TRABAJO FIN DE GRADO
INGENIERÍA EN INFORMÁTICA

Desarrollo de librería sobre Algoritmos Evolutivos en Julia

Autor
Eloy Bedia García

Director
Daniel Molina Cabrera

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, Noviembre de 2020

Desarrollo de librería sobre Algoritmos Evolutivos en Julia

Autor
Eloy Bedia García

Director
Daniel Molina Cabrera

Desarrollo de librería sobre Algoritmos Evolutivos en Julia

Eloy Bedia García

Palabras clave: Julia, Python, Algoritmos Evolutivos, Metaheurísticas

Resumen

Un lenguaje de programación es hoy en día una necesidad para las personas dedicadas a las ciencias o ingenierías en sus múltiples ámbitos La mayoría eligen Python por su sencillez a la hora de aprenderlo aunque en ocasiones el rendimiento no es el esperado o surge la necesidad de recurrir a librerías externas para poder acelerarlo. Julia es un lenguaje que cumplirá con todas estas expectativas ya que es muy fácil de aprender y además es rápido.
En este proyecto se desarrolla una librería compuesta por algoritmos evolutivos desarrollada en Julia, un lenguaje de programación bastante reciente. Además de un excelente rendimiento, también posee una interfaz muy simple para que pueda integrarse en cualquier entorno con la mayor brevedad posible.

Library Development about Evolutionary Algorithm in Julia

Eloy Bedia García

Keywords: Julia, Python, Evolutionary Algorithm, Metaheuristic

Abstract

A programming language is today a necessity for people dedicated to science or engineering in their multiple ambitions. Most choose Python for its simplicity when it comes to learning it, although sometimes performance is not as expected or the need arises to resort to external libraries in order to accelerate it. Julia is a language that will meet all these expectations as it is very easy to learn and also fast.
This project develops a library composed of evolutionary algorithms developed in Julia. In addition to excellent performance, it is also a very simple interface so that it can be integrated into any environment as soon as possible.

Yo, **Eloy Bedia García**, alumno de la titulación TITULACIÓN de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 45119918J, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Eloy Bedia García

Granada a 7 de Noviembre de 2020 .

D. **Daniel Molina Cabrera**, Profesor del Área XXXX del Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado **Desarrollo de librería sobre Algoritmos Evolutivos en Julia**, ha sido realizado bajo su supervisión por **Eloy Bedia García**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 7 de Noviembre de 2020 .

El director:

Daniel Molina Cabrera

Indice

1. Introducción

A menudo la sociedad se enfrenta a problemas de gran dificultad. Para resolver estos problemas, la tecnología tiene la responsabilidad de ofrecerse como apoyo con el objetivo proporcionar las herramientas adecuadas.

Algún ejemplo de estos problemas podrían ser la predicción del tiempo [1] o seismos [2] cuyo pilar fundamental, en ambos casos, es la inteligencia artificial, en concreto el aprendizaje automático. En el caso de la arquitectura, se utilizan modelos de simulación para evaluar la seguridad estructural [3] o incluso arrojar estimaciones de los costes de producción.

Para resolver este tipo de problemas existen lenguajes de programación que ofrecen gran flexibilidad sacrificando el rendimiento. Estos podrían ser MATLAB, R o Python, encontrándose éste último en el top 3 de los lenguajes mas populares de 2020 según el indice TIOBE, por esta razón lo utilizaremos como lenguaje de referencia a lo largo del documento.

MATLAB [4] es un lenguaje que permite utilizar operaciones vectoriales y matriciales de una manera muy natural; R [5] es un lenguaje de programación orientado al análisis estadístico siendo muy popular en áreas como aprendizaje automático; Python es un lenguaje de propósito general cuya principal ventaja es su legibilidad y facilidad de aprendizaje.

Estos tres lenguajes, como todos los demás, tienen sus ventajas e inconvenientes, pero para el desarrollo de este trabajo es muy importante resaltar un inconveniente que es común a los tres, y es el problema del doble lenguaje. Esto es que para poder obtener un rendimiento razonable en algoritmos computacionalmente complejos, es necesario hacer uso de bibliotecas programadas en C o C++ que son cargadas dinámicamente.

Para resolver este problema nace Julia [6], un lenguaje de programación pensado para la comunidad científica tan flexible como Python pero con un rendimiento equiparable a lenguajes como C/C++. Este lenguaje tan joven cuenta ya con una vasta colección de bibliotecas científicas con las que se ha ganado el apoyo de la comunidad.

Además, Julia no solo cuenta con un gran rendimiento, si no que también cuenta con una sintaxis muy similar a la de Python, lo que quiere decir que, como hemos dicho anteriormente de este lenguaje, cuenta con una sintaxis muy fácil de leer y aprender.

Habiendo sido pensado para la comunidad científica es de suponer que el lenguaje debe contar con bibliotecas para resolver estos problemas, y así es. JuMP es un paquete de optimización exacta de funciones. Pero, ¿Y si para nuestro problema no es necesario tener el óptimo de la función y es suficiente con una aproximación? Para ello existen alternativas como los Algoritmos Evolutivos.

Julia cuenta con paquetes de Algoritmos Evolutivos desarrollados por la comunidad, como por ejemplo CMAES.jl que, como su nombre indica, implementa el algoritmo CMAES. Pero no existe un paquete con una variedad de algoritmos evolutivos.

Para mitigar esta carencia, a lo largo de esta memoria se va a mostrar el desarrollo de un paquete de algoritmos evolutivos que implementa algoritmos genéticos, GA, en sus variantes generacional y estacionaria; evolución diferencial, DE; optimización de enjambre de partículas, PSO, en sus variantes local y global; algoritmo gravitacional, GSA y algoritmo de forrajeo bacteriano, BFO.

Esta librería programada en Julia promete superar en rendimiento a librerías similares desarrolladas en lenguajes como Python. Además, tendrá una interfaz tan flexible que será posible utilizar código ajeno a la librería para programar nuevos tipos de cruces, mutaciones, selecciones, etc.

Sería posible también añadir incluso nuevos algoritmos respetando, obviamente, la interfaz. Para facilitar el análisis del resultado del algoritmo, se habilita una función callback, también personalizable, para poder mostrar datos al comienzo de cada iteración.

Para probar que las expectativas de rendimiento son ciertas, se elaborará una comparación de algunos de los algoritmos programados en la librería con otros programados en Python. Para hacerla se utilizarán las 16 primeras funciones del benchmark CEC 2014 [26].

2. Objetivos

Como se ha dejado entrever en la sección anterior, este trabajo consta de tres objetivos claros.

- Desarrollar un paquete de algoritmos evolutivos para Julia
- Realizar una comparación entre los distintos algoritmos desarrollados
- Elaborar una comparación de rendimientos de Julia frente a Python

El primero de ellos, como bien se dice, es el desarrollo de un paquete con cierta variedad de algoritmos evolutivos y que además, al ser Julia un lenguaje muy utilizado por matemáticos, es decir que no tienen por qué tener un nivel alto de programación, se precisa de una interfaz flexible y fácil de usar.

Los algoritmos que se incluyen en el paquete son los siguientes:

- Algoritmos Genéticos (Genetic Algorithms, GA)
  - Generacional (Generational, GGA)
  - Estacionario (Steady-State, SSGA)
- Algoritmo Evolución Diferencial (Differential Evolution, DE)
- Algoritmo de Optimización con Enjambre de Partículas (Particle Swarm Optimization, PSO)
  - Global (Global Particle Swarm Optimization, GPSO)
  - Local (Local Particle Swarm Optimization, LPSO)
- Algoritmo de Búsqueda Gravitacional (Gravitational Search Algorithm, GSA)
- Algoritmo de Forrajeo Bacteriano (Bacterial Foraign Optimization, BFO)

En la medida de lo posible, los algoritmos tendrán una parametrización que permita al usuario poder llegar a personalizar el uso del algoritmo. Por ejemplo, en el caso de los algoritmos genéticos, permitir programar un tipo de cruce que no venga predefinido y poder utilizarlo de manera sencilla.

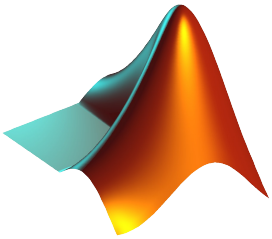
Además, se hará una comparación de los distintos algoritmos implementados a nivel de resultados para poder ofrecerle al usuario más información acerca de qué algoritmo usar según qué casos.

Para ello, lanzaremos cada algoritmo con funciones pertenecientes al CEC14 Benchmark Suite con dimensiones 20, 30, 50 y 100.

El siguiente objetivo es comprobar hasta qué punto Julia es más rápido que Python. Haciendo uso de algunos de los algoritmos mencionados anteriormente, programados tanto en Python como en Julia (utilizando nuestro propio paquete) se elaborará una comparación.

3. Contexto Tecnológico

Para facilitar el contexto de los dos lenguajes utilizados para elaborar este proyecto vamos a hacer uso del índice TIOBE [7] proporcionado en el mes de Junio de 2020. Este índice es un indicador de la popularidad de cada lenguaje. La posición que ocupa cada lenguaje está basada en el personal cualificado que hace uso de ellos, la cantidad de información que existe, o las empresas que lo utilizan.



MATLAB [4] es un lenguaje interpretado con licencia propiedad de *The Mathworks,* esto último significa que para hacer uso de él es necesario pagar el costo de la licencia. Este lenguaje permite utilizar operaciones vectoriales y matriciales de una manera muy natural, además de proveer bibliotecas para distintos fines como por ejemplo optimización, tratamiento de señales o análisis de texto entre otras.

Este ocupa la 15º posición con un ratio del 0,90%, y con expectativas de que siga subiendo posiciones en el ranking. Desde mayo de 2017, este lenguaje lleva una tendencia descendente en este ranking.

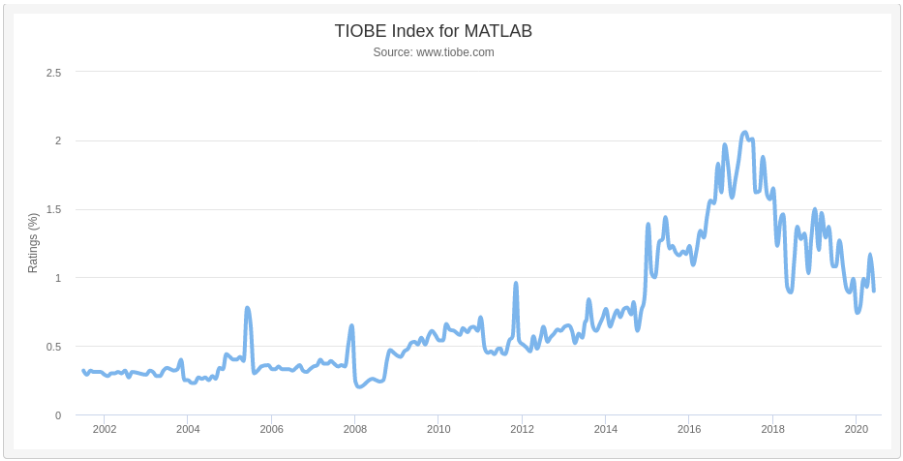


Figura 1: Evolución de Matlab en TIOBE

R [5] es un lenguaje de programación orientado al análisis estadístico siendo muy popular en áreas como aprendizaje automático, minería de datos, biomedicina o bioinformática. Además, este forma parte de un proyecto colaborativo en el que usuarios pueden publicar sus propios paquetes y actualmente goza de 2697 paquetes. A diferencia de MATLAB [4], este lenguaje tiene una licencia de software libre y código abierto, por lo que utilizarlo es totalmente gratuito.

Este ocupa la 9º posición con un ratio del 2,19%, y con expectativas de que siga subiendo posiciones en el ranking. Desde enero de 2018, este lenguaje lleva una tendencia a la descendente en el ranking, sin embargo desde principios de este año ha subido casi un 2%.

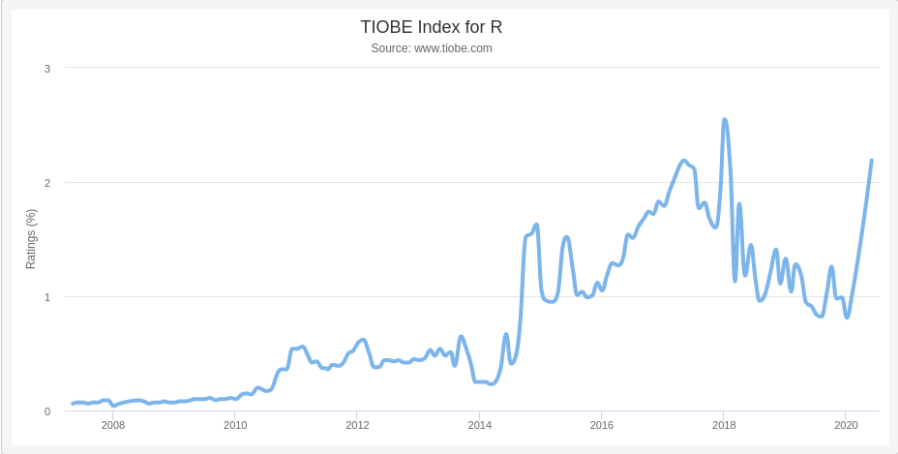


Figura 2: Evolución de R en TIOBE

l

Python es un lenguaje de propósito general cuya principal ventaja es su egibilidad y facilidad de aprendizaje. Al igual que los anteriores lenguajes, este también posee un amplia biblioteca estándar que permite realizar una gran variedad de tareas. Entre sus usos más comunes se encuentran la programación de interfaces gráficas, programación web, gestión de bases de datos o testing de aplicaciones. Además es un lenguaje con una gran aceptación en la comunidad científica.

Este ocupa la 3º posición con un ratio del 8,36%, y con expectativas de que siga subiendo posiciones en el ranking. Desde septiembre de 2017, este lenguaje lleva una tendencia a la alza en este ranking, esto podría ser por la importancia que esta tomando la ciencia de datos estos últimos años.

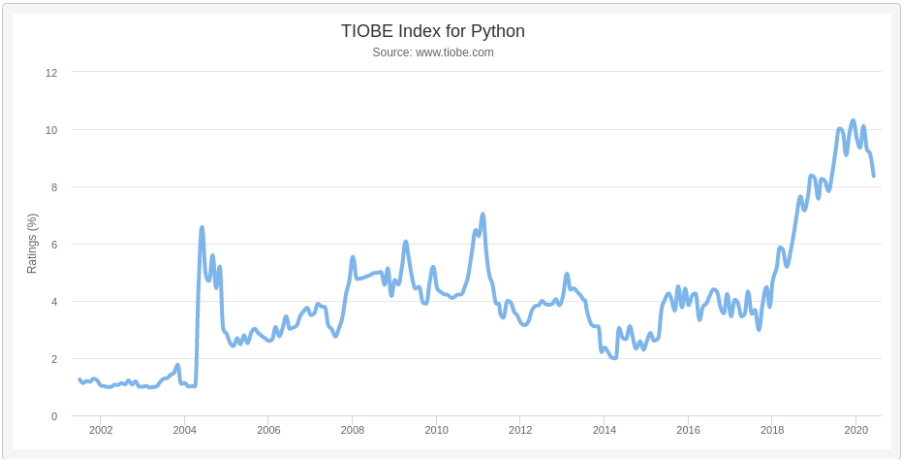


Figura 3: Evolución de Python en TIOBE

Empresas muy importantes en el sector tecnológico mantienen software en este lenguaje. Netflix, entre otros muchos usos que le da al lenguaje, lo utiliza para análisis de datos. Google tuvo durante años en plantilla al creador de Python, Guido van Rossum, que desarrolló un sistema de revisión de código para uso interno. Facebook o Instagram son plataformas que también usan Python en sus servidores. Contar con el apoyo de empresas de tal volumen es una gran ventaja a

la hora de popularizar un lenguaje.

Por otro lado, está Julia. Este lenguaje ocupa el puesto 33 del ranking con un ratio del 0.41 %. Sobre este lenguaje no hay más información en este ranking. Este lenguaje es de reciente creación (año 2012), sin embargo, está compitiendo contra lenguajes que ya están muy consolidados en la comunidad como son R, Matlab o el propio Python. Por ello, resultará difícil que pueda ser uno de los lenguajes más populares.

Julia también cuenta con bibliotecas desarrolladas por algunas empresas como Pfizer, una de las farmacéuticas más grandes del mundo, que utiliza Julia para acelerar las simulaciones de nuevas terapias. Racefox, una empresa dedicada al asesoramiento deportivo digital en tiempo real para corredores, es otro ejemplo de negocio que utiliza Julia en su software. Julia también es utilizado instituciones oficiales como El Banco Nacional de Desarrollo Económico y Social, vinculada al Ministerio de Desarrollo, Industria y Comercio Exterior de Brasil, para acelerar sus modelos matemáticos.

Otro proyecto interesante que destaca la potencia de Julia es el proyecto celeste. En 2014 un grupo de científicos e ingenieros se unieron para trabajar en este proyecto con el fin de desarrollar un nuevo método para catalogar objetos celestes. Nace así un modelo de computación paralela capaz de procesar 188M de objetos en 14 minutos.

Dejando a un lado la popularidad de ambos lenguajes, procedemos a entrar en aspectos técnicos. Python y Julia, aunque con sintaxis bastante similar, tiene muchas diferencias a nivel técnico.

3.1 Aspectos de Python

Python es un lenguaje de "scripting", concebido dentro del paradigma de programación orientada a objetos, cuyo diseño se focalizó en la legibilidad del código. Fue creado, como sucesor del lenguaje de programación ABC [8], por Guido van Rossum a finales de la década de 1980 en el Centro para las Matemáticas y la Informática (CWI).

Python cuenta con unos principios denominados como "The Zen of Python" [9] que deja ver que el principal objetivo de Python es su legibilidad.

- Bello es mejor que feo.
- Explícito es mejor que implícito.
- Simple es mejor que complejo.
- Complejo es mejor que complicado.
- Plano es mejor que anidado.
- Espaciado es mejor que denso.
- La legibilidad es importante.
- Los casos especiales no son lo suficientemente especiales como para romper las reglas.
- Sin embargo la practicidad le gana a la pureza.
- Los errores nunca deberían pasar silenciosamente.
- A menos que se silencien explícitamente.
- Frente a la ambigüedad, evitar la tentación de adivinar.
- Debería haber una, y preferiblemente solo una, manera obvia de hacerlo.
- A pesar de que esa manera no sea obvia a menos que seas Holandés.
- Ahora es mejor que nunca.
- A pesar de que nunca es muchas veces mejor que *ahora* mismo.
- Si la implementación es difícil de explicar, es una mala idea.
- Si la implementación es fácil de explicar, puede que sea una buena idea.

- Los espacios de nombres son una gran idea, ¡tengamos más de esos!

En Python existen múltiples implementaciones del lenguaje. CPython, la implementación original escrita en C, es la que nos podemos encontrar en el sitio oficial de Python; por otro lado, PyPy es una implementación de Python escrita en Python optimizada mediante un compilador just-in-time (JIT).

De la primera de ellas, tiene la ventaja de que al estar escrita en C, es muy sencillo crear bibliotecas escritas en C para que sean llamadas desde código Python. Esto nos permite desarrollar y utilizar bibliotecas como NumPy o ScyPy que incrementan el rendimiento de funciones propias de la implementación del lenguaje.

Por otro lado, PyPy es una versión que ya tiene un rendimiento reseñable, pero tiene un inconveniente, y es que no nos permite llamar a código C, por lo que bibliotecas como las mencionadas en el párrafo anterior quedan totalmente descartadas.

3.2 Aspectos de Julia

En la actualidad muchos científicos realizan su trabajos con lenguajes de tipado dinámico aunque C o Fortran siguen siendo lenguajes a los que recurrir cuando el problema es requiere alta carga computacional. Visto de esta forma, los que programan en lenguajes de tipado dinámico están perdiendo rendimiento mientras que los programadores de C o Fortran pierden productividad, ya que los códigos en este tipo de lenguajes suelen requerir muchas más líneas.

Julia es un lenguaje de reciente creación que resuelve este problema combinando productividad y alto rendimiento. Esto es posible gracias a que durante la etapa del diseño del lenguaje, las tecnologías utilizadas fueron cuidadosamente seleccionadas para que pudieran convivir adecuadamente produciendo así el resultado esperado.

Gracias a la filosofía seguida durante su diseño, Julia resuelve el problema del doble lenguaje. Esta es que toda funcionalidad básica debe poder implementarse en Julia, lo que nos lleva a su siguiente principio; no forzar a los desarrolladores a tener que utilizar otros lenguajes como C. Funcionalidades básicas pueden ser aritmética, que en lenguajes como Python (utilizando NumPy), R o MATLAB esconden tras de si código C o Fortran.

El manejo de tipos en un lenguaje es algo que marcará su rendimiento, y es que cuanto más sepa la computadora acerca de los datos mejor se adaptará a ellos optimizando así su rendimiento. Los tipos son metadatos que informan a la computadora sobre los datos que maneja. Por un lado, los de tipado estático tienen una definición de tipos durante el tiempo de compilación que dotan al lenguaje de un excelente rendimiento; por otro lado, los de tipado dinámico, prescinden totalmente de definiciones de tipo, lo que aumenta su productividad sacrificando su rendimiento.

Julia por el contrario, a pesar de ser un lenguaje de tipado dinámico, tiene un sistema de inferencia de tipos por lo que al tener más información sobre los datos, tiene un rendimiento notable. Además muchos de los usuarios de Julia es posible que no sepan nada acerca de los tipos de datos, por lo que es una opción muy ventajosa en este sentido.

Relacionado con lo anterior tenemos también los tipos definidos por el usuario. Y es que normalmente, los tipos definidos por el lenguaje (*built-in*) son mucho más rápidos que los definidos por los usuarios. En el caso de Julia, esto no es así, no existe una diferencia significativa entre ambos tipos.

La mayor parte de usuarios de la comunidad de Julia pertenecen al ámbito científico, donde el alfabeto griego está muy extendido a la hora de definir fórmulas. En este aspecto Julia también tiene ventaja sobre otros lenguajes, ya que escribir expresiones matemáticas en Julia es muy parecido escribirlas en la realidad.

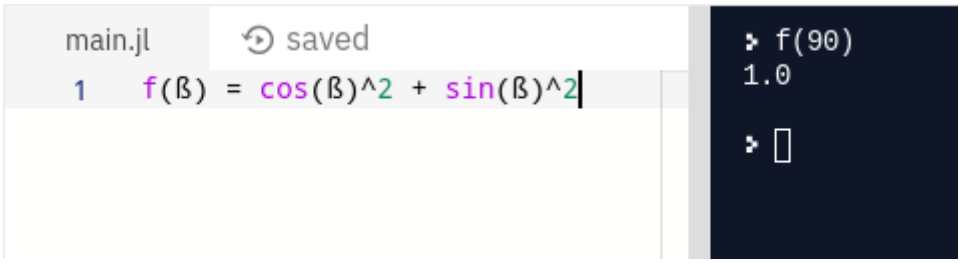


Figura 4: Ejemplo de uso de alfabeto griego en Julia

Julia también da facilidades para implementar paralelización. La programación distribuida en Julia se construye sobre primitivas. Cabe destacar las primitivas, sirviendo como ejemplo de lo que se ha dicho anteriormente, están completamente escritas en Julia.

```julia
function potencia_modular(a, b, c)
    p = 1

    while(b > 0)
        if(b % 2 == 1)
            p = (p * a) % c
        end

        b = div(b, 2) #division entera
        a = (a * a) % c
    end
    return p
end
```

Figura 5: Potencia Modular en Julia

```python
def potenciamodular(a, b, c):
    p = 1

    while(b > 0):
        if (b % 2 == 1):
            p = (p * a) % c



        b = b // 2 #division entera
        a = (a * a) % c


    return p
```

Figura 6: Potencia Modular en Python

4. Planificación y Presupuestos

4.1 Aprender Julia

La primera tarea que se plantea para el desarrollo del trabajo es aprender las nociones básicas del lenguaje. Si bien es cierto que al conocer lenguajes con similar sintaxis como puede ser Python facilita la tarea, cada lenguaje tiene sus propias características y estilos, por lo que es necesario familiarizarse con ellos.

4.2 Recopilación de Información sobre Algoritmos

Antes de comenzar cualquier desarrollo es importante tener un conocimiento, cuanto más detallado mejor , sobre lo que se va a realizar, en este caso, buscamos información sobre los algoritmos evolutivos que se van a incluir en el paquete.

- **Algoritmos Genéticos**: La intención es buscar información de las dos modalidades que manejan este tipo de algoritmos: modelos generacional y modelo estacionario. Además de buscar información sobre los diferentes métodos de selección, cruce, mutación y remplazo característicos de este tipo de metaheurísticas.

  Esta tarea fue bastante rápida ya que se tenia un conocimiento previo, tanto teórico como práctico, sobre los algoritmos.

- **Algoritmos de Evolución Diferencial**: Al igual que anterior grupo de algoritmos, esta tarea fue rápida ya que poseía conocimientos teóricos previos.

- **Algoritmos de Enjambre de Partículas:** Junto con los dos grupos anteriores, de este tipo de algoritmos ya tenía conocimiento previo, por lo que esta tarea sirvió para recordar y afianzar. En este caso la búsqueda se centro en dos modalidades: enjambre de partículas local y enjambre de partículas global.

- **Algoritmo de Luciérnagas:** Este es un algoritmo del cual no tenía ningún conocimiento previo por lo que fue necesario indagar más acerca de él. Fue sencillo de entender al estar inspirado en un proceso de la naturaleza bastante común como es el cortejo.

- **Algoritmo de Forrajeo Bacteriano:** Al igual que pasaba con el anterior algoritmo, tampoco tenía conocimientos previos pero por el contrario, fue más difícil de entender, ya que se basa en un proceso más complejo.

- **Algoritmo Gravitacional:** De este algoritmo si que tenía conocimientos previos y además al estar basado en un proceso físico como es la ley de gravitación universal, es un algoritmo muy fácil de entender.

4.3 Diseñar Interfaz

En paralelo, una de las tareas fundamentales del proyecto fue el diseño de la interfaz. Esta debía ser flexible y poco complicada al mismo tiempo, recordemos que el objetivo del paquete es que usuarios de todos los niveles puedan hacer uso de las ventajas que ofrecen las diferentes metaheurísticas sin necesidad de tener conocimientos avanzados del lenguaje.

4.4 Implementación

Esta tarea fue llevada en paralelo junto con las dos anteriores. Durante el periodo de tiempo que se le dedicó a esta tarea, se abordaron cuestiones de implementación de cada algoritmo, ciñéndose siempre al diseño de interfaz realizado previamente. Antes de implementar cada algoritmo, se hacia una búsqueda de información sobre cada uno de ellos.

Después de cada algoritmo se implementaban test unitarios para verificar que el algoritmo cumpliese unos requisitos mínimos.

Nombre del Algoritmo	Tiempo Real	Tiempo Estimado
Genético Generacional	75h	75h
Genético Estacionario		
Evolución Diferencial	12h	15h
Forrajeo Bacteriano	12h	15h
Enjambre de Particulas Local	12h	24h
Enjambre de Particulas Global		
Cambio de Interfaz	25h	50h
Luciernagas	12h	20h
Gravitacional	12h	20h
Corrección de Errores	25h	15h

Figura 7: Planificación para Implementación

La tarea “Cambio de Interfaz” consistió en una mejora de la interfaz para unificar todos los algoritmos en una sola función, esto se hizo concibiendo cada algoritmo como una estructura que es recibida como parámetros a través de esta función.

4.5 Experimentación

Tras la fase de implementación, es importante en nuestro caso comprobar la hipótesis de la que partimos: Julia supera a Python en cuanto a rendimiento. Para probar esto, se han tomado la implementaciones de Python de los algoritmos genéticos, evolución diferencial y algoritmo gravitacional; y hemos calculado el tiempo que tardan en realizar las optimizaciones pertenecientes al benchmark. A continuación se ha hecho el mismo procedimiento con los algoritmos pertenecientes a nuestro paquete.

Para optimizar el proceso, se han creado varios scripts:

- **Script de Tiempos en Python**: recibe como parámetros el número de función con el que se desea ejecutar el cálculo y lo realiza con los algoritmos de Python. Al terminar graba los tiempos en un fichero CSV.
- **Script de Tiempos en Julia**: recibe como parámetros el número de función con el que

se desea ejecutar el cálculo y lo realiza con los algoritmos de Julia. Al terminar graba los tiempos en un fichero CSV.

- **Script de Optimización en Python**: recibe como parámetros el número de función con el que se desea ejecutar el cálculo y lo realiza con los algoritmos de Python. Al terminar graba los resultados en un fichero CSV.
- **Script de Optimización en Julia**: recibe como parámetros el número de función con el que se desea ejecutar el cálculo y lo realiza con los algoritmos de Julia. Al terminar graba los resultados en un fichero CSV.
- **Script generacion.dat**: Toma los datos del fichero CSV y los pasa a un fichero.dat con almacenando en cada linea el número de dimensión y el tiempo invertido (o el resultado).
- **Script de Gráficos Gnuplot**: toma los datos del fichero .dat de un algoritmo y realiza la gráfica tomando los datos de ambos lenguajes.

Como experimentación adicional, también se comparan los resultados obtenidos por cada algoritmo.

Script	Tiempo Real	Tiempo Estimado
Tiempos en Python	18h	18h
Optimización en Python		
Tiempos en Julia	18h	18h
Optimización en Julia		

Figura 8: Planificación para Experimentación

4.6 Presupuesto

A continuación, se da una aproximación de la remuneración económica que se hubiera reclamado en el caso de haber sido un trabajo por contrato.

Concepto del Gasto		A pagar	Horas
Material	Acer Aspire 5 A515-54-735N	649,00 €	
Salario del Personal		20 €/h	185h
Total		4.349,00 €	

Figura 9: Presupuesto

5. Revisión de la Literatura

La optimización de problemas con codificación real ha sido objeto de estudio en las últimas décadas siendo un apoyo para las investigaciones realizadas sobre los Algoritmos Evolutivos. En este caso, vamos a trabajar con varios algoritmos: Algoritmos Genéticos, Evolución Diferencial, Forrajeo Bacteriano, Enjambre de Particulas, Luciérnagas y Búsqueda Gravitacional.

Los algoritmos genéticos [10] [11][30] fueron ideados por John H. Holland en el año 1961. Apoyado sobre los trabajos de genética teórica iniciados por R. A. Fisher y Sewall Wright, estos algoritmos se basan en la capacidad de evolución y adaptación de los sistemas biológicos. Dicho en términos de ingeniería, estos algoritmos dotan al sistema de la capacidad de adaptarse al problema haciendo evolucionar la solución hacia el óptimo. Algunos de los usos que se le ha dado a este tipo de algoritmos está relacionado con las redes neuronales.

Los algoritmos de evolución diferencial [15][27] fueron creados por Rainer Storn en el año 1996. Surge como un nuevo enfoque para minimizar funciones no lineales en espacio continuo. Debe su éxito a su rápida convergencia y precisión en la búsqueda del optimo global. Su baja cantidad de parámetros ajustables y su facilidad de uso lo hacen ideal para entornos en los que no se tiene un conocimiento profundo del algoritmo. El problema del despacho económico dinámico consiste en predecir la demanda de energía durante un periodo de tiempo atendiendo a unas restricciones impuestas por el sistema. Este problema ha sido reconocido como uno de los problemas de optimización más difíciles debido a su alta dimensionalidad [16]. Differential Evolution fue utilizado para resolver este problema junto con el PSO y GA [17]. DE demostró presentar mejores planificaciones que las generadas por los dos algoritmos restantes.

El algoritmo de forrajeo bacteriano [18][31] se originó en el año 2002 por Hans Bremermann. En 1971 el físico Max Ludwig Henning Delbrück publicó un trabajo sobre la chemotaxis, una acción que lleva a cabo la bacteria con el fin de encontrar nutrientes. Este algoritmo surge a raíz de las simulación realizadas sobre estos procesos biológicos. Uno de los problemas resueltos por este algoritmo esta relacionado con las telecomunicaciones. Este problema se denomina directividad optima (ODG), un problema de optimización no lineal cuya solución determina la configuración de angulos. Por ejemplo, se trata de aplicar este problema a un tipo de antena llamada dipolo en V [19].

El algoritmo de enjambre de partículas [20][32] fue diseñado por James Kennedy y Russell Eberhart en 1995. Al igual que pasaba con el algoritmo anterior, este tambien surge de una simulación, en este caso utilizada para estudiar las dínamicas de grupos existentes en las bandadas de pájaros. Un ejemplo de aplicación de este algoritmo es la resolución del problema de diseño de red de transporte (TNDP) [21]. Dada una red de carreteras definida por un grafo, la solución al problema consiste en obtener la ruta que optimize algún parámetro del viaje, por ejemplo: el presupuesto, el tiempo de viaje, la distancia recorrida...

El algoritmo de luciernagas [22] fue creado por Xin-She Yang en el año 2008. Este algoritmo, fue probado para selección de características [23] en problemas de clasificación con técnicas de aprendizaje automático. El mismo experimento fue probado en varios conjuntos de datos además de con lo salgoritmos GA y PSO. En la mayoria de los casos, FFA obtenia mejores resultados.

El algoritmo de búsqueda gravitacional [24][33] fue publicado en Junio de 2009 por Esmat Rashedi, Hossein Nezamabadi-pour y Saeid Saryazdi. A diferencia del resto de algoritmos, este no se inspira en un proceso biológico o social, está basado en la ley de la gravitación universal formulada por el conocido físico Isaac Newton. Uno de los problemas en los que se ha probado este algoritmo tiene que ver con la ingeniería eléctrica, más concretamente con los filtros adaptativos [25]. Un filtro adaptativo es un sistema con un filtro lineal con una función de transferencia y un algoritmo de optimización que se utiliza para ajustar los parámetros de dicha función. Los

resultados ofrecidos por el algoritmos fueron comparables a algoritmos más consolidados como pueden ser los ya mencionados GA o PSO.

6. Análisis

En el presente apartado se estudia la eficacia de cada uno de los algoritmos que se han integrado en el paquete desarrollado para este trabajo.

A continuación se explicará cada algoritmo individualmente.

6.1 Inicialización

La inicialización es una parte fundamental de cada algoritmo, ya que marcará el punto de partida de nuestra población. Cada individuo de la población será un vector de dimensión D que representara un a posible solución al problema de optimización abordado.

$$X_i=[X_{i,1}, X_{i,2}, X_{i,3}, X_{i,4}, ..., X_{i,D}]$$

Es importante que los individuos, en primera instancia, estén uniformemente distribuidos a lo largo del dominio del problema para poder cubrir el mayor área posible. Además, es posible definir restricciones en el dominio.

Algorithm 1 Población de dimensión NxD inicializada con una distribución uniforme

```
1: procedure INITPOP(N, D)
2:     P := {x_{i,j} | x_{i,j} ∈ U(0,1), i < N, j < D}
3:     return P
4: end procedure
```

Algoritmo 1: Inicialización

6.2 Evolución Diferencial, DE

El algoritmo DE [2], nace hace mas de dos décadas como uno de los algoritmos de optimización continua más prometedores. Este algoritmo es tan poderoso, que actualmente los algoritmos ganadores de competiciones como CEC están fundamentados en él.

Este tipo de algoritmos constan de 4 fases.

Mutación

Biologicamente hablando, una mutación es una perturbación en el genotipo. En este caso, una mutación consiste en una perturbación en el vector llamado *target vector*. Un vector que ha sido mutado pasa a ser llamado *donor vector*.

Existen varias formas de realizar mutaciones en DE . Dos de las más conocidas son:

- DE/best/1

  En este caso, la mutación parte del mejor individuo hallado hasta el momento. La pertur-

bación que se realiza consiste en desplazar este individua hacia otro punto del plano dejándose guiar por dos individuos elegidos de manera aleatoria.

$$V = X_{best} + F \cdot (X_1 - X_2)$$

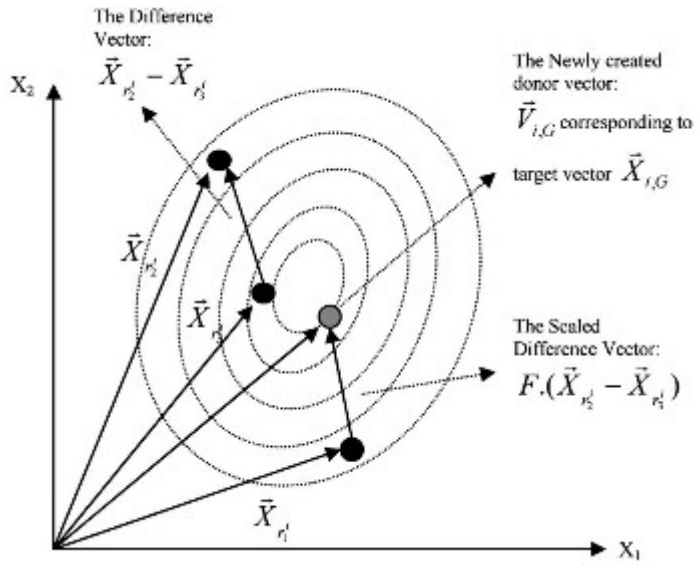


Figure 10: Ejemplo de mutación en un plano

Algorithm 2 Operador de mutación DE/best/1

```
1: procedure BEST1(X,F)
2:     for i := 0 do#X
3:         j ∈ U(0, N)
4:         k ∈ U(0, N)
5:         V_i := {X_best + F * (X_j − X_k)}
6:     end for
7:     return V
8: end procedure
```

Algoritmo 2: DE/best/1

- DE/current-to-best/1 (Jeyakumar y C.Shanmugavelayutham 2011)

  En este caso, la mutación se aplica sobre cada individuo de la población. El objetivo de esta mutación es dirigir el individuo hacia la mejor solución X' hallada hasta el momento. Por tanto, tomaremos X' y además 2 elementos aleatorios, X_1 y X_2 , como en la mutación Best. El procedimiento será parecido al anterior, solo que en este caso tendremos dos vectores diferencia. El primero se realiza con el *target* vector y con X'; el segundo, con X_1 y X_2. Ambos vectores definirán el cambio de dirección del *target* vector.

$$V_i = X_i + F_1 \cdot (X_{best} - X_i) + F_2 \cdot (X_1 - X_2)$$

Algorithm 3 Operador de mutación DE/current-to-best/1

```
1: procedure CURRENT2BEST1(P,F_1, F_2)
2:     for i := 0 do#P
3:         j ∈ U(0, N)
4:         k ∈ U(0, N)
5:         V_i := {P_i + F_1 * (P_best − X_i) + F_2 * (X_j − X_k)}
6:     end for
7:     return V
8: end procedure
```

Algoritmo 3: DE/current-to-best/1

Recombinación

Junto con la mutación, esta operación es la responsable del balance exploración-explotación que se obtiene en la población durante el paso de las generaciones. El calculo se realiza sobre el *target vector* y su *donor vector correspondiente* obtenido durante la mutación. De aquí se obtendrá el denominado *trial vector*.

La recombinación consiste en mezclar al individuo original, perteneciente a la generación actual; y al mismo individuo después de haber pasado por el proceso de mutación. Esta combinación se hace escogiendo qué elemento escoger de cada individuo aleatoriamente.

$$U_i = \begin{cases} V_i \text{ si } U(0,1) < GR \\ X_i \text{ si } U(0,1) \geq GR \end{cases}$$

Siendo GR el parámetro que controla la probabilidad de recombinación.

Algorithm 4 Recombinar población para obtener la nueva generación

```
1: procedure RECOMBINATION(X, V, GR)
2:     U := ∅
3:     for i := 0 do#X
4:         if U(0, 1) ≤ GR then
5:             U_i := V_i
6:         else
7:             U_i := X_i
8:         end if
9:     end for
10:    return U
11: end procedure
```

Algoritmo 4: Recombinación

Selección

Este el último paso del algoritmo, en este punto tenemos que formar la siguiente generación de soluciones a partir de dos poblaciones diferentes, la formado por los *target vector* y la formada

por el *trial vector*. En otras palabras, en este paso se decide si a la siguiente generación pertenecerá el *target vector* o el *trial vector* correspondiente a cada solución. Esto se decide por el valor de la función objetivo.

$$X_i = \begin{cases} X_i \text{ si } f(x_i) \leq f(U_i) \\ U_i \text{ si } f(x_i) > f(U_i) \end{cases}$$

Algorithm 5 Selección

```
1: procedure SELECTION(X, U)
2:     for i := 0 do#X
3:         if f(X_i) ≤ f(U_i) then
4:             X_i := X_i
5:         else
6:             X_i := U_i
7:         end if
8:     end for
9:     return X
10: end procedure
```

Algoritmo 5: Selección

6.3 Algoritmos Genéticos, GA

Los GA son algoritmos de optimización inspirados en la selección natural. En este caso vamos a distinguir dos modelos: generacional y estacionario. Ambos modelos tienen un esquema común que detallaremos a continuación sin embargo difieren cómo se forman las distintas generaciones.

Estos algoritmos se dividen en cinco pasos.

Selección

Esta sección del algoritmo es la encargada de escoger cuáles serán los padres de la siguiente generación. Depende del modelo que se esté utilizando, generacional o estacionario, se seleccionará un número de padres. En el caso del modelo generacional, toda la población será escogida en algún momento mientras que en el modelo estacionario, solo dos padres serán elegidos.

Selección por Torneo

Se eligen aleatoriamente a *n* individuos de la población y se elige al que mejor valor de la función objetivo tenga.

Algorithm 6 Selección por Torneo

1: **procedure** TOURNAMENT(X, k)
2: $\triangleright$ $X = (X_{11}, ..., X_{1d}; X_{21}, ..., X_{2d}; X_{n1}, ..., X_{nd})$
3: $\{p_1, ..., p_k\} \in X$
4: $\{t_1, ..., t_k\} \subset X \setminus \{p\}$
5: $S := \{x \in p | \nexists y \in p, f(y) > f(x)\} \cup \{x \in t | \nexists y \in t, f(y) > f(x)\}$
6: **return** S
7: **end procedure**

Algoritmo 6: Selección por Torneo

Emparejamiento Variado Inverso

El primer padre se elige de manera aleatoria. Para seleccionar al otro padre, se toman de la población *n* individuos al azar y se escoge el que más lejano este del primero.

Algorithm 7 Reverse Mixed Pairing

1: **procedure** REVERSEMIXEDPAIRING(X, $nnam$)
2: $\triangleright$ $X = (X_{11}, ..., X_{1d}; X_{21}, ..., X_{2d}; X_{n1}, ..., X_{nd})$
3: $P_1 \in X$
4: $\{p_1 ... p_{nnam}\} \subset X \setminus \{P_1\}$
5: $S := \{x \in p | \nexists y, \| P_1 - y \|_2 > \| P_1 - x \|_2\} \cup \{P_1\}$
6: **return** S
7: **end procedure**

Algoritmo 7: Emparejamiento Variado Inverso

Orden Lineal

En función del valor de la función objetivo que tenga cada individuo, la población es ordenada y se asocia a cada uno una probabilidad de ser escogido dependiendo la posición que ocupe.

Algorithm 8 Linear

1: **procedure** LINEAR(X)
2: $\triangleright$ $X = (X_{11}, ..., X_{1d}; X_{21}, ..., X_{2d}; X_{n1}, ..., X_{nd})$
3: $Y \in U(f_{min}(X), f_{max}(X))$
4: $Pr(i) = P(Y \leq f(X_i))$
5: $S := \{X_i | \nexists j, Pr(j) > Pr(i)\} \cup \{X_i | \nexists j, Pr(j) > Pr(i)\}$
6: **return** S
7: **end procedure**

Algoritmo 8: Selección Lineal

Selección por Ruleta

Este procedimiento es similar a "Orden Lineal", sin embargo, en lugar de asociar la probabilidad en función del orden, se hace en función del valor de la función objetivo de cada individuo.

Algorithm 9 Selección por Ruleta

1: **procedure** ROULETTEWHEEL(X)
2: $Y \in U(f(X_0), \sum_i f(X_i))$
3: $Pr(i) = P(Y \leq \sum_i f(X_i))$
4: $S := \{X_i | \nexists j, Pr(j) > Pr(i)\} \cup \{X_i | \nexists j, Pr(j) > Pr(i)\}$
5: **return** S
6: **end procedure**

Algoritmo 9: Selección por Ruleta

Cruce

Otro de los pasos que se lleva a cabo en este algoritmo es el cruce cuyo objetivo es buscar la convergencia de la población. El cruce se aplica sobre los individuos seleccionados con una probabilidad *Pc,* tomando esta siempre el valor uno en caso de trabajar con un modelo estacionario. De esta operación saldrán nuevos individuos que acabarán incorporándose a la siguiente generación. Dos de los cruces más conocidos son:

- Cruce Media

Este cruce es también para poblaciones con representación real, en este caso, el individuo que hará de hijo y podrá optar a formar parte de la siguiente generación será el punto medio entre ambos padres.

Algorithm 10 Cruce Media

1: **procedure** MEANCR(C_1, C_2)
2: ▷ $C_1 = (C_{11}, ..., C_{1d})$
3: ▷ $C_2 = (C_{21}, ..., C_{2d})$
4: $H = \frac{1}{2} * (C_1 + C_2)$
5: **return** H
6: **end procedure**

Algoritmo 10: Cruce Media

- BLX-α

Este es un tipo de cruce para poblaciones con representación real. Este cruce define un intervalo a partir de los dos padres, elegidos anteriormente, en el cual aleatoriamente se escogen los valores que formarán a dos nuevos individuos que actuaran como hijos.

Parámetros:

  - α →Este parámetro controla la amplitud del intervalo en el que se escogen lo elementos de ambos hijos. Cuanto mayor sea el valor de α mayor será su amplitud.

Algorithm 11 Cruce $BLX-\alpha$

1: **procedure** BLXCR(C_1, C_2, α)
2: ▷ $C_1 = (C_{11}, ..., C_{1d})$
3: ▷ $C_2 = (C_{21}, ..., C_{2d})$
4: $H_k = (h_{k1}, .., h_{kn})$ ▷ $k = 1, 2$
5: $C_{max} = max\{C_{1i}, C_{2i}\}$
6: $C_{min} = min\{C_{1i}, C_{2i}\}$
7: $I = C_{max} - C_{min}$
8: $H_k \in [C_{min} - I * \alpha, C_{max} + I * \alpha]^d$ ▷ $k = 1, 2$
9: **return** H
10: **end procedure**

Algoritmo 11: Cruce BLX-α

Mutación

La mutación es un operador que nos permite aumentar la capacidad de exploración de nuestro algoritmo. Esto se consigue introduciendo una perturbación aleatoria sobre el individuo fruto del cruce anterior y, en el caso de no haberse producido, sobre los padres. Este operación se aplica con una probabilidad P_m sobre el individuo, debiendo ser esta muy baja.

Algorithm 12 Mutación

1: **procedure** MUTACION(H_1)
2: ▷ $H_1 = (C_{11}, ..., C_{1d})$
3: $H'_{1i} = H_{1i} + N(0, \sigma_i)$ ▷ $i = 1, .., d$
4: **return** H'
5: **end procedure**

Algoritmo 12: Mutación

6.4 Optimización de Forrajeo Bacteriano, BFO

El BFO [18] es un algoritmo bioinspirado que trata de imitar el movimiento que realizan las bacterias mientras buscan nutrientes, este movimiento se denomina chemotaxis. En este caso, el movimiento se realizará sobre el espacio de búsqueda de dimensión N con el objetivo último de hallar el óptimo global.

Este algoritmo, al igual que el proceso al que trata de imitar, se divide en 4 bloques. Estas fases al contrario que puede pasar en otros algoritmos, no son secuenciales si no que cada etapa abarca a las siguientes. Estas etapas son:

Chemotaxis

Simula el movimiento de una bacteria. Los dos movimientos posible son en linea recta o una rotación. Ambos movimiento son alternados durante el tiempo que dura viva la bacteria. Suponga-

mos que $\theta^i(j,k,l)$ se trata de la bacteria i en el j-esimo proceso de chemotaxis, k-esimo proceso de reproducción y l-eseimo proceso de dispersión y eliminación. Tenemos entonces que:

$$\theta^i(j+1,k,l)=\theta^i(j,k,l)+C(i)\frac{\Delta i}{\sqrt{\Delta^T i\,\Delta i}}$$

Siendo C(i) el espacio recorrido en una dirección aleatoria tomada por la rotación y Δi un vector aleatorio cuyos valores se encuentran en el intervalo [-1, 1]

Swarming

Movimiento que ayuda a propagarse colectivamente como patrones concéntricos de enjambres con alta densidad bacteriana mientras se mueven hacia el optimo en el gradiente de la función.

Reproduction

La bacteria con menos nivel de salud probablemente muera, por lo que el resto de bacterias más saludables se duplicarán permitiendo así mantener el tamaño de población constante.

Elimination and Dispersal

En ocasiones, espacios con alta temperatura pueden matar a bacterias que están en un lugar con una elevada concentración de nutrientes (o en nuestro caso una región del espacio prometedora). En ocasiones lo que ocurre es que el grupo de bacterias que se encuentra en esa región se dispersa. Esta consiste en, con una probabilidad muy baja, eliminar aleatoriamente a parte de la población de bacterias. Estas serán reemplazadas por vectores inicializados al azar.

Algorithm 13 BFO

```
1: procedure BFO(X)
2:                                   ▷ X = (X_11, ..., X_1d; X_21, ..., X_2d; X_n1, ..., X_nd)
3:     for i := 1 doN_ed
4:         for j := 1 doN_re
5:             for k := 1 doN_c
6:                 for n := 1 doS
7:                     Jlast := J(n)
8:                     Generar ángulo de caída para la bacteria
9:                     Actualizar la posición de la n-enesima bacteria
10:                    Recalcular J(n)
11:                    m := 0
12:                    while m < N_s do
13:                        if J(n) < Jlast then
14:                            Jlast = J(n)
15:                            Recalcular J(n)
16:                            m := m + 1
17:                        else
18:                            m := N_s
19:                        end if
20:                    end while
21:                end for
22:                Actualizar el mejor valor alcanzado
23:            end for
24:            Ordenar la población según su valor J
25:        end for
26:        for m := 1 do S/2
27:            X(m + S/2) = X(m)
28:        end for
29:        for l := 1 doS
30:            if rand < P_e then
31:                Mover X(l) a una posición aleatoria
32:            end if
33:        end for
34:    end for
35:    return X_best
36: end procedure
```

Algoritmo 13: BFO

6.5 Optimización de Enjambre de Partículas, PSO

Este algoritmo [20] se inspira en cómo se comportan las bandadas de pájaros. Los pájaros son capaces de volar de manera coordinada, cambiar de dirección en un momento dado, incluso replegarse o agruparse. Este hecho está muy relacionado con las dinámicas de grupos o comportamiento social que es el pilar fundamental de este tipo de algoritmos.

Calculo de Velocidad

Una población de pájaros es aleatoriamente inicializada a lo largo del espacio de búsqueda. A cada uno de ellos se le otorga una velocidad en cada una de las dimensiones. En cada iteración del algoritmo las velocidades de cada pájaro variaran según sus vecinos más cercanos.

Además, para que el proceso se asemeje más a la realidad, se suma una variable aleatoria a las velocidades.

Calculo de Posición

Como se ha explicado en el punto anterior, la velocidad de cada la velocidad pájaro se recalcula en cada iteración del algoritmo. En consecuencia, la posición de cada pájaro se cambia en cada iteración.

Vector de campo

Se añade un elemento sobre el cual los pájaros tienden a ser atraídos. En la vida real, esto podría equivaler al pájaro que lidera. Este elemento en cuestión se corresponderá con la mejor posición obtenida por cada agente en el espacio de búsqueda, entendiéndose como mejor posición como aquella con mejor valor de la función objetivo.

PSO Local

En el momento de calcular la velocidad que toma cada partícula en cada iteración, si se adopta el modelo local, se realiza de la siguiente manera.

$$V = V + \phi_1 * p * (pbest - current)$$

Siendo pbest, la mejor posición encontrada por cada partícula.

Algorithm 14 PSO Local

```
1: procedure PSOL(X)
2:                                  ▷ X = (X_11, ..., X_1d; X_21, ..., X_2d; X_n1, ..., X_nd)
3:                                  ▷ V = (V_11, ..., V_1d; V_21, ..., V_2d; V_n1, ..., V_nd)
4:     t := 0
5:     inicializar X y V
6:     while no se cumple la condicion de parada do
7:         t := t + 1
8:         for i := 1 don
9:             Evaluar f(X_i)
10:            if f(x) > f(pBest) then
11:                pBest_i := X_i
12:                f(pBest_i) := f(X_i)
13:            end if
14:        end for
15:        for i := 1 don
16:            Escoger IBest_i, la partícula con mejor fitness del entorno de X_i
17:            Calcular V_i, la velocidad de X_i, de acuerdo a pBest_i y IBest_i
18:            Calcular la nueva posición X_i, de acuerdo a X_i y V_i
19:        end for
20:    end while
21:    return X_best
22: end procedure
```

Algoritmo 14: PSO Local

PSO Global

En el momento de calcular la velocidad que toma cada partícula en cada iteración, si se adopta el modelo global, se realiza de la siguiente manera.

$$V = V + \phi_1 * (pbest - current) + \phi_2 * (pbest[global] - current)$$

Siendo pbest, la mejor posición encontrada por cada partícula y pbest[global] la mejor posición encontrada por todo el enjambre.

Algorithm 15 PSO Global

```
1: procedure PSOG(X)
2:                                  ▷ X = (X_11, ..., X_1d; X_21, ..., X_2d; X_n1, ..., X_nd)
3:                                  ▷ V = (V_11, ..., V_1d; V_21, ..., V_2d; V_n1, ..., V_nd)
4:     t := 0
5:     inicializar X y V
6:     while no se cumple la condicion de parada do
7:         t := t + 1
8:         for i := 1 don
9:             Evaluar f(X_i)
10:            if f(x) > f(pBest) then
11:                pBest_i := X_i
12:                f(pBest_i) := f(X_i)
13:            end if
14:            if f(pBest) > f(gBest) then
15:                gBest_i := pBest_i
16:                f(gBest_i) := f(pBest_i)
17:            end if
18:        end for
19:        for i := 1 don
20:            Calcular V_i, la velocidad de X_i, de acuerdo a pBest_i y gBest_i
21:            Calcular la nueva posición X_i, de acuerdo a X_i y V_i
22:        end for
23:    end while
24:    return X_best
25: end procedure
```

Algoritmo 15: PSO Global

6.6 Gravitational Search Algorithm

Para poder entender mejor el funcionamiento de GSA [24], antes debemos comprender los pilares dónde se fundamenta este algoritmo. Los pilares fundamentales de GSA son:

- Algoritmos de Enjambre de Partículas, PSO [20]
- Ley de Gravitación Universal (Física)
- Ley Fundamental de la Dinámica (Física)
- Movimiento Rectilíneo Uniformemente Acelerado, MRUA (Física)

Ley de Gravitación Universal

La Ley de Gravitación Universal, desarrollada por Isaac Newton, establece la fuerza con la que se atraen dos cuerpos por el mero hecho de tener masa.

Esta ley dice que la fuerza de atracción de dos cuerpos es directamente proporcional al producto de sus masas e inversamente proporcional al cuadrado de la distancia que las separa.

$$F = G\frac{M*m}{R^2}$$

Siendo G la constante de gravitación.

Haciendo una analogía con el GSA, las masas en la Ley de Gravitación Universal equivaldrían a las soluciones del algoritmo.

Segunda Ley de Newton

"El cambio de movimiento es directamente proporcional a la fuerza motriz impresa y ocurre según la línea recta a lo largo de la cual aquella fuerza se imprime."

A partir de esta ley se puede deducir la llamada ecuación fundamental de la dinámica:

$$F = m \cdot a$$

La fuerza neta que actúa sobre un cuerpo, también llamada fuerza resultante, es el vector suma de todas las fuerzas que sobre él actúan.

$$\sum F = m \cdot a$$

Movimiento Rectilíneo Uniformemente Acelerado

En mecánica clásica el movimiento rectilíneo uniformemente acelerado (MRUA) presenta dos características fundamentales:

1. La trayectoria es rectilínea
2. La aceleración sobre la partícula son constantes.

$$F = m \cdot a$$

Por lo tanto, esto determina que:

1. La velocidad varía linealmente respecto del tiempo.
$$v(t) = a \cdot t + v_0$$
2. La posición varía según una relación cuadrática respecto del tiempo.

$$x(t) = \frac{1}{2} \cdot a \cdot t^2 + v_0 \cdot t + x_0$$

Cuando se explicó el PSO, se dijo que las soluciones se movían siguiendo a las mejores soluciones. En el GSA, el movimiento es marcada por esta ecuación (Esto no es del todo cierto, se explicará más adelante).

Análisis de GSA

Conociendo los fundamentos del algoritmo GSA estamos en condiciones de analizar cada una de las partes del algoritmo. En concreto nos centraremos en analizar la utilidad de cada uno de los parámetros para posteriormente ser capaces de adecuar el algoritmo a cada problema.

Anteriormente hemos explicado que el algoritmo GSA está basado en los algoritmos PSO. A continuación vamos a comentar la analogía entre GSA y PSO.

La población de soluciones de PSO, en GSA se materializa como un conjunto de cuerpos celestes cuya masa depende del valor de la función. Como hemos dicho en la Ley de Gravitación Universal, la masa es muy importante en la dinámica.

El movimiento que realizan las soluciones de GSA, se hacen según las ecuaciones del Movimiento Rectilíneo Uniformemente Acelerado. Junto con la Ley de Gravitación Universal y la Segunda Ley de Newton, somos capaces de definir el movimiento que realiza un cuerpo.

La siguiente imagen presenta el pseudocódigo del Algoritmo de Búsqueda Gravitacional.

Algorithm 16 Busqueda Gravitacional

```
1: procedure GSA
2:     inicializar X                ▷ X = (X_11, ..., X_1d; X_21, ..., X_2d; X_n1, ..., X_nd)
3:     inicializar V                ▷ V = (V_11, ..., V_1d; V_21, ..., V_2d; V_n1, ..., V_nd)
4:     for i := 1 domax_iter
5:         t := t + 1
6:         fitness := ∪_i f(X_i)
7:         best := min(fitness)
8:         if i = 1 then
9:             Fbest := best
10:            Lbest := x tq. f(x) = best
11:        else if best ≤ Fbest then
12:            Fbest := best
13:            Lbest := x tq. f(x) = best
14:        end if
15:        M := massCalculation(fitness)
16:        G := GConstant(i, max_iter)
17:        a := Gfield(M, X, G, Rnorm, Rpower, i, max_iter)
18:        [X, V] := move(X, a, V, fitness)
19:    end for
20:    return X_best
21: end procedure
```

Algoritmo 16: Búsqueda Gravitacional

En las 4 primeras líneas de ese pseudocódigo nos encontramos los 4 primeros parámetros que deberemos fijar para adaptar el algoritmo al problema.

- N ← Es el tamaño de la población de soluciones, en este caso habrá 50 potenciales soluciones. El resultado del algoritmo será la mejor de ellas.
- Iteration ← Es un parámetro fijado en la propia competición, se especifica que debe evaluar la función objetivo 1000**dim* veces (entendemos *dim* como el tamaño del vector que se le pasa a la función *f*)
- Rpower ← Denota a que potencia es elevada la distancia entre las masas en la Ley de Gravitación Universal. Cuanto más pequeño sea este valor, mayor será la fuerza de atracción y por tanto, mayor distancia se recorrerá en cada movimiento. Este parámetro esta ligado a la exploración del algoritmo.
- Rnorm ← La distancia entre dos masas es calculada mediante la norma de la diferencia de las posiciones de ambas masas. Cuanto más grande es el valor de Rnorm, más pequeño es el resultado de la norma. Al igual que Rpower, este parámetro también esta ligado a la exploración.

$$||x||_{Rnorm} = \sqrt[Rnorm]{|x_1|^{Rnorm} + |x_2|^{Rnorm} + ... + |x_n|^{Rnorm}}$$

En las siguientes líneas, evaluamos la población y nos quedamos con la mejor solución. *Fbest* es el valor de la mejor evaluación (en términos matemáticos, el valor de la coordenada dependiente) y Lbest es el punto donde *f* toma el valor Fbest (en términos matemáticos, los valores de las coordenadas independientes).

Por último, desde la linea 18 hasta la 21 están las funciones que componen la parte física del algoritmo. Antes de mostrar el pseudocódigo de las funciones explicaremos cual es el papel de cada función:

- massCalculation ← A partir de la evaluación de una solución, le asigna una masa a esta. Esto es para posteriormente poder aplicar las Leyes de Newton.
- Gconstant ← En la Ley de Gravitación Universal, se mencionó una constante G que colaboraba en el cálculo de la fuerza de atracción. Pues esta es la función que se encarga de calcular esta constante. Aunque definamos G como una constante y la estemos tratando como tal, en este algoritmo, la constante G depende de la iteración donde nos encontremos.
- Gfield ← Esta función, se encarga de aplicar la Ley de Gravitación Universal y la Segunda ley de Newton para calcular la aceleración que adquiere cada cuerpo (solución) en cada iteración.
- Move ← Aplica las ecuaciones del Movimiento Rectilíneo Uniformemente Acelerado para adquirir la nueva velocidad y posición de cada cuerpo en el espacio.

Cálculo de la Constante Gravitacional

En este algoritmo, la constante gravitacional cambia a lo largo del tiempo, es decir, en cada iteración tiene un valor diferente. Esta constante se calcula con una fórmula que en Enfriamiento Simulado llamamos Esquema de Enfriamiento.

Al tener un número de iteraciones concretas, el esquema que se ha utilizado para calcular G es uno que permite que G alcance su valor máximo en la primera iteración y su valor mínimo en la iteración N.

$$G = G_0 * e^{\frac{-\alpha * iteracion}{max\,it}}$$

Este esquema tiene dos parámetros:

- α ← Este parámetro controla la velocidad a la que decrece G. Tenemos que tener en cuenta que es una función exponencial, por lo que su decrecimiento será bastante rápido.
- G_0 ← Es el valor máximo alcanzable por G, por lo que G tiene valores en el intervalo $[0, G_0]$.

Como todos los parámetros comentados anteriormente, estos también tienen influencia en el nivel de exploración del algoritmo. Aunque la parte importante de la exploración en este apartado no son los parámetros, si no el esquema escogido para calcular G. Los parámetros dependen del esquema que elijamos.

Algorithm 17 GConstant

```
1: procedure GCONSTANT(iteration, maz_iter)
2:     α := 20
3:     G_0 := 100
4:     G := G_0 * e^(-α iteration / max_iter)
5:     return G
6: end procedure
```

Algoritmo 17: GConstant

Aunque el código sea sencillo, esta parte del algoritmo es fundamental, ya que la fuerza de atracción de dos cuerpos, también es directamente proporcional a G. Por lo que cuanto mayor es G, mayor será el movimiento que realice el cuerpo (solución).

Calculo de la Masa

Como hemos explicado antes, existe una parte del algoritmo que tiene como objetivo asignarle a cada cuerpo (solución) una masa. La encargada de esto es la función *massCalculation*, que se muestra a continuación:

Algorithm 18 massCalculation

```
1:  procedure MASSCALCULATION(fit)
2:      inicializar M                                   ▷ M := 0_n
3:      Fmax := max(fit)
4:      Fmin := min(fit)
5:      Fmean = mean(fit)
6:      if Fmax == Fmax then
7:          M := 1_n
8:      else
9:          for i := 1 don
10:             M_i := (fit − worst) / (best − worst)
11:         end for
12:     end if
13:     return (∑_i M_i) * M
14: end procedure
```

Algoritmo 18: Cálculo de Masa

En primer lugar, las evaluaciones normalizadas (en el intervalo [0,1]), se introducen en el conjunto M. Por ultimo cada elemento de M es dividido entre la sumatoria total del conjunto.

Recordemos que en la Ley de Gravitación Universal, se específica que la fuerza de atracción de dos cuerpos es directamente proporcional al producto de sus masas. Por lo que la forma de calcular las masas también esta ligada a la exploración.

6.7 Firefly Algorithm

Este algoritmo fue desarrollado por Xin-She Yang a finales del año 2007. Esta inspirado en el comportamiento de las luciérnagas y el uso de su linterna.

La luz producida por las luciérnagas es debido a un proceso bioluminiscente y su uso aun está en estudio. Dos de sus funciones principales son la atracción de compañeros o posibles parejas. Algunas especies lo usan p ara organizarse y actuar en sociedad; esto es en lo que se basa el algoritmo que se describirá a continuación.

Para simplificar el funcionamiento del algoritmo, partiremos de una serie de suposiciones:

- Todas las luciérnagas serán atraídas por las demás independientemente de su sexo.
- El nivel de atracción es proporcional al brillo emitido por la luciérnaga Dadas dos luciérnagas que estén emitiendo luz, la que menos brilla es la que se moverá.
- La intensidad el brillo esta relacionado con el valor de la función objetivo.

Cálculo de la Intensidad/Atracción

Para calcular la intensidad recibida por una luciérnaga a través de un medio dado, podemos utilizar el siguiente calculo:

$$I = I_0 e^{-\gamma r^2}$$

Como hemos dicho antes, el nivel de atracción está estrechamente relacionado con el brillo, por lo que este se calcula de la misma forma:

$$\beta = \beta_0 e^{-\gamma r^2}$$

Siendo β_0 el valor de la función objetivo.

Algorithm 19 Firefly Algorithm

```
1:  procedure FIREFLY(fit)
2:      inicializar X                ▷ X = (X_11, ..., X_1d; X_21, ..., X_2d; X_n1, ..., X_nd)
3:      Calcular la intensidad I_i en X_i determinada por f(x_i)
4:      while t < maxGen do
5:          for i := 0 do
6:              for j := 0 don
7:                  if I_i < I_j then
8:                      Mover X_i hacia X_j
9:                  end if
10:                 Calcular atracción en función de r via [−γr^2]
11:                 Evaluar las soluciones y actualizar la intensidad
12:             end for
13:         end for
14:     end while
15:     return X
16: end procedure
```

Algoritmo 19: FA

7 Diseño

A lo largo de este apartado se analizaran las decisiones tomadas en la fase de diseño.

7.1 Interfaz de Usuario

En un primer momento, por cada algoritmo existiría una función especifica muy parecidas entre sí. La diferencia serían los parámetros específicos que necesitase cada uno. Esto podría ser válido si nos limitáramos a un pequeño paquete como este que tiene 8 algoritmos diferentes, pero como en todo proyecto, existe la posibilidad de que el paquete se amplie en un futuro, por lo que la interfaz se tornaría muy complicada.

Para solucionar este problema se crea una única función, cuyos parámetros son los mismos que recibían las funciones del diseño anterior (recordemos que la única diferencia eran los parámetros específicos) con la diferencia de que además también recibirá una estructura que define el algoritmo que será llamado junto con sus parámetros específicos.

Esta solución permite que añadir un algoritmo más sea casi transparente para el usuario. Si el usuario ya está familiarizado con el paquete, la única diferencia es que existe una nueva estructura correspondiente a un nuevo algoritmo.

7.2 Diagramas de Diseño

Por cada tipo de algoritmo se diseña una estructura con sendos parámetros específicos. Dichas estructuras serán las que permitan al usuario distinguir qué algoritmo utilizar.

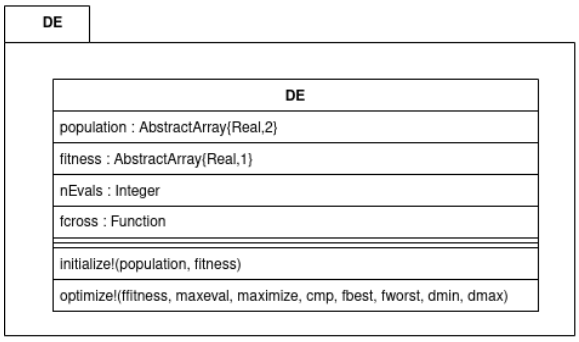


Figura 20: DE

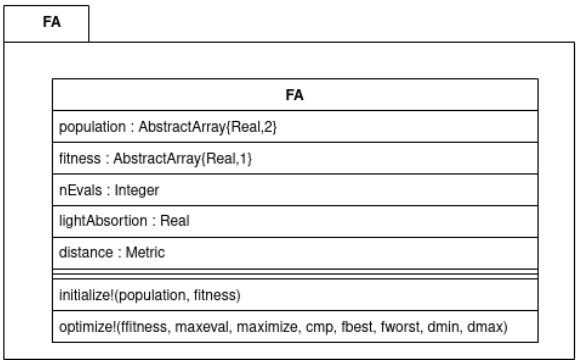


Figura 21: FA

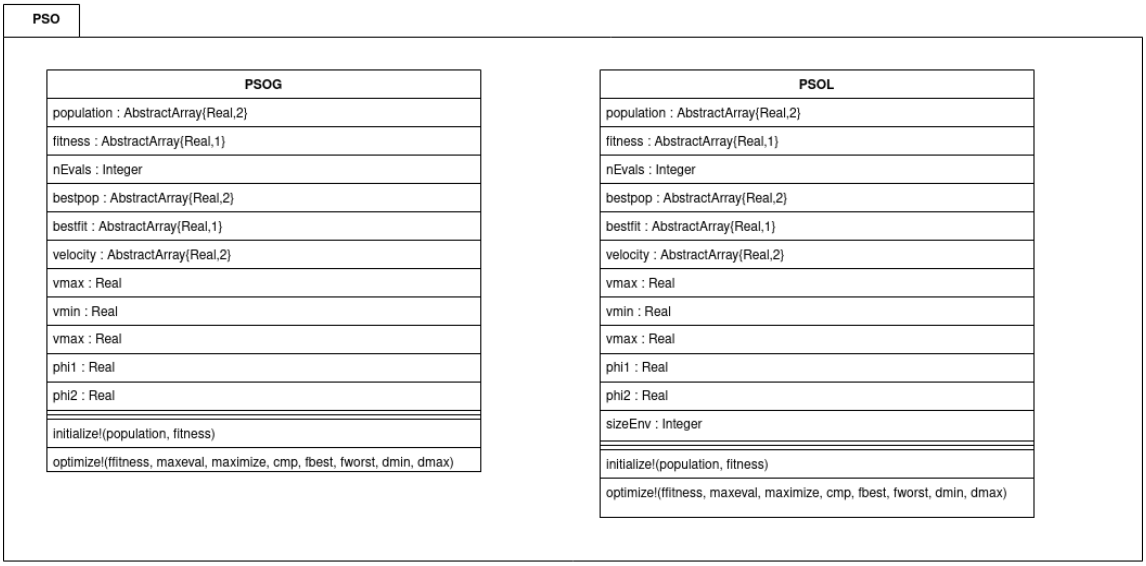


Figura 22: Diseño PSO

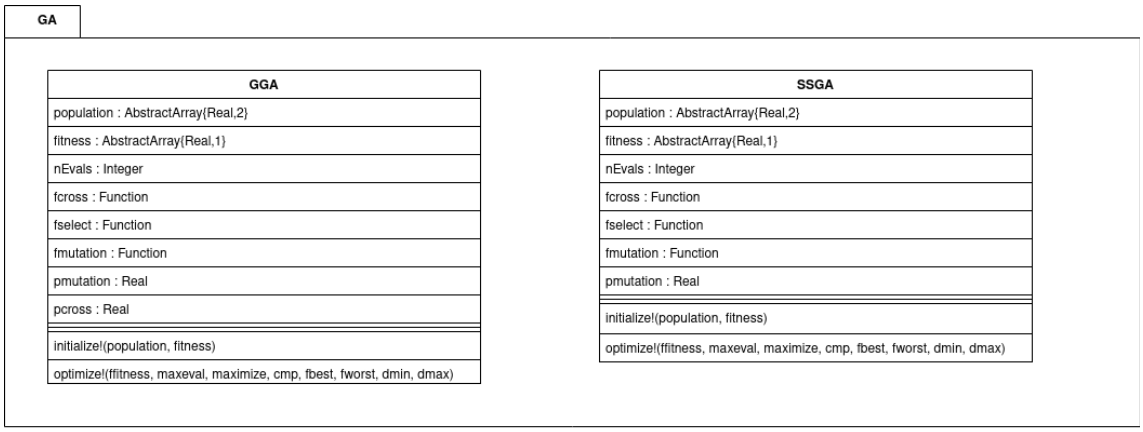


Figura 23: Diseño GA

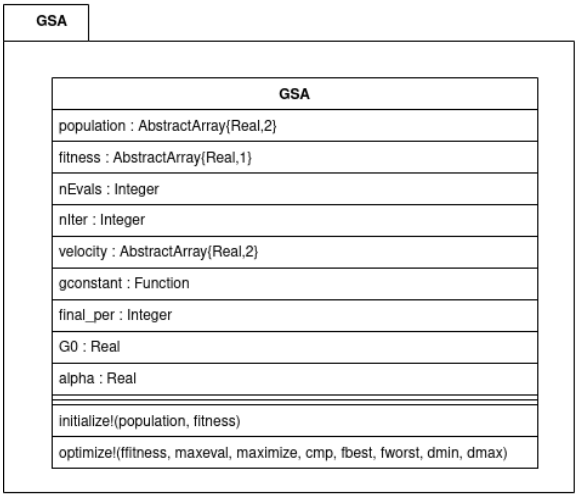


Figura 24: GSA

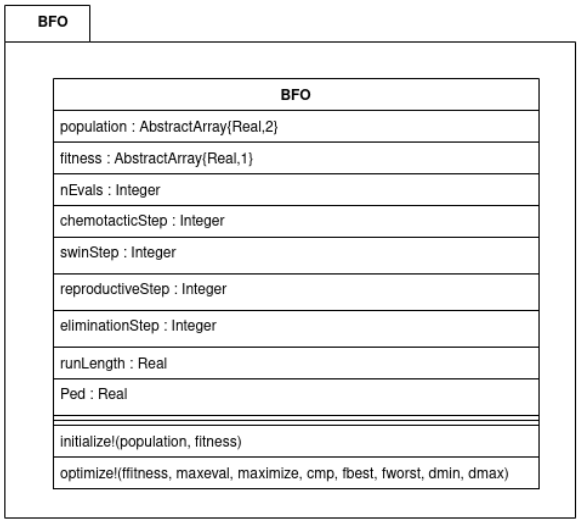


Figura 25: BFO

Se observa que todos las estructuras tienen en común dos métodos: *initialize* y *optimize!*. Estos son llamados desde la función *optimize* accesible por el usuario. Esta es la función en la que el usuario configurará a su gusto la optimización permitiendo lo siguiente:

- Elegir la función a optimizar
- El tamaño de la población
- Elegir minimizar o maximizar la función
- Número máximo de evaluaciones de la función objetivo
- El algoritmo a utilizar
- El dominio de la función objetivo

Esta función funciona como nexo de unión entre los distintos algoritmos desarrollados. Además facilita la adaptación del usuario a futuras ampliaciones del paquete debido a que es el único metodo que debe conocer el usuario. El el caso de añadir nuevos algoritmos el usuario debería conocer los parámetros especificos de este, pero en ningún caso, debería aprender nuevas funciones .

Esto es así porque el algoritmo a ejecutar es un parámetro más de la función optimize. Julia cuenta con Multi Dispatch, lo que permite alterar el funcionamiento de una función en tiempo de ejecución, en este caso, mediante un parámetro.

A continuación, se presenta la documentación detallada para el correcto uso del paquete.

EvolutionaryAlgs.jl

Documentation for EvolutionaryAlgs.jl

EvolutionaryAlgs.optimize — Function

optimize(ffitness, maxeval; maximize, population, fitness, popsize, ndim, dmin, dmax, alg, fcallback)

Optimizacion

ffitness: Función objetivo

maxeval: Nº maximo de iteraciones

maximize : ¿Maximización?

population: Poblacion previa (opcional)

fitness: Valor de fitness previa (opcional)

popsize: Tamaño de población

ndim: Dimensión del problema

dmin: Cota inferior del dominio

dmax: Cota superior del dominio

alg: Algoritmo escogido para la optimización

fcallback: Función llamada al comienzo de cada iteración

Algoritmos Genéticos

EvolutionaryAlgs.GGA — Function

GGA(; fcross, fselect, fmutation, pmutation, pcross)

Algoritmo Génetico Generacional

fcross: Operador de Cruce

fselect: Operador de Selección

fmutation: Operador de Mutacion

pmutation: Probilidad de Mutacion

pcross: Probabilidad de Cruce

EvolutionaryAlgs.SSGA — Function

SSGA(; fcross, fselect, fmutation, pmutation)

Algoritmo Génetico Estacionario

fcross: Operador de Cruce

fselect: Operador de Selección

fmutation: Operador de Mutación

pmutation: Probabilidad de Mutación

Operadores de Selección

EvolutionaryAlgs.reverse_mixed_pairing_selection — Function

reverse_mixed_pairing_selection(population, fitness; distance, nnam)

Se coge una solución aleatoria, y a partir de otro grupo de nnam soluciones se escoge la solucion más lejana junto con la primera solución escogida.

population: Array de soluciones

distance: Metodo de calculo de distancia

nnam: Numero de sluciones escogidas

EvolutionaryAlgs.linear_selection — Function

linear_selection(population, fitness)

Para cada gen, se asigna una probabilidad de ser escogido en función de su fitness

fitness: Array con valores de fitness

EvolutionaryAlgs.roulette_wheel_selection — Function

roulette_wheel_selection(population, fitness)

Para cada gen, se asigna una probabilidad acorde con la siguiente formula:

$$\frac{ffitness(gen)}{\Sigma_{i=0}^{n} ffitness(gen_i)}$$

fitnes : Array con valores de fitness

EvolutionaryAlgs.tournament_selection — Function

tournament_selection(population, fitness; k)

Coge al azar *k* soluciones, y escoge el que mejor fitness tenga

fitnes : Array con valores fitness

k : Numero de competidores

Operadores de Cruce

EvolutionaryAlgs.blx_cross — Function

blx_cross(p1, p2; alpha)

El hijo se genera a partir de un intervalo aleatorio formado a partir de ambos padres

p1: Padre 1

p1: Padre 2

EvolutionaryAlgs.arithmetic_cross — Function

arithmetic_cross(p1, p2)

El hijo se calcula con la media aritmetica de los dos padres

p1: Padre 1

p1: Padre 2

Operadores de Mutación

EvolutionaryAlgs.perm_mutation! — Function

perm_mutation!(h)

Intercambia la posicion de dos genes

h: Cromosoma que muta

EvolutionaryAlgs.norm_mutation! — Function

norm_mutation!(h; sigma)

Suma un valor en N(0, σ)

h: Cromosoma que muta

sigma ->Desviación estandar

Differential Evolution

EvolutionaryAlgs.DE — Function

DE(; fcross)

fcross: Operador de Cruce

Operadores de Cruce

EvolutionaryAlgs.bestCross — Function

bestCross(population, fitness, current, fbest; f, p)

Se obtiene una solución desde la mejor solucion actual hasta dos soluciones aleatorias

population : Array de soluciones

fitness: Array con valores fitness

current: Solución actual

fbest: Funcion para obtener el mejor individuo

f: Fuerza de cruce

p: Probabilidad de cruce

EvolutionaryAlgs.current2bestCross — Function

current2bestCross(population, fitness, current, fbest; f, p)

Genera una solución partiendo de la actual hasta la mejor solución

population : Array de soluciones

fitness : Array con valores fitness

current : Solución actual

fbest : Función para obtener el mejor individuo

f : Fuerza de cruce

p : Probabilidad de cruce

Enjambre de Partículas

EvolutionaryAlgs.PSOG — Function

PSOG(; vmax, vmin, phi1, phi2)

Algoritmo de Enjambres de Partículas Global

vmin: Mínima velocidad de partícula

vmax: Máxima velocidad de partícula

phi1: Influencia del óptimo real

phi2: Influencia del óptimo global

EvolutionaryAlgs.PSOL — Function

PSOL(; vmax, vmin, phi1, phi2, sizeEnv)

Algoritmo de Enjambre de Partículas Local

vmin: Mínima velocidad de partículas

vmax: Máxima velocidad de partícula

phi1: Influencia del óptimo real

phi2: Influencia del óptimo global

sizeEnv: Tamaño de entorno de vecindad

Forrajeo Bacteriano

EvolutionaryAlgs.BFO — Function

BFO(; chemotacticStep, swimStep, reproductiveStep, eliminationStep, Ped, runLength)

Algoritmo de Forrajeo Bacteriano

chemotacticStep: Número de movimientos

swimStep : Número de avances en cada movimiento

reproductiveStep : Número de reproducciones de por cada generación

Ped : Probabilidad de dispersión - eliminación

runLength : Distancia recorrida por cada bacteria

Luciernagas

EvolutionaryAlgs.FA — Function

FA(; lightAbsortion, distance)

lightAbsortion : Nivel de absorcion de luz

distance : Método de cálculo de distancia

Busqueda Gravitacional

EvolutionaryAlgs.GSA — Function

GSA(; g_constant, final_per, G0, alpha)

g_constant : Esquema de cálculo de la constante gravitacional

final_per : Número de cuerpos celestes que ejercen fuerza en la ultima iteracion

G0 : Valor inicial de la constante gravitacional

alpha : Valor que marca como decrece la constante gravitacional

En cuanto a la función principal del paquete, *optimize*, se presenta a continuación un diagrama de flujo en el que se pueden distinguir *grosso modo* las diferentes etapas de dicha función.

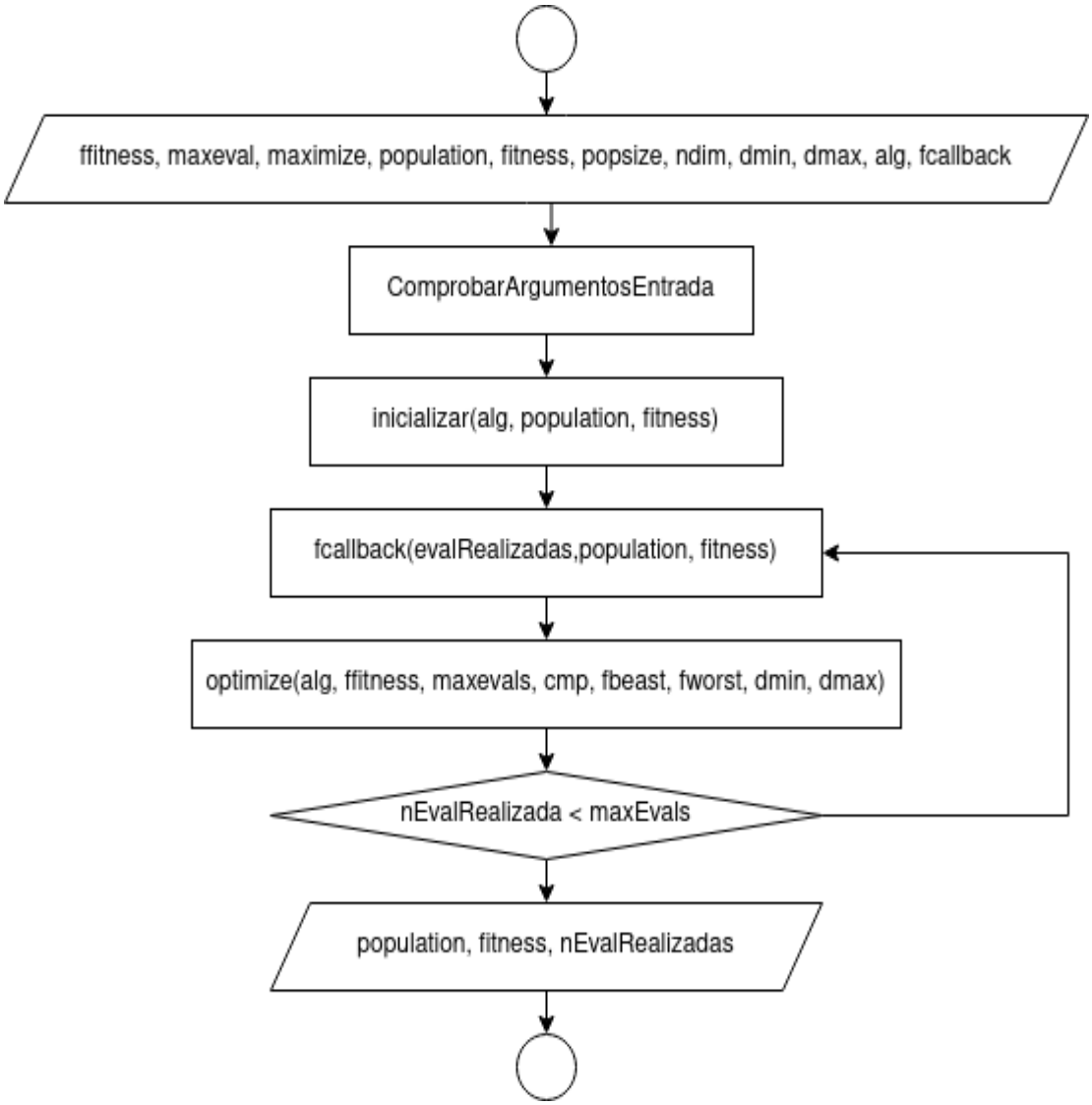


Figura 26: Diagrama de Flujo de la función Optimize

8. Experimentación

En este apartado se va a realizar dos comparativas: una de rendimiento entre los lenguajes Julia y Python; y otra para comparar los resultados obtenidos por los diferentes algoritmos. Para la primera de ellas vamos a analizar los tiempos de algunos de los algoritmos mostrados anteriormente implementados en ambos lenguajes. Los algoritmos de Python se tomaran del paquete *inspyred* [28] y *GSA* [29]

Para realizar este análisis, se han recurrido a funciones del "Congress on Evolutionary Computation 2014" [26] para utilizarlo como estándar. En concreto vamos a utilizar las diez primeras funciones con dimensiones 2, 10, 30, 50.

Para cada dimensión se tienen unos parámetros de ejecución distintos. El número máximo de evaluaciones se obtiene multiplicando el número de dimensión por 1000, es decir, para dimensión 2 el máximo de evaluaciones es 2000.

8.1 Análisis de Tiempo de EvolutionaryAlgs

En este apartado se analizará el tiempo invertido con cada algoritmo para cada una de las 16 primeras funciones del benchmark. A continuación se inserta una gráfica en escala logarítmica el tiempo invertido, en promedio, por cada algoritmo en cada dimensión.

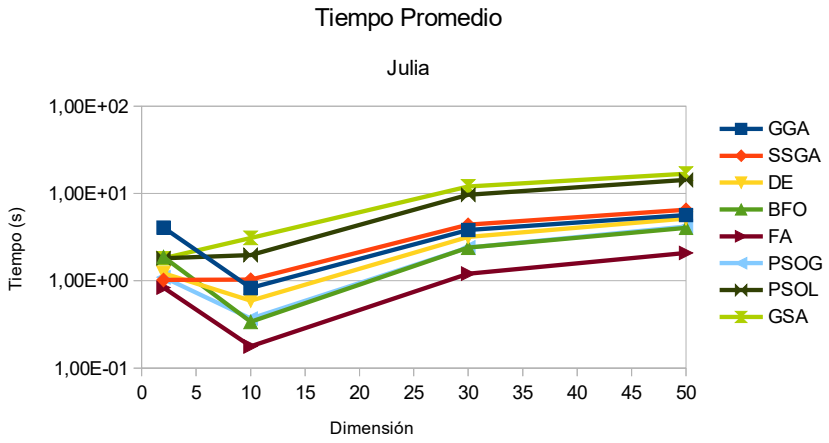


Figura 27: Tiempo Promedio (Julia)

Viendo la gráfica, podemos cómo, evidentemente, a medida que aumentamos el tamaño del problema, el tiempo invertido en la optimización aumenta; a excepción del paso desde la dimensión 2 a la 10, cuyos tiempos disminuyen en la mayoría de los casos, no obstante, debido a que dimensión 2 es un tamaño de problema demasiado pequeño, despreciaremos esa parte de la gráfica.

En cuanto a los diferentes algoritmos, podemos ver que FA es el algoritmo más rápido en cada una de las dimensiones. Los 2 siguientes mejores algoritmos son PSOG y BFO, que gozan de

unos tiempos prácticamente idénticos.

GGA, SSGA y DE; quedarían en la mitad de la tabla, con unos tiempos un poco por encima de los algoritmos anteriores. Los dos peores algoritmos, con gran diferencia, son PSOL y GSA.

	FUNCIONES							
	1	2	3	4	5	6	7	8
GGA	6,07E+00	6,29E+00	5,53E+00	5,51E+00	3,43E+00	3,57E+00	3,42E+00	3,40E+00
SSGA	1,49E+00	1,43E+00	1,32E+00	1,32E+00	8,83E-01	1,04E+00	8,81E-01	8,78E-01
DE	1,82E+00	1,78E+00	1,60E+00	1,60E+00	1,05E+00	1,20E+00	1,04E+00	1,04E+00
BFO	2,79E+00	2,77E+00	2,49E+00	2,50E+00	1,58E+00	1,71E+00	1,58E+00	1,58E+00
FA	**1,26E+00**	**1,21E+00**	**1,05E+00**	**1,04E+00**	**7,13E-01**	**7,78E-01**	**7,14E-01**	**7,10E-01**
PSOG	1,57E+00	1,56E+00	1,40E+00	1,39E+00	9,52E-01	1,13E+00	9,40E-01	9,08E-01
PSOL	2,68E+00	2,61E+00	2,48E+00	2,51E+00	1,52E+00	1,66E+00	1,52E+00	1,57E+00
GSA	2,67E+00	2,56E+00	2,34E+00	2,33E+00	1,56E+00	1,72E+00	1,54E+00	1,58E+00
Dimension: 2 Rango [-100.0 100.0] MaxEvals 20000								

Tabla 1: Tiempos Julia F1-F8 / D2

	FUNCIONES							
	1	2	3	4	5	6	7	8
GGA	7,64E-01	7,32E-01	6,95E-01	7,00E-01	5,44E-01	3,75E+00	5,57E-01	5,33E-01
SSGA	1,02E+00	9,99E-01	9,23E-01	9,68E-01	7,20E-01	4,07E+00	7,27E-01	7,07E-01
DE	5,07E-01	4,62E-01	4,44E-01	4,63E-01	3,22E-01	3,31E+00	3,29E-01	3,22E-01
BFO	1,09E-01	7,30E-02	7,66E-02	7,87E-02	8,81E-02	3,13E+00	1,03E-01	7,13E-02
FA	**7,03E-02**	**5,58E-02**	**5,94E-02**	**5,64E-02**	**5,08E-02**	**1,60E+00**	**5,66E-02**	**4,61E-02**
PSOG	1,46E-01	1,10E-01	1,06E-01	1,09E-01	1,06E-01	3,31E+00	1,19E-01	9,26E-02
PSOL	2,13E+00	2,06E+00	2,04E+00	1,98E+00	1,61E+00	4,74E+00	1,64E+00	1,61E+00
GSA	3,61E+00	3,41E+00	3,24E+00	3,29E+00	2,63E+00	6,19E+00	2,62E+00	2,66E+00
Dimension: 10 Rango [-100.0 100.0] MaxEvals 100000								

Tabla 2: Tiempos Julia F1-F8 / D10

	FUNCIONES							
	1	2	3	4	5	6	7	8
GGA	2,67E+00	2,47E+00	2,36E+00	2,34E+00	1,97E+00	2,45E+01	2,05E+00	1,73E+00
SSGA	3,53E+00	3,36E+00	3,10E+00	3,11E+00	2,43E+00	2,50E+01	2,51E+00	2,20E+00
DE	2,07E+00	1,89E+00	1,81E+00	1,86E+00	1,42E+00	2,21E+01	1,47E+00	1,23E+00
BFO	8,58E-01	6,12E-01	6,03E-01	6,28E-01	7,50E-01	2,13E+01	8,91E-01	4,24E-01
FA	**4,90E-01**	**3,66E-01**	**3,58E-01**	**3,71E-01**	**4,25E-01**	**1,02E+01**	**4,62E-01**	**2,50E-01**
PSOG	9,19E-01	6,67E-01	6,58E-01	6,86E-01	7,55E-01	2,05E+01	8,80E-01	4,21E-01
PSOL	1,00E+01	9,72E+00	9,33E+00	9,25E+00	7,82E+00	2,59E+01	7,93E+00	7,50E+00
GSA	1,42E+01	1,30E+01	1,24E+01	1,26E+01	9,68E+00	2,83E+01	9,83E+00	9,59E+00
Dimension: 30 Rango [-100.0 100.0] MaxEvals 300000								

Tabla 3: Tiempos Julia F1-F8 / D30

	FUNCIONES							
	1	2	3	4	5	6	7	8
GGA	4,24E+00	3,85E+00	3,63E+00	3,71E+00	3,12E+00	3,44E+01	3,24E+00	2,43E+00
SSGA	5,41E+00	5,24E+00	4,80E+00	4,85E+00	3,74E+00	3,57E+01	3,80E+00	3,11E+00
DE	3,48E+00	3,30E+00	3,16E+00	3,22E+00	2,59E+00	3,38E+01	2,72E+00	1,91E+00
BFO	1,65E+00	1,37E+00	1,43E+00	1,44E+00	1,71E+00	3,28E+01	1,91E+00	7,19E-01
FA	**9,75E-01**	**7,82E-01**	**7,64E-01**	**7,95E-01**	**8,88E-01**	**1,64E+01**	**9,66E-01**	**3,36E-01**
PSOG	1,95E+00	1,51E+00	1,53E+00	1,56E+00	1,71E+00	3,36E+01	1,88E+00	6,35E-01
PSOL	1,47E+01	1,44E+01	1,37E+01	1,37E+01	1,11E+01	4,14E+01	1,13E+01	1,07E+01
GSA	1,97E+01	1,84E+01	1,79E+01	1,76E+01	1,31E+01	4,39E+01	1,30E+01	1,24E+01
Dimension: 50 Rango [-100.0 100.0] MaxEvals 500000								

Tabla 4: Tiempos Julia F1-F8 / D50

En estas tablas se pueden observar los tiempos de las 8 primeras funciones del benchmark. Cabe destacar que FA, no solo es más rápido en promedio, si no que es más rápido en todas y cada una de las funciones.

	FUNCIONES							
	9	10	11	12	13	14	15	16
GGA	3,41E+00	3,44E+00	3,55E+00	3,53E+00	3,49E+00	3,49E+00	3,49E+00	3,49E+00
SSGA	8,79E-01	8,84E-01	8,96E-01	9,16E-01	8,76E-01	8,69E-01	8,72E-01	8,76E-01
DE	1,04E+00	1,05E+00	1,08E+00	1,09E+00	1,05E+00	1,05E+00	1,04E+00	1,05E+00
BFO	1,58E+00	1,58E+00	1,64E+00	1,65E+00	1,60E+00	1,60E+00	1,60E+00	1,62E+00
FA	**7,11E-01**	**7,19E-01**	**7,51E-01**	**7,60E-01**	**7,32E-01**	**7,32E-01**	**7,32E-01**	**7,42E-01**
PSOG	9,27E-01	9,46E-01	9,79E-01	1,00E+00	9,52E-01	9,52E-01	9,52E-01	9,63E-01
PSOL	1,52E+00	1,52E+00	1,57E+00	1,56E+00	1,54E+00	1,54E+00	1,54E+00	1,54E+00
GSA	1,56E+00	1,58E+00	1,63E+00	1,56E+00	1,58E+00	1,55E+00	1,57E+00	1,56E+00
Dimension: 2 Rango [-100.0 100.0] MaxEvals 20000								

Tabla 5: Tiempos Julia F9-F16 / D2

	FUNCIONES							
	9	10	11	12	13	14	15	16
GGA	5,40E-01	5,77E-01	5,39E-01	1,36E+00	4,73E-01	4,70E-01	4,91E-01	4,97E-01
SSGA	7,08E-01	7,69E-01	7,27E-01	1,54E+00	6,55E-01	6,34E-01	6,70E-01	6,66E-01
DE	3,19E-01	3,66E-01	3,67E-01	1,05E+00	2,89E-01	2,84E-01	3,02E-01	3,10E-01
BFO	8,64E-02	1,32E-01	1,53E-01	9,93E-01	6,29E-02	6,32E-02	9,05E-02	9,95E-02
FA	**5,22E-02**	**7,19E-02**	**7,99E-02**	**4,36E-01**	**4,29E-02**	**4,21E-02**	**5,31E-02**	**5,72E-02**
PSOG	1,06E-01	1,59E-01	1,76E-01	9,58E-01	8,57E-02	8,48E-02	1,13E-01	1,21E-01
PSOL	1,55E+00	1,74E+00	1,65E+00	2,51E+00	1,56E+00	1,53E+00	1,61E+00	1,59E+00
GSA	2,60E+00	2,67E+00	2,70E+00	3,68E+00	2,55E+00	2,53E+00	2,58E+00	2,56E+00
Dimension: 10 Rango [-100.0 100.0] MaxEvals 100000								

Tabla 6: Tiempos Julia F9-F16 / D10

	FUNCIONES							
	9	10	11	12	13	14	15	16
GGA	1,93E+00	2,10E+00	2,14E+00	7,83E+00	1,65E+00	1,65E+00	1,82E+00	1,84E+00
SSGA	2,44E+00	2,65E+00	2,70E+00	8,44E+00	2,15E+00	2,11E+00	2,29E+00	2,29E+00
DE	1,42E+00	1,52E+00	1,78E+00	7,03E+00	1,21E+00	1,21E+00	1,36E+00	1,39E+00
BFO	7,55E-01	1,09E+00	1,41E+00	6,22E+00	5,22E-01	5,16E-01	7,80E-01	8,45E-01
FA	**4,31E-01**	**5,41E-01**	**7,36E-01**	**3,21E+00**	**2,96E-01**	**2,91E-01**	**4,32E-01**	**4,56E-01**
PSOG	7,60E-01	1,10E+00	1,43E+00	6,98E+00	5,33E-01	5,23E-01	7,81E-01	8,56E-01
PSOL	7,84E+00	8,13E+00	7,98E+00	1,34E+01	7,36E+00	7,37E+00	7,64E+00	7,61E+00
GSA	9,68E+00	1,02E+01	1,01E+01	1,54E+01	9,22E+00	9,16E+00	9,67E+00	9,53E+00
Dimension: 30 Rango [-100.0 100.0] MaxEvals 300000								

Tabla 7: Tiempos Julia F9-F16 / D30

	FUNCIONES							
	9	10	11	12	13	14	15	16
GGA	3,09E+00	3,00E+00	3,44E+00	1,15E+01	2,56E+00	2,56E+00	2,83E+00	2,93E+00
SSGA	3,71E+00	3,77E+00	4,14E+00	1,22E+01	3,20E+00	3,17E+00	3,48E+00	3,56E+00
DE	2,59E+00	2,64E+00	3,16E+00	1,09E+01	2,14E+00	2,15E+00	2,40E+00	2,50E+00
BFO	1,71E+00	1,79E+00	2,24E+00	9,67E+00	1,29E+00	1,29E+00	1,72E+00	1,82E+00
FA	**8,81E-01**	**8,94E-01**	**1,34E+00**	**5,11E+00**	**6,55E-01**	**6,57E-01**	**8,77E-01**	**9,30E-01**
PSOG	1,69E+00	1,69E+00	2,69E+00	1,06E+01	1,26E+00	1,27E+00	1,71E+00	1,81E+00
PSOL	1,11E+01	1,15E+01	1,19E+01	1,91E+01	1,09E+01	1,07E+01	1,12E+01	1,13E+01
GSA	1,32E+01	1,35E+01	1,39E+01	2,11E+01	1,28E+01	1,27E+01	1,35E+01	1,33E+01
Dimension: 50 Rango [-100.0 100.0] MaxEvals 500000								

Tabla 8: Tiempos Julia F9-F16 / D50

En estas tablas se pueden observar los tiempos de las 8 útimas funciones del benchmark.

8.2 Análisis de Resultados de EvolutionaryAlgs

En este apartado, se mostrará un ranking para cada dimensión, en los cuales se posicionarán los algoritmo en función del resultado obtenido en cada función. Al final, se insertará un ranking que posicione los algoritmos en función de los resultados obtenidos en cada dimensión.

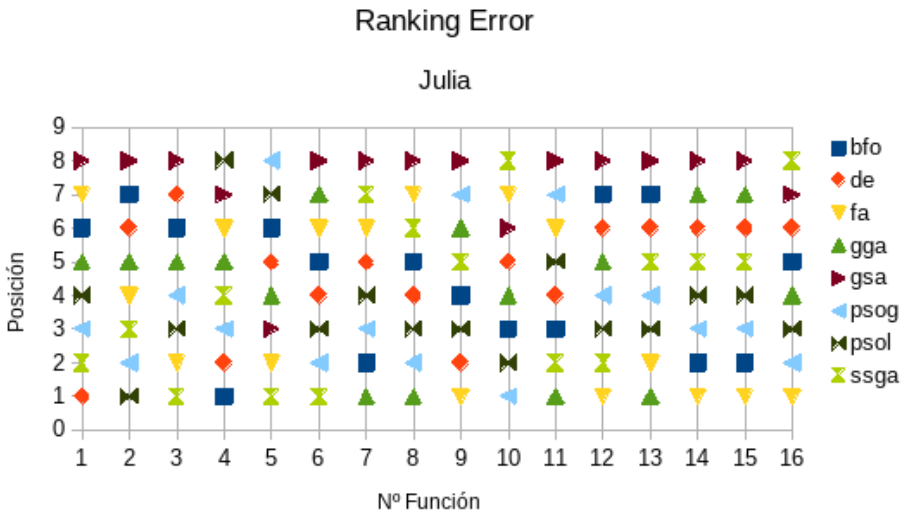


Figura 28: Ranking Error Julia D2

En este ranking, podemos ver qué tan bueno son los algoritmos al optimizar cada función en dimensión 2. Podemos ver cómo en las primeras posiciones se repite hasta 6 y 8 veces SSGA y FA respectivamente. También obtienen buenos resultados PSOL y PSOG, que al contrario que SSGA y FA, no aparecen, a excpción de 2 y 3 veces, en las últimas 3 posiciones.

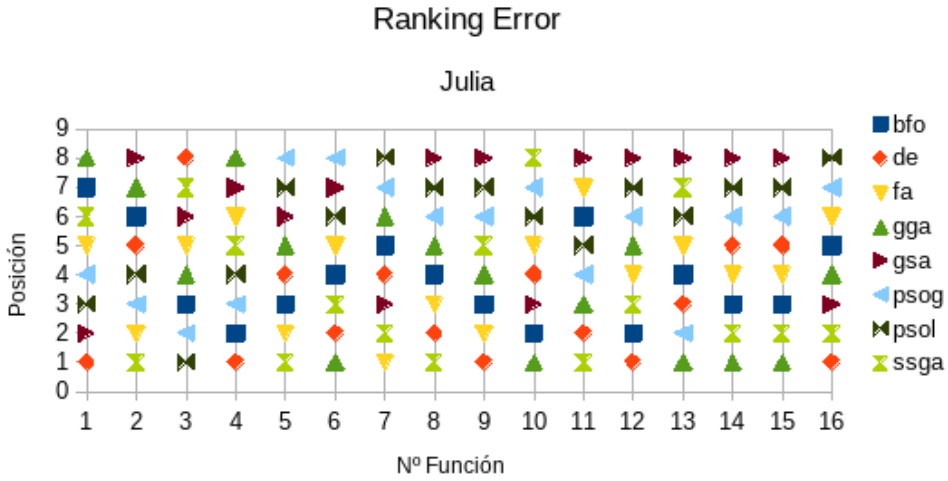


Figura 29: Ranking Error Julia D10

Para dimensión 10, podemos ver que varían ligeramente los algoritmos con respecto al ranking anterior. Las primeras son ocupadas por DE y SSGA, ambas 8 veces. FA pasa a ocupar posiciones más altas, al igual que PSOG y PSOL que ahora pasa a ocupar las últimas posiciones del ranking.

Figura 30: Ranking Error Julia D30

Para dimensión 10, podemos ver que las primeras posiciones se mantienen con respecto al ranking anterior. Las primeras son ocupadas por DE y SSGA, ambas 7 veces. FA mejora un poco ocupando la mayoria de veces el top 3, al igual que PSOG y PSOL que empieza a tener mayor ocupación en las primeras posiciones.

Figura 31: Ranking Error Julia D50

En esta ocasión, los algoritmos genéticos, GGA y SSGA toman la delantera apareciendo en las primeras 2 posiciones 9 y 7 veces respectivamente.

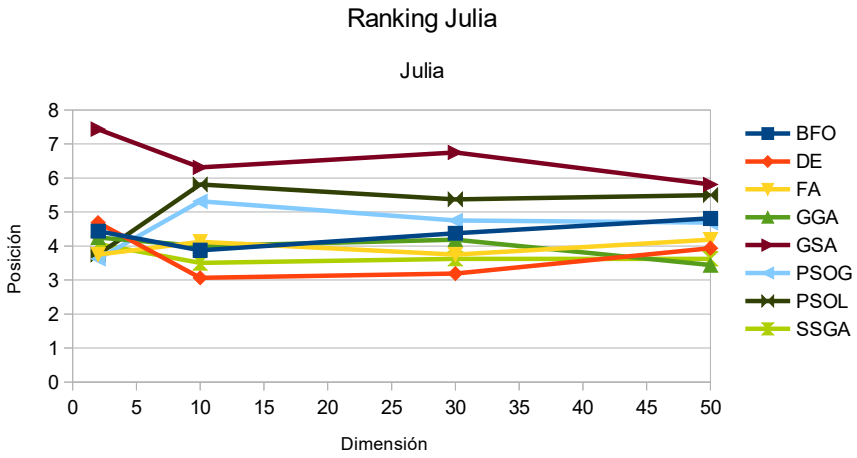


Figura 32: Ranking Julia Dimensiones

Aquí se puede ver el promedio de ranking obtenido por los algoritmos en cada dimensión, lo que nos da una idea acerca de qué algoritmos son mejores dependiendo de la dimensión. Podemos observar que ambos algoritmos geneticos tienden a mejorar posiciones conforme aumenta la dimensión quedando en las primeras posiciones. BFO tiene una clara tendencia a obtener peores posiciones a medida que aumenta la dimensión. El único algoritmo que apenas varía su posición en el ranking es FA,el cuál recordemos es el más rápido en todos los casos.

	FUNCIONES							
	1	2	3	4	5	6	7	8
GGA	7,52E+06	2,84E+07	8,34E+06	2,59E+00	1,70E+01	1,56E+00	2,88E+00	7,20E+00
SSGA	6,22E+06	2,10E+07	5,56E+06	2,53E+00	1,63E+01	1,35E+00	3,50E+00	7,98E+00
DE	5,67E+06	3,06E+07	1,00E+07	1,97E+00	1,72E+01	1,43E+00	3,25E+00	7,66E+00
BFO	1,13E+07	5,21E+07	8,88E+06	1,35E+00	1,72E+01	1,47E+00	3,03E+00	7,94E+00
FA	1,16E+07	2,45E+07	7,86E+06	4,37E+00	1,66E+01	1,48E+00	3,48E+00	8,46E+00
PSOG	6,81E+06	1,37E+07	8,05E+06	2,12E+00	1,76E+01	1,42E+00	3,23E+00	7,29E+00
PSOL	6,83E+06	1,36E+07	7,99E+06	1,09E+01	1,76E+01	1,42E+00	3,24E+00	7,34E+00
GSA	1,28E+08	6,58E+08	2,02E+07	4,56E+00	1,67E+01	1,59E+00	3,96E+00	1,09E+01
Dimension: 2 Rango [-100,0 100,0] MaxEvals 20000								

Tabla 9: Error Promedio Julia F1-F8 / D2

	FUNCIONES							
	1	2	3	4	5	6	7	8
GGA	5,24E+08	1,64E+10	6,55E+06	3,45E+03	2,10E+01	1,36E+01	2,64E+02	1,32E+02
SSGA	4,87E+08	**1,46E+10**	8,87E+06	3,19E+03	**2,10E+01**	1,37E+01	2,48E+02	**1,26E+02**
DE	**3,75E+08**	1,56E+10	1,57E+07	**2,81E+03**	2,10E+01	1,36E+01	2,58E+02	1,27E+02
BFO	5,17E+08	1,61E+10	3,44E+06	2,84E+03	2,10E+01	1,38E+01	2,60E+02	1,31E+02
FA	4,77E+08	1,48E+10	6,58E+06	3,27E+03	2,10E+01	1,40E+01	**2,47E+02**	1,30E+02
PSOG	4,75E+08	1,50E+10	3,25E+06	3,06E+03	2,11E+01	1,42E+01	2,66E+02	1,34E+02
PSOL	4,74E+08	1,56E+10	**3,24E+06**	3,17E+03	2,11E+01	1,41E+01	2,68E+02	1,34E+02
GSA	4,69E+08	1,77E+10	7,31E+06	3,27E+03	2,10E+01	1,42E+01	2,51E+02	1,35E+02
Dimension: 10 Rango [-100,0 100,0] MaxEvals 100000								

Tabla 10: Error Promedio Julia F1-F8 / D10

	FUNCIONES							
	1	2	3	4	5	6	7	8
GGA	3,84E+09	1,47E+11	1,78E+07	4,63E+04	2,14E+01	4,88E+01	1,35E+03	5,17E+02
SSGA	4,18E+09	1,48E+11	1,27E+07	**4,14E+04**	2,14E+01	**4,82E+01**	1,32E+03	5,27E+02
DE	4,04E+09	**1,42E+11**	**1,03E+07**	4,30E+04	2,14E+01	4,97E+01	1,36E+03	5,20E+02
BFO	4,20E+09	1,43E+11	1,37E+07	4,64E+04	2,14E+01	4,92E+01	**1,32E+03**	5,19E+02
FA	3,99E+09	1,43E+11	1,53E+07	4,52E+04	**2,13E+01**	4,88E+01	1,33E+03	5,28E+02
PSOG	4,00E+09	1,49E+11	1,37E+07	4,15E+04	2,14E+01	4,90E+01	1,34E+03	5,25E+02
PSOL	4,06E+09	1,50E+11	1,37E+07	4,20E+04	2,14E+01	4,89E+01	1,35E+03	5,26E+02
GSA	4,02E+09	1,57E+11	1,67E+08	4,31E+04	2,14E+01	4,91E+01	1,38E+03	5,36E+02
Dimension: 30 Rango [-100,0 100,0] MaxEvals 300000								

Tabla 11: Error Promedio Julia F1-F8 / D30

	FUNCIONES							
	1	2	3	4	5	6	7	8
GGA	9,81E+09	3,05E+11	8,83E+06	1,28E+05	2,15E+01	8,52E+01	2,94E+03	9,77E+02
SSGA	9,48E+09	3,00E+11	1,93E+07	1,34E+05	2,15E+01	8,54E+01	2,87E+03	9,62E+02
DE	1,01E+10	2,92E+11	3,45E+06	1,31E+05	2,15E+01	8,54E+01	2,85E+03	9,91E+02
BFO	9,95E+09	3,03E+11	8,54E+06	1,32E+05	2,15E+01	8,59E+01	2,94E+03	9,64E+02
FA	1,00E+10	3,02E+11	9,82E+06	1,34E+05	2,14E+01	8,59E+01	2,87E+03	9,81E+02
PSOG	1,03E+10	3,02E+11	4,02E+06	1,39E+05	2,14E+01	8,55E+01	2,92E+03	1,00E+03
PSOL	1,03E+10	3,03E+11	4,02E+06	1,44E+05	2,14E+01	8,55E+01	2,95E+03	1,00E+03
GSA	1,03E+10	3,05E+11	1,05E+07	1,38E+05	2,15E+01	8,54E+01	3,02E+03	9,85E+02
Dimension: 50 Rango [-100,0 100,0] MaxEvals 500000								

Tabla 12: Error Promedio Julia F1-F8 / D50

	FUNCIONES							
	9	10	11	12	13	14	15	16
GGA	8,16E+00	2,12E+02	1,73E+02	1,06E+00	8,99E-01	1,92E+01	1,92E+01	5,90E-01
SSGA	8,07E+00	2,36E+02	1,89E+02	1,04E+00	9,44E-01	1,51E+01	1,51E+01	6,27E-01
DE	7,36E+00	2,13E+02	2,10E+02	1,07E+00	9,93E-01	1,83E+01	1,83E+01	6,07E-01
BFO	8,00E+00	2,07E+02	2,02E+02	1,09E+00	1,00E+00	5,91E+00	5,91E+00	6,00E-01
FA	7,32E+00	2,18E+02	2,15E+02	1,04E+00	9,00E-01	5,33E+00	5,33E+00	5,64E-01
PSOG	8,64E+00	1,94E+02	2,20E+02	1,06E+00	9,13E-01	7,42E+00	7,42E+00	5,80E-01
PSOL	7,50E+00	1,95E+02	2,15E+02	1,05E+00	9,10E-01	7,42E+00	7,42E+00	5,83E-01
GSA	9,38E+00	2,18E+02	2,41E+02	1,57E+00	2,19E+00	2,08E+05	2,08E+05	6,15E-01
Dimension: 2 Rango [-100,0 100,0] MaxEvals 20000								

Tabla 13: Error Promedio Julia F9-F16 / D2

	FUNCIONES							
	9	10	11	12	13	14	15	16
GGA	1,35E+02	2,45E+03	2,57E+03	5,61E+00	6,11E+01	3,88E+05	3,88E+05	4,48E+00
SSGA	1,36E+02	2,56E+03	**2,55E+03**	5,52E+00	6,81E+01	3,91E+05	3,91E+05	4,44E+00
DE	**1,32E+02**	2,52E+03	2,57E+03	**5,37E+00**	6,22E+01	4,31E+05	4,31E+05	**4,44E+00**
BFO	1,35E+02	2,51E+03	2,63E+03	5,49E+00	6,48E+01	4,04E+05	4,04E+05	4,50E+00
FA	1,34E+02	2,54E+03	2,65E+03	5,53E+00	6,62E+01	4,11E+05	4,11E+05	4,52E+00
PSOG	1,38E+02	2,56E+03	2,58E+03	5,64E+00	6,21E+01	6,03E+05	6,03E+05	4,52E+00
PSOL	1,41E+02	2,56E+03	2,59E+03	5,73E+00	6,62E+01	6,07E+05	6,07E+05	4,53E+00
GSA	1,44E+02	2,52E+03	2,66E+03	5,80E+00	7,05E+01	8,87E+06	8,87E+06	4,47E+00
Dimension: 10 Rango [-100,0 100,0] MaxEvals 100000								

Tabla 14: Error Promedio Julia F9-F16 / D10

	FUNCIONES							
	9	10	11	12	13	14	15	16
GGA	6,48E+02	9,27E+03	9,54E+03	1,10E+01	4,71E+02	4,83E+07	4,83E+07	1,43E+01
SSGA	**6,31E+02**	9,49E+03	**9,45E+03**	1,15E+01	**4,50E+02**	4,24E+07	4,24E+07	1,43E+01
DE	6,34E+02	9,33E+03	9,51E+03	1,14E+01	4,57E+02	**3,88E+07**	**3,88E+07**	**1,42E+01**
BFO	6,39E+02	9,46E+03	9,67E+03	1,13E+01	4,52E+02	4,26E+07	4,26E+07	1,43E+01
FA	6,42E+02	9,43E+03	9,53E+03	**1,08E+01**	4,60E+02	3,89E+07	3,89E+07	1,43E+01
PSOG	6,38E+02	9,34E+03	9,64E+03	1,15E+01	4,51E+02	4,63E+07	4,63E+07	1,43E+01
PSOL	6,43E+02	9,31E+03	9,63E+03	1,15E+01	4,53E+02	4,64E+07	4,64E+07	1,43E+01
GSA	6,71E+02	9,49E+03	9,62E+03	1,15E+01	4,94E+02	1,96E+08	1,96E+08	1,43E+01
Dimension: 30 Rango [-100,0 100,0] MaxEvals 300000								

Tabla 15: Error Promedio Julia F9-F16 / D30

	FUNCIONES							
	9	10	11	12	13	14	15	16
GGA	1,23E+03	1,65E+04	1,70E+04	1,27E+01	7,54E+02	1,88E+08	1,88E+08	2,42E+01
SSGA	1,23E+03	1,67E+04	1,68E+04	1,27E+01	7,68E+02	2,01E+08	2,01E+08	2,41E+01
DE	1,24E+03	1,66E+04	1,69E+04	1,32E+01	7,55E+02	2,25E+08	2,25E+08	2,41E+01
BFO	1,24E+03	1,68E+04	1,67E+04	1,30E+01	7,90E+02	2,19E+08	2,19E+08	2,41E+01
FA	1,26E+03	1,66E+04	1,68E+04	1,29E+01	7,64E+02	2,19E+08	2,19E+08	2,42E+01
PSOG	1,26E+03	1,66E+04	1,68E+04	1,28E+01	7,82E+02	2,08E+08	2,08E+08	2,42E+01
PSOL	1,26E+03	1,66E+04	1,68E+04	1,27E+01	7,90E+02	2,25E+08	2,25E+08	2,42E+01
GSA	1,25E+03	1,63E+04	1,68E+04	1,30E+01	8,03E+02	6,08E+08	6,08E+08	2,42E+01
Dimension: 50 Rango [-100,0 100,0] MaxEvals 500000								

Tabla 16: Error Promedio Julia F9-F16 / D50

	Dimensión			
	2	10	30	50
BFO	4,44	3,88	4,38	4,81
DE	4,69	3,06	3,19	3,94
FA	3,75	4,13	3,75	4,19
GGA	4,25	4,00	4,19	3,44
GSA	7,44	6,31	6,75	5,81
PSOG	3,63	5,31	4,75	4,69
PSOL	3,75	5,81	5,38	5,50
SSGA	4,06	3,50	3,63	3,63

Tabla 17: Ranking Error Promedio

8.3 Julia vs. Python

Este apartado se dedicará ha realizar una comparación entre los algoritmos programados en Julia y en Python. En primer lugar, siguiendo el orden de los apartados anteriores, haremos la comparación entre los tiempos en escala logarítmica.

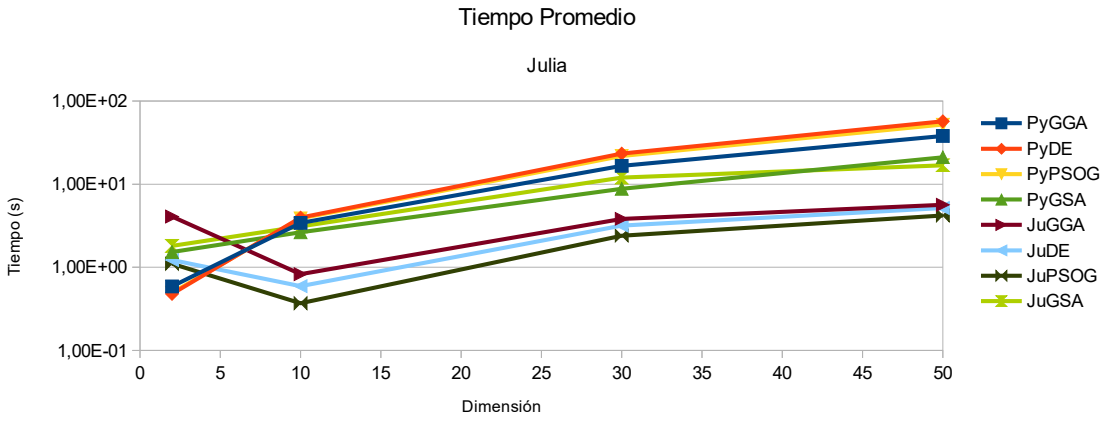


Figura 33: Tiempo Promedio Julia - Python

Tal y como venimos previendo a lo largo del trabajo, podemos observar que en general todos los algoritmos programados en Julia son superiores a los de Python. Aunque podemos ver cómo el GSA de Julia es más lento que el de Python hasta la dimensión 50.

	FUNCIONES							
	1	2	3	4	5	6	7	8
PyGGA	6,87E-01	6,25E-01	5,75E-01	5,17E-01	5,14E-01	7,62E-01	6,55E-01	5,99E-01
PyDE	5,90E-01	5,90E-01	**4,77E-01**	**4,75E-01**	**4,79E-01**	**5,99E-01**	**4,35E-01**	**4,36E-01**
PyPSOG	**5,89E-01**	**5,89E-01**	4,87E-01	4,81E-01	4,92E-01	6,16E-01	4,60E-01	4,60E-01
PyGSA	1,46E+00	1,45E+00	1,59E+00	1,51E+00	1,52E+00	1,67E+00	1,50E+00	1,54E+00
JuGGA	6,07E+00	6,29E+00	5,53E+00	5,51E+00	3,43E+00	3,57E+00	3,42E+00	3,40E+00
JuDE	1,82E+00	1,78E+00	1,60E+00	1,60E+00	1,05E+00	1,20E+00	1,04E+00	1,04E+00
JuPSOG	1,57E+00	1,56E+00	1,40E+00	1,39E+00	9,52E-01	1,13E+00	9,40E-01	9,08E-01
JuGSA	2,67E+00	2,56E+00	2,34E+00	2,33E+00	1,56E+00	1,72E+00	1,54E+00	1,58E+00
Dimension: 2 Rango [-100.0 100.0] MaxEvals 20000								

Tabla 18: Tiempos Julia - Python F1-F8 / D2

	FUNCIONES							
	1	2	3	4	5	6	7	8
PyGGA	3,74E+00	3,72E+00	3,14E+00	3,14E+00	3,17E+00	6,81E+00	2,98E+00	2,95E+00
PyDE	4,43E+00	4,42E+00	3,71E+00	3,70E+00	3,73E+00	7,32E+00	3,49E+00	3,48E+00
PyPSOG	4,21E+00	4,19E+00	3,59E+00	3,58E+00	3,73E+00	7,22E+00	3,45E+00	3,42E+00
PyGSA	2,26E+00	2,23E+00	2,27E+00	2,20E+00	2,26E+00	6,88E+00	2,24E+00	2,25E+00
JuGGA	7,64E-01	7,32E-01	6,95E-01	7,00E-01	5,44E-01	3,75E+00	5,57E-01	5,33E-01
JuDE	5,07E-01	4,62E-01	4,44E-01	4,63E-01	3,22E-01	3,31E+00	3,29E-01	3,22E-01
JuPSOG	**1,46E-01**	**1,10E-01**	**1,06E-01**	**1,09E-01**	**1,06E-01**	**3,31E+00**	**1,19E-01**	**9,26E-02**
JuGSA	3,61E+00	3,41E+00	3,24E+00	3,29E+00	2,63E+00	6,19E+00	2,62E+00	2,66E+00
Dimension: 10 Rango [-100.0 100.0] MaxEvals 100000								

Tabla 19: Tiempos Julia - Python F1-F8 / D10

	FUNCIONES							
	1	2	3	4	5	6	7	8
PyGGA	1,65E+01	1,65E+01	1,40E+01	1,40E+01	1,41E+01	4,78E+01	1,34E+01	1,29E+01
PyDE	2,40E+01	2,39E+01	2,09E+01	2,08E+01	2,09E+01	5,44E+01	2,00E+01	1,96E+01
PyPSOG	2,19E+01	2,18E+01	1,92E+01	1,92E+01	1,99E+01	5,26E+01	1,86E+01	1,82E+01
PyGSA	5,88E+00	5,65E+00	5,70E+00	5,62E+00	6,33E+00	4,57E+01	5,65E+00	5,48E+00
JuGGA	2,67E+00	2,47E+00	2,36E+00	2,34E+00	1,97E+00	2,45E+01	2,05E+00	1,73E+00
JuDE	2,07E+00	1,89E+00	1,81E+00	1,86E+00	1,42E+00	2,21E+01	1,47E+00	1,23E+00
JuPSOG	**9,19E-01**	**6,67E-01**	**6,58E-01**	**6,86E-01**	**7,55E-01**	**2,05E+01**	**8,80E-01**	**4,21E-01**
JuGSA	1,42E+01	1,30E+01	1,24E+01	1,26E+01	9,68E+00	2,83E+01	9,83E+00	9,59E+00
Dimension: 30 Rango [-100.0 100.0] MaxEvals 300000								

Tabla 20: Tiempos Julia - Python F1-F8 / D30

	FUNCIONES							
	1	2	3	4	5	6	7	8
PyGGA	3,60E+01	3,59E+01	3,14E+01	3,14E+01	3,17E+01	1,15E+02	3,05E+01	2,85E+01
PyDE	5,83E+01	5,81E+01	5,17E+01	5,17E+01	5,18E+01	1,25E+02	4,99E+01	4,81E+01
PyPSOG	5,19E+01	5,14E+01	4,66E+01	4,64E+01	4,79E+01	1,20E+02	4,50E+01	4,32E+01
PyGSA	1,25E+01	1,22E+01	1,21E+01	1,19E+01	1,39E+01	1,14E+02	1,37E+01	1,12E+01
JuGGA	4,24E+00	3,85E+00	3,63E+00	3,71E+00	3,12E+00	3,44E+01	3,24E+00	2,43E+00
JuDE	3,48E+00	3,30E+00	3,16E+00	3,22E+00	2,59E+00	3,38E+01	2,72E+00	1,91E+00
JuPSOG	**1,95E+00**	**1,51E+00**	**1,53E+00**	**1,56E+00**	**1,71E+00**	**3,36E+01**	**1,88E+00**	**6,35E-01**
JuGSA	1,97E+01	1,84E+01	1,79E+01	1,76E+01	1,31E+01	4,39E+01	1,30E+01	1,24E+01
Dimension: 50 Rango [-100.0 100.0] MaxEvals 500000								

Tabla 21: Tiempos Julia - Python F1-F8 / D50

	FUNCIONES							
	9	10	11	12	13	14	15	16
PyGGA	5,35E-01	4,87E-01	5,92E-01	6,32E-01	5,85E-01	5,90E-01	4,85E-01	5,51E-01
PyDE	**4,33E-01**	**4,42E-01**	**4,37E-01**	**4,79E-01**	**4,34E-01**	**4,36E-01**	**4,37E-01**	**4,46E-01**
PyPSOG	4,48E-01	4,61E-01	4,64E-01	5,11E-01	4,60E-01	4,56E-01	4,55E-01	4,62E-01
PyGSA	1,50E+00	1,52E+00	1,51E+00	1,53E+00	1,52E+00	1,52E+00	1,51E+00	1,50E+00
JuGGA	3,41E+00	3,44E+00	3,55E+00	3,53E+00	3,49E+00	3,49E+00	3,49E+00	3,49E+00
JuDE	1,04E+00	1,05E+00	1,08E+00	1,09E+00	1,05E+00	1,05E+00	1,04E+00	1,05E+00
JuPSOG	9,27E-01	9,46E-01	9,79E-01	1,00E+00	9,52E-01	9,52E-01	9,52E-01	9,63E-01
JuGSA	1,56E+00	1,58E+00	1,63E+00	1,56E+00	1,58E+00	1,55E+00	1,57E+00	1,56E+00
Dimension: 2 Rango [-100.0 100.0] MaxEvals 20000								

Tabla 22: Tiempos Julia - Python F9-F16 / D2

	FUNCIONES							
	9	10	11	12	13	14	15	16
PyGGA	2,93E+00	3,01E+00	3,02E+00	4,01E+00	2,94E+00	2,94E+00	2,94E+00	3,04E+00
PyDE	3,45E+00	3,53E+00	3,55E+00	4,52E+00	3,47E+00	3,48E+00	3,46E+00	3,53E+00
PyPSOG	3,36E+00	3,46E+00	3,50E+00	4,49E+00	3,41E+00	3,40E+00	3,44E+00	3,50E+00
PyGSA	2,21E+00	2,49E+00	2,49E+00	3,23E+00	2,22E+00	2,21E+00	2,31E+00	2,30E+00
JuGGA	5,40E-01	5,77E-01	5,39E-01	1,36E+00	4,73E-01	4,70E-01	4,91E-01	4,97E-01
JuDE	3,19E-01	3,66E-01	3,67E-01	1,05E+00	2,89E-01	2,84E-01	3,02E-01	3,10E-01
JuPSOG	**1,06E-01**	**1,59E-01**	**1,76E-01**	**9,58E-01**	**8,57E-02**	**8,48E-02**	**1,13E-01**	**1,21E-01**
JuGSA	2,60E+00	2,67E+00	2,70E+00	3,68E+00	2,55E+00	2,53E+00	2,58E+00	2,56E+00
Dimension: 10 Rango [-100.0 100.0] MaxEvals 100000								

Tabla 23: Tiempos Julia - Python F9-F16 / D10

	FUNCIONES							
	9	10	11	12	13	14	15	16
PyGGA	1,32E+01	1,35E+01	1,37E+01	2,25E+01	1,32E+01	1,31E+01	1,33E+01	1,36E+01
PyDE	1,98E+01	2,01E+01	2,05E+01	2,93E+01	1,98E+01	1,99E+01	1,98E+01	2,03E+01
PyPSOG	1,83E+01	1,87E+01	1,92E+01	2,79E+01	1,86E+01	1,82E+01	1,86E+01	1,89E+01
PyGSA	5,68E+00	5,49E+00	5,71E+00	1,50E+01	5,68E+00	5,64E+00	5,70E+00	5,86E+00
JuGGA	1,93E+00	2,10E+00	2,14E+00	7,83E+00	1,65E+00	1,65E+00	1,82E+00	1,84E+00
JuDE	1,42E+00	1,52E+00	1,78E+00	7,03E+00	1,21E+00	1,21E+00	1,36E+00	1,39E+00
JuPSOG	**7,60E-01**	**1,10E+00**	**1,43E+00**	**6,98E+00**	**5,33E-01**	**5,23E-01**	**7,81E-01**	**8,56E-01**
JuGSA	9,68E+00	1,02E+01	1,01E+01	1,54E+01	9,22E+00	9,16E+00	9,67E+00	9,53E+00
Dimension: 30 Rango [-100.0 100.0] MaxEvals 300000								

Tabla 24: Tiempos Julia - Python F9-F16 / D30

	FUNCIONES							
	9	10	11	12	13	14	15	16
PyGGA	2,99E+01	3,00E+01	3,16E+01	5,60E+01	2,97E+01	2,96E+01	3,02E+01	3,09E+01
PyDE	4,95E+01	4,95E+01	5,11E+01	7,21E+01	4,91E+01	4,92E+01	4,94E+01	5,03E+01
PyPSOG	4,44E+01	4,43E+01	4,62E+01	6,48E+01	4,45E+01	4,41E+01	4,49E+01	4,60E+01
PyGSA	1,28E+01	1,76E+01	1,90E+01	3,51E+01	1,23E+01	1,20E+01	1,44E+01	1,38E+01
JuGGA	3,09E+00	3,00E+00	3,44E+00	1,15E+01	2,56E+00	2,56E+00	2,83E+00	2,93E+00
JuDE	2,59E+00	2,64E+00	3,16E+00	1,09E+01	2,14E+00	2,15E+00	2,40E+00	2,50E+00
JuPSOG	**1,69E+00**	**1,69E+00**	**2,69E+00**	**1,06E+01**	**1,26E+00**	**1,27E+00**	**1,71E+00**	**1,81E+00**
JuGSA	1,32E+01	1,35E+01	1,39E+01	2,11E+01	1,28E+01	1,27E+01	1,35E+01	1,33E+01
Dimension: 50 Rango [-100.0 100.0] MaxEvals 500000								

Tabla 25: Tiempos Julia - Python F9-F16 / D50

A continuación, haremos la comparativa del promedio de errores obtenidos en cada función por cada uno de los algoritmos en escala logarítmica.

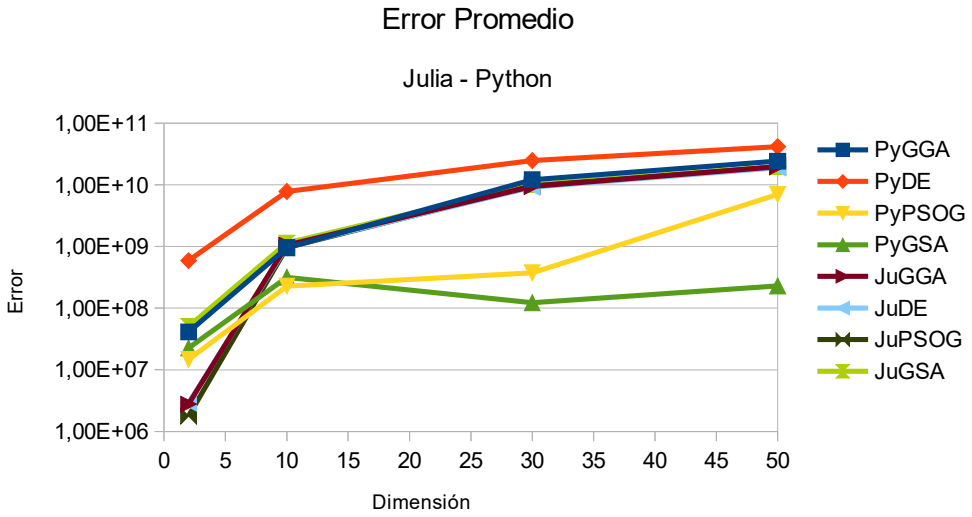


Figura 34: Error Promedio Julia - Python

Podemos ver que mientras los errores obtenidos por los algoritmos de Python, varían mucho en función del algoritmo en el que nos fijemos, los de Julia obtienen un error bastante parecido. No obstante, el conjunto de algoritmos de Julia obtiene mejores resultados que dos de los algoritmos de Python, y además tiene tendencia a obtener errores más bajos que dichos algoritmos.

	FUNCIONES							
	1	2	3	4	5	6	7	8
PyGGA	1,18E+08	5,37E+08	2,07E+06	5,99E+02	2,00E+01	2,25E+00	3,73E+00	1,43E+01
PyDE	4,40E+09	4,90E+09	1,20E+08	1,52E+02	2,18E+01	4,62E+00	9,16E+01	6,24E+01
PyPSOG	2,09E+08	1,63E+06	2,32E+07	0,00E+00	5,68E-14	0,00E+00	7,40E-03	9,95E-01
PyGSA	1,50E+07	1,75E+08	4,48E+07	3,44E+04	2,29E+01	2,46E+01	3,64E+02	2,14E+02
JuGGA	7,52E+06	2,84E+07	8,34E+06	2,59E+00	1,70E+01	1,56E+00	2,88E+00	7,20E+00
JuDE	5,67E+06	3,06E+07	1,00E+07	1,97E+00	1,72E+01	1,43E+00	3,25E+00	7,66E+00
JuPSOG	6,81E+06	1,37E+07	8,05E+06	2,12E+00	1,76E+01	1,42E+00	3,23E+00	7,29E+00
JuGSA	1,28E+08	6,58E+08	2,02E+07	4,56E+00	1,67E+01	1,59E+00	3,96E+00	1,09E+01
Dimension: 2 Rango [-100,0 100,0] MaxEvals 20000								

Tabla 26: Error Promedio Julia - Python F1-F8 / D2

	FUNCIONES							
	1	2	3	4	5	6	7	8
PyGGA	1,10E+09	1,40E+10	9,60E+07	1,65E+03	2,12E+01	1,25E+01	1,69E+02	1,19E+02
PyDE	2,83E+09	1,21E+11	2,92E+08	3,66E+04	2,11E+01	1,56E+01	1,17E+03	2,23E+02
PyPSOG	2,87E+05	3,64E+09	8,04E+05	2,18E+01	2,19E+01	8,84E+00	7,52E-01	2,63E+01
PyGSA	1,39E+06	6,74E+06	1,98E+05	7,20E+04	2,28E+01	2,48E+01	6,74E+02	3,02E+02
JuGGA	5,24E+08	1,64E+10	6,55E+06	3,45E+03	2,10E+01	1,36E+01	2,64E+02	1,32E+02
JuDE	3,75E+08	1,56E+10	1,57E+07	2,81E+03	2,10E+01	1,36E+01	2,58E+02	1,27E+02
JuPSOG	4,75E+08	1,50E+10	3,25E+06	3,06E+03	2,11E+01	1,42E+01	2,66E+02	1,34E+02
JuGSA	4,69E+08	1,77E+10	7,31E+06	3,27E+03	2,10E+01	1,42E+01	2,51E+02	1,35E+02
Dimension: 10 Rango [-100,0 100,0] MaxEvals 100000								

Tabla 27: Error Promedio Julia - Python F1-F8 / D10

	FUNCIONES							
	1	2	3	4	5	6	7	8
PyGGA	7,66E+09	1,86E+11	3,90E+07	7,71E+04	2,15E+01	5,23E+01	1,42E+03	5,16E+02
PyDE	1,69E+10	3,64E+11	1,19E+10	3,05E+05	2,14E+01	5,39E+01	3,01E+03	1,00E+03
PyPSOG	8,71E+06	5,96E+09	7,73E+04	8,88E+02	2,17E+01	1,83E+01	2,88E+02	1,90E+02
PyGSA	1,23E+06	9,84E+06	2,74E+05	8,57E+04	2,28E+01	2,57E+01	8,08E+02	3,73E+02
JuGGA	3,84E+09	1,47E+11	1,78E+07	4,63E+04	2,14E+01	4,88E+01	1,35E+03	5,17E+02
JuDE	4,04E+09	1,42E+11	1,03E+07	4,30E+04	2,14E+01	4,97E+01	1,36E+03	5,20E+02
JuPSOG	4,00E+09	1,49E+11	1,37E+07	4,15E+04	2,14E+01	4,90E+01	1,34E+03	5,25E+02
JuGSA	4,02E+09	1,57E+11	1,67E+08	4,31E+04	2,14E+01	4,91E+01	1,38E+03	5,36E+02
Dimension: 30 Rango [-100,0 100,0] MaxEvals 300000								

Tabla 28: Error Promedio Julia - Python F1-F8 / D30

	FUNCIONES							
	1	2	3	4	5	6	7	8
PyGGA	9,13E+09	3,77E+11	2,15E+07	2,03E+05	2,15E+01	9,42E+01	2,98E+03	9,60E+02
PyDE	2,74E+10	6,31E+11	2,88E+08	5,98E+05	2,13E+01	9,08E+01	7,28E+03	1,63E+03
PyPSOG	1,18E+09	1,10E+11	5,11E+07	1,22E+04	2,19E+01	5,87E+01	3,58E+02	3,34E+02
PyGSA	2,01E+06	8,82E+06	2,84E+05	9,50E+04	2,28E+01	2,63E+01	9,22E+02	3,58E+02
JuGGA	9,81E+09	3,05E+11	8,83E+06	1,28E+05	2,15E+01	8,52E+01	2,94E+03	9,77E+02
JuDE	1,01E+10	2,92E+11	3,45E+06	1,31E+05	2,15E+01	8,54E+01	2,85E+03	9,91E+02
JuPSOG	1,03E+10	3,02E+11	4,02E+06	1,39E+05	2,14E+01	8,55E+01	2,92E+03	1,00E+03
JuGSA	1,03E+10	3,05E+11	1,05E+07	1,38E+05	2,15E+01	8,54E+01	3,02E+03	9,85E+02
Dimension: 50 Rango [-100,0 100,0] MaxEvals 500000								

Tabla 29: Error Promedio Julia - Python F1-F8 / D50

	FUNCIONES							
	9	10	11	12	13	14	15	16
PyGGA	2,05E+01	2,24E+02	7,72E+01	1,54E+01	1,33E+00	2,22E+00	1,90E+00	9,54E-01
PyDE	3,35E+01	8,23E+02	5,28E+02	1,88E+01	1,47E+01	3,58E+01	9,23E+06	9,91E-01
PyPSOG	0,00E+00	3,12E-01	1,68E+01	1,10E+01	4,87E-01	2,79E-01	0,00E+00	0,00E+00
PyGSA	3,79E+02	3,84E+02	4,22E+02	3,51E+01	5,44E+01	1,49E+02	1,26E+08	4,00E+01
JuGGA	8,16E+00	2,12E+02	1,73E+02	1,06E+00	8,99E-01	1,92E+01	1,92E+01	5,90E-01
JuDE	7,36E+00	2,13E+02	2,10E+02	1,07E+00	9,93E-01	1,83E+01	1,83E+01	6,07E-01
JuPSOG	8,64E+00	1,94E+02	2,20E+02	1,06E+00	9,13E-01	7,42E+00	7,42E+00	5,80E-01
JuGSA	9,38E+00	2,18E+02	2,41E+02	1,57E+00	2,19E+00	2,08E+05	2,08E+05	6,15E-01
Dimension: 2 Rango [-100,0 100,0] MaxEvals 20000								

Tabla 30: Error Promedio Julia - Python F9-F16 / D2

	FUNCIONES							
	9	10	11	12	13	14	15	16
PyGGA	1,76E+02	2,54E+03	2,98E+03	5,66E+00	6,53E+00	3,97E+01	1,45E+05	4,76E+00
PyDE	3,74E+02	2,35E+03	3,18E+03	9,37E+00	1,56E+01	3,12E+02	3,67E+08	4,98E+00
PyPSOG	4,04E+01	4,77E+02	1,21E+03	8,27E-02	2,28E+00	4,15E+00	2,13E+01	4,45E+00
PyGSA	3,66E+02	2,51E+02	2,91E+02	3,48E+01	9,24E+01	2,96E+02	5,05E+09	4,01E+01
JuGGA	1,35E+02	2,45E+03	2,57E+03	5,61E+00	6,11E+01	3,88E+05	3,88E+05	4,48E+00
JuDE	1,32E+02	2,52E+03	2,57E+03	5,37E+00	6,22E+01	4,31E+05	4,31E+05	4,44E+00
JuPSOG	1,38E+02	2,56E+03	2,58E+03	5,64E+00	6,21E+01	6,03E+05	6,03E+05	4,52E+00
JuGSA	1,44E+02	2,52E+03	2,66E+03	5,80E+00	7,05E+01	8,87E+06	8,87E+06	4,47E+00
Dimension: 10 Rango [-100,0 100,0] MaxEvals 100000								

Tabla 31: Error Promedio Julia - Python F9-F16 / D10

	FUNCIONES							
	9	10	11	12	13	14	15	16
PyGGA	7,51E+02	1,10E+04	1,05E+04	7,72E+00	1,43E+01	7,23E+02	4,87E+07	1,44E+01
PyDE	1,30E+03	9,31E+03	1,06E+04	5,69E+00	1,72E+01	9,85E+02	1,71E+09	1,46E+01
PyPSOG	2,55E+02	4,41E+03	3,93E+03	2,79E-01	4,97E+00	1,85E+02	2,15E+03	1,41E+01
PyGSA	4,12E+02	1,76E+02	2,10E+02	3,46E+01	7,18E+01	5,02E+02	1,94E+09	4,02E+01
JuGGA	6,48E+02	9,27E+03	9,54E+03	1,10E+01	4,71E+02	4,83E+07	4,83E+07	1,43E+01
JuDE	6,34E+02	9,33E+03	9,51E+03	1,14E+01	4,57E+02	3,88E+07	3,88E+07	1,42E+01
JuPSOG	6,38E+02	9,34E+03	9,64E+03	1,15E+01	4,51E+02	4,63E+07	4,63E+07	1,43E+01
JuGSA	6,71E+02	9,49E+03	9,62E+03	1,15E+01	4,94E+02	1,96E+08	1,96E+08	1,43E+01
Dimension: 30 Rango [-100,0 100,0] MaxEvals 300000								

Tabla 32: Error Promedio Julia - Python F9-F16 / D30

	FUNCIONES							
	9	10	11	12	13	14	15	16
PyGGA	1,29E+03	1,73E+04	1,79E+04	1,08E+01	1,28E+01	8,74E+02	5,06E+08	7,42E-01
PyDE	2,11E+03	1,83E+04	1,83E+04	8,05E+00	2,27E+01	1,81E+03	4,55E+09	6,67E-01
PyPSOG	4,83E+02	5,78E+03	7,08E+03	1,02E+01	5,34E+00	1,98E+02	7,81E+07	6,80E-01
PyGSA	4,90E+02	1,74E+02	2,02E+02	3,48E+01	5,68E+01	4,20E+02	3,65E+09	4,03E+01
JuGGA	1,23E+03	1,65E+04	1,70E+04	1,27E+01	7,54E+02	1,88E+08	1,88E+08	2,42E+01
JuDE	1,24E+03	1,66E+04	1,69E+04	1,32E+01	7,55E+02	2,25E+08	2,25E+08	2,41E+01
JuPSOG	1,26E+03	1,66E+04	1,68E+04	1,28E+01	7,82E+02	2,08E+08	2,08E+08	2,42E+01
JuGSA	1,25E+03	1,63E+04	1,68E+04	1,30E+01	8,03E+02	6,08E+08	6,08E+08	2,42E+01
Dimension: 50 Rango [-100,0 100,0] MaxEvals 500000								

Tabla 33: Error Promedio Julia - Python F9-F16 / D50

9. Conclusiones

El paquete desarrollado en Julia a cumplido la expectativas esperadas. El tiempo que invierten los algoritmos de este lenguaje en la optimización de funciones está por debajo de los desarrollados en Python y según la tendencia que siguen los tiempos, esta diferencia se hará más pronunciada conforme aumente la dimensión.

En cuanto a los resultados obtenidos, los algoritmos de nuestro paquete pueden competir con los algoritmos de los paquetes utilizados en Python aunque los errores esten un poco por encima de estos. Ajustando los parámetros adecuadamente a cada función, podríamos lograr mejorar mucho más los resultados.

El lenguaje de Julia, a pesar de ser de reciente creación, sobrepasa las expectativas de rendimiento esperadas al inicio de este trabajo. Es sorprendente la gran diferencia que existe con respecto a Python teniendo en cuenta que tienen una sintaxis muy similar, lo que a simple vista podría parecer que comparten tambien similitudes en cuanto aspectos de rendimiento se refiere.

En cuanto a la experiencia personal con el lenguaje, agradezco profundamente la rápidez de ejecución de este lenguaje, ya que al ser un lenguaje nuevo para mi, tuve que hacer modificaciones constantes en el codigo aún estando en fase de experimentación, y por ende, repetir los experimentos realizados. La cantidad de tiempo invertido en todos los experimentos realizados en este lenguaje, ya sean fallidos o no, no es para nada comparable con el coste de tiempo que requirió una única ejecución de los algoritmos en Python.

10. Referencias

[1] V. M. Krasnopolsky y M. S. Fox-Rabinovitz, «Complex hybrid models combining deterministic and machine learning components for numerical climate modeling and weather prediction», *Neural Networks*, vol. 19, n.º 2, pp. 122-134, 2006, doi: 10.1016/j.neunet.2006.01.002.

[2] J. Reyes, A. Morales-Esteban, y F. Martínez-Álvarez, «Neural networks to predict earthquakes in Chile», *Applied Soft Computing Journal*, vol. 13, n.º 2, pp. 1314-1328, 2013, doi: 10.1016/j.asoc.2012.10.014.

[3] A.-T. Nguyen, S. Reiter, y P. Rigo, «A review on simulation-based optimization methods applied to building performance analysis», *Applied Energy*, vol. 113, pp. 1043-1058, 2014, doi: 10.1016/j.apenergy.2013.08.061.

[4] MATLAB. (2018). *9.7.0.1190202 (R2019b)*. Natick, Massachusetts: The MathWorks Inc.

[5] R Core Team (2017). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

[6] J. Bezanson, A. Edelman, S. Karpinski, y V. B. Shah, «Julia: A Fresh Approach to Numerical Computing», *SIAM Rev.*, vol. 59, n.º 1, pp. 65-98, ene. 2017, doi: 10.1137/141000671.

[7] «index | TIOBE - The Software Quality Company». https://www.tiobe.com/tiobe-index/ (accedido jul. 02, 2020).

[8] S. Pemberton, «An Alternative Simple Language and Environment for PCs», *IEEE Software*, vol. 4, n.º 1, pp. 56-64, ene. 1987, doi: 10.1109/MS.1987.229797.

[9] «PEP 20 – The Zen of Python», *Python.org*. https://www.python.org/dev/peps/pep-0020/ (accedido jun. 27, 2020).

[10] K. F. Man, K. S. Tang, y S. Kwong, «Genetic algorithms: concepts and applications [in engineering design]», *IEEE Transactions on Industrial Electronics*, vol. 43, n.º 5, pp. 519-534, oct. 1996, doi: 10.1109/41.538609.

[11] J. H. Holland, «Outline for a Logical Theory of Adaptive Systems», *J. ACM*, vol. 9, n.º 3, pp. 297–314, jul. 1962, doi: 10.1145/321127.321128.

[12] D. Whitley, T. Starkweather, y C. Bogart, «Genetic algorithms and neural networks: optimizing connections and connectivity», *Parallel Computing*, vol. 14, n.º 3, pp. 347-361, ago. 1990, doi: 10.1016/0167-8191(90)90086-O.

[13] K. Stanley y R. Miikkulainen, «Evolving Neural Networks through Augmenting Topologies», *Evolutionary computation*, vol. 10, pp. 99-127, feb. 2002, doi: 10.1162/106365602320169811.l

[14] K. O. Stanley y R. Miikkulainen, «Efficient reinforcement learning through evolving neural network topologies», en *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, New York City, New York, jul. 2002, pp. 569–577, Accedido: jul. 07, 2020. [En línea].

[15] R. Storn, «On the usage of differential evolution for function optimization», jul. 1996, pp. 519-523, doi: 10.1109/NAFIPS.1996.534789.

[16] X. S. Han, H. B. Gooi, y D. S. Kirschen, «Dynamic economic dispatch: feasible and opti-

mal solutions», *IEEE Transactions on Power Systems*, vol. 16, n.º 1, pp. 22-28, feb. 2001, doi: 10.1109/59.910777.

[17] L. Lakshminarasimman y S. Subramanian, «Applications of Differential Evolution in Power System Optimization», en *Advances in Differential Evolution*, U. K. Chakraborty, Ed. Berlin, Heidelberg: Springer, 2008, pp. 257-273.

[18] H. Bremermann, «Chemotaxis and optimization», *Journal of the Franklin Institute*, vol. 297, n.º 5, pp. 397-404, may 1974, doi: 10.1016/0016-0032(74)90041-6.

[19] B. Mangaraj, I. Misra, y A. Barisal, «Optimizing included angle of symmetrical V dipoles for higher directivity using bacteria foraging optimization algorithm», *Progress in Electromagnetics Research B*, vol. 3, pp. 295-314, ene. 2008, doi: 10.2528/PIERB07121005.

[20] J. Kennedy y R. Eberhart, «Particle swarm optimization», en *Proceedings of ICNN'95 - International Conference on Neural Networks*, nov. 1995, vol. 4, pp. 1942-1948 vol.4, doi: 10.1109/ICNN.1995.488968.

[21] A. Babazadeh, H. Poorzahedy, y S. Nikoosokhan, «Application of particle swarm optimization to transportation network design problem», *Journal of King Saud University - Science*, vol. 23, n.º 3, pp. 293-300, jul. 2011, doi: 10.1016/j.jksus.2011.03.001.

[22] X.-S. Yang, *Nature-Inspired Metaheuristic Algorithms*. Luniver Press, 2008.

[23] E. Emary, H. M. Zawbaa, K. K. A. Ghany, A. E. Hassanien, y B. Parv, «Firefly Optimization Algorithm for Feature Selection», en *Proceedings of the 7th Balkan Conference on Informatics Conference*, Craiova, Romania, sep. 2015, pp. 1–7, doi: 10.1145/2801081.2801091.

[24] E. Rashedi, H. Nezamabadi-pour, y S. Saryazdi, «GSA: A Gravitational Search Algorithm», *Information Sciences*, vol. 179, n.º 13, pp. 2232-2248, jun. 2009, doi: 10.1016/j.ins.2009.03.004.

[25] E. Rashedi, H. Nezamabadi-pour, y S. Saryazdi, «Filter modeling using gravitational search algorithm», *Engineering Applications of Artificial Intelligence*, vol. 24, n.º 1, pp. 117-122, feb. 2011, doi: 10.1016/j.engappai.2010.05.007.

[26] J. Liang, B.-Y. Qu, P. Suganthan, Problem definitions and evaluation criteria forthe CEC 2014 special session and competition on single objective real-parameternumerical optimization, Tech. rep., Zhengzhou University and Nanyang Tech-nological University (2013).

[27] S. Das y P. N. Suganthan, «Differential Evolution: A Survey of the State-of-the-Art», IEEE Trans. Evol. Computat., vol. 15, n.º 1, pp. 4-31, feb. 2011, doi: 10.1109/tevc.2010.2059031.

[28] Tonda, A. Inspyred: Bio-inspired algorithms in Python. *Genet Program Evolvable Mach* 21, 269–272 (2020). https://doi.org/10.1007/s10710-019-09367-z

[29] A. Deepanshu, Gravitational-Search-Algorithm (2019), GitHub repository, https://github.com/deepanshu1999/Gravitational-Search-Algorithm

[30] M. Mitchell, «Genetic algorithms: An overview», Complexity, vol. 1, n.º 1, pp. 31-39, sep. 1995, doi: https://doi.org/10.1002/cplx.6130010108.

[31] M. A. T. Estrada, «Bacterial Foraging Optimization Algorithm (BFOA)». Unpublished, 2015, doi: https://doi.org/10.13140/RG.2.2.10053.24803.

[32] M. Imran, R. Hashim, y N. E. A. Khalid, «An Overview of Particle Swarm Optimization Variants», Procedia Engineering, vol. 53, pp. 491-496, 2013, doi: https://doi.org/10.1016/j.proeng.2013.02.063.

[33] N. M. Sabri, M. Puteh, y M. R. Mahmood, «An overview of Gravitational Search Algorithm utilization in optimization problems», presented at the 2013 IEEE 3rd International Conference on System Engineering and Technology (ICSET), ago. 2013, doi: https://doi.org/10.1109/ICSEngT.2013.6650144.