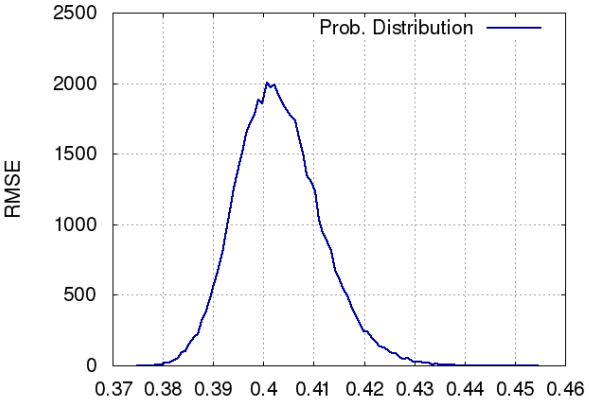


Fig. 81 Función de Distribución de la ELM

Por último, a la derecha se muestra la gráfica de regresión para 50 neuronas y 15000 VTr. El resultado en sí ya es bastante bueno pero nuestro objetivo es mejorarlo, encontrando los parámetros de los que más depende la altura de la ola.

Durante el proceso de selección de características inicial se utilizarán únicamente 50 neuronas por motivos de tiempo computacional. Esto puede interpretarse como una **relajación** del problema.

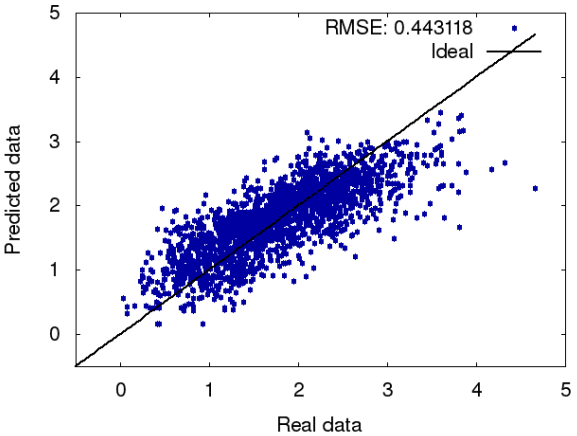


Fig. 82 Gráfica de regresión para todo el conjunto de parámetros

En los apartados finales se aumentará el número de neuronas para así aumentar la precisión de la ELM y encontrar mejores conjuntos de características.

9.3.1 Resultados deterministas

En este apartado se realizará un estudio determinista de los parámetros para intentar descubrir aquellos que tienen una mayor relación con la salida.

La gráfica de la derecha muestra el RMSE asociado a cada uno de los parámetros de entrada por separado. Como se puede observar, el menor error se obtiene para los parámetros 16-19 y 31-34 que son las alturas efectivas y momentos de las 2 boyas más cercanas a la boya 42058, lo cual es razonable.

Con 50 neuronas y 15000 VTr y 2000 VTe tenemos que realizamos un estudio para encontrar los mejores VS.

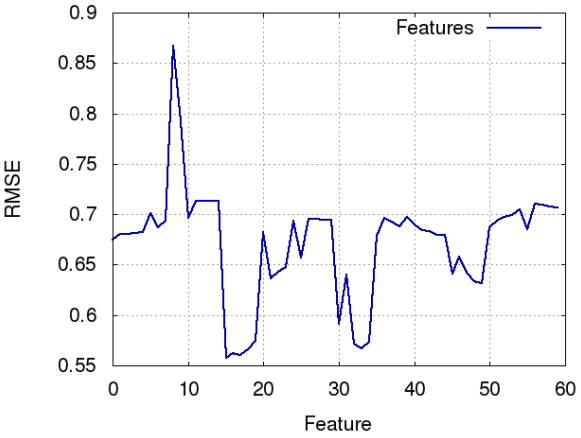


Fig. 83 RMSE de cada uno de los parámetros.

Los mejores VS obtenidos con el algoritmo SFS son:

100000000000000 100000001010000 0001000000000000 00000000000000
100000000000000 111010101110000 0001101001000000 00001000000000
100000000000000 111010111110000 0001101001000001 00001000000000
100000000000000 111010111110000 0001101001000000 00001000000000
100000000000000 101000001010000 0001100001000000 00001000000000

Los mejores VS obtenidos con el algoritmo SBE son:

000000110000000 101000101101000 1000100000000000 00001000000000
000000100000000 101000101100000 1000100000000000 00001000000000
000000100000000 101000101101000 1000100000000000 00001000000000
000000110000000 101000101101010 1000100000001000 00001000000000
000000100000000 100000101100000 1000100000000000 00000000000000

Con esta información y los datos obtenidos en el apartado anterior se ha decidido realizar una Búsqueda Exahustiva sobre el siguiente VS, que contiene los mejores parámetros y los más utilizados. El vector de ES es:

100000100000000 111000101110000 1001100000000000 10001000000000

Los mejores VS de la selección son:

000000100000000 011000100100000 1001100000000000 00000000000000
100000000000000 011000100100000 0001100000000000 00000000000000
000000100000000 100000001010000 0000100000000000 00000000000000
100000000000000 011000001010000 0001100000000000 00000000000000
000000100000000 101000100100000 1000100000000000 10001000000000
000000100000000 101000100100000 1000000000000000 10000000000000
100000000000000 011000101010000 0000000000000000 10000000000000

El RMSE de estos VS varía tan solo en 0.3580 y 0.3590 por lo que su capacidad de producción puede considerarse la misma. Tienen en común que:

- Poseen un parámetro de la primera boya
- Poseen uno o más momentos de la segunda y tercera boya.
- Poseen parámetros secundarios de la segunda boya.

9.3.2 Resultados de algoritmos metaheurísticos

Es hora de utilizar algoritmos metaheurísticos para comprobar si son capaces de encontrar la mejor solución del problema. Estos algoritmos se han ejecutado diversas veces con parámetros muy bajos con respecto a los óptimos calculados para este problema para que el tiempo de ejecución de los algoritmos nos sea excesivo. Los parámetros utilizados son:

30 neuronas 5000 VTr 2000 VTe Función de transferencia sigmoide.

Las siguientes gráficas muestran la ejecución de los algoritmos 3 algoritmos, SA, GA y AIS respectivamente

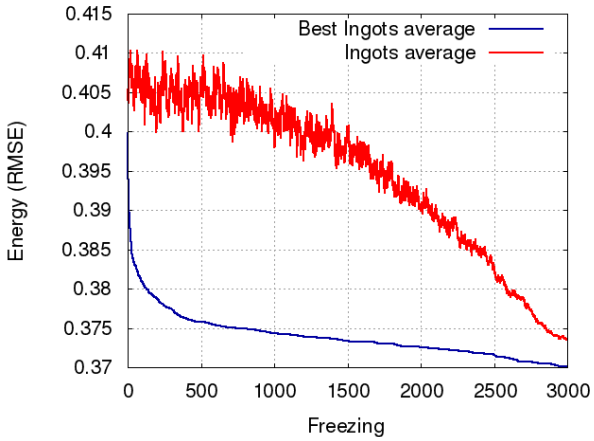


Fig. 85 Resultados finales algoritmo SA

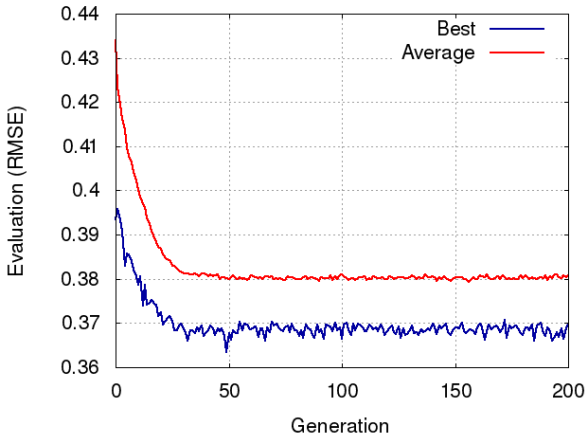


Fig. 84 Resultados finales algoritmo GA

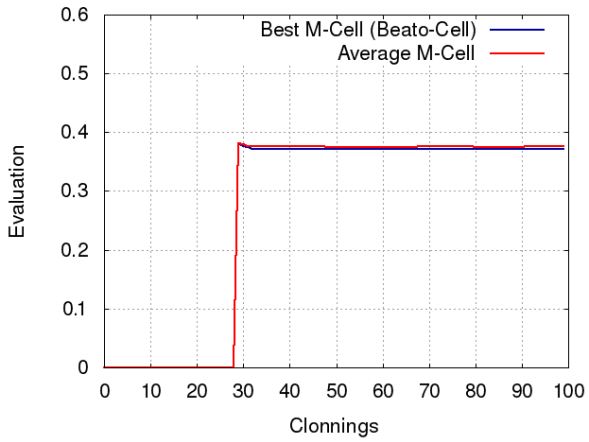


Fig. 86 Resultados finales algoritmoAIS

Los 3 algoritmos se saturan en el valor de RMSE 0.375. Los mejores VS obtenidos por todos los algoritmos comparten la siguiente base común, que se corresponde principalmente con los momentos de las boyas:

100010000000000 110000101010000 0000100000000000 10000000000000

Las características más comunes son:

100010100000000 111000101110000 1001101000000000 10001000000000

Existen multitud de VS complementarios a estos que añaden más características que no perjudican el RMSE del problema. Es posible que utilizando un mayor número de neuronas, estos parámetros sean capaces de aportar más información, mejorando el RMSE. Es por ello que se realizará un último procedimiento de optimización más corto pero utilizando un mayor número de neuronas y VTr.

9.3.3 Optimización de resultados

A partir de los mejores VS obtenidos en el apartado anterior se procederá a utilizarlos como población inicial en los algoritmos SA y GA utilizando un mayor número de neuronas y VTr. Se espera que esto incremente la información que pueden ofrecer los parámetros, reduciendo así el RMSE.

Las diversas pruebas llevadas a cabo no muestran una gran mejoría del RMSE y nos enseñan que solo son necesarias unas pocas características para realizar una predicción óptima.

Los VS finales que daremos como solución son:

100010000000000 100000100110000 0000100000000000 10000000000000
100010100000000 111000101110000 1001101000000000 10001000000000

El primero de ellos contiene los parámetros mínimos que dan el mejor resultado y el segundo añade otros tantos que complementan ligeramente la solución haciéndola más estable.

Las siguientes curvas de regresión representan las mejores soluciones del problema:

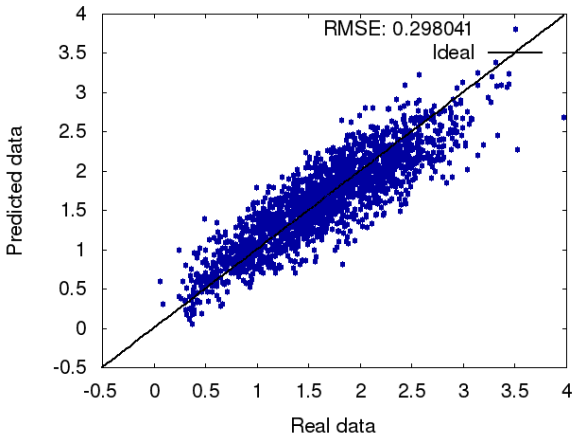


Fig. 88 Regresión final 1

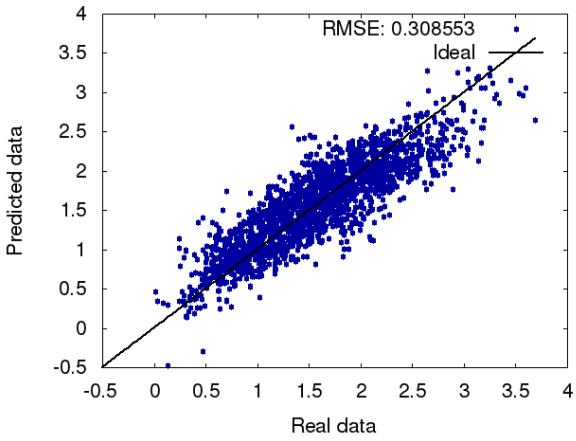


Fig. 87 Regresión final 2

Por último comentar que entre los VTr y VTe hay mediciones de olas que se salen fuera del rango normal de 0,5 a 3 metros que hacen aumentar el error de la predicción, si filtramos estas medidas obtenemos un mejor RMSE como el que se presenta a continuación:

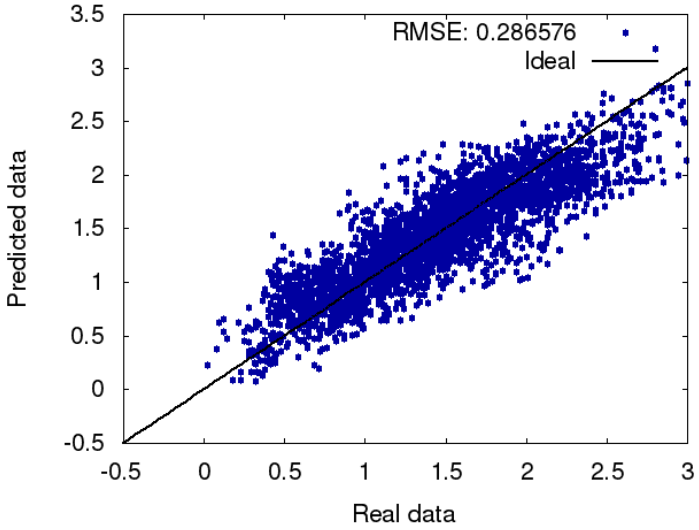


Fig. 89 Regresión final 3

Capítulo 10
Conclusiones

10. CONCLUSIONES

Por último, en este capítulo se presentan las conclusiones del proyecto, así como diversas líneas de trabajo futuro sobre las cuales se puede continuar con la investigación llevada a cabo hasta el momento.

Como se ha podido comprobar en el apartado de resultados, la altura efectiva de ola para la boya 42058 puede predecirse con tan solo un puñado de parámetro de las boyas que se encuentran a su alrededor, basta con saber la altura efectiva de ola de las 2 boyas más próximas para obtener un resultado bastante próximo al óptimo, lo cual hace sospechar que las condiciones marítimas en ambos puntos están gobernadas por las mismas influencias y que el resto de parámetros apenas introduce información complementaria. Para una misma boya se ha comprobado que la altura efectiva de ola H_s puede ser totalmente caracterizada a partir de uno de sus momentos, mientras que el resto de parámetros aporta poca información complementaria, excepto el periodo T_p y el factor de ancho de banda de Longuet-Higgings nu.

Este estudio ha servido por tanto para demostrar que las características del oleaje en un punto del mar caribe son fácilmente predecibles a partir de las características de un punto cercano a este. Esto es un indicador de que la energía de ola en el mar caribe permanece casi constante a lo largo de este, reduciendo la complejidad del cálculo de estimaciones de energía para implementación de WECs.

10.1 Líneas de trabajo futuro

En este apartado se comentan diversas líneas de trabajo a seguir para continuar con la mejora de sistemas de predicción de energía renovables:

- Realizar el estudio de predicción de otros parámetros de la ola a parte de la altura efectiva como pueden ser el periodo T_p para conseguir más información sobre la energía de ola disponible en un área.
- En este proyecto solo se ha intentado predecir una característica del oleaje a partir del oleaje de otros puntos en ese mismo instante. Una línea de trabajo interesante sería predecir la altura efectiva a partir de parámetros en instantes anteriores, ya sea de la misma boya o de otras boyas no muy lejanas a esta. Para hacer esto viable se deberá hacer una preselección de características a utilizar, ya que si utilizamos datos de N instantes anteriores, el número de características se multiplica por este factor. Este proyecto ha podido servir para hacer esta preselección.
- Añadir parámetros temporales y climatológicos a las entradas. Estos datos podrían suponer una gran complementariedad.
- Calcular la energía disponible de las olas.
- Utilizar el programa para otros problemas de selección de características. Dado que gran parte del tiempo destinado a este proyecto ha sido utilizado en el desarrollo de un programa de carácter general, este puede utilizarse en cualquier otro problema de selección de características, con el ahorra temporal que ello conlleva.
- Mejorar el programa en alguna de las siguientes líneas:
    - Añadir un sistema de interacción en tiempo real mediante teclado para poder dar órdenes al programa mientras realiza la búsqueda. Esto puede servir por ejemplo para forzar la finalización de un algoritmo cuando este se ha estancado en un mínimo local.
    - Mejorar el algoritmo genético. Añadiendo mecanismos para controlar la presión selectiva y el estado de la población.

10.2 Estudio de costes

Para completar la memoria se presenta el presupuesto total del proyecto, donde se incluyen las herramientas utilizadas (tanto hardware como software) y la mano de obra. A continuación se desglosan dichos costes.

Tabla 18 Coste de materiales.

Material	Precio (€)	Amortización	Meses de uso	Total (€)
Ordenador	900	5 años	6	90
S.O Ubuntu 12.4	0	5 años	6	0
MATLAB R2013b	2500	5 años	6	250
Herramientas de software libre (gcc, geany, valgrind, gnuplot, LibreOffice)	0	5 años	6	0
Total				340

Tabla 19 Coste de desarrollo.

Función	Precio (€/h)	Nº horas	Total (€)
Estudio y análisis del problema	30	50	1.500
Programación	50	200	10.000
Mecanografía	20	200	4.000
Gastos de impresión	2.500		
Gastos de electricidad	0.01678	1000	16,78
Total			15.516,78

Añadiendo los impuestos del IVA (21%), se obtiene un valor total de 19.186,71 €.

Bibliografía

11. BIBLIOGRAFÍA

[1] Clever Algorithms, Nature-Inspired Programming Recipes, Jason Brawly, First Edition. LuLu. January 2011.

[2] Metaheurísticas: Concepto y Propiedades, José A. Moreno Pérez. Departamento de Estadística, I.O. y Computación. Universidad de La Laguna.

[3] Discrete Optimization – Heuristics Geir Haslen SINTEF ICT, Applied Mathematics, Oslo, Norway University of Jyväskylä, Finland eVITA Winter School 2009 Geilo, January 11.-16. 2009.

[4] Computational Intelligence. An introduction. Second Edition. University of Pretoria.

[5] Artificial Neural Networks. http://web.cs.swarthmore.edu/~meeden/cs63/f11/ml-ch4.pdf

[6] Artificial Neural Networks. http://www.cedar.buffalo.edu/~srihari/CSE555/Chap6.Part1.pdf

[7] Artificial Neural Networks. http://shodhganga.inflibnet.ac.in:8080/jspui/bitstream/10603/48/6/chaper%204_c%20b%20bangal.pdf

[8] Extreme learning machine: Theory and applications Guang-Bin Huang, Qin-Yu Zhu, Chee-Kheong Siew. Neurocomputing 70 (2006) 489–501.

[9] Artificial Neural Networks. Module 22. www.cse.iitd.ac.in/~saroj/AI/ai2013/L22.ppt

[10] Fast Computation of Moore-Penrose Inverse Matrices. Pierre Courrieu. Vol.8, No.2, August 2005. Laboratoire de Psychologie Cognitive, UMR CNRS 6146, Université de Provence.

[11] Neural Networks and Back Propagation Algorithm. Mirza Cilimkovic. Institute of Technology Blanchardstown

[12] TRENDS & CONTROVERSIES. Extreme Learning Machines. Erik Cambria, MIT Media Laboratory Guang-Bin Huang, Nanyang Technological University, Singapore.

[13] Introduction to Extreme Learning Machines. Guang-Bin HUANG. Assistant Professor School of Electrical and Electronic Engineering. Nanyang Technological University, Singapore.

[14] AI – Search Techniques. Abdennour El Rhalibi Simulated Annealing - Threshold algorithms www.dm.uba.ar/materias/optimizacion/2008/1/9SiAnneal.doc

[15] Inteligencia Artificial. Búsqueda local. Primavera 2007. Profesor: Luigi Ceccaroni.

[16] Simulates Annealing. Dimitris Bertsimas and John Tsitsiklis. Satistical Science 1998. Vol 8, No. 1, 10-15.

[17] Chapter 10 THE THEORY AND PRACTICE OF SIMULATED ANNEALING Darrall Henderson.

[18] Simulated Annealing. Rafael Fernández Fernando García.

[19] Parámetros típicos del oleaje J.C. Nieto Borge Universidad de Alcalá 2 de octubre de 2013.

[20] Capítulo 22 ENERGÍA DEL OLEAJE. http://comunidad.eduambiental.org/file.php/1/curso/contenidos/docpdf/capitulo22.pdf

[21] A Review of Computational Intelligence Techniques in Wave Energy Production L. Cuadra. , J. C. Nieto, E. Alexandre , G. Rodr__guez , S. Salcedo-Sanz .

[22] Basic Wave Theory Review. Graham Warren Bureau of Meteorology Australia.

[23] LINEAR WAVE THEORY PART A. Regular waves. HARALD E. KROGSTAD AND ØIVIND A. ARNTSEN.

[24] Búsqueda Local . Grupo 1: Verónica Giaudrone Marcelo Vaccaro.

[25] Problem Solving and Search in Artificial Intelligence. Nysret Musliu Database and Artificial Intelligence Group Institut für Informationssysteme, TU-Wien.

[26] Administración de procesos: Procesos e hilos Gunnar Wolf Facultad de Ingeniería, UNAM.

[27] Sistemas Operativos. Aspectos internos y principios de diseño. William Stallings. 5ª Edicion. Editorial Pearson.

[28] C/C++ Curso de Programación. Francisco Javier Ceballos. 3ª Edición. Editorial Ra-Ma.

[29] Inteligencia en Redes de Comunicaciones. Ingeniería de Telecomunicación. Algoritmos Evolutivos y Algoritmos Genéticos. Alfonso Mateos Andaluz N.I.A.: 100027597. Universidad Carlos III.

[30] What is an Evolutionary Algorithm? http://www.cs.vu.nl/~gusz/ecbook/Eiben-Smith-Intro2EC-Ch2.pdf

[31] Genetic and Evolutionary Algorithms. Gareth Jones University of Sheffield, UK.

[32] Introducción a los Algoritmos Genéticos. Marcos Gestal Pose. Depto. Tecnologías de la Información y las Comunicaciones Universidade da Coruña.

[33] Evolutionary Algorithms Piero P. Bonissone GE Corporate Research & Development.

[34] ALGORITMOS GENÉTICOS. Arranz de la Peña, Jorge y Parra Truyol, Antonio. Universidad Carlos III.

[35] Algoritmos Evolutivos y Memeticos Curso de Postgrado – UC3M – Junio 16,17,18 – 2004

[36] Feature Selection Algorithms: A Survey and Experimental Evaluation Luis Carlos Molina, Lluís Belanche, Àngela Nebot. Universitat Politècnica de Catalunya.

[37] FEATURE SELECTION METHODS AND ALGORITHMS. L.Ladha et al. / International Journal on Computer Science and Engineering (IJCSE).

[38] Artificial Intelligence 97 (1997) 273-324 Wrappers for feature subset selection Ron Kohavi and George H. John.

[39] An Introduction to Variable and Feature Selection. Isabelle Guyon. Journal of Machine Learning Research 3 (2003) 1157-1182.

[40] A Few Useful Things to Know about Machine Learning. Pedro Domingos. Department of Computer Science and Engineering. University of Washington.

[41] Machine learning methods for bioinformatics INFO-F-528. Gianluca Bontempi Département d'Informatique. Boulevard de Triomphe - CP 212.

[42] Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques Edmund K. Burke (Editor), Graham Kendall (Editor). Chapter 13 ARTIFICIAL IMMUNE SYSTEMS.

[43] ARTIFICIAL IMMUNE SYSTEMS – MODELS, ALGORITHMS AND APPLICATIONS 1J.R. Al-Enezi, M.F. Abbod & S. Alsharhan. IJRRAS 3 (2) . May 2010 Al-Enezi & al. Artificial Immune Systems.

[44] ARTIFICIAL IMMUNE SYSTEMS: PRINCIPLE, ALGORITHMS AND APPLICATIONS. BY SATYASAI JAGANNATH NANDA.

[45] Algoritmos genéticos http://www.obitko.com/tutorials/genetic-algorithms/

[46] Trabajo Fin de Máster: Mejora de la resolución en predicción de viento con herramientas de Inteligencia Computacional. Autor: Álvaro Pastor Sánchez.

[47] http://www.ndbc.noaa.gov/

[48] Trabajo Fin de Carrera: Rediseño automático de configuraciones viales urbanas basado en computación evolutiva. Autor: Álvaro Pastor Sánchez