



ugr | Universidad
de Granada

TRABAJO FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA

ALGORITMOS BIOINSPIRADOS CON INTERACCIONES NEGATIVAS

Autor

Carmen Biedma Rodríguez

Directores

Daniel Molina Cabrera
Francisco Herrera Triguero



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE
TELECOMUNICACIÓN

—
Granada, Septiembre de 2018

Algoritmos Bioinspirados con Interaciones Negativas

CARMEN BIEDMA RODRIGUEZ

Palabras clave: Metaheuristica, interacción, poblacional, bioinspirado, atracción, repulsión, optimización

Resumen

En cualquier metaheurística, sabemos que hay un aspecto muy importante a tener en cuenta para su buen funcionamiento: la diversidad. Muchas son las técnicas utilizadas para conseguir dicha calidad, pero en los últimos años se ha comenzado a utilizar la llamada repulsión. Con ella conseguiremos que nuestro conjunto de soluciones mantenga la diversidad en todo momento, consiguiendo así mejorar las soluciones finales y evitando el estancamiento en óptimos locales.

Este proyecto está enfocado a estudiar la influencia de dos técnicas recientes en el ámbito de la metaheurística, éstas son la atracción y la repulsión. Dichos métodos están estrechamente relacionados con la interacción entre soluciones, ya que hablamos de atracción cuando dos soluciones o más se atraen entre sí y de repulsión cuando se repelen. Para ello, es necesario tener una población de soluciones ya que si tuviésemos una sola no podría interaccionar con ninguna más. Es por ello por lo que se ha decidido ver su aplicación en algoritmos de tipo poblacional y, concretamente, de carácter bioinspirado. Los algoritmos basados en el comportamiento de seres vivos son bastante adecuados para este estudio porque poseen una gran interacción entre individuos ya que en la vida real, también es así.

Los algoritmos que se han elegido para trabajar han sido:

- **Grasshopper Optimisation Algorithm:** Basado en el comportamiento de los saltamontes que se atraen entre ellos para moverse en colonia a la hora de buscar comida, además de ser fuertemente atraídos por el que más cerca se encuentra de la comida.
- **Firefly Algorithm:** Implementa la tendencia que tienen las luciérnagas de moverse junto a las que más brillan.

- **Dragonfly Algorithm:** Refleja cómo las libélulas se ven atraídas por la más prometedora a la hora de buscar alimento y repelidas por los depredadores.
- **Bacterial Foraging Algorithm:** Este algoritmo está inspirado en el comportamiento de la bacteria E-Coli a la hora de verse atraída por las zonas con más nutrientes del ser humano y repelida por las más dañinas para ellas.

Como vemos, hay algunos algoritmos que poseen tanto atracción como repulsión mientras que otros solamente poseen atracción. Esto nos ayudará a comprobar si realmente ayuda tener una componente repulsiva o no. Cabe destacar que una de las características de todos ellos es que tanto la atracción como la repulsión viene determinada por la distancia a las soluciones que se toman como referencia. Esto nos permite comparar con otros algoritmos como PSO que tienen en cuenta otras soluciones pero no se guían por la distancia a ellas.

Las aplicaciones de este tipo de metaheurísticas son de lo mas variadas, pero en éste caso nos centraremos en su eficacia en el ámbito de la optimización. Son algoritmos que obtienen muy buenos resultados en dicho campo y además los tiempos de ejecución son muy aceptables. Para ello se utilizará el benchmark propuesto en la competición de optimización global en el CEC 2014.

En resumen, el propósito final de este trabajo es ver el comportamiento de los algoritmos elegidos en funciones complejas. Una vez obtenidos estos resultados se realizará un análisis y un balance de los mismos. Como conclusión se pretende determinar si de cara al futuro las nuevas metaheurísticas deberían tener en cuenta utilizar la repulsión en sus comportamientos.

Bioinspired Algorithms Using Negative Interactions

CARMEN BIEDMA RODRIGUEZ

Key words: Metaheuristic, interaction, swarming, bio-inspired, attraction, repulsion, optimization

Abstract

We know that all metaheuristics have a very important aspect to consider if we want good results, the diversity. There are a lot of techniques to provide this quality, but a few years ago appeared once called repulsion. With this component we will achieve that our set of solutions keep the diversity every moment, consequenquentially we will have better solutions and the avoidance of local minimums.

The aim of this project is to study the influence of two techniques in the court of metaheuristics called attraction and repulsion. Those methods are very attached to the interaction between solutions. That is because we talk about attraction of two or more solutions when they attract each other and about repulsion when they repell each other.

One important fact is that we need at least two solutions to have interaction. That is the reason why we choose to apply this techniques with swarming algorithms, and specifically in bio-inspired ones. Those algorithms based in alive criatures are very apropiate because they hace a lot of interaction between individes as well as happens in the real life.

The algorithms we have choosen are this ones:

- **Grasshopper Optimisation Algorithm:** Based in the swarming behaviour of grasshoppers, that attract one to the others to look for food .
- **Firefly Algorithm:** Implements the trend of firelies to move around those ones that shine brighter.
- **Dragonfly Algorithm:** Reflect how the dragonflies are attracted to the one that is nearest the food and they run off predators.

- **Bacterial Foraging Algorithm:** This algorithm is inspired in the behaviour of E-Coli bacteria. It is attracted by the zones of the humans that have more nutrients and repelled from those ones that have noxious sustances for him.

We realize that we have algorithms that have attraction as well as repulsion instead the others ones that only have attraction. This will help us to check if repulsion really improve the result of our algorithms. It is important to say that one of the characteristic of this tecnicas is that they are determined by the distance between the solutions that we consider as reference. This let us compare with other algorithms like PSO that have interaction between solutions but without using the distance.

The aplications of this type of metaheuristic are very varied but we will focus the investigation in the effectiveness with optimization problems. This algorithms get very well results in this area and the execution times are really acceptable. To prove this, we will use the benchmark proposed in the global competition of optimization in CEC 2014.

Finally, the propose of this work is to check the behaviour of the choosen algorithms using complex functions. When we have obtained the results, we will analize them and see what are the best. To conclude, we purport to determine if in the future we should use this repulsion tecnicas in the implementation of new metaheuristic.

Yo, **Carmen Biedma Rodríguez**, alumna de la titulación GRADO EN INGENIERIA INFORMATICA de la Escuela Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada, con DNI 26510125A, autorizo la ubicación de la siguiente copia de mi Trabajo de Fin de Grado a la biblioteca del centro para ser consultada por las personas que lo deseen.

Fdo: Carmen Biedma Rodríguez

Granada a 6 de Septiembre de 2018

D. **Daniel Molina Cabrera (tutor1)**, Profesor del departamento Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada

D. **Francisco Herrera Triguero (tutor2)**, Profesor del departamento Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada

Informan:

Que el presente trabajo, titulado **Algoritmos Bioinspirados con interacciones negativas**, ha sido realizado bajo su supervisión por **Carmen Biedma Rodriguez**, y autorizo la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a X de mes de 2018.

Los directores:

Daniel Molina Cabrera y Francisco Herrera Triguero

Agradecimientos

En primer lugar, quiero agradecer y dedicar este proyecto a mis padres porque gracias a ellos hoy he sido capaz de llegar hasta este punto de mi vida. También por aguantarme y apoyarme incondicionalmente estos años de carrera.

El segundo agradecimiento es para mi hermana, por ser un pilar fundamental en mi vida y estar conmigo siempre que lo he necesitado.

Otras personas a las que le debo muchos de mis logros a lo largo de esta carrera es a mis amigas del colegio mayor, pero especialmente a Alba. Seguramente ella sea la que más sufrido en primera persona mis agobios y llantos los días malos y la que ha sabido mejor que nadie tranquilizarme de una forma especial.

Y por último, a mi tutor Daniel quiero agradecerle toda la ayuda y paciencia que me ha brindado durante la realización de este proyecto.

Índice

1	Introducción	11
2	Planificación	13
2.1	Tareas realizadas	13
2.2	Tiempo dedicado	14
2.3	Precios	14
3	Revisión de la literatura: atracción y repulsión como técnicas	16
4	Análisis de los algoritmos	20
4.1	Algoritmos implementados con atracción	20
4.1.1	Grasshopper Optimization Algorithm	20
4.1.2	Firefly Algorithm	22
4.2	Algoritmos implementados tanto con atracción como con repulsión	24
4.2.1	Dragonfly Algorithm	24
4.2.2	Bacterial Foraging Algorithm	29
5	Implementación	34
5.1	Requisitos	34
5.2	Instalación	35
5.3	Representación de los datos	35
6	Diseño experimental	37
6.1	Benchmark	37
6.2	Parámetros importantes	37
6.3	Comparativas a realizar	38
6.4	Representación de los resultados	38
7	Análisis de resultados	40
7.1	Comparativa de la componente atractiva para dimensión 10	40
7.2	Comparativa de la componente repulsiva para dimensión 10	51
7.3	Comparativa de la componente atractiva para dimensión 30	57
7.4	Comparativa de la componente repulsiva para dimensión 30	63
7.5	Comparativa entre algoritmos	66
8	Conclusiones	73

Índice de tablas

2.1.	Planificación genérica inicial	14
------	--	----

Índice de figuras

4.1.	Componentes del algoritmo Dragonfly.	27
4.2.	Ejemplo de reproducción	32
5.1.	Modelo de solución.	36
7.1.	Comparativa de atracción algoritmo GOA con dimensión 10.	41
7.2.	Función 11. Shifted Scherfel's Function	42
7.3.	Función 15. Shifted and Rotated Expanded Griewank's plus Rosenbrock's Function	43
7.4.	Comparativa de repulsión algoritmo FA con dimensión 10.	45
7.5.	Comparativa de atracción algoritmo DA con dimensión 10.	47
7.6.	Gráfica 3D de la función 26.	48
7.7.	Comparativa de atracción algoritmo BFA con dimensión 10.	50
7.9.	Mejores resultados sin repulsión DA.	52
7.8.	Comparativa de repulsión algoritmo DA con dimensión 10.	53
7.10.	Comparativa sin repulsión y ponderada 0.75 de DA en dimensión 10.	54
7.11.	Comparativa de repulsión algoritmo BFA con dimensión 10.	56
7.12.	Comparativa de atracción algoritmo GOA con dimensión 30.	58
7.13.	Comparativa de atracción algoritmo FA con dimensión 30.	59
7.14.	Comparativa de atracción algoritmo DA con dimensión 30.	61
7.15.	Comparativa de atracción algoritmo BFA con dimensión 30.	62
7.16.	Comparativa de repulsión algoritmo DA con dimensión 30.	64
7.17.	Comparativa de repulsión algoritmo BFA con dimensión 30.	65
7.18.	Comparativa de los algoritmos con parámetros del autor en dimensión 10.	67
7.19.	Comparativa de los algoritmos con mejores factores de atracción-repulsión en dimensión 10.	68
7.20.	Gráficas de las funciones 4 y 7.	69
7.21.	Gráfica 3D de la función 11.	69
7.22.	Comparativa de los algoritmos dados por el autor en dimensión 30.	71
7.23.	Comparativa de los mejores algoritmos conseguidos en dimensión 30.	72

Capítulo 1: Introducción

1. Introducción

A lo largo de la historia, se ha demostrado que los algoritmos evolutivos funcionan muy bien en problemas de optimización. Estos tipos de metaheurísticas dan buenas soluciones a problemas complejos en un tiempo de computación razonable. En este tipo clasificaríamos los conocidos algoritmos genéticos, así como PSO, DE y muchos algoritmos bioinspirados como los implementados en este proyecto. [1]

La gran mayoría de los algoritmos evolutivos son poblacionales. Esto significa que utilizan un conjunto de soluciones al problema que de alguna forma influirán unas sobre las otras para conseguir evolucionar hacia las soluciones óptimas. Esta influencia puede ser de muchos tipos, por ejemplo, en los genéticos consiste en una combinación a pares de las soluciones. Otro tipo de interacción entre soluciones, que es el que trataremos en este trabajo, son la atracción y la repulsión. [2]

Cuando hablamos de *atracción* tendremos una componente en nuestro algoritmo que atraerá a las soluciones entre sí. Dentro de los criterios de atracción, podemos distinguir dos que subdividen este tipo de técnica: la atracción guiada solamente por el mejor de la población y influenciada por todas las demás o parte de ellas. Un ejemplo del primer comportamiento es el algoritmo PSO, en el cuál todas las partículas se guian por aquella que está más cerca del óptimo global [3]. Por otra parte, podemos ampliar el campo de influencia y no sólo tener la mejor en cuenta. De esta manera, podemos considerar las X mejores soluciones de la población, un vecindario de soluciones menor que la población completa y considerar tanto las mejores como las peores, etc.

Además de qué soluciones vamos a tomar como atractivas, tenemos que tener en cuenta qué característica de ellas utilizaremos para realizar la modificación de nuestras soluciones. En este caso, nosotros utilizaremos la distancia que hay de la solución a las demás. Esta componente nos permite evolucionar de forma correcta nuestras soluciones.

Por otra parte, la *repulsión* nos permite evitar de alguna manera que el algoritmo se estanke en óptimos locales. Para ello, haremos lo contrario que con la técnica de atracción, es decir, mantendremos siempre una distancia entre las soluciones repeliendo unas de las otras. De esta manera obtendremos una mayor diversidad y, por consiguiente, una mayor exploración del espacio de búsqueda.

En los últimos años se han propuesto muchos algoritmos que utilizan dichas técnicas, algunos de ellos son implementados y estudiados más adelante. [11]

El objetivo del trabajo es evaluar si estos algoritmos dan buenos resultados y si las técnicas contribuyen a que las soluciones sean mejores, de cara a seguir utilizándolas en un futuro. Veremos como se comportan los algoritmos en el ámbito de la optimización utilizando un benchmark y compararemos cuáles de ellos son mejores. De cara a ver la influencia de las técnicas, se realizarán variaciones de los algoritmos usando ésta componente como parámetro de estudio.

Capítulo 2: Planificación

2. Planificación

La planificación previa de éste proyecto se ha realizado siguiendo una metodología ágil. Es el modelo de planificación que más se ajusta a él ya que a priori se desconoce la dificultad de las tareas a realizar.

2.1. Tareas realizadas

- **Planteamiento y comprensión del problema:** Revisión del trabajo a realizar y reuniones con el tutor para comprender mejor y aclarar todos los matices del TFG.
- **Búsqueda de información y lecturas:** Búsqueda y lectura comprensiva de todos los artículos y documentos necesarios para la realización del proyecto.
- **Planificación del proyecto:** Se planificaron algunos aspectos iniciales, así como las tareas que eran necesarias inicialmente.
- **Comprensión de los algoritmos a implementar:** Búsqueda de información sobre los algoritmos que se van a implementar y lectura comprensiva de los papers. El fin de esta fase es prepararse para la posterior implementación.
- **Implementación:** Fase de implementación de los algoritmos propuestos. En esta parte también se incluyen muchas reuniones para ver el avance en la implementación y corrección de fallos.
- **Diseño experimental:** En esta fase, posterior a la programación y visto el comportamiento de los algoritmos, se decide cómo se van a estudiar las características de ellos que nos interesan. Se eligió un benchmark, las ejecuciones que se iban a realizar para cada prueba, la estructura de las tablas de resultados, etc.
- **Ejecución de la batería de pruebas:** Acorde a lo previamente planificado, se ejecutan los algoritmos con todas las funciones propuestas. Se recogen los datos y se crean las tablas correspondientes.¹.

¹Se especificarán 2 tiempos, el de ejecución y el de preparación de las pruebas.

- **Análisis de los resultados:** Con los resultados obtenidos en las pruebas, se realiza un estudio de cara a dar una explicación de los mismos.
- **Desarrollo de la documentación:** El desarrollo de la memoria es algo que se ha realizado a lo largo de todo el proyecto. Aquí se va plasmando todo el trabajo que se va realizando.
- **Revisión del proyecto:** Una vez terminado el trabajo, se ha hecho una revisión exhaustiva tanto de memoria como de códigos.

2.2. Tiempo dedicado

Tabla 2.1: Planificación genérica inicial

TAREA	TIEMPO
Planteamiento y comprensión del problema	10 horas
Búsqueda de información y lecturas	10 horas
Planificación del proyecto	2 horas
Comprendión de los algoritmos a implementar	30 horas
Implementación	70 horas
Elegir benchmark y concretar las pruebas	1 hora
Ejecución de la batería de pruebas	(3 + 192 horas)
Análisis de los resultados	20 horas
Desarrollo de la memoria	30 horas
Revisión del proyecto	10 horas

2.3. Precios

Si quisiésemos valorar económicoamente el proyecto tenemos que tener en cuenta dos aspectos: el precio de la mano de obra y el de cómputo si tuviésemos que pagarlos.

El precio de la mano de obra son 20€ la hora. Para poder estimar el costo que tendría el cómputo realizado en este proyecto buscaré un servicio en Amazon Web Services que proporcione las mismas prestaciones que mi ordenador. El ordenador portátil usado para la realización de las ejecuciones ha sido un MSI-GP62-2QE Leopard Pro con un procesador Intel® Core™ i7-5700HQ CPU @ 2.70GHz × 8 y 8GB de RAM. Aunque sabemos que

en servidores el rendimiento no es igual que el de un portátil, seleccionaré el que más se asemeje a dichas características. En servidores de cómputo podemos ver que hay una instancia llamada t2.small que podría ser similar a nuestro ordenador aunque tiene menos memoria. Tiene una única CPU que funciona a una velocidad de hasta 2.5GHz y cuesta 0,023 USD por hora. Si hacemos un calculo de las horas que nos ha llevado ejecutar todo el proyecto nos sale que $192h * 0.023\text{USD} = 4,416$ USD, que al cambio actual de la moneda son 3,79 €.

Capítulo 3: Revisión de la literatura: atracción y repulsión como técnicas

3. Revisión de la literatura: atracción y repulsión como técnicas

Si hacemos un repaso a lo largo de la historia de la metaheuristica, podemos observar que los algoritmos que poseen características similares a los desarrollados en este trabajo se han utilizado en infinidad de campos. En concreto, la mayoría son problemas de optimización aplicados a muchos ámbitos. Por lo general, dichos problemas son NP-Duros y se ha comprobado que los resultados tanto en calidad como en tiempo son muy buenos. Después de hablar de las técnicas como tal, se hará una revisión de algunas de las aplicaciones reales que han tenido.

El primer algoritmo en el que podemos observar la técnica de la atracción es en el Particle Swarm Optimization. En este caso, las soluciones se guiaban y se veían atraídas por la mejor global o local. Este concepto se ha generalizado y se ha extendido con muchas variantes, como por ejemplo tener en cuenta no solo la mejor solución sino todas ellas, o teniendo en cuenta una penalización de la distancia a la que se encuentran las soluciones.

Todo algoritmo de optimización debe tener un mecanismo de mejora o modificación de las soluciones, ya sea para maximizar o minimizar. Estos métodos pueden ser muy variados, desde una combinación de soluciones, como por ejemplo en los genéticos, hasta un desplazamiento de ellas. Este último caso es en el que englobaríamos estas dos técnicas de generación de nuevas soluciones.

Sea cual sea el tipo de mecanismo que tengamos, hay que tener claro que además necesitamos saber qué soluciones queremos tener en cuenta para modificar la que estamos estudiando en un momento concreto. Para ello podremos usar las mejores, usarlas todas, usar solamente la mejor, etc. Cualquiera de estos criterios se pueden aplicar para

ambas técnicas, pero es importante saber en qué caso usaremos las mejores y las peores soluciones.

En el caso de la atracción, está claro que utilizaremos siempre soluciones que sean mejores, ya que guiar la actualización de una solución hacia otra peor no tendría validez alguna. Esto nos hace tener una mayor explotación de los mejores entornos de búsqueda pero, como veremos a continuación, también nos hace perder la diversidad.

Con respecto a la repulsión, no está tan claro si se utilizarán las mejores, las peores o todas ellas, esto dependerá de la funcionalidad que queramos darle a la componente. En un primer pensamiento, podemos decir que usaríamos la repulsión para alejar las soluciones de la peor y que, además, se usa de forma frecuente en este tipo de algoritmos. Por otra parte, la repulsión también se utiliza para mantener la diversidad de las soluciones que vamos perdiendo conforme avanza el algoritmo. De esta manera, no solo deberíamos alejarlas de los peores, sino también de las que son mejores que ellas para tener una mayor exploración del entorno.

Ambas técnicas necesitan forzosamente de una población para ser aplicadas. De otra manera no tendríamos soluciones que atraer ni que repeler. Por ello, en el ámbito que más se utilizan es en el de las metaheurísticas de carácter bioinspirado que, por lo general, son poblacionales.

A continuación, se dará una breve explicación de cada algoritmo y veremos alguna aplicación importante en algún problema resuelto con los mismos.

El primero que comentaré es el algoritmo Bacterial Foraging Algorithm, que refleja el comportamiento de la bacteria E-coli. Su funcionamiento de basa en la manera que tiene dicha bacteria para ir hacia sitios con nutrientes (atracción) y huir de los que poseen toxinas (repulsión). BFA se ha utilizado mucho en problemas de optimización desde que se desarrolló por primera vez en 2007. Los resultados que se obtenían eran buenos pero el carácter aleatorio que tiene hacia que en otro tipo de problemas no funcionase tan bien..

En el año 2008, Ahmed Y. Saber y Ganesh K. Venayagamoorthy publicaron un artículo en el que proponían eliminar la aleatoriedad de las soluciones. En su lugar, las

partículas serían guiadas por las mejores soluciones proporcionadas por un PSO. Esto se aplicó para resolver el problema ELD (Economic Load Dispatch) y los resultados fueron bastante buenos [4]. En el mismo año, además de para resolver el problema ELD, el algoritmo BFA se utilizó para muchas otras aplicaciones. En una publicación realizada en septiembre podemos ver como se utilizó en el ámbito de la electrónica para diseñar sistemas estabilizadores de potencia. En éste caso los resultados fueron buenos pero no tanto como los que proporcionó el algoritmo SPPSO (Small Population PSO). [5]. El problema OPF (Optimal Power Flow) también se ha conseguido resolver con unos resultados bastante prometedores utilizando el algoritmo BFA. En este caso también se hace una hibridación del algoritmo llamada DBFA (Dynamic Bacterial Foraging Algorithm). La alteración del algoritmo consiste en adaptar las bacterias cuando los ambiente son dinámicos. [6]

Si hablamos del algoritmo Grasshopper Optimization Algorithm, tenemos que decir que se basa en el comportamiento de los saltamontes que se atraen unos a otros según estén mas cerca de la comida. Fue propuesto por primera vez en el año 2007 al igual que el anterior. Originalmente se planteó para resolver problemas de optimización, pero con el tiempo ha tenido bastantes aplicaciones de distinto tipo.

En el año de su aparición se implementó una variante llamada AGOA cuya primera inicial le añade al original la característica de adaptativo. En este caso se usó para optimización de trayectorias en placas en aviones que utilizan placas solares para su movimiento. Estos necesitan de unas trayectorias específicas para obtener la mayor energía posible del sol, por tanto no deja de ser un problema de optimización. Las comparaciones realizadas se hicieron con respecto al GOA original, GWO y PSO, pero el que mejores resultados produjo fue el propuesto en dicho paper. [7]. Al año siguiente (2008), además de otros muchos, se publicaron dos artículos que también hacían referencia al algoritmo GOA para resolver problemas. El primero pertenece a uno de los propuestos en la Conferencia Internacional de Ingenierías Eléctrica y Electrónica y trata la aplicación de dicho algoritmo en reguladores automáticos de voltaje para determinar la proporción óptima entre los modos integral y derivativo. Los resultados obtenidos fueron óptimos en la mayoría de los casos. [8]

Otro algoritmo a mencionar es el Firefly Algorithm. Como todos los demás también

se basa en el comportamiento de un ser vivo, en este caso las luciérnagas. En ellas podemos observar un comportamiento de atracción hacia la que más brilla, por lo que será la componente atractiva que utilizaremos en nuestro algoritmo. Anteriormente se ha mencionado el problema ELD, que se ha conseguido también resolver con dicho algoritmo [9] en el año 2011.

Para terminar, el algoritmo Dragonfly ha sido otro de los implementados en este proyecto. En concreto éste se basa en el comportamiento de las libélulas y se puede observar tanto componente atractiva como repulsiva. La atracción viene dada por el individuo más cercano a la comida (el mejor) y la repulsión por el depredador (la peor solución). Además de estos dos tipos de interacciones entre soluciones, veremos que entre ellas tienen muchos más. En el año 2016 dicho algoritmo se utilizó para resolver el problema ECED (economic emission dispatch problem) y sus resultados podemos verlos en la referencia [10].

Como podemos ver, las posibilidades que nos brindan estos algoritmos son muy variadas, desde problemas económicos hasta electrónicos. Además, una buena opción que vemos que funciona es hibridar los algoritmos con otras técnicas para poder adaptarlos a todo tipo de aplicaciones según nos convenga.

Capítulo 4: Análisis de los algoritmos

4. Análisis de los algoritmos

En esta sección se da una explicación detallada del funcionamiento de los algoritmos implementados. Todos ellos son algoritmos bioinspirados, es decir, que reflejan comportamientos de seres vivos. Además, tenemos 2 bloques de algoritmos: los que poseen atracción y repulsión y los que solo poseen atracción.

4.1. Algoritmos implementados con atracción

En primer lugar se describirán los algoritmos desarrollados que solo tienen atracción y no repulsión. Los dos propuestos con estas características son el Grasshopper Optimization Algorithm y el Firefly Algorithm.

4.1.1. Grasshopper Optimization Algorithm

Al igual que todos los algoritmos bioinspirados, Grasshopper Optimization Algorithm está basado en el comportamiento de un ser vivo, concretamente los saltamontes. Estos se ven atraídos por los demás saltamontes que hay a su alrededor guiándose por la distancia a la que se encuentran. Además, también tienen una atracción hacia el mejor de todos para no perder el camino hacia el óptimo. [16]

Para empezar a hablar de este algoritmo, hay que decir que partimos de un conjunto de soluciones que representarán una posición dentro del dominio de una función. Los fitness de dichas soluciones vienen calculados por una función evaluación que queremos optimizar.

En primer lugar, tendremos una parte de inicialización en la que se creará una población aleatoria de soluciones, se calculará cada uno de los fitness y guardaremos el mejor

de todos ellos.

Para cada una de nuestras soluciones, lo primero que haremos será normalizar las distancias hacia los otros individuos como se ve en 4.1. Para realizar esta normalización utilizaremos la distancia a pares en la dimensión. Es decir, que para cada par de coordenadas de la solución tendremos la misma distancia normalizada. De esta manera, calcularemos todas con la siguiente fórmula:

$$Normalizada_{i,j,d} = \sqrt{(X_{j,k} - X_{i,k})^2 + (X_{j,k+1} - X_{i,k+1})^2} \quad (4.1)$$

siendo X las posiciones de cada una de las soluciones y d la dimensión en la que estamos calculando la distancia.

Una vez que hemos normalizado todas las distancias, actualizaremos la posición de todas las soluciones. Para ello utilizaremos la siguiente fórmula:

$$X_i^d = c \left(\sum_{j=1}^N c \frac{\maxPos - \minPos}{2} s * (|x_j^d - x_i^d|) \frac{x_j^d - x_i^d}{Normalizada_{i,j}} \right) + Best^d \quad (4.2)$$

en la cual, \maxPos y \minPos representan los límites superior e inferior tomados para el dominio de la función, $Best$ la mejor solución encontrada hasta el momento y s nos define la fuerza de la interacción social y viene dado por la siguiente fórmula:

$$s = fe^{-\frac{r}{l}} - e^{-r} \quad (4.3)$$

siendo r la distancia entre las dos soluciones.

Otra de las componentes que encontramos en la ecuación utilizada para actualizar la posición es la variable c . Ésta representa la amplitud del paso que se va a dar, es decir, el tamaño de la variación que va a sufrir la solución. El paso se vuelve más pequeño conforme se avanza el número de iteraciones para favorecer la explotación al final y la exploración al principio.

$$c = cmax - l \frac{cmax - cmin}{L} \quad (4.4)$$

tomando l como la iteración en la que estamos y L el número máximo de iteraciones. cmax y cmin son constantes que en este caso han tomado el valor 1 y 0.0000001 respectivamente.

Un aspecto que hay que tener en cuenta es que cuando se modifica una solución, ésta puede quedar fuera de los límites establecidos en el dominio, por lo que tendremos que devolverla dentro de ellos. En este caso si salen del dominio se sustituye el valor por uno aleatorio.

Inicializar las soluciones aleatoriamente

$$X_i (i = 1, 2, \dots, n)$$

```
while condición de parada no satisfecha do
    Actualizar valor de c Eq 4.4
    Normalizar las distancias Eq 4.1
    Modificar las posiciones Eq 4.2
    Actualizar el mejor si fuese necesario
end
```

Algorithm 1: Pseudocódigo de Grasshopper Optimization Algorithm

4.1.2. Firefly Algorithm

Este algoritmo está inspirado en el comportamiento de las luciernagas. Éstas se atraen unas a las otras debido a su luminosidad, siempre a las que brillan más que ellas. En nuestro caso, la luminosidad estará representada por la función de evaluación. De esta manera, una solución se verá atraída por otra siempre que sea mejor, es decir, que su valor en la función sea más pequeño ya que nuestro objetivo es minimizar. [17]

En primer lugar, inicializaremos las soluciones aleatoriamente al igual que en los casos anteriores y calcularemos su fitness acorde a la función que estemos evaluando.

La atracción en este algoritmo viene dada por la distancia a las soluciones mejores,

por tanto solo modificaremos las posiciones utilizando las que son mejores que ella. La distancia que usaremos será la cartesiana, calculada de la siguiente manera:

$$Distancia_{ij} = \sqrt{\sum_{k=1}^{\text{dim}} (X_{j,k} - X_{i,k})^2} \quad (4.1)$$

al igual que en los casos anteriores, X representa las posiciones y dim la dimensión en la que estamos trabajando.

Acorde a dicha distancia, calcularemos las nuevas posiciones de nuestras soluciones. Para ello, utilizaremos una componente atractiva y además una componente aleatoria para no perder la diversidad de las mismas. La ecuación para ello es la siguiente:

$$X_i = X_i + e^{-r_{ij}^2} * (X_j - X_i) + \alpha * rand \quad (4.2)$$

siendo α un valor igual a 0.2 y $rand$ un vector aleatorio del tamaño de la dimensión.

Inicializar las soluciones aleatoriamente

$$X_i (i = 1, 2, \dots, n)$$

Calcular el valor fitness de cada solución;

while condición de parada no satisfecha **do**

for i in TodasLasSoluciones **do**

for j in TodasLasSoluciones **do**

if $\text{fitness}(j) < \text{fitness}(i)$ **then**

end

 Calcular distancia de i a j . Eq 4.1

 Actualizar posición. Eq 4.2

 Si se sale de los límites, generar una nueva aleatoriamente

 Calcular nuevo fitness

 Actualizar mejor si fuese necesario

end

end

end

Algorithm 2: Pseudocódigo de Firefly

4.2. Algoritmos implementados tanto con atracción como con repulsión

A continuación estudiaremos los algoritmos que además de atracción presentan repulsión en su comportamiento. Dichos algoritmos son Dragonfly Algorithm y Bacterial Foraging Algorithm.

4.2.1. Dragonfly Algorithm

Dragonfly Algorithm está inspirado en el comportamiento de las libélulas. En él podemos observar tanto una componente atractiva como repulsiva [11]. Como veremos, la componente atractiva se realizará hacia la mejor de ellas, mientras que la repulsiva intentará alejarlas de la peor.

En primer lugar, partimos de un conjunto de soluciones que representarán las *posiciones* de nuestras libélulas. Dichas soluciones tendrán que estar dentro de los límites de nuestra función evaluación, que será la que nos dé el valor fitness de las soluciones.

Además, haremos uso de un vector *step* que determinará la amplitud del movimiento de las soluciones y la dirección del mismo.

En cada iteración del algoritmo se actualizarán las soluciones haciendo uso de las componentes que se mencionan a continuación y dependiendo de los vecinos que posean. Los vecinos de una solución serán aquellas soluciones que disten de ella menos de un radio *r* que irá incrementando su valor a medida que avanza el algoritmo. La ecuación que ajusta dicho valor viene dada por la siguiente expresión:

$$r = D/4 + (\text{iter}/\text{maxIter}) * D * 2 \quad (4.1)$$

en la que *D* representa el dominio de la función a optimizar, *iter* representa la iteración actual y *maxIter* el total de iteraciones.

Las nuevas soluciones resultantes de cada iteración dependerán de si la mejor solución pertenece a sus vecinos o no y de que su número de vecinos sea mayor que 1. La ecuación general que utilizaremos será la siguiente:

$$X_{i+1} = X_i + Step_{i+1} \quad (4.2)$$

siendo *i* la iteración actual e *i+1* la siguiente.

El vector de movimiento (*step*) será el que guiará la dirección del movimiento, por lo que aquí es donde aplicaremos los términos de repulsión y atracción. La actualización de dicho vector se realizará mediante las siguientes componentes:

- **Separación (*S*):** Componente que se utilizará para mantener las distancias de las soluciones :

$$S_i = - \sum_{j=1}^n X_i - X_j \quad (4.3)$$

donde, *n* es el número de vecinos de la solución *i*

- **Alineamiento (*A*):** Alineará las direcciones del movimiento de las soluciones:

$$A_i = \frac{\sum_{j=1}^n Step_j}{n} \quad (4.4)$$

- **Cohesión (C)** : Tendencia que tendrán los individuos al centro del vecindario:

$$C_i = \frac{\sum_{j=1}^n X_j}{n} - X_i \quad (4.5)$$

- **Atracción hacia el mejor (F)** : Ésta es una de las componentes del estudio realizado. Las soluciones tendrán una atracción hacia la mejor solución, componente que viene dada de la siguiente manera:

$$F_i = X_{best} - X_i \quad (4.6)$$

- **Repulsión del peor (E)** : Componente repulsiva que alejará las soluciones de la peor conocida.

$$E_i = X_{worst} + X_i \quad (4.7)$$

En la Figura 4.1 podemos ver gráficamente como se comportan las libélulas teniendo en cuenta estas componentes.

Cada una de estas componentes tendrá un peso que ponderará su importancia a la hora de calcular el vector de movimiento ($s=0.1$, $a=0.1$, $c=0.7$). Los pesos de la atracción y la repulsión dependerán de si la mejor solución pertenece o no al vecindario. Si no pertenece $f=0$ y $e=0$, mientras que si pertenecen serán $f=1$ y $e=1$. Con todo esto, podremos definir la actualización de la velocidad de la siguiente manera:

$$Step_{i+1} = sS_i + aA_i + sS_i + cC_i + fF_i + eE_i + w * Step_i \quad (4.8)$$

siendo w el valor dado por:

$$w = 0,9 - \frac{i}{maxIter} 0,5 \quad (4.9)$$

En la fórmula 4.8 podemos observar el gran número de parámetros que se ven involucrados. Con ellos podemos variar el comportamiento explorativo y explotativo del

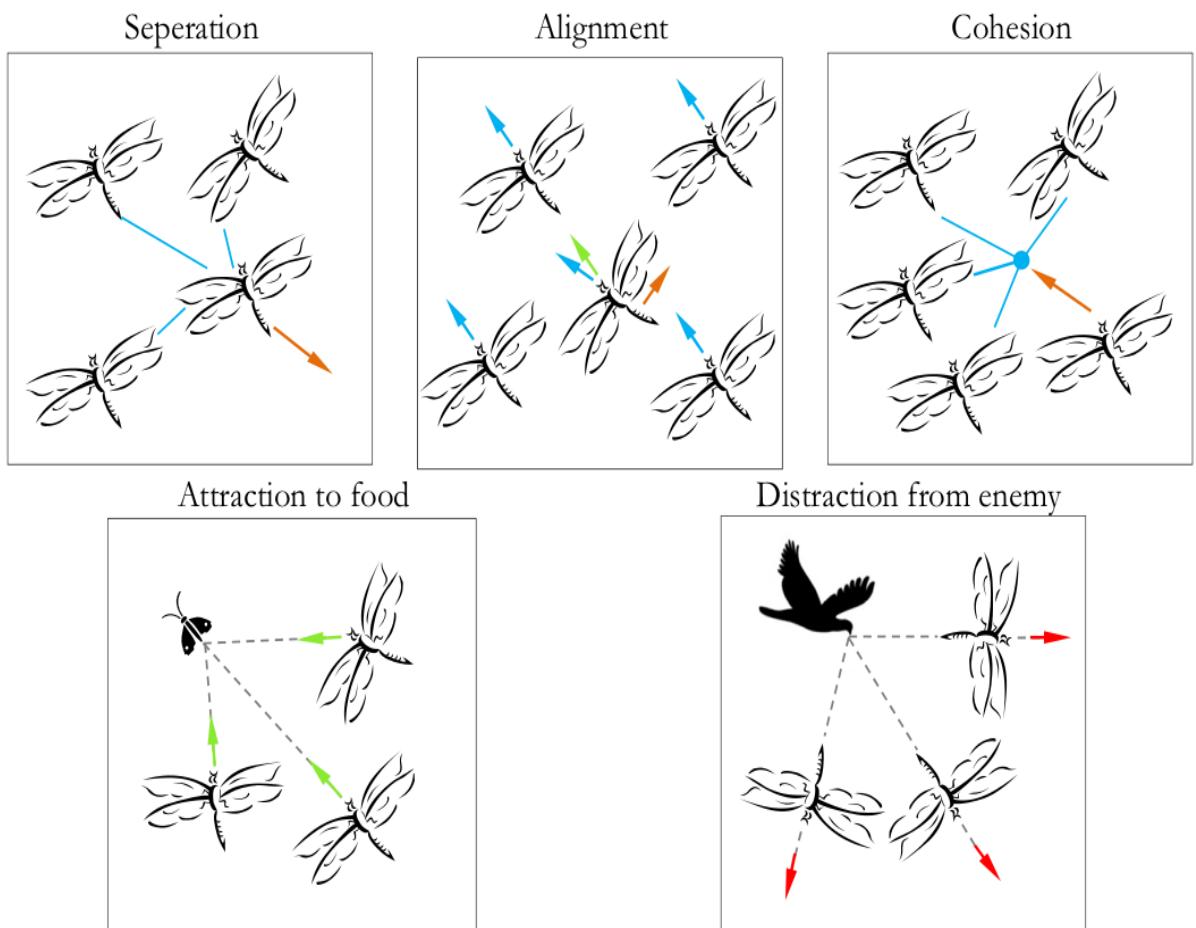


Figura 4.1: Componentes del algoritmo Dragonfly.

algoritmo. Por ejemplo, si el coeficiente de separación s es muy grande, tendremos una diversidad alta en las soluciones por lo que el algoritmo tendrá un comportamiento más explorativo. Por otra parte, si la cohesión c es muy alta, la tendencia de las soluciones a concentrarse en un punto será mayor y por lo tanto el algoritmo tendrá un carácter más explotativo.

También tenemos que tener en cuenta que puede haber soluciones que no tengan vecinos, por lo que realizar todos estos cálculos anteriores no serviría de nada. Para actualizar las soluciones en esta situación utilizaremos un movimiento aleatorio que vendrá dado por la ecuación de Lèvy:

$$Levy(x) = 0,01 * \frac{r1 * \sigma}{|r2|^{\frac{1}{\beta}}} \quad (4.10)$$

siendo r1 y r2 números aleatorios entre 0 y 1, beta = 1.5 y sigma :

$$\sigma = \left(\frac{\Gamma(1 + \beta) * \sin(\frac{\pi\beta}{2})}{\Gamma(\frac{1+\beta}{1}) * \beta * 2^{\frac{\beta-1}{2}}} \right)^{\frac{1}{\beta}} \quad (4.11)$$

Siempre que se modifique una posición, comprobaremos que está dentro de los límites considerados. En este trabajo se ha tomado el criterio de devolver la solución a la menor posición si sobrepasa el límite superior y viceversa.

A continuación en el Algorithm 3 se muestra el pseudocódigo del algoritmo:

Iniciar las soluciones aleatoriamente

$$X_i(i = 1, 2, \dots, n)$$

Iniciar los movimientos

$$Step_i(i = 1, 2, \dots, n)$$

while condición de parada no satisfecha **do**

 Calcular el valor fitness de cada solución;

 Actualizar la mejor posición y la peor si fuese necesario;

for TodasLasSoluciones **do**

 Actualizar r Eq(3.1);

 Obtener sus vecinos;

 Calcular S,A,C,F y E Eqs(3.3 - 3.7);

if mejorNoVecino **then**

if vecinos > 1 **then**

 Actualizar movimientos pesos f=0 y e=0 Eq(3.8);

 Actualizar posición con movimiento Eq(3.2);

else

 Actualizar posición con Lévy Eqs(3.10 - 3.12);

end

else

 Actualizar movimientos pesos f=1 y e=1 Eq(3.8);

 Actualizar posición con movimiento Eq(3.2);

end

 Comprobar que las posiciones están dentro de los límites;

end

end

Algorithm 3: Pseudocódigo de Dragonfly

4.2.2. Bacterial Foraging Algorithm

El siguiente algoritmo que se estudiará está inspirado en el comportamiento de una bacteria que vive en el intestino llamada E-Coli. Al igual que en el ejemplo anterior, contaremos con una componente atractiva y otra repulsiva. En este caso, la bacteria se sentirá atraída por las zonas que ofrezcan más nutrientes y repelida por las que posean más sustancias nocivas. [13].

Este comportamiento puede ser perfectamente aplicado a problemas de optimización ya que éste siempre pretende maximizar el ratio Energía Obtenida/ Tiempo de búsqueda.

Para comenzar, tendremos una población de soluciones inicializadas de forma aleatoria por el espacio de búsqueda. Además tenemos que diferenciar 3 partes importantes en este algoritmo:

- **Quimiotaxia:** Fase en la que las bacterias buscan las zonas con nutrientes. En esta fase actualizaremos las soluciones simulando los dos movimientos de la bacteria (tumbling y run).
- **Reproducción:** Etapa en la que conseguimos quedarnos siempre con las mejores soluciones.
- **Eliminación de dispersión:** En esta parte del algoritmo, se seleccionarán soluciones con una probabilidad de 0.25 para dispersarlas aleatoriamente en el entorno de búsqueda. Así conseguimos que no se estanquen en óptimos locales y poder tener una mayor exploración del entorno.

QUIMIOTAXIA

En esta parte del código se itera sobre todas las soluciones tantas veces como delimita la constante Nc . Lo primero que se hace en esta fase es actualizar las posiciones mediante un movimiento aleatorio (tumble) [14]. La fórmula que aplicaremos es la siguiente:

$$X(\text{iter_actual}) = X(\text{iter_anterior}) + C * \text{random} \quad (4.1)$$

donde random es un vector aleatorio con valores entre [-1,1] y C es el tamaño del paso, actualizado en la fase de eliminación de dispersión que veremos más adelante.

A continuación, el movimiento de nuestra solución ya no será aleatorio, sino que vendrá delimitado por la dirección en la que nos acabamos de mover. Si este movimiento ha mejorado el fitness, continuaremos desplazandola en dicha dirección, en caso contrario se quedará donde está y finalizará esta fase. Además, hay que tener en cuenta que tendremos un número máximo Ns de pasos en este movimiento.

En esta parte del código es en la que podemos observar el uso de la atracción y la repulsión. Un dato importante es que solamente se tienen en cuenta las demás soluciones

para decidir si se sigue moviendo y no para obtener la dirección del mismo, ya que se calculó aleatoriamente en la fase previa. De esta manera, calcularemos un valor de fitness que no es el real de la función, sino que reflejará la atracción y la repulsión que tiene en ese instante con respecto a las demás soluciones.

Este dato de interacción con las demás soluciones viene calculado por una función que llamamos *Cell-to-Cell Communication* [15]. Por una parte, calculamos la atracción en dicha posición con las demás soluciones, tomando la distancia al cuadrado de la misma con todas las demás. Por otra parte, con la misma distancia calcularemos la repulsión para que el movimiento no haga que las soluciones queden muy juntas. La fórmula que usaremos será la siguiente: De la misma manera y

$$Jcc(X_i, X) = \sum_{i=1}^S \left[-d_a \exp(-w_a \sum_{d=1}^D (X_d - X_{i,d})^2) \right] + \sum_{i=1}^S \left[-h_r \exp(-w_r \sum_{d=1}^D (X_d - X_{i,d})^2) \right] \quad (4.2)$$

siendo X_i todas las soluciones y X la actual. S es el número total de soluciones de nuestra población y D la dimensión del problema. También tenemos unos valores que ponderan la cantidad de atracción y repulsión que queremos aplicar: d_a y w_a son los pesos asociados a la atracción, mientras que h_r y w_r se relacionan con la repulsión.

De esta manera, cada vez que hagamos un movimiento, calcularemos un valor usando esta ecuación que nos dirá si el movimiento ha sido favorecedor. En el caso de serlo, seguiremos moviendo nuestra solución en la misma dirección en la que iba. De lo contrario, pasaremos al siguiente paso dejándola donde estaba.

REPRODUCCIÓN

En esta sección, veremos que este algoritmo propone un sistema de guardado de las mejores soluciones. Para ello, después de mover todas ellas se asegurará de que la población se quede con las mejores obtenidas.

El mecanismo consiste en ordenar todas las soluciones de mejor a peor fitness y quedarse con la mitad que posee las mejores. Para no perder la otra mitad restante, lo queharemos será hacer una replica de las mismas. Por tanto tendremos una población nueva en la que la mitad de las peores se convertirá en la mitad de las mejores. Este paso se

realizará un máximo de Nre veces.

En el ejemplo de la Figura 4.2 siguiente podemos ver como se realizaría la reproducción gráficamente. Supongamos que se está minimizando una función x^2 , cada uno de los rectángulos representa la población y cada cuadrado de dentro un individuo con la posición de x en la que se encuentra. A la izquierda se ve la población desordenada tal cual está, se ordenan de mejor a peor y por último eliminamos las peores para añadir las mejores.

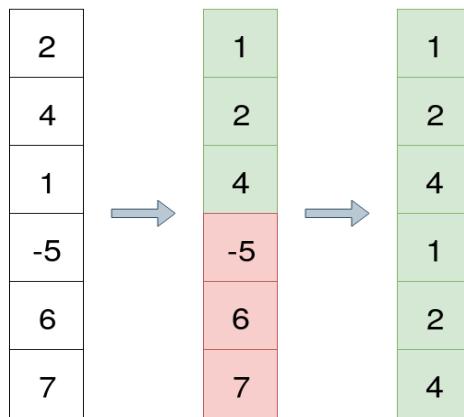


Figura 4.2: Ejemplo de reproducción

ELIMINACIÓN DE LA DISPERSIÓN

Uno de los problemas que suelen tener los algoritmos que eliminan las peores soluciones es la pérdida de la diversidad en la población. Para solucionarlo haremos una especie de *mutación* de las soluciones para volver a ampliar la superficie de exploración.

En este algoritmo, la mutación se realiza moviendo algunas de las soluciones de la población a posiciones aleatorias. Para que esto no produzca una gran pérdida de las mejores soluciones, se realizará con una probabilidad de 0.25. Si utilizando esa probabilidad toca realizar una modificación se moverá una de las soluciones, seleccionada de forma aleatoria, a una posición también aleatoria.

Iniciar las soluciones aleatoriamente

$$X_i(i = 1, 2, \dots, n)$$

```
for l=0 to Ned do
    for k=0 to Nre do
        for j=0 to Nc do
            for current in todasLasSoluciones do
                lastFitness = fitness(current) + Jcc(Soluciones,actual)
                Actualizar posición de current. Eq 4.1
                Actualizar fitness de current currentFitness = fitness(current) +
                Jcc(Soluciones,actual)
                for a=0 to Ns do
                    if currentFitness < lastFitness then
                        lastFitness = fitness(current) + Jcc(Soluciones,actual)
                        Actualizar posición de current. Eq 4.1
                        Actualizar fitness de current currentFitness = fit-
                        ness(current) + Jcc(Soluciones,actual)

                    end
                    else
                        a=Ns
                    end
                end
            end
        end
    end
end
Realizar fase de reproducción acorde el apartado 4.2
end
Realizar fase de eliminación de dispersión acorde el apartado 4.3
end
```

Algorithm 4: Pseudocódigo de Bacterial Foraging Algorithm

Capítulo 5 : Implementación

5. Implementación

En primer lugar, destacar que los algoritmos se han implementado desde cero en todos los casos ya que en el lenguaje que se querían desarrollar no estaban a mi disposición. Mediante los papers mencionados para cada algoritmo en el apartado anterior se han estudiado y desarrollado a través de los pseudocódigos explicados. Cabe destacar la dificultad de algunos de ellos por la no similitud entre lo que el autor proponía en los pseudocódigos y lo que luego realmente estaba implementado.

5.1. Requisitos

El lenguaje elegido para la implementación ha sido C++ con el uso de una biblioteca llamada Armadillo². Dicha biblioteca está desarrollada para realizar trabajos de álgebra lineal y computación científica, similar a Matlab. La razón de utilizar dicha biblioteca es que todos los algoritmos hacen uso de matrices y vectores en gran medida. Armadillo nos ayuda mucho a la hora de realizar operaciones con estas estructuras de datos ya que tiene implementadas muchas de ellas (sumas, productos, etc) y podemos prescindir de una gran cantidad de bucles.

Otra herramienta muy útil que se ha utilizado ha sido Cmake. Con ella podemos generar los archivos makefile necesarios para posteriormente generar los ejecutables. Sobre todo es muy necesaria cuando tenemos muchos archivos que dependen unos de otros. Las dependencias entre ellos se realizan automáticamente con éste tipo de herramientas.

Por tanto, para poder ejecutar el proyecto correctamente necesitaremos tener instalados tanto la librería Armadillo como la herramienta Cmake. También necesitaremos tener un compilador de C++. Además, la arquitectura utilizada para el desarrollo y ejecución del proyecto ha sido un portátil con un Intel® Core™ i7-5700HQ CPU @ 2.70GHz × 8 y la versión de compilador utilizada es c++ (Ubuntu 7.3.0-16ubuntu3) 7.3.0 Copyright©

²<http://arma.sourceforge.net/>

5.2. Instalación

En el proyecto encontraremos la siguiente estructura de carpetas:

- armadillo-8.400.0 : En ésta carpeta tendremos todos los archivos necesarios para instalar la libreria Armadillo.
- Una carpeta por algoritmo (BFA,GOA,DA,FA): Cada carpeta que contiene los codigos fuente del algoritmo y todo lo necesario para ejecutarlo.

Para instalar la librería en primer lugar tendremos que instalar los paquetes llapack y lblas, que podemos obtener directamente con apt-get. Para instalar la librería basta con entrar en la carpeta especificada en el primer punto e introducir los comandos *make* y *make install* respectivamente, con eso bastaría para poder ejecutar los algoritmos.

En cada una de las carpetas de los algoritmos tenemos un archivo *cmake* que se encarga de construir el makefile necesario para crear el ejecutable. Por lo tanto, para crear el ejecutable (main) tendremos que hacer primero un *cmake .* y a continuación un *make*. Una vez que tengamos el ejecutable, tendremos que ejecutarlo pasando un parámetro que se corresponde con el número de función del benchmark que queremos ejecutar. Por ejemplo *./main 1* en la carpeta BFA, ejecutaremos el algoritmo BFA para la función 1 y obtendremos un valor decimal como resultado.

5.3. Representación de los datos

Si hablamos de aspectos más técnicos, podemos decir que las soluciones vienen representadas por vectores. Es decir, una solución será un vector de tamaño igual a la dimensión del problema en el que cada una de sus posiciones almacena el valor asociado a la variable correspondiente. Dichos valores almacenados siempre tienen que estar dentro del espacio de búsqueda preestablecido en el problema. Por ejemplo, si tuviésemos una solución de dimensión 3 (supongamos que las variables son x,y,z) gráficamente tendríamos lo que podemos ver en la Figura 5.1.

0	1	2
3.5	80.3	-40.2
X	Y	Z

Figura 5.1: Modelo de solución.

Para cada una de las poblaciones tendremos una matriz en la que cada una de las filas será un individuo en forma de vector solución como el descrito anteriormente. El número de filas de cada una de las matrices será el tamaño de la población, por lo que tendremos una matriz de N (Número de individuos) x D (Dimensión del problema).

Capítulo 6: Diseño experimental

6. Diseño experimental

En la parte experimental de este proyecto se ejecutarán los algoritmos planteados para ver que resultados producen sobre un benchmark. La finalidad es poder hablar de su eficacia y comparar unos con otros.

Además de ver cuáles de ellos es mejor, nos interesa saber de cada uno como mejora o empeora el hecho de tener atracción o repulsión. Para ello se han utilizado unos parámetros de los que se hablará más adelante.

6.1. Benchmark

El benchmark utilizado en esta experimentación ha sido el propuesto en la competición de optimización global en el CEC 2014 [18]. En él encontramos un total de 30 funciones de 4 tipos: unimodales, multimodales simples, híbridas y composiciones. Como veremos más adelante, el tipo de cada función es importante a la hora de considerar las soluciones obtenidas.

El criterio de parada en todos los algoritmos será un número de iteraciones predefinido. Además, si el algoritmo consigue un valor menor a 10E-08, terminaremos la ejecución y se considerará que el valor final obtenido es 0. El número de iteraciones viene determinado por la dimensión en la que estemos trabajando, de forma que será $dimensión * 10000$. Se realizarán 10 ejecuciones por cada una de las funciones y el valor que consideraremos será la media de todas ellas.

6.2. Parámetros importantes

Uno de los parámetros más importantes a tener en cuenta a la hora de evaluar algoritmos poblacionales es el *tamaño de la población*. En este caso, las poblaciones son todas de tamaño 10. Las dimensiones que utilizamos para resolver los problemas son 10

y 30.

Otro aspecto que hay que destacar es que para ver si la atracción y la repulsión son favorables o no tendremos que ponderar dichas componentes. Para ello se ha añadido una variable en cada algoritmo que modificaremos según el grado de atracción/repulsión que queramos darle en cada momento al algoritmo. Esto se aplicará al valor por defecto que proporciona el autor de los algoritmos.

6.3. Comparativas a realizar

En primer lugar vamos a ponderar el parámetro de atracción. Para ello utilizaremos la variable asociada a la atracción y la modificaremos aumentando y reduciendo su impacto a la hora de modificar la solución. De esa manera, tendremos como caso base aquel en el que la ponderación de la variable es 1. Reduciremos la componente usando un valor de variable 0.5 y 0.75 y la aumentaremos multiplicando por 1.5 y 1.75.

A continuación haremos lo mismo con la repulsión. Esta vez la componente a ponderar será la repulsiva y, además, se realizará un estudio para ver si eliminándola el algoritmo mejora o empeora. Esto es algo que con la atracción no podemos hacer porque si eliminamos el criterio por el cual las soluciones se modifican hacia otras mejores el algoritmo no funcionaría correctamente. Hay que señalar que esta comparación se realiza con el modelo de algoritmo que mejores resultados haya proporcionado en la comparativa anterior.

Además, queremos ver de todos los algoritmos cuáles dan mejores resultados. Esta comparación se realizará con la versión que haya producido mejores resultados en las comparaciones anteriores.

6.4. Representación de los resultados

Para cada comparativa se ha realizado una tabla con los resultados obtenido para cada función. Como se ha dicho antes, este valor es la media de las 10 ejecuciones realizadas para cada una de las 30 funciones que posee el benchmark.

En cada una de las filas encontramos la función que estamos ejecutando, que a su

vez están agrupadas por tipos. Las columnas de izquierda a derecha representan: el tipo de función, el número de función y las siguientes el valor aplicado para ponderar la componente.

Para resaltar el mejor resultado de cada función se ha utilizado un color más oscuro de azul. Además, hay que tener en cuenta que si hay al menos 2 ponderaciones que obtienen el mismo valor en una función, ésta no se considerará como mejor en ninguno de los casos.

Al final de cada tabla, tendremos un apartado resumen. En él aparece para cada variante del algoritmo el número de mejores resultados que ha obtenido y, a la derecha, el puesto en el que ha quedado con respecto a los demás, tomando como mejor el 1º.

Capítulo 7: Análisis de los resultados

7. Análisis de resultados

Una vez se han realizado todas las pruebas preestablecidas para el estudio de los distintos algoritmos, se analizan las mismas. En esta sección encontraremos los resultados reales obtenidos de la batería de pruebas realizada, así como la interpretación de los mismos.

Para dejar más claro el análisis lo dividiré en 5 apartados: atracción y repulsión en dimensión 10, lo mismo para dimensión 30 y por último una comparativa de todos los algoritmos.

7.1. Comparativa de la componente atractiva para dimensión 10

Grasshopper Optimization Algorithm

Como podemos observar en la figura 7.1, la primera comparación que se ha realizado ha sido la ponderación de la componente atractiva en el algoritmo GOA. Dicha ponderación está relacionada directamente con la explotación que tenemos en el algoritmo y esto a su vez afecta a la diversidad del mismo.

En primer lugar, y a grandes rasgos, podemos observar que el valor por defecto que da el autor para la componente atractiva (columna con valor 1) no es el más idóneo. De hecho, solo obtiene el mejor valor en las funciones 22 y 24. Además podemos ver que reduciendo la componente mejora los resultados bastante más que aumentándola. Esto puede estar producido porque al aumentar la componente atractiva se pierde la diversidad de las soluciones muy rápido, ya que se atraen unos a otros con un factor muy grande. Además, al estar atraídos a su vez por el mejor de todos, van hacia él muy pronto, produciendo que no se explore el espacio de soluciones correctamente. Cuando tenemos menos atracción de la que hay por defecto tendremos un radio de exploración

GOA	FUNCIONES	0,5	0,75	1	1,25	1,5
Unimodal Functions	F1	7,6728E+06	9,0536E+06	8,5202E+06	9,2166E+06	8,2678E+06
	F2	1,3356E+09	1,3201E+09	1,4797E+09	1,4687E+09	1,3838E+09
	F3	9,6551E+03	9,8137E+03	1,0452E+04	9,4398E+03	1,0553E+04
Simple Multimodal Functions	F4	1,2901E+02	1,3672E+02	1,4000E+02	1,3578E+02	1,3315E+02
	F5	2,0095E+01	1,9878E+01	2,0205E+01	2,0043E+01	2,0225E+01
	F6	7,6574E+00	7,4557E+00	7,6783E+00	7,8415E+00	7,5357E+00
	F7	2,2374E+01	2,2660E+01	2,3894E+01	2,3511E+01	2,2684E+01
	F8	4,7711E+01	4,8938E+01	4,7752E+01	4,7485E+01	4,7500E+01
	F9	5,0179E+01	4,6608E+01	5,3395E+01	5,1005E+01	5,0565E+01
	F10	1,1056E+03	1,0268E+03	1,0549E+03	9,9995E+02	9,7480E+02
	F11	7,8116E-01	1,1310E+03	1,1006E+03	1,0482E+03	1,1452E+03
	F12	1,2254E+00	8,4287E-01	8,3667E-01	7,6070E-01	8,4297E-01
	F13	5,9401E+00	1,2210E+00	1,2597E+00	1,2794E+00	1,3000E+00
	F14	9,3217E+01	6,3809E+00	6,0895E+00	5,8019E+00	6,3534E+00
	F15	1,0481E+03	7,4990E+01	8,0105E+01	6,2188E+01	1,1465E+02
	F16	3,2499E+00	3,3207E+00	3,3010E+00	3,3085E+00	3,2821E+00
Hybrid Function	F17	1,3991E+04	1,4284E+04	1,8393E+04	1,7535E+04	1,6778E+04
	F18	1,2615E+04	6,3718E+03	9,6723E+03	8,9086E+03	1,8410E+04
	F19	6,1232E+00	5,8922E+00	6,0185E+00	6,1397E+00	6,2024E+00
	F20	2,1742E+02	1,9348E+02	2,6846E+02	2,2167E+02	2,4069E+02
	F21	2,0991E+03	1,6905E+03	2,0524E+03	1,8803E+03	1,8207E+03
	F22	5,5257E+01	5,4578E+01	5,0883E+01	5,3851E+01	5,7152E+01
Composition Functions	F23	3,5104E+02	3,4913E+02	3,4974E+02	3,4897E+02	3,4729E+02
	F24	1,6126E+02	1,6379E+02	1,6086E+02	1,6599E+02	1,6356E+02
	F25	1,7242E+02	1,7310E+02	1,7552E+02	1,7237E+02	1,7585E+02
	F26	1,0099E+02	1,0096E+02	1,0097E+02	1,0092E+02	1,0086E+02
	F27	7,8970E+01	8,3611E+01	8,7087E+01	8,5584E+01	8,5081E+01
	F28	6,4318E+02	6,2794E+02	6,4091E+02	6,5954E+02	6,4914E+02
	F29	4,2840E+03	6,5138E+03	6,8667E+03	5,7536E+03	4,9205E+03
	F30	2,0176E+03	2,1627E+03	2,0505E+03	2,0634E+03	1,9549E+03
Resumen		9 (2º)	10 (1º)	2 (5º)	5 (3º)	4 (4º)

Figura 7.1: Comparativa de atracción algoritmo GOA con dimensión 10.

más grande y, como vemos, obtendremos mejores soluciones.

Observando la complejidad de las funciones, podemos ver que algunos valores de la atracción destacan frente a otros. Con el parámetro establecido en 0.75 vemos que la eficacia aumenta sobre todo en las funciones híbridas. En las unimodales claramente los mejores resultados se obtienen en las ponderaciones mas pequeñas, sobre todo en la de 0.5. Si hablamos de las funciones multimodales vemos que hay algunas en las que se obtienen resultados muy diferentes y se mencionarán a continuación.

Una de las funciones que merece ser nombrada es la número 11 ya que la mejora que se produce al reducir la componente a 0.5 es muy significativa. Si observamos su forma en la Figura 7.2 vemos que es una función con muchos mínimos y muy juntos entre sí. Esto hace que la atracción y la repulsión tengan problemas debido a las distancias aplicadas para mover las soluciones. Teniendo en cuenta la fórmula usada para realizar los movimientos de las soluciones, cuánto más valor le pongamos a la atracción menos penalización se le da a la distancia y por tanto más influirá. Por esta razón, al tener tantos mínimos puede que una solución se mueva al mínimo más lejano teniendo a otro mejor en una posición mas cercana. Cuando ponderamos poco la atracción (caso de 0.5), la distancia influye menos y por lo tanto se acercará a un mínimo mas cercano, por lo que la mejoría de las soluciones será mucho mas rápida.

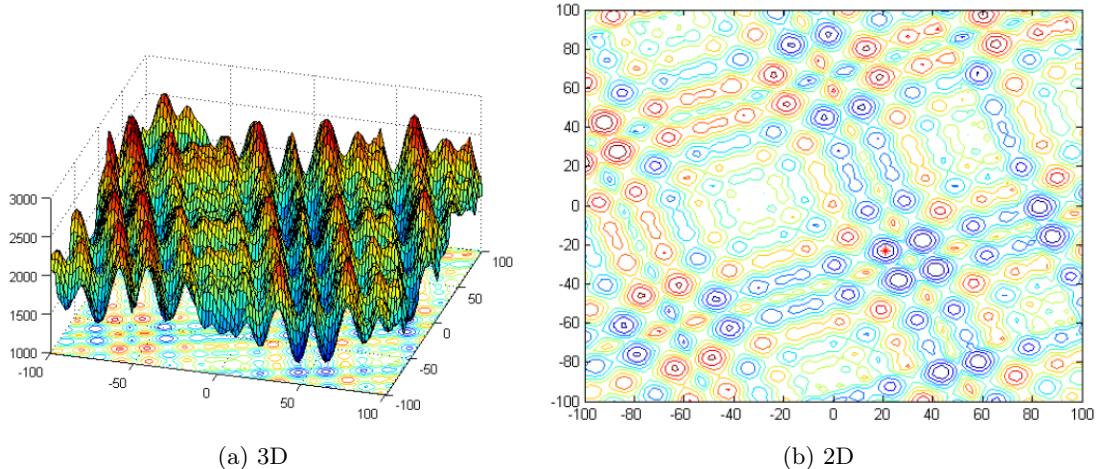


Figura 7.2: Función 11. Shifted Scherfel's Function

Otra de las funciones que experimentan un comportamiento llamativo es la 15. En este caso la ponderación 0.5 da un resultado mucho peor que la 1.25. Si vemos la Figura 7.3 observamos que hay también muchos mínimos locales pero que el global se encuentra relativamente separado de los demás. Es ésta la razón por la que mejora en ponderaciones altas ya que, como hemos dicho antes, en estas variantes del algoritmo la distancia se penaliza menos y las soluciones se irán a óptimos más lejanos independientemente de los que tengan cerca si son mejores. Por lo tanto, al encontrarse el óptimo global más lejos de los locales, las soluciones se aproximarán a él antes que a uno cercano que sea peor, mientras que con ponderaciones bajas, se penaliza más la distancia y se acercará antes a los óptimos cercanos que al mejor más lejano.

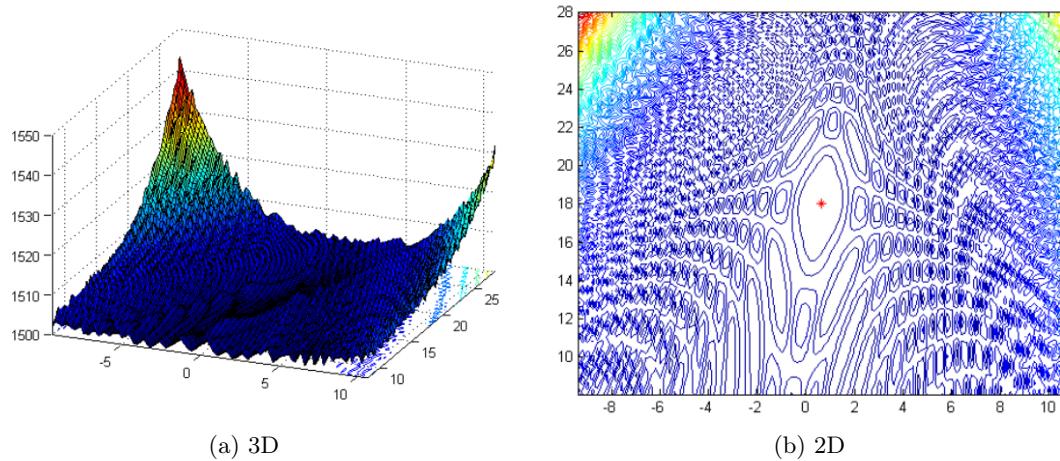


Figura 7.3: Función 15. Shifted and Rotated Expanded Griewank's plus Rosenbrock's Function

Firefly Algorithm

En la Figura 7.4 podemos ver los resultados obtenidos para este algoritmo ponderando su componente atractiva. Lo primero que hay que mencionar es que las variantes que mejores resultados dan son la 0.75 y la 1.25. Esto no nos deja muy claro cuál es el comportamiento del algoritmo con respecto a este parámetro. Pero a continuación vamos a ver su comportamiento por grupos de funciones.

Hay tres grupos de funciones en los que queda especialmente claro con que valor del parámetro se comporta mejor el algoritmo. En el caso de las funciones multimodales podemos ver claramente que los mejores valores son los que están por debajo del 1. Como se ha dicho en apartados anteriores, la mayoría de funciones multimodales tienen un espacio de búsqueda mas o menos homogéneo. En este algoritmo, cuanto menor sea el valor del parámetro que pondera la atracción, menos se penaliza la distancia, por lo que las soluciones irán a óptimos más lejanos independientemente de su distancia, que es lo que nos interesa en éste tipo de funciones.

Por otro lado, en las funciones más complejas como son las híbridas y las compuestas pasa totalmente al contrario que en el anterior caso. Cuanto más grande es el factor que le ponemos a la atracción, menos se penaliza la distancia y por lo tanto más se tiene en cuenta. Si los óptimos están muy cerca y son muchos como en el caso de éstas funciones, lo que mejor resultado da es ir a los más cercanos porque si guiamos las soluciones a los más lejanos que en ese momento sean mejores, podemos perder un cercano que en ese momento sea peor pero tenga un óptimo global.

FA	FUNCIONES	0,5	0,75	1	1,25	1,5
Unimodal Functions	F1	9,3940E+08	1,1149E+09	8,8585E+08	1,3891E+09	6,9522E+08
	F2	2,3201E+10	2,2479E+10	1,8157E+10	1,9916E+10	2,4238E+10
	F3	1,0214E+08	1,6449E+08	6,7800E+07	8,0549E+07	1,8322E+08
Simple Multimodal Functions	F4	8,0759E+03	6,7371E+03	7,5988E+03	7,7059E+03	7,8853E+03
	F5	2,1068E+01	2,1031E+01	2,1006E+01	2,1064E+01	2,1050E+01
	F6	1,4993E+01	1,4272E+01	1,4940E+01	1,5195E+01	1,5049E+01
	F7	3,7242E+02	3,4335E+02	4,2172E+02	3,5811E+02	3,8654E+02
	F8	1,4448E+02	1,5799E+02	1,5728E+02	1,6206E+02	1,5887E+02
	F9	1,5301E+02	1,4136E+02	1,5488E+02	1,4700E+02	1,6536E+02
	F10	2,9589E+03	3,0256E+03	3,0841E+03	2,9105E+03	3,1100E+03
	F11	2,7205E+03	2,8176E+03	2,8148E+03	2,8438E+03	2,7643E+03
	F12	5,1473E+00	4,6490E+00	5,1860E+00	4,8755E+00	4,8120E+00
	F13	7,4227E+00	7,0278E+00	7,5580E+00	7,8885E+00	6,5063E+00
	F14	8,9283E+01	9,7346E+01	9,1648E+01	9,5857E+01	8,9674E+01
	F15	1,3526E+06	8,7736E+05	1,1479E+06	9,6701E+05	1,9545E+06
	F16	4,4107E+00	4,4731E+00	4,3516E+00	4,3522E+00	4,3278E+00
Hybrid Function	F17	1,4867E+07	3,0066E+07	2,5447E+07	2,0999E+07	1,3997E+07
	F18	1,1080E+09	1,1450E+09	1,2513E+09	6,5722E+08	1,3159E+09
	F19	3,2935E+02	4,8012E+02	4,0318E+02	2,6432E+02	2,5958E+02
	F20	9,5186E+04	3,7705E+04	1,8292E+04	2,1687E+04	3,0237E+04
	F21	1,6306E+08	4,3224E+08	1,2701E+08	9,1226E+07	9,8314E+07
	F22	1,0233E+03	1,0727E+03	9,6927E+02	1,0081E+03	9,6748E+02
Composition Functions	F23	1,2621E+03	1,0074E+03	1,0579E+03	1,1166E+03	1,1343E+03
	F24	2,7898E+02	2,7498E+02	2,7030E+02	2,7394E+02	2,7662E+02
	F25	2,4516E+02	2,4070E+02	2,6116E+02	2,3118E+02	2,3871E+02
	F26	1,3545E+02	1,6500E+02	1,7349E+02	1,7811E+02	1,7319E+02
	F27	8,4121E+02	1,2282E+03	9,2762E+02	1,0310E+03	9,9819E+02
	F28	2,4670E+03	2,4263E+03	2,8003E+03	2,3898E+03	2,3961E+03
	F29	1,0890E+08	1,2436E+08	9,9795E+07	8,2685E+07	1,0247E+08
	F30	5,8003E+06	7,2574E+06	2,4055E+06	1,9608E+06	5,6089E+06
Resumen		5 (3º)	7 (1º)	5 (3º)	7 (1º)	6 (2º)

Figura 7.4: Comparativa de repulsión algoritmo FA con dimensión 10.

Dragonfly Algorithm

A continuación, vamos a observar como reacciona el algoritmo Dragonfly a la hora de variar su componente atractiva al igual que en los casos anteriores. Mas adelante realizaremos el mismo estudio pero con la componente repulsiva para ver como afecta al algoritmo.

Si observamos la Figura 7.5 vemos que las mejores soluciones están bastante dispersas entre todas las variantes. En primer lugar se puede ver que en el recuento gana la ponderación 0.75, lo que significaría que la ponderación que el autor le da a la atracción no es del todo correcta pero se acerca bastante a la óptima hasta ahora conocida. Cabe destacar que dicha variante del algoritmo es la que mejores resultados proporciona en funciones híbridas.

La razón por la que la variación de la componente no produce una gran mejora es que se trata de un algoritmo en el que, para realizar un movimiento, se ven involucrados muchos parámetros y muchas componentes. De hecho, si observamos la fórmula usada para desplazar las soluciones, vemos que depende de un valor *step* que se calcula de la siguiente manera:

$$Step_{i+1} = sS_i + aA_i + sS_i + cC_i + fF_i + eE_i + w * Step_i \quad (7.1)$$

Si tenemos en cuenta esto, podemos ver que la componente atractiva es solo uno de los 7 parámetros que influyen a la hora de calcular una nueva solución. Para notar una modificación significativa tendríamos que modificar más parámetros o hacerlo de una manera más brusca.

Si observamos los resultados atendiendo a los tipos de funciones, vemos que las variantes en las que la ponderación es más baja son las que mejores resultados dan en funciones unimodales y multimodales simples. Por otra parte, las ponderaciones altas son las que nos dan unos resultados mejores en funciones híbridas y compuestas. Esto es debido a que cuando las ponderaciones son pequeñas, la influencia de la distancia es mayor y al ser funciones con un entorno más homogéneo irá mas rápidamente hacia los óptimos globales por muy lejos que estén. De la misma manera, cuando las ponderaciones son muy altas, tenemos una influencia menor de la distancia. Las funciones híbridas y

DA	FUNCIONES	0,5	0,75	1	1,25	1,5
Unimodal Functions	F1	4,4360E+08	4,2103E+08	6,3420E+08	8,5076E+08	6,7915E+08
	F2	1,0203E+10	9,9610E+09	1,0211E+10	9,5168E+09	1,0618E+10
	F3	1,2142E+04	1,1036E+04	1,1240E+04	1,1084E+04	1,0855E+04
Simple Multimodal Functions	F4	4,4229E+03	4,6740E+03	4,5713E+03	5,0447E+03	4,6282E+03
	F5	2,0320E+01	2,0400E+01	2,0345E+01	2,0332E+01	2,0341E+01
	F6	1,1209E+01	1,1363E+01	1,1283E+01	1,1213E+01	1,1583E+01
	F7	1,2192E+02	1,2076E+02	1,1364E+02	1,3006E+02	1,3703E+02
	F8	9,7094E+01	9,0596E+01	9,4949E+01	9,1473E+01	9,1525E+01
	F9	8,1342E+01	8,9784E+01	8,3076E+01	9,1323E+01	9,5673E+01
	F10	1,7542E+03	1,7180E+03	1,7550E+03	1,7639E+03	1,6887E+03
	F11	1,8450E+03	2,0684E+03	1,9307E+03	1,7933E+03	1,8999E+03
	F12	1,3963E+00	1,3181E+00	1,2639E+00	1,3907E+00	1,4181E+00
	F13	5,6134E+00	5,7886E+00	5,6420E+00	6,3871E+00	5,8546E+00
	F14	3,9256E+01	3,1998E+01	3,6928E+01	3,5525E+01	3,7475E+01
	F15	1,9185E+04	1,3722E+04	2,3484E+04	5,5999E+04	1,8693E+04
	F16	3,4944E+00	3,6823E+00	3,5737E+00	3,6300E+00	3,5661E+00
Hybrid Function	F17	4,3678E+06	4,2838E+05	3,5832E+05	4,5469E+05	4,6704E+05
	F18	4,6839E+06	1,2544E+07	3,9633E+06	2,4264E+06	3,7888E+06
	F19	4,6016E+01	4,0968E+01	3,9633E+01	5,5433E+01	5,9673E+01
	F20	9,9202E+03	8,1618E+03	1,2819E+04	1,4828E+04	2,4705E+05
	F21	4,9119E+05	3,3399E+05	1,7242E+05	3,4317E+05	3,6822E+05
	F22	3,3690E+02	3,8884E+02	2,8067E+02	3,5022E+02	3,1804E+02
Composition Functions	F23	2,4609E+02	2,1161E+02	2,1775E+02	2,0004E+02	2,2577E+02
	F24	1,9843E+02	2,1095E+02	2,0091E+02	2,0003E+02	2,0004E+02
	F25	1,9969E+02	1,9886E+02	1,9971E+02	1,9950E+02	1,9990E+02
	F26	1,0985E+02	1,1512E+02	1,1236E+02	1,1386E+02	1,0850E+02
	F27	2,4662E+02	2,0000E+02	2,0000E+02	3,9646E+02	2,7711E+02
	F28	2,0016E+02	4,2447E+02	2,0030E+02	3,0023E+02	2,0009E+02
	F29	2,0678E+04	1,0313E+05	6,2395E+05	2,3423E+06	1,6968E+05
	F30	2,8769E+04	3,2065E+03	3,8448E+03	2,5762E+03	4,2316E+03
Resumen		6 (3º)	8 (1º)	5 (4º)	7 (2º)	3 (5º)

Figura 7.5: Comparativa de atracción algoritmo DA con dimensión 10.

compuestas tienen entornos muy heterogéneos, con muchos mínimos y muy juntos. Al tener menos en cuenta la distancia a la hora de mover las soluciones, no se desvía tanto de los mínimos cercanos y son más efectivas las búsquedas.

De acuerdo a lo que se acaba de comentar, nos damos cuenta de que no siempre en un mismo algoritmo un valor es bueno para todas las funciones que tengamos. Si observamos la función 26 en la Figura 7.6, vemos que el espacio de búsqueda tiene zonas muy diferentes. Es por esto que podemos pensar en realizar una ponderación de la atracción variable dependiendo de como sean los entornos de las funciones.

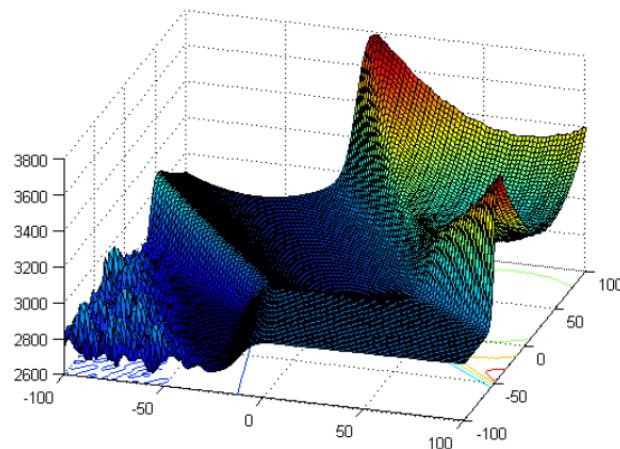


Figura 7.6: Gráfica 3D de la función 26.

Bacterial Foraging Algorithm

Si observamos la tabla 7.7, vemos que los resultados obtenidos en esta comparativa son a primera vista muy poco informativos. En primer lugar, observando el ranking, podemos ver que tenemos 3 variantes con la mejor puntuación.

En el ámbito de las funciones unimodales podemos ver que, de las 3 mejores variantes del algoritmo, la única que obtiene buenas puntuaciones es aquella que pondera la atracción en un 1.5. Sin embargo, esta es la que peores resultados proporciona en funciones multimodales simples, ya que no se lleva ninguna de las mejores puntuaciones en ninguna de ellas. Cabe destacar que en las funciones composición esta variante del algoritmo no

tiene un comportamiento del todo malo, de hecho es de las que mas puntuación obtiene en este tipo.

En general, los mejores resultados los dan los 3 que más ponderación le dan a la atracción, por tanto podemos decir que en este algoritmo cuánto menos influya la distancia en la atracción, mejores resultados obtendremos. Si ponderasemos la atracción un valor por encima de 1.5 podríamos ver si alguno de esos valores nos da el valor óptimo para el parámetro. Este aspecto se podría plantear para una posible ampliación del proyecto.

BFA	FUNCIONES	0,5	0,75	1	1,25	1,5
Unimodal Functions	F1	3,0169E+05	2,5956E+05	1,9671E+05	2,2758E+05	1,8666E+05
	F2	1,4036E+05	1,2375E+05	1,2443E+05	1,2949E+05	1,2054E+05
	F3	1,3613E+04	1,1946E+04	1,3767E+04	1,4483E+04	1,3863E+04
	F4	1,8318E+00	2,0328E+00	1,0540E+00	2,2741E+00	1,3228E+00
	F5	2,0372E+01	2,0380E+01	2,0414E+01	2,0377E+01	2,0407E+01
	F6	1,0940E+01	1,0857E+01	1,0649E+01	1,0395E+01	1,0990E+01
	F7	5,1167E-01	4,7795E-01	5,0760E-01	4,8955E-01	4,9178E-01
	F8	6,1046E+01	5,6122E+01	5,8592E+01	5,3994E+01	5,8045E+01
	F9	6,0806E+01	5,7887E+01	6,2565E+01	6,5109E+01	5,5956E+01
	F10	8,7723E+02	9,6632E+02	9,3873E+02	8,9921E+02	8,8826E+02
Simple Multimodal Functions	F11	8,5182E+02	8,9525E+02	7,7266E+02	8,0968E+02	8,8993E+02
	F12	3,3063E-01	3,3781E-01	3,0673E-01	3,5389E-01	3,3100E-01
	F13	2,6781E-01	2,5857E-01	2,5145E-01	2,7338E-01	2,7216E-01
	F14	2,3569E-01	2,2168E-01	2,3760E-01	2,1198E-01	2,1541E-01
	F15	1,2429E+01	1,1704E+01	1,1297E+01	1,1897E+01	1,1700E+01
	F16	3,9545E+00	3,9504E+00	3,9206E+00	3,9576E+00	4,0034E+00
Hybrid Function	F17	1,6674E+03	1,7413E+03	1,6820E+03	1,9419E+03	1,5926E+03
	F18	1,7326E+02	1,6378E+02	1,5594E+02	1,4060E+02	1,6350E+02
	F19	1,1375E+01	8,9679E+00	8,7855E+00	8,9984E+00	7,5414E+00
	F20	5,3428E+02	4,2834E+02	5,7338E+02	5,1317E+02	9,9542E+02
	F21	7,0674E+02	8,0534E+02	7,3301E+02	6,7157E+02	7,9162E+02
	F22	1,1638E+02	1,2217E+02	1,1988E+02	1,2290E+02	1,4335E+02
Composition Functions	F23	3,1483E+02	3,2145E+02	2,7151E+02	3,2947E+02	3,0767E+02
	F24	1,8959E+02	1,9087E+02	1,8994E+02	1,8980E+02	1,8556E+02
	F25	1,8910E+02	1,8720E+02	1,8177E+02	1,9058E+02	1,8831E+02
	F26	1,0027E+02	1,0028E+02	9,7936E+01	1,0026E+02	1,0027E+02
	F27	2,6935E+02	2,6255E+02	1,6600E+02	2,6683E+02	1,8467E+02
	F28	5,3515E+02	5,8377E+02	5,8160E+02	5,6793E+02	5,8296E+02
Resumen	F29	2,3739E+02	2,3822E+02	2,3665E+02	2,3371E+02	2,3112E+02
	F30	4,7638E+02	5,0532E+02	4,7493E+02	4,7488E+02	5,0346E+02
Resumen		5 (2º)	4 (3º)	7 (1º)	7 (1º)	7 (1º)

Figura 7.7: Comparativa de atracción algoritmo BFA con dimensión 10.

7.2. Comparativa de la componente repulsiva para dimensión 10

Dragonfly Algorithm

Como se ha dicho en el apartado de diseño de experimentación, las pruebas de repulsión que se ven a continuación se han realizado con el mejor valor obtenido en el estudio de la atracción. En este caso, el valor que proporcionaba mejores resultados era 0.75, por lo que partiendo de este valor en la ponderación de la atracción se ha modificado la componente repulsiva.

Lo primero que destacaremos de la Figura 7.8 es la poca diferencia que hay entre las soluciones de todas las variantes del algoritmo. Esto es lo mismo que ocurría cuando estudiábamos la componente atractiva y es porque la fórmula en la que se utiliza esta componente tiene muchas más.

Otra de los resultados que se observan claramente es que el valor por defecto que nos ofrece el artículo no es el más adecuado. En este caso el valor óptimo es 0.75, un poco por debajo del que teníamos. Cabe destacar que para las funciones composición destaca mucho la mejoría con respecto a los otros valores de ponderación. Teniendo en cuenta que solo consideramos 6 de las funciones que forman este grupo, por haber obtenido empate en las restantes, esta variante obtiene el mejor resultado en la mitad de ellas.

Si observamos la variante que pondera la repulsión con un 1.5, vemos que por lo general obtiene resultados peores que las otras. Pero en las funciones híbridas obtiene el mejor resultado en 4 funciones de 6 que forman esta categoría. Esto nos hace pensar que no debemos tomar un mismo patrón para todos los tipos de problemas, dado que el hecho de que una ponderación puede hacer que el algoritmo funcione muy bien con unos problemas no nos garantiza que funcione igual con otros de distinto tipo. Es decir, que cuando queramos resolver un problema concreto, primero deberíamos hacer un estudio similar a éste y ver cuáles son los valores óptimos para los parámetros en ese tipo de problemas.

En esta parte del estudio, hemos eliminado la componente repulsiva para ver como se comporta el algoritmo. Como vemos, eliminar la componente repulsiva no siempre es la mejor opción, de hecho no mejora las soluciones en la mayoría de los casos. Esto puede

llevarnos a la conclusión de que, si no incrementa el tiempo de ejecución, es algo que deberíamos considerar a la hora de realizar un algoritmo porque por poco que mejoren las soluciones siempre será útil.

Vamos a observar la forma que tienen las funciones en las que mejora el algoritmo si eliminamos la repulsión. En la Figura 7.9 podemos ver que excepto la función 2 que es diferente, las otras dos son bastante parecidas. La primera es una función unimodal mientras que las otras dos son multimodales simples.

En el caso de la función unimodal podríamos pensar que, al no tener “obstáculos” de ningún tipo para encontrar el mínimo, la repulsión nos ayuda bastante porque el camino que hay que seguir es claro. Es decir, que siempre que la solución sea repelida de la peor, irá en buen camino ya que solo hay un mínimo que considerar en toda la función por lo que no tenemos mínimos locales de los que preocuparnos.

Las funciones multimodales son totalmente lo contrario. Se caracterizan por tener muchos mínimos y máximos locales, lo que nos haría pensar que lo dicho en el párrafo anterior se contradice. En este caso la repulsión podría separar nuestras soluciones de máximos que por otro lado llevasen a mínimos mejores que al que se dirigen en ese momento.

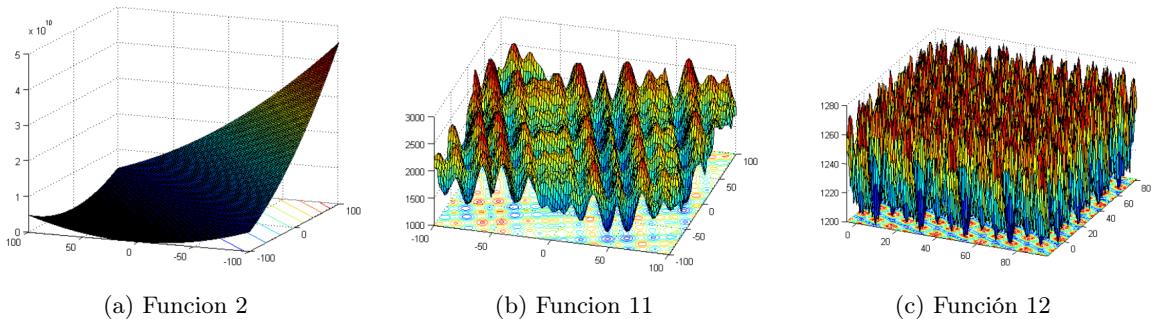


Figura 7.9: Mejores resultados sin repulsión DA.

Para poder analizar si la repulsión mejora las soluciones lo suficiente como para tenerlo en cuenta, haremos una tabla en la que compararemos el mejor algoritmo obtenido con el algoritmo sin repulsión. El resultado se muestra en la Figura 7.10 y como podemos observar en esta comparación, sí se ve una mejoría notable de los resultados.

DA	FUNCIONES	Sin repulsion	0,5	0,75	1	1,25	1,5
Unimodal Functions	F1	7,7649E+08	7,6501E+08	5,8951E+08	4,2103E+08	6,8901E+08	4,7594E+08
	F2	9,3211E+09	1,1903E+10	1,1126E+10	9,9610E+09	1,1851E+10	1,0294E+10
	F3	1,0404E+04	1,1333E+04	1,0145E+04	1,1036E+04	1,1269E+04	3,2130E+07
Simple Multimodal Functions	F4	4,9724E+03	3,7626E+03	4,4413E+03	4,6740E+03	3,9705E+03	4,2112E+03
	F5	2,0362E+01	2,0420E+01	2,0354E+01	2,0400E+01	2,0404E+01	2,0390E+01
	F6	1,1161E+01	1,1103E+01	1,1534E+01	1,1363E+01	1,1257E+01	1,1715E+01
	F7	1,3725E+02	1,1518E+02	1,3615E+02	1,2076E+02	1,1672E+02	1,2968E+02
	F8	9,5116E+01	8,8545E+01	9,4993E+01	9,0596E+01	1,0373E+02	9,5222E+01
	F9	8,6450E+01	8,3626E+01	8,5446E+01	8,9784E+01	8,0589E+01	8,1099E+01
	F10	1,7534E+03	1,7296E+03	1,6887E+03	1,7180E+03	1,7557E+03	1,6915E+03
	F11	1,9330E+03	2,0173E+03	1,9707E+03	2,0684E+03	1,9372E+03	1,9555E+03
	F12	1,2336E+00	1,3221E+00	1,3424E+00	1,3181E+00	1,4316E+00	1,4180E+00
	F13	6,3129E+00	5,5370E+00	6,1667E+00	5,7886E+00	5,3561E+00	5,8296E+00
	F14	4,1185E+01	3,4491E+01	3,5843E+01	3,1998E+01	3,5983E+01	3,5464E+01
	F15	1,1840E+04	3,6578E+04	3,9548E+04	1,3722E+04	1,6866E+04	9,8382E+03
	F16	3,6052E+00	3,6031E+00	3,5791E+00	3,6823E+00	3,5464E+00	3,5408E+00
Hybrid Function	F17	5,0777E+05	4,7282E+05	4,6439E+05	4,2838E+05	4,6095E+05	3,9845E+05
	F18	6,4652E+06	6,0375E+06	1,0230E+07	1,2544E+07	3,9801E+07	5,5693E+06
	F19	4,9362E+01	7,6426E+01	4,6477E+01	4,0968E+01	5,1523E+01	5,0665E+01
	F20	3,4605E+04	1,3913E+04	1,0916E+04	8,1618E+03	4,1911E+04	7,2847E+03
	F21	4,8273E+05	2,2962E+05	3,6712E+05	3,3399E+05	3,5092E+05	1,4754E+05
Composition Functions	F22	3,6032E+02	3,3280E+02	3,1048E+02	3,8884E+02	3,2467E+02	3,7776E+02
	F23	2,0002E+02	2,3886E+02	2,0001E+02	2,1161E+02	2,0002E+02	2,0002E+02
	F24	2,0003E+02	2,0211E+02	1,9889E+02	2,1095E+02	2,0063E+02	1,9956E+02
	F25	2,0020E+02	1,9994E+02	1,9985E+02	1,9886E+02	2,0140E+02	1,9965E+02
	F26	1,1909E+02	1,3516E+02	1,1584E+02	1,1512E+02	1,0780E+02	1,1445E+02
	F27	2,4549E+02	2,0000E+02	2,0001E+02	2,0000E+02	2,4011E+02	2,6391E+02
	F28	2,0003E+02	3,0038E+02	3,9474E+02	4,2447E+02	3,8567E+02	2,0003E+02
	F29	1,2079E+04	6,1865E+05	2,6027E+05	1,0313E+05	6,8289E+05	5,8642E+04
	F30	1,0722E+04	3,0879E+03	3,7315E+03	3,2065E+03	2,6707E+03	5,1601E+03
Resumen		4 (3º)	3 (4º)	8 (1º)	3 (4º)	5 (2º)	5 (2º)

Figura 7.8: Comparativa de repulsión algoritmo DA con dimensión 10.

DA	FUNCIONES	Sin repulsion	0,75
Unimodal Functions	F1	7,7649E+08	5,8951E+08
	F2	9,3211E+09	1,1126E+10
	F3	1,0404E+04	1,0145E+04
Simple Multimodal Functions	F4	4,9724E+03	4,4413E+03
	F5	2,0362E+01	2,0354E+01
	F6	1,1161E+01	1,1534E+01
	F7	1,3725E+02	1,3615E+02
	F8	9,5116E+01	9,4993E+01
	F9	8,6450E+01	8,5446E+01
	F10	1,7534E+03	1,6887E+03
	F11	1,9330E+03	1,9707E+03
	F12	1,2336E+00	1,3424E+00
	F13	6,3129E+00	6,1667E+00
	F14	4,1185E+01	3,5843E+01
	F15	1,1840E+04	3,9548E+04
	F16	3,6052E+00	3,5791E+00
Hybrid Function	F17	5,0777E+05	4,6439E+05
	F18	6,4652E+06	1,0230E+07
Composition Functions	F19	4,9362E+01	4,6477E+01
	F20	3,4605E+04	1,0916E+04
	F21	4,8273E+05	3,6712E+05
	F22	3,6032E+02	3,1048E+02
	F23	2,0002E+02	2,0001E+02
	F24	2,0003E+02	1,9889E+02
	F25	2,0020E+02	1,9985E+02
	F26	1,1909E+02	1,1584E+02
	F27	2,4549E+02	2,0001E+02
	F28	2,0003E+02	3,9474E+02
Resumen	F29	1,2079E+04	2,6027E+05
	F30	1,0722E+04	3,7315E+03
Resumen		8 (2º)	22 (1º)

Figura 7.10: Comparativa sin repulsión y ponderada 0.75 de DA en dimensión 10.

Bacterial Foraging Algorithm

Este estudio con respecto a la repulsión es uno de los que nos ha proporcionado unos resultados más claros. Como podemos observar en la Figura 7.11, la puntuación en el ranking de la variante sin repulsión ha sido de 0 puntos. Esto demuestra que en este algoritmo tener en cuenta dicha componente mejora bastante su comportamiento.

Cabe destacar que el tipo de funciones en las que más mejoría notamos es en las unimodales. Esto es debido a que estas funciones tienen un entorno muy homogéneo y un mínimo muy claro. En estas situaciones la repulsión funciona muy bien porque siempre que se aleje una solución de otra peor estará acercándose al único óptimo que tiene la función, que es el global.

Un comportamiento similar lo encontramos en las funciones multimodales, en las que la mejoría de los resultados también es significativa. Un aspecto que podemos observar en la Figura 7.11 es la importancia de adaptar la repulsión ya que hay algunas funciones, como por ejemplo la 19, en las que la diferencia entre ponderarla con valor 1 o valor 0.5 nos hace que varíe mucho el resultado.

BFA	FUNCIONES	Sin repulsión	0,5	0,75	1	1,25	1,5
Unimodal Functions	F1	4,0336E+05	2,4096E+05	3,4520E+05	1,8666E+05	2,7779E+05	4,2058E+05
	F2	1,4159E+05	1,3376E+05	1,1963E+05	1,2054E+05	1,4460E+05	1,3328E+05
	F3	1,6903E+04	1,0039E+04	1,4250E+04	1,3863E+04	1,4010E+04	1,7185E+04
Simple Multimodal Functions	F4	1,9733E+00	1,7983E+00	1,2341E+00	1,3228E+00	3,2039E+00	3,0361E+00
	F5	2,0428E+01	2,0395E+01	2,0403E+01	2,0407E+01	2,0467E+01	2,0431E+01
	F6	1,1619E+01	1,1308E+01	1,1578E+01	1,0990E+01	1,0833E+01	1,1742E+01
	F7	4,9527E-01	5,0489E-01	5,0912E-01	4,9178E-01	5,7732E-01	5,1644E-01
	F8	7,3473E+01	5,4269E+01	5,2408E+01	5,8045E+01	7,4980E+01	7,2589E+01
	F9	7,0116E+01	5,6236E+01	6,1548E+01	5,5956E+01	7,2342E+01	8,0578E+01
	F10	9,5951E+02	8,0157E+02	8,1181E+02	8,8826E+02	1,0020E+03	9,1411E+02
	F11	1,0012E+03	7,9044E+02	8,5646E+02	8,8993E+02	9,1583E+02	9,1377E+02
	F12	3,6167E-01	3,4578E-01	3,2940E-01	3,3100E-01	3,5528E-01	3,3863E-01
	F13	3,3020E-01	2,7588E-01	2,7646E-01	2,7216E-01	2,8471E-01	3,1749E-01
	F14	2,2720E-01	2,0067E-01	2,1017E-01	2,1541E-01	2,6756E-01	2,5668E-01
	F15	1,4490E+01	1,2008E+01	1,2673E+01	1,1700E+01	1,2902E+01	1,4664E+01
	F16	4,2964E+00	3,7337E+00	3,9662E+00	4,0034E+00	4,2617E+00	4,0397E+00
Hybrid Function	F17	2,1296E+03	2,1727E+03	1,7506E+03	1,5926E+03	2,9557E+03	2,7082E+03
	F18	1,9159E+02	1,5646E+02	1,6118E+02	1,6350E+02	1,8937E+02	1,8439E+02
	F19	1,0495E+01	9,9224E+00	8,8418E+00	7,5414E+00	1,1664E+01	1,0692E+01
	F20	1,4290E+03	9,3118E+02	1,0815E+03	9,9542E+02	9,6288E+02	1,6676E+03
	F21	1,0041E+03	8,6046E+02	6,9710E+02	7,9162E+02	1,2236E+03	9,2858E+02
	F22	2,0286E+02	1,2570E+02	1,4829E+02	1,4335E+02	1,8953E+02	2,2246E+02
Composition Functions	F23	3,2948E+02	3,0780E+02	3,2947E+02	3,0767E+02	3,2707E+02	3,2948E+02
	F24	1,9310E+02	1,8277E+02	1,9399E+02	1,8556E+02	1,9448E+02	1,9171E+02
	F25	1,8993E+02	1,8634E+02	1,8632E+02	1,8831E+02	1,8570E+02	1,9114E+02
	F26	1,0031E+02	9,9403E+01	1,0024E+02	1,0027E+02	1,0032E+02	1,0033E+02
	F27	3,6146E+02	2,6983E+02	2,1706E+02	1,8467E+02	3,1383E+02	2,0322E+02
	F28	7,2411E+02	5,2098E+02	5,7335E+02	5,8296E+02	7,3608E+02	8,6722E+02
Resumen	F29	2,5070E+02	2,4446E+02	2,4597E+02	2,3112E+02	2,7664E+02	2,5095E+02
	F30	6,7078E+02	4,5681E+02	4,7972E+02	5,0346E+02	5,3693E+02	4,9397E+02
Resumen		0 (5º)	14 (1º)	5 (3º)	9 (2º)	2 (4º)	0 (5º)

Figura 7.11: Comparativa de repulsión algoritmo BFA con dimensión 10.

7.3. Comparativa de la componente atractiva para dimensión 30

Grasshopper Optimization Algorithm

Si comparamos la Figura 7.12 con su equivalente en dimensión 10 (Figura 7.1) podemos ver que el algoritmo en gran parte se comporta de la misma forma. El valor de parámetro 0.75 sigue siendo el mejor de ellos pero el 0.5 ha perdido mucha eficacia.

Comparando los resultados obtenidos en ambas versiones, podemos ver que son mucho peores los de dimensión 30 que los de 10. Esto era algo de esperar ya que es un problema más complejo y se han utilizado unas poblaciones del mismo tamaño de las anteriores. Los mayores cambios se notan sobre todo en las funciones unimodales, en las que por ejemplo en la función 1 obteníamos un 7.6728E+06 como mejor valor y en este caso se obtiene 3.0233E+09.

Tenemos que destacar sobre estos resultados que la función 15 sigue obteniendo unos resultados muy malos con respecto a las demás funciones.

Firefly Algorithm

En estas variantes del algoritmo (Figura 7.13), la componente de atracción deja de ser tan robusta ya que se observan diferencias significativas entre los resultados de las distintas variantes. Una diferencia que encontramos grande con respecto a la dimensión 10 es que ya el mejor no es el mismo, si no que es el valor más alto de todos, el 1.5. Otra observación que podemos hacer es que el que antes era uno de los mejores (1.25) ahora es uno de los peores.

Un aspecto que merece ser nombrado de este algoritmo es que las soluciones en dimensión 30 no son mucho peores que las que obteníamos en dimensión 10. Esto sorprende porque el número de individuos de la población es el mismo en ambas pruebas. Si pensamos en tiempo de ejecución, al incrementar la dimensión del problema el tiempo aumenta de forma importante y si además incrementamos el tamaño de la población el tiempo aumentará mucho más aún. Por tanto, es interesante saber que en este caso el número de individuos de la población no influye demasiado a la hora de incrementar la dimensión.

GOA	FUNCIONES	0,5	0,75	1	1,25	1,5
Unimodal Functions	F1	3,6762E+09	3,2759E+09	3,4313E+09	2,9562E+09	3,0233E+09
	F2	1,3485E+11	1,2654E+11	1,1744E+11	1,2372E+11	1,2850E+11
	F3	5,8776E+06	4,5748E+06	1,3954E+06	7,0082E+06	5,0388E+07
Simple Multimodal Functions	F4	3,5962E+04	2,9317E+04	3,8222E+04	3,9520E+04	3,3870E+04
	F5	2,1352E+01	2,1258E+01	2,1321E+01	2,1297E+01	2,1302E+01
	F6	4,8194E+01	4,7886E+01	4,7501E+01	4,8316E+01	4,8097E+01
	F7	1,2882E+03	1,0712E+03	1,1403E+03	1,1363E+03	1,1804E+03
	F8	4,9854E+02	4,7350E+02	5,0963E+02	4,7162E+02	5,0182E+02
	F9	6,1662E+02	5,7338E+02	5,7242E+02	5,6611E+02	6,1655E+02
	F10	9,0008E+03	8,8847E+03	9,0128E+03	9,2884E+03	9,1700E+03
	F11	9,3816E+03	9,3765E+03	9,2890E+03	9,4103E+03	9,2104E+03
	F12	6,3361E+00	5,9398E+00	7,1846E+00	6,4745E+00	5,9495E+00
	F13	1,1079E+01	9,7764E+00	1,0570E+01	9,9107E+00	1,0070E+01
	F14	3,4982E+02	3,9286E+02	3,5117E+02	3,7830E+02	3,8621E+02
	F15	5,7991E+07	2,2830E+07	1,3941E+07	1,9197E+07	2,0324E+07
	F16	1,4098E+01	1,4324E+01	1,4222E+01	1,4213E+01	1,4216E+01
Hybrid Function	F17	4,8319E+08	1,6535E+08	3,8370E+08	2,7767E+08	2,2633E+08
	F18	1,0973E+10	7,5794E+09	8,8954E+09	9,7533E+09	6,6395E+09
	F19	1,1010E+03	9,9035E+02	1,3994E+03	1,4001E+03	1,0767E+03
	F20	1,0236E+07	2,5041E+07	9,6256E+06	1,0872E+07	2,9478E+07
	F21	3,3354E+08	1,6873E+08	1,9293E+08	1,1433E+08	2,3835E+08
	F22	2,7112E+04	9,1195E+04	1,2100E+05	1,3417E+04	1,3848E+04
Composition Functions	F23	1,9907E+03	1,9697E+03	1,8302E+03	2,1070E+03	1,5932E+03
	F24	5,2660E+02	5,0845E+02	5,2635E+02	5,1911E+02	5,1248E+02
	F25	4,3091E+02	3,9886E+02	4,0669E+02	4,3155E+02	3,8433E+02
	F26	2,3378E+02	3,1726E+02	2,7494E+02	3,8282E+02	3,2285E+02
	F27	1,8873E+03	1,8651E+03	2,0311E+03	1,9670E+03	1,7099E+03
	F28	8,7146E+03	8,0937E+03	7,4541E+03	7,3151E+03	8,0480E+03
	F29	7,3705E+08	5,3782E+08	6,0646E+08	7,1650E+08	1,0526E+09
	F30	8,2762E+06	1,2329E+07	8,5930E+06	1,0931E+07	9,6505E+06
Resumen		4 (3º)	10 (1º)	5 (4º)	4 (3º)	7 (2º)

Figura 7.12: Comparativa de atracción algoritmo GOA con dimensión 30.

FA	FUNCIONES	0,5	0,75	1	1,25	1,5
Unimodal Functions	F1	5,9727E+09	5,3414E+09	6,3369E+09	5,8940E+09	6,5881E+09
	F2	1,7851E+11	1,8824E+11	1,7461E+11	1,7723E+11	1,8340E+11
	F3	2,4273E+06	2,9973E+07	1,7628E+07	2,5480E+07	3,0623E+07
Simple Multimodal Functions	F4	5,8125E+04	6,1836E+04	6,1521E+04	6,6576E+04	5,3022E+04
	F5	2,1287E+01	2,1334E+01	2,1340E+01	2,1314E+01	2,1323E+01
	F6	5,1499E+01	5,1833E+01	5,1766E+01	5,0674E+01	5,1856E+01
	F7	1,5422E+03	1,6286E+03	1,6680E+03	1,7114E+03	1,5083E+03
	F8	5,8705E+02	5,9375E+02	5,7059E+02	5,9731E+02	5,7833E+02
	F9	7,1232E+02	7,1094E+02	7,1933E+02	7,2115E+02	6,9973E+02
	F10	1,0325E+04	1,0400E+04	9,9755E+03	1,0273E+04	1,0087E+04
	F11	1,0192E+04	1,0109E+04	1,0226E+04	9,9649E+03	1,0071E+04
	F12	6,4819E+00	6,1362E+00	6,3921E+00	6,7760E+00	6,4417E+00
	F13	1,3341E+01	1,3802E+01	1,3166E+01	1,3419E+01	1,2461E+01
	F14	5,5293E+02	5,2007E+02	5,9521E+02	5,5699E+02	5,7635E+02
	F15	9,1324E+07	9,6259E+07	8,2937E+07	9,3330E+07	8,2819E+07
	F16	1,4402E+01	1,4322E+01	1,4397E+01	1,4397E+01	1,4319E+01
Hybrid Function	F17	9,9474E+08	1,0110E+09	1,1903E+09	1,3624E+09	8,1917E+08
	F18	2,2217E+10	1,7634E+10	1,9283E+10	1,8512E+10	1,6904E+10
	F19	3,2319E+03	2,3242E+03	1,9086E+03	2,3523E+03	2,9772E+03
	F20	2,2829E+08	4,0072E+08	3,3293E+08	6,0895E+08	2,1267E+08
	F21	3,7238E+08	4,0123E+08	5,1519E+08	4,7841E+08	2,9915E+08
	F22	1,4478E+06	4,4844E+05	1,3228E+06	5,0566E+05	1,5375E+06
Composition Functions	F23	3,1468E+03	3,4365E+03	3,5947E+03	3,3817E+03	3,3678E+03
	F24	6,4658E+02	6,1337E+02	6,3336E+02	6,9374E+02	6,3369E+02
	F25	6,1552E+02	5,6552E+02	6,0139E+02	5,7979E+02	5,7457E+02
	F26	5,6995E+02	5,9217E+02	5,5112E+02	5,8314E+02	5,6736E+02
	F27	2,7061E+03	2,8945E+03	3,0540E+03	2,8516E+03	2,8016E+03
	F28	1,0985E+04	1,0496E+04	1,0898E+04	1,0163E+04	1,0204E+04
	F29	1,4286E+09	1,3823E+09	1,4119E+09	1,4281E+09	1,4940E+09
	F30	2,7284E+07	3,8065E+07	3,7554E+07	2,6131E+07	4,6174E+07
Resumen		4 (3º)	6 (2º)	6 (2º)	4 (3º)	10 (1º)

Figura 7.13: Comparativa de atracción algoritmo FA con dimensión 30.

Dragonfly Algorithm

En la Figura 7.5 vemos el comportamiento que tiene dicho algoritmo para dimensión 10, que es parecido al que vemos en la Figura 7.14 (dimensión 30) ya que el mejor sigue siendo el que pondera la atracción con 0.75. Sin embargo una diferencia que encontramos muy significativa es que en dimensión 10 las ponderaciones que proporcionaban mejores resultados en las funciones composición eran las altas y ahora es la 0.75.

También podemos apreciar que para dimensión 30 en funciones unimodales es mejor el valor de atracción más alto, en este caso el 1.5. Esto es así porque al aumentar la atracción hacia la mejor solución de la población en entornos homogéneos funciona muy bien ya que solo hay un mínimo y se ve claramente donde está. De esta forma, podremos ir más rápido hacia él aunque perdamos la diversidad.

Hablando del tamaño de la población, vemos que en este algoritmo tampoco hace falta que lo aumentemos conforme incrementamos la complejidad del problema ya que las soluciones no empeoran mucho con respecto a las obtenidas en dimensión 10.

Bacterial Foraging Algorithm

En este algoritmo, el comportamiento cambia bastante cuando cambiamos la dimensión del problema. Como vemos en la Figura 7.15 el mejor, cuando la dimensión es 30, es el que pondera la atracción con 0.75, mientras que en el estudio de dimensión 10 (Figura 7.7) era el peor de todos. Además, uno de los mejores en dimensión 10 eran el 1 y 1.25 mientras que aquí son los dos peores.

También podemos observar es que los resultados son además bastante poco informativos ya que no hay ningún tipo de funciones en las que alguna variante del algoritmo predomine significativamente.

DA	FUNCIONES	0,5	0,75	1	1,25	1,5
Unimodal Functions	F1	1,6184E+09	1,6315E+09	1,5453E+09	1,5976E+09	1,5748E+09
	F2	9,0425E+10	9,3475E+10	9,1963E+10	9,2069E+10	8,7290E+10
	F3	5,0333E+06	8,5097E+04	8,2596E+04	8,2885E+04	8,2105E+04
Simple Multimodal Functions	F4	1,9434E+04	1,9507E+04	2,4138E+04	2,6194E+04	1,9325E+04
	F5	2,0993E+01	2,1002E+01	2,1026E+01	2,0980E+01	2,0993E+01
	F6	4,5328E+01	4,4774E+01	4,5051E+01	4,5144E+01	4,4761E+01
	F7	7,6430E+02	7,8986E+02	7,7454E+02	7,6529E+02	7,8356E+02
	F8	3,6321E+02	3,6696E+02	3,8999E+02	3,6730E+02	3,7988E+02
	F9	4,0825E+02	4,1047E+02	4,2579E+02	4,1205E+02	4,1223E+02
	F10	7,6937E+03	7,6874E+03	7,6938E+03	7,6687E+03	7,6849E+03
	F11	8,1901E+03	8,1682E+03	8,2003E+03	8,3348E+03	8,2704E+03
	F12	3,1045E+00	3,0001E+00	3,0980E+00	3,0212E+00	2,9849E+00
	F13	9,8599E+00	9,5989E+00	9,7834E+00	9,7513E+00	9,6642E+00
	F14	2,7504E+02	2,8466E+02	2,8468E+02	2,8043E+02	2,7705E+02
	F15	2,6735E+05	2,4911E+05	2,3587E+05	2,4656E+05	2,4313E+05
	F16	1,3131E+01	1,3129E+01	1,3006E+01	1,3021E+01	1,3094E+01
Hybrid Function	F17	2,5976E+08	3,1254E+08	3,2347E+08	3,1992E+08	2,6969E+08
	F18	1,0162E+10	1,0302E+10	1,0746E+10	9,8299E+09	9,7430E+09
	F19	7,4225E+02	6,9083E+02	7,1889E+02	7,2108E+02	6,4776E+02
	F20	2,7876E+05	3,6544E+05	3,3008E+05	2,7972E+05	4,4894E+05
	F21	2,9430E+08	2,8131E+08	1,2697E+08	2,0571E+08	1,3190E+08
	F22	1,3178E+06	1,3938E+05	2,5869E+05	1,9360E+05	3,5879E+05
Composition Functions	F23	2,0003E+02	2,0000E+02	2,0005E+02	2,0036E+02	2,0040E+02
	F24	2,0003E+02	2,0028E+02	2,0002E+02	2,0002E+02	2,3337E+02
	F25	2,0871E+02	2,0001E+02	2,2551E+02	2,0000E+02	2,0000E+02
	F26	1,5600E+02	1,4666E+02	1,5478E+02	1,5067E+02	1,5952E+02
	F27	3,0325E+02	2,0001E+02	5,7793E+02	4,3659E+02	6,1326E+02
	F28	2,0007E+02	5,1306E+02	5,7919E+02	2,0002E+02	1,0275E+03
	F29	1,6972E+06	7,4957E+04	8,0891E+07	3,3923E+05	7,6204E+07
	F30	1,4653E+06	2,1721E+03	5,6176E+05	2,1589E+06	4,5737E+05
Resumen		6 (3º)	8 (1º)	4 (4º)	3 (5º)	7(2º)

Figura 7.14: Comparativa de atracción algoritmo DA con dimensión 30.

BFA	FUNCIONES	0,5	0,75	1	1,25	1,5
Unimodal Functions	F1	2,0236E+06	2,0787E+06	2,2086E+06	2,2014E+06	2,2500E+06
	F2	7,3361E+06	7,2722E+06	7,3684E+06	6,7155E+06	7,0678E+06
	F3	9,7776E+04	1,0423E+05	1,0304E+05	1,1374E+05	1,1848E+05
Simple Multimodal Functions	F4	1,8126E+01	1,7281E+01	1,9104E+01	1,7974E+01	1,8496E+01
	F5	2,0954E+01	2,0930E+01	2,0948E+01	2,0959E+01	2,0948E+01
	F6	3,1119E+01	3,2285E+01	3,1281E+01	3,1451E+01	3,0937E+01
	F7	1,0618E+00	1,0590E+00	1,0618E+00	1,0644E+00	1,0653E+00
	F8	2,8640E+02	3,0287E+02	2,9583E+02	3,0520E+02	3,1098E+02
	F9	4,4732E+02	4,2136E+02	4,4218E+02	3,9773E+02	4,3537E+02
	F10	3,3008E+03	3,7380E+03	3,4022E+03	3,5513E+03	3,7207E+03
	F11	3,4489E+03	3,3693E+03	3,4953E+03	3,4228E+03	3,5695E+03
	F12	1,1153E+00	1,1804E+00	1,0939E+00	1,2717E+00	1,1688E+00
	F13	4,5890E-01	4,5731E-01	4,6933E-01	4,5720E-01	4,5648E-01
	F14	2,6146E-01	2,6335E-01	2,5656E-01	2,5715E-01	2,5155E-01
	F15	2,8326E+01	2,9995E+01	2,9098E+01	2,9507E+01	3,0963E+01
	F16	1,3246E+01	1,3261E+01	1,3275E+01	1,3403E+01	1,3341E+01
Hybrid Function	F17	6,6666E+04	5,2220E+04	6,1421E+04	4,1167E+04	5,5788E+04
	F18	4,5531E+04	4,0867E+04	4,4819E+04	4,5390E+04	4,2513E+04
	F19	1,2978E+01	1,3357E+01	1,3187E+01	1,3158E+01	1,1475E+01
	F20	1,2670E+04	1,0746E+04	1,2902E+04	1,6591E+04	8,9177E+03
	F21	4,7328E+04	3,8013E+04	4,5864E+04	4,0804E+04	4,1265E+04
	F22	4,9415E+02	4,9229E+02	4,7087E+02	4,1962E+02	5,1948E+02
Composition Functions	F23	3,1431E+02	3,1433E+02	3,1433E+02	3,1432E+02	3,1436E+02
	F24	2,4079E+02	2,4457E+02	2,4214E+02	2,4136E+02	2,3853E+02
	F25	2,0102E+02	2,0130E+02	2,0101E+02	2,0109E+02	2,0107E+02
	F26	1,0058E+02	1,0054E+02	1,0056E+02	1,0055E+02	1,0055E+02
	F27	4,0261E+02	4,0196E+02	4,0254E+02	4,0254E+02	4,0220E+02
	F28	3,8393E+03	4,0571E+03	3,6947E+03	3,8997E+03	4,0946E+03
	F29	2,1387E+02	2,1406E+02	2,1392E+02	2,1485E+02	2,1343E+02
	F30	8,4787E+02	9,0811E+02	8,3251E+02	8,3265E+02	9,2368E+02
Resumen		7 (2º)	8 (1º)	4 (3º)	4 (3º)	7 (2º)

Figura 7.15: Comparativa de atracción algoritmo BFA con dimensión 30.

7.4. Comparativa de la componente repulsiva para dimensión 30

Dragonfly Algorithm

En la Figura 7.16 se muestra una tabla con los resultados obtenidos del estudio de repulsión en dimensión 30 del algoritmo Dragonfly. Lo que más destaca en dicha tabla y la diferencia de los demás es que una de las variante que mejor se comporta del algoritmo es aquella en la que quitamos la repulsión. Como se ha dicho antes, se ha aumentado la dimensión del problema pero no el número de individuos de la población. Esto hace que haya muy pocas soluciones explorando en un espacio de búsqueda muy grande y la repulsión deje de tener efecto.

Sin embargo, hay ciertas funciones en las que las mejoras que se producen cuando añadimos la componente repulsiva son muy significativas. Por ejemplo las funciones compuestas 29 y 30 destacan por este comportamiento. Al no tener las funciones dibujadas gráficamente no podemos observar su forma pero lo que si sabemos de ellas es que son las únicas dos compuestas que están formadas en su totalidad por funciones híbridas (la F29 está formada por F17, F18 y F19 y la F30 por F20, F21 y F22).

Bacterial Foraging Algorithm

Al ver la tabla de la Figura 7.17 afirmamos más las conclusiones sacadas para el algoritmo Dragonfly en este aspecto: la repulsión no va bien cuando las dimensiones son grandes y las poblaciones pequeñas. En este algoritmo cambia los resultados de forma radical. Si observamos lo que se obtuvo en la experimentación de la repulsión con dimensión 10 (Figura 7.11) vemos que la variante sin repulsión era la peor con diferencia.

Por lo demás si podemos observar similitudes de comportamiento con respecto a la dimensión 10. Pese a que la repulsión en dimensión 10 si mejoraba el comportamiento del algoritmo bastante, sus ponderaciones altas no eran las más adecuadas y eso pasa igual en dimensión 30, las ponderaciones 1.25 y 1.5 siguen siendo las dos peores. Con esto podemos llegar a la conclusión de que aunque BFA es un algoritmo que mejora con la repulsión, hay que tener en cuenta que tampoco es interesante aumentar la componente en exceso.

DA	FUNCIONES	Sin repulsion	0,5	0,75	1	1,25	1,5
Unimodal Functions	F1	1,7032E+09	1,6226E+09	1,7570E+09	1,6315E+09	1,6701E+09	1,5611E+09
	F2	9,1517E+10	8,9910E+10	9,1029E+10	9,3475E+10	8,6676E+10	9,0639E+10
	F3	8,0769E+04	8,1225E+04	9,0390E+04	8,5097E+04	8,4910E+04	8,4599E+04
Simple Multimodal Functions	F4	1,9379E+04	1,8800E+04	1,9213E+04	1,9507E+04	2,0506E+04	1,9216E+04
	F5	2,0973E+01	2,1023E+01	2,1024E+01	2,1002E+01	2,1000E+01	2,1033E+01
	F6	4,5713E+01	4,5892E+01	4,4274E+01	4,4774E+01	4,5276E+01	4,5344E+01
	F7	8,5359E+02	7,4980E+02	7,9299E+02	7,8986E+02	7,7879E+02	7,5508E+02
	F8	3,6928E+02	3,6066E+02	3,7716E+02	3,6696E+02	3,7128E+02	3,6114E+02
	F9	4,0186E+02	4,0290E+02	4,0700E+02	4,1047E+02	4,1617E+02	4,1774E+02
	F10	7,7707E+03	7,7144E+03	7,6621E+03	7,6874E+03	7,5568E+03	7,6640E+03
	F11	8,1200E+03	8,0470E+03	8,1991E+03	8,1682E+03	8,0008E+03	8,2684E+03
	F12	3,3595E+00	2,9514E+00	2,7800E+00	3,0001E+00	2,9726E+00	2,5962E+00
	F13	9,3621E+00	9,5846E+00	9,5267E+00	9,5989E+00	9,9119E+00	9,9887E+00
	F14	2,7692E+02	2,8457E+02	2,6970E+02	2,8466E+02	2,8683E+02	2,6762E+02
	F15	2,9541E+06	2,4975E+05	5,6446E+05	2,4911E+05	2,4693E+05	1,9664E+05
	F16	1,3045E+01	1,3033E+01	1,2990E+01	1,3129E+01	1,3068E+01	1,3003E+01
Hybrid Function	F17	2,2452E+08	5,1863E+08	2,5442E+08	3,1254E+08	2,8478E+08	2,5028E+08
	F18	1,0742E+10	1,0449E+10	1,2079E+10	1,0302E+10	1,0593E+10	1,2435E+10
	F19	6,8919E+02	6,8558E+02	6,3802E+02	6,9083E+02	6,9305E+02	7,9803E+02
	F20	2,3864E+05	2,4316E+05	2,4149E+05	3,6544E+05	2,7621E+05	3,0986E+05
	F21	2,5810E+08	1,5311E+08	1,7497E+08	2,8131E+08	2,1194E+08	2,4326E+08
	F22	2,0985E+05	4,5028E+05	1,2810E+05	1,3938E+05	5,7656E+05	4,9150E+05
Composition Functions	F23	2,0001E+02	2,0005E+02	3,7800E+02	2,0000E+02	5,7821E+02	2,0002E+02
	F24	2,3399E+02	2,0002E+02	2,3465E+02	2,0028E+02	2,0003E+02	2,2138E+02
	F25	2,7699E+02	2,0211E+02	2,0000E+02	2,0001E+02	2,0000E+02	2,0188E+02
	F26	1,4675E+02	1,6190E+02	1,7386E+02	1,4666E+02	1,6878E+02	1,7817E+02
	F27	2,0001E+02	7,1724E+02	4,1014E+02	2,0001E+02	2,0000E+02	2,0001E+02
	F28	2,0003E+02	2,0000E+02	9,9522E+02	5,1306E+02	2,0011E+02	7,8949E+02
Resumen	F29	8,8544E+07	2,0000E+02	1,4215E+03	7,4957E+04	2,8387E+07	1,1263E+04
	F30	1,0704E+06	2,6245E+06	8,1612E+05	2,1721E+03	2,2250E+03	2,0000E+02

Figura 7.16: Comparativa de repulsión algoritmo DA con dimensión 30.

BFA	FUNCIONES	Sn repulsión	0,5	0,75	1	1,25	1,5
Unimodal Functions	F1	2,2757E+06	1,9354E+06	2,2467E+06	2,0787E+06	2,0574E+06	2,1214E+06
	F2	7,1563E+06	7,0433E+06	7,0479E+06	7,2722E+06	7,0821E+06	7,2751E+06
	F3	1,1017E+05	1,1336E+05	1,2140E+05	1,0423E+05	1,1304E+05	1,0766E+05
Simple Multimodal Functions	F4	1,7529E+01	1,9040E+01	1,8739E+01	1,7281E+01	1,9249E+01	1,9057E+01
	F5	2,0951E+01	2,0971E+01	2,0945E+01	2,0930E+01	2,0950E+01	2,0961E+01
	F6	3,0776E+01	3,1268E+01	3,2093E+01	3,2285E+01	3,1930E+01	3,1727E+01
	F7	1,0612E+00	1,0610E+00	1,0603E+00	1,0590E+00	1,0610E+00	1,0612E+00
	F8	3,3008E+02	3,0700E+02	2,7201E+02	3,0287E+02	3,1772E+02	3,0661E+02
	F9	3,9291E+02	4,2947E+02	4,7110E+02	4,2136E+02	4,3339E+02	4,2933E+02
	F10	3,5299E+03	3,6274E+03	3,6874E+03	3,7380E+03	3,6280E+03	3,6804E+03
	F11	3,3204E+03	3,5637E+03	3,4515E+03	3,3693E+03	3,5110E+03	3,4908E+03
	F12	1,1581E+00	1,1521E+00	1,1461E+00	1,1804E+00	1,1482E+00	1,1637E+00
	F13	4,5190E-01	4,4515E-01	4,4396E-01	4,5731E-01	4,6172E-01	4,4587E-01
	F14	2,7249E-01	2,5355E-01	2,6145E-01	2,6335E-01	2,7125E-01	2,6014E-01
	F15	3,2756E+01	3,1894E+01	3,0588E+01	2,9995E+01	3,0974E+01	3,1729E+01
	F16	1,3432E+01	1,3466E+01	1,3418E+01	1,3261E+01	1,3326E+01	1,3531E+01
Hybrid Function	F17	7,6846E+04	5,4387E+04	5,9369E+04	5,2220E+04	5,6113E+04	6,9545E+04
	F18	4,1833E+04	4,3477E+04	4,6805E+04	4,0867E+04	4,0265E+04	3,7373E+04
	F19	1,2054E+01	1,4000E+01	1,2760E+01	1,3357E+01	1,1836E+01	1,3794E+01
	F20	1,1103E+04	1,0554E+04	1,3640E+04	1,0746E+04	1,1462E+04	1,2198E+04
	F21	4,6036E+04	4,6321E+04	3,8036E+04	3,8013E+04	5,5295E+04	5,0163E+04
	F22	4,1285E+02	4,1857E+02	4,7604E+02	4,9229E+02	5,8633E+02	4,7174E+02
Composition Functions	F23	3,1436E+02	3,1436E+02	3,1430E+02	3,1433E+02	3,1435E+02	3,1431E+02
	F24	2,4219E+02	2,3835E+02	2,4638E+02	2,4457E+02	2,3974E+02	2,3601E+02
	F25	2,0102E+02	2,0112E+02	2,0111E+02	2,0130E+02	2,0123E+02	2,0111E+02
	F26	1,0052E+02	1,0053E+02	1,0053E+02	1,0054E+02	1,1049E+02	1,0052E+02
	F27	4,0252E+02	4,0242E+02	4,0243E+02	4,0196E+02	4,0216E+02	4,0237E+02
	F28	3,7951E+03	3,8176E+03	3,9185E+03	4,0571E+03	3,9254E+03	3,8709E+03
	F29	2,1434E+02	2,1331E+02	2,1378E+02	2,1406E+02	2,1446E+02	2,1451E+02
	F30	8,7987E+02	7,7205E+02	8,5179E+02	9,0811E+02	8,7569E+02	8,3271E+02
Resumen		7 (2º)	6 (3º)	4 (4º)	9 (1º)	3 (5º)	1 (6º)

Figura 7.17: Comparativa de repulsión algoritmo BFA con dimensión 30.

7.5. Comparativa entre algoritmos

En esta comparativa, veremos qué algoritmos dan los mejores resultados. Para ello, en primer lugar, se ha realizado una tabla de comparación para los algoritmos tal cual los proporciona el autor, sin modificar ningun parámetro. En otra tabla se han contemplado las mejores versiones de los algoritmos conseguidas en este proyecto modificando la atracción y la repulsión.

A continuación se muestran y se comentan 2 Figuras. En la primera (Figura 7.18) podemos ver los resultados medios obtenidos para los algoritmos con los parámetros establecidos por el autor. Seguidamente podemos ver en la Figura 7.19 las mejores variantes conseguidas de cada algoritmo, también mostrando la media conseguida en cada una de las funciones.

En la Figura 7.18 vemos que los mejores algoritmos son Grasshopper Optimization Algorithm y Bacterial Foraging Algorithm. El que obtiene el primer puesto es un algoritmo que posee atracción y repulsión y el segundo puesto es para uno que solo posee atracción. Esto nos da a entender que añadir una componente repulsiva puede mejorar un algoritmo pero no es suficiente, ya que el algoritmo Dragonfly también posee repulsión y es peor que GOA. Se podría comprobar en un futuro es si añadiendo una componente repulsiva al algoritmo GOA, conseguimos mejorar las soluciones lo suficiente como para decir que la técnica es importante.

Además, en la Figura 7.19 vemos que los dos mejores siguen siendo los dichos anteriormente y además en el mismo orden. Por lo tanto se muestra que aunque la mejora de las soluciones es significativa, no es suficiente como para cambiar el orden de los mejores algoritmos.

Hablando de grupos de funciones no podemos decir que ningún algoritmo destaque en alguno de ellos ya que las mejores soluciones están muy repartidas entre todos. En las funciones unimodales podemos observar que BFA consigue mejorar mucho los resultados en las dos primeras, esto es porque al ser entornos homogéneos, la repulsión funciona bastante bien. Por otro lado vemos que en las funciones multimodales en las que BFA es mejor que GOA, las diferencias de las soluciones son muy significativas.

	Funciones	GOA	FA	DA	BFA
Unimodal Functions	F1	8,5202E+06	8,8585E+08	6,3420E+08	1,9671E+05
	F2	1,4797E+09	1,8157E+10	1,0211E+10	1,2443E+05
	F3	1,0452E+04	6,7800E+07	1,1240E+04	1,3767E+04
Simple Multimodal Functions	F4	1,4000E+02	7,5988E+03	4,5713E+03	1,0540E+00
	F5	2,0205E+01	2,1006E+01	2,0345E+01	2,0414E+01
	F6	7,6783E+00	1,4940E+01	1,1283E+01	1,0649E+01
	F7	2,3894E+01	4,2172E+02	1,1364E+02	5,0760E-01
	F8	4,7752E+01	1,5728E+02	9,4949E+01	5,8592E+01
	F9	5,3395E+01	1,5488E+02	8,3076E+01	6,2565E+01
	F10	1,0549E+03	3,0841E+03	1,7550E+03	9,3873E+02
	F11	1,1006E+03	2,8148E+03	1,9307E+03	7,7266E+02
	F12	8,3667E-01	5,1860E+00	1,2639E+00	3,0673E-01
	F13	1,2597E+00	7,5580E+00	5,6420E+00	2,5145E-01
	F14	6,0895E+00	9,1648E+01	3,6928E+01	2,3760E-01
	F15	8,0105E+01	1,1479E+06	2,3484E+04	1,1297E+01
	F16	3,3010E+00	4,3516E+00	3,5737E+00	3,9206E+00
Hybrid Function	F17	1,8393E+04	2,5447E+07	3,5832E+05	1,6820E+03
	F18	9,6723E+03	1,2513E+09	3,9633E+06	1,5594E+02
	F19	6,0185E+00	4,0318E+02	3,9633E+01	8,7855E+00
	F20	2,6846E+02	1,8292E+04	1,2819E+04	5,7338E+02
	F21	2,0524E+03	1,2701E+08	1,7242E+05	7,3301E+02
Composition Functions	F22	5,0883E+01	9,6927E+02	2,8067E+02	1,1988E+02
	F23	3,4974E+02	1,0579E+03	2,1775E+02	2,7151E+02
	F24	1,6086E+02	2,7030E+02	2,0091E+02	1,8994E+02
	F25	1,7552E+02	2,6116E+02	1,9971E+02	1,8177E+02
	F26	1,0097E+02	1,7349E+02	1,1236E+02	9,7936E+01
	F27	8,7087E+01	9,2762E+02	2,0000E+02	1,6600E+02
	F28	6,4091E+02	2,8003E+03	2,0030E+02	5,8160E+02
	F29	6,8667E+03	9,9795E+07	6,2395E+05	2,3665E+02
	F30	2,0505E+03	2,4055E+06	3,8448E+03	4,7493E+02
Resumen		12 (2º)	0 (4º)	2 (3º)	16 (1º)

Figura 7.18: Comparativa de los algoritmos con parámetros del autor en dimensión 10.

	Funciones	GOA	FA	DA	BFA
Unimodal Functions	F1	7,6728E+06	8,8585E+08	5,8951E+08	2,4096E+05
	F2	1,3356E+09	1,8157E+10	1,1126E+10	1,3376E+05
	F3	9,6551E+03	6,7800E+07	1,0145E+04	1,0039E+04
Simple Multimodal Functions	F4	1,2901E+02	7,5988E+03	4,4413E+03	1,7983E+00
	F5	2,0095E+01	2,1000E+01	2,0354E+01	2,0395E+01
	F6	7,6574E+00	1,4940E+01	1,1534E+01	1,1308E+01
	F7	2,2374E+01	4,2172E+02	1,3615E+02	5,0489E-01
	F8	4,7711E+01	1,5728E+02	9,4993E+01	5,4269E+01
	F9	5,0179E+01	1,5488E+02	8,5446E+01	5,6236E+01
	F10	1,1056E+03	3,0841E+03	1,6887E+03	8,0157E+02
	F11	7,8116E-01	2,8148E+03	1,9707E+03	7,9044E+02
	F12	1,2254E+00	5,1860E+00	1,3424E+00	3,4578E-01
	F13	5,9401E+00	7,5580E+00	6,1667E+00	2,7588E-01
	F14	9,3217E+01	9,1648E+01	3,5843E+01	2,0067E-01
	F15	1,0481E+03	1,1479E+06	3,9548E+04	1,2008E+01
	F16	3,2499E+00	4,3516E+00	3,5791E+00	3,7337E+00
Hybrid Function	F17	1,3991E+04	2,5447E+07	4,6439E+05	2,1727E+03
	F18	1,2615E+04	1,1450E+09	1,0230E+07	1,5646E+02
Composition Functions	F19	6,1232E+00	4,0318E+02	4,6477E+01	9,9224E+00
	F20	2,1742E+02	1,8292E+04	1,0916E+04	9,3118E+02
	F21	2,0991E+03	1,2701E+08	3,6712E+05	8,6046E+02
	F22	5,5257E+01	9,6927E+02	3,1048E+02	1,2570E+02
	F23	3,5104E+02	1,0579E+03	2,0001E+02	3,0780E+02
	F24	1,6126E+02	2,7030E+02	1,9889E+02	1,8277E+02
	F25	1,7242E+02	2,6116E+02	1,9985E+02	1,8634E+02
	F26	1,0099E+02	1,7349E+02	1,1584E+02	9,9403E+01
	F27	7,8970E+01	9,2762E+02	2,0001E+02	2,6983E+02
	F28	6,4318E+02	2,8003E+03	3,9474E+02	5,2098E+02
Resumen	F29	4,2840E+03	9,9795E+07	2,6027E+05	2,4446E+02
	F30	2,0176E+03	2,4055E+06	3,7315E+03	4,5681E+02
Resumen		13 (2º)	0 (4º)	2 (3º)	15 (1º)

Figura 7.19: Comparativa de los algoritmos con mejores factores de atracción-repulsión en dimensión 10.

Si vemos la función 4 y 7 (Figura 7.20), como hemos dicho antes son entornos homogéneos en los que la repulsión realiza una mejora grande de las soluciones. Por lo tanto es el algoritmo BFA el que consigue un mínimo más pequeño con una diferencia significativa. El mismo comportamiento lo observamos en las funciones 14 y 15, que también poseen un entorno homogéneo aunque en menor medida que las anteriores.

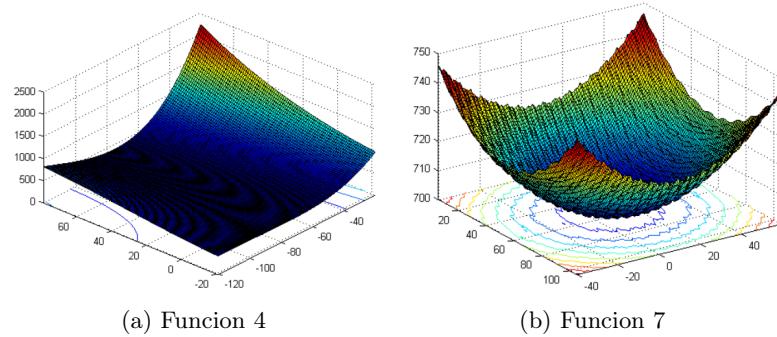


Figura 7.20: Gráficas de las funciones 4 y 7.

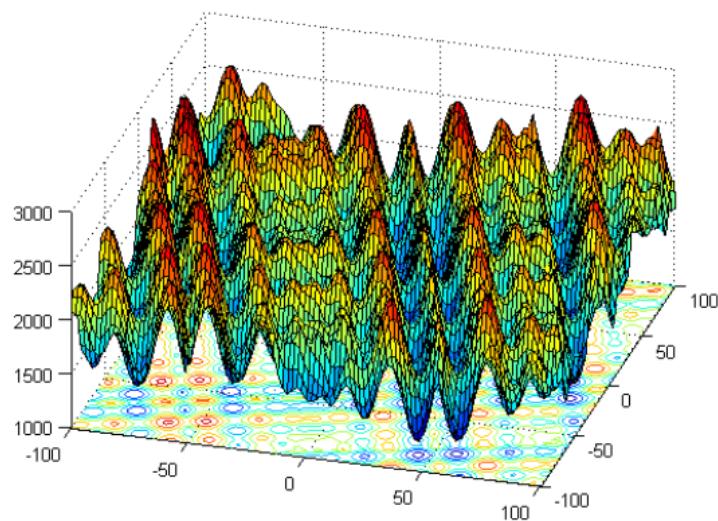


Figura 7.21: Gráfica 3D de la función 11.

Por otra parte, el algoritmo GOA también consigue mejorar los resultados significativamente en algunas funciones concretas como por ejemplo la 11. Viendo su forma en

la Figura 7.21 vemos que es todo lo contrario a las funciones que se han mencionado en el párrafo anterior. Es del tipo de funciones que poseen muchos mínimos y además están muy juntos. La repulsión en éste tipo de funciones no funciona tan bien, por lo que es un aspecto a tener en cuenta.

La comparación de algoritmos en dimensión 30 cambia por completo con respecto a la vista en dimensión 10. Si observamos las tablas que muestran las Figuras 7.22 y 7.23 vemos que en este caso los mejores algoritmos son los dos que poseen repulsión. Por lo tanto, aunque hayamos visto que la repulsión en dimensión 30 con tan pocos individuos en la población funciona peor que cuando la dimensión es más pequeña, podemos afirmar que con repulsión funcionan mejor que otros algoritmos.

		Funciones	GOA	FA	DA	BFA
Unimodal Functions	F1	3,4313E+09	6,3369E+09	1,5453E+09	2,2086E+06	
	F2	1,1744E+11	1,7461E+11	9,1963E+10	7,3684E+06	
	F3	1,3954E+06	1,7628E+07	8,2596E+04	1,0304E+05	
Simple Multimodal Functions	F4	3,8222E+04	6,1521E+04	2,4138E+04	1,9104E+01	
	F5	2,1321E+01	2,1340E+01	2,1026E+01	2,0948E+01	
	F6	4,7501E+01	5,1766E+01	4,5051E+01	3,1281E+01	
	F7	1,1403E+03	1,6680E+03	7,7454E+02	1,0618E+00	
	F8	5,0963E+02	5,7059E+02	3,8999E+02	2,9583E+02	
	F9	5,7242E+02	7,1933E+02	4,2579E+02	4,4218E+02	
	F10	9,0128E+03	9,9755E+03	7,6938E+03	3,4022E+03	
	F11	9,2890E+03	1,0226E+04	8,2003E+03	3,4953E+03	
	F12	7,1846E+00	6,3921E+00	3,0980E+00	1,0939E+00	
	F13	1,0570E+01	1,3166E+01	9,7834E+00	4,6933E-01	
	F14	3,5117E+02	5,9521E+02	2,8468E+02	2,5656E-01	
	F15	1,3941E+07	8,2937E+07	2,3587E+05	2,9098E+01	
	F16	1,4222E+01	1,4397E+01	1,3006E+01	1,3275E+01	
Hybrid Function	F17	3,8370E+08	1,1903E+09	3,2347E+08	6,1421E+04	
	F18	8,8954E+09	1,9283E+10	1,0746E+10	4,4819E+04	
	F19	1,3994E+03	1,9086E+03	7,1889E+02	1,3187E+01	
	F20	9,6256E+06	3,3293E+08	3,3008E+05	1,2902E+04	
	F21	1,9293E+08	5,1519E+08	1,2697E+08	4,5864E+04	
	F22	1,2100E+05	1,3228E+06	2,5869E+05	4,7087E+02	
Composition Functions	F23	1,8302E+03	3,5947E+03	2,0005E+02	3,1433E+02	
	F24	5,2635E+02	6,3336E+02	2,0002E+02	2,4214E+02	
	F25	4,0669E+02	6,0139E+02	2,2551E+02	2,0101E+02	
	F26	2,7494E+02	5,5112E+02	1,5478E+02	1,0056E+02	
	F27	2,0311E+03	3,0540E+03	5,7793E+02	4,0254E+02	
	F28	7,4541E+03	1,0898E+04	5,7919E+02	3,6947E+03	
	F29	6,0646E+08	1,4119E+09	8,0891E+07	2,1392E+02	
	F30	8,5930E+06	3,7554E+07	5,6176E+05	8,3251E+02	
Resumen		0 (3º)	0 (3º)	3 (2º)	27 (1º)	

Figura 7.22: Comparativa de los algoritmos con parámetros del autor en dimensión 30.

	Funciones	GOA	FA	DA	BFA
Unimodal Functions	F1	3,2759E+09	6,5881E+09	1,6226E+09	2,0787E+06
	F2	1,2654E+11	1,8340E+11	8,9910E+10	7,2722E+06
	F3	4,5748E+06	3,0623E+07	8,1225E+04	1,0423E+05
Simple Multimodal Functions	F4	2,9317E+04	5,3022E+04	1,8800E+04	1,7281E+01
	F5	2,1258E+01	2,1323E+01	2,1023E+01	2,0930E+01
	F6	4,7886E+01	5,1856E+01	4,5892E+01	3,2285E+01
	F7	1,0712E+03	1,5083E+03	7,4980E+02	1,0590E+00
	F8	4,7350E+02	5,7833E+02	3,6066E+02	3,0287E+02
	F9	5,7338E+02	6,9973E+02	4,0290E+02	4,2136E+02
	F10	8,8847E+03	1,0087E+04	7,7144E+03	3,7380E+03
	F11	9,3765E+03	1,0071E+04	8,0470E+03	3,3693E+03
	F12	5,9398E+00	6,4417E+00	2,9514E+00	1,1804E+00
	F13	9,7764E+00	1,2461E+01	9,5846E+00	4,5731E-01
	F14	3,9286E+02	5,7635E+02	2,8457E+02	2,6335E-01
	F15	2,2830E+07	8,2819E+07	2,4975E+05	2,9995E+01
	F16	1,4324E+01	1,4319E+01	1,3033E+01	1,3261E+01
Hybrid Function	F17	1,6535E+08	8,1917E+08	5,1863E+08	5,2220E+04
	F18	7,5794E+09	1,6904E+10	1,0449E+10	4,0867E+04
Composition Functions	F19	9,9035E+02	2,9772E+03	6,8558E+02	1,3357E+01
	F20	2,5041E+07	2,1267E+08	2,4316E+05	1,0746E+04
	F21	1,6873E+08	2,9915E+08	1,5311E+08	3,8013E+04
	F22	9,1195E+04	1,5375E+06	4,5028E+05	4,9229E+02
	F23	1,9697E+03	3,3678E+03	2,0005E+02	3,1433E+02
	F24	5,0845E+02	6,3369E+02	2,0002E+02	2,4457E+02
	F25	3,9886E+02	5,7457E+02	2,0211E+02	2,0130E+02
	F26	3,1726E+02	5,6736E+02	1,6190E+02	1,0054E+02
	F27	1,8651E+03	2,8016E+03	7,1724E+02	4,0196E+02
	F28	8,0937E+03	1,0204E+04	2,0000E+02	4,0571E+03
Resumen	F29	5,3782E+08	1,4940E+09	2,0000E+02	2,1406E+02
	F30	1,2329E+07	4,6174E+07	2,6245E+06	9,0811E+02
Resumen		0 (3º)	0 (3º)	7 (2º)	23 (1º)

Figura 7.23: Comparativa de los algoritmos con mejores factores de atracción-repulsión en dimensión 30.

Capítulo 8: Conclusiones

8. Conclusiones

El objetivo de este trabajo es estudiar el comportamiento de las componentes atractiva y repulsiva en algoritmos aplicados a la optimización de funciones. Para poder realizar una comparativa los algoritmos que se han estudiado han sido dos que poseen atracción (Grasshopper Optimization Algorithm y Firefly Algorithm) y otros dos que poseen atracción y repulsión (Dragonfly Algorithm y Bacterial Foraging Algorithm).

Para el estudio de las componentes mencionadas en el párrafo anterior se han realizado 2 tipos de experimentos. En el primero se ha realizado un ajuste de los parámetros de atracción y repulsión de cada algoritmo. Una vez conseguida la mejora del comportamiento de dichos algoritmos, se ha realizado una comparación de todos ellos.

Después de observar el comportamiento de la repulsión podemos decir que, vistas las comparativas que se han realizado en el último apartado, es un aspecto a tener en cuenta para futuras implementaciones. Aunque en dimensiones más pequeñas como es el caso de dimensión 10 no sea suficiente, cuando se aumenta dicha variable vemos que mejora con diferencia los resultados de los algoritmos que no la poseen.

La técnica de repulsión será algo a tener en cuenta siempre que necesitemos crear más diversidad o acelerar un poco la obtención de las mejores soluciones. Si la utilizamos para separar las soluciones tendremos menos concentración de las mismas por lo que el carácter explorativo del algoritmo aumentará. Si la usamos para huir de las peores soluciones obtenidas, evitamos explorar zonas poco prometedoras por lo que se avanzará hacia las mejores zonas de forma más rápida.

Cabe destacar que la atracción es otra técnica que funciona bastante bien y que ajustándola en la medida correcta puede darnos algoritmos que obtienen muy buenos resultados.

La simplicidad de estas técnicas es algo también a tener en cuenta ya que sería fácil

introducirlas en cualquier algoritmo y podría mejorar bastante los resultados. Más adelante se hablará de posibles extensiones del trabajo que por falta de tiempo no se han llegado a realizar pero que podrían haber sido de gran carácter informativo.

Si hablamos de tiempos de ejecución, he podido comprobar que las diferencias de tiempos con y sin repulsión son mínimas. Con este dato tenemos un incentivo más para utilizar dicha técnica en otros algoritmos ya que con las mismas condiciones mejoraremos las soluciones en la medida de lo posible.

Otra conclusión que podemos sacar del estudio es que ajustar los parámetros de la atracción y la repulsión también es muy importante. En la gran mayoría de los casos vistos las soluciones mejoran cuando se realiza un buen ajuste del parámetro a observar. Esto es importante pero también hay que decir que aún ajustando los parámetros no hemos conseguido cambiar el orden de los algoritmos con respecto a los que daba el autor.

Si hablamos de los algoritmos podemos decir que el que mejor se comporta en todos los aspectos es el Bacterial Foraging Algorithm. Como podemos observar en la última comparativa realizada, ha obtenido el mejor puesto en todas las variantes, tanto en dimensión 10 como en 30 y con los parámetros del autor y los ajustados previamente. Además, otro algoritmo que vemos que produce muy buenos resultados en dimensión 10 es el Grasshopper Optimization Algorithm, pero en dimensión 30 no ha conseguido obtener ninguno de los mejores resultados.

Un aspecto que se ha visto en los algoritmos y que merece ser nombrado es la influencia del tamaño de las poblaciones. En los dos estudios realizados, dimensión 10 y 30, el tamaño de las poblaciones ha sido el mismo, sin embargo los resultados no han empeorado demasiado. Hablando de tiempos de ejecución, aumentar el tamaño de la población ralentiza mucho los algoritmos por lo que si se nota tan poca diferencia en los comportamientos no merece la pena asumir el aumento de tiempo de ejecución ya que son algoritmos que de por si tienen tiempos de ejecución muy elevados.

Para terminar, veremos que el proyecto se puede ampliar en varios aspectos que enumeraremos a continuación:

- **Añadir repulsión a un algoritmo que no la tenga:** Una buena forma de comprobar que efectivamente la repulsión mejora los resultados de un algoritmo es añadirle dicha componente a un algoritmo que en su origen no la tiene. Así podríamos afirmar de forma más exacta que es una técnica fácil de añadir y que no aumenta los tiempos de ejecución.
- **Adaptar los parámetros dinámicamente:** Como hemos visto, hay veces en las que un valor del parámetro de atracción o repulsión nos da muy buenos resultados en un tipo de funciones pero no en otros. Esto es debido a la diferencia que tienen los entornos de cada uno de los grupos de funciones. Es por ello que sería útil ajustar el parámetro con respecto al entorno que se esté estudiando.
- **Tomar más valores para el ajuste de parámetros:** En éste proyecto debido al tiempo solo se han podido tomar 4 valores para cada parámetro (0.5,0.75,1.25 y 1.5). Sería de interés poder seguir ajustando el parámetro con valores próximos a los que nos dan en este momento las mejores variantes del algoritmo.

Referencias

- [1] A.E Eiben y G. Rudolph. “*Theory of evolutionary algorithms: a bird’s eye view*”, Theoretical Computer Science, VOL 229, págs 3-9, año 1999. DOI:10.1016/s0304-3975(99)00089-4
- [2] Iztok Fister Jr., Xin-She Yang, Iztok Fister , Janez Brest y Dušan Fister. “*A Brief Review of Nature-Inspired Algorithms for Optimization*”, E Lektrotehni Ški Vestnik VOL 80(3), págs 116–122, año 2013.
- [3] <http://www.swarmintelligence.org/tutorials.php>
- [4] Ahmed Y. Saber y Ganesh K. Venayagamoorthy, “*Economic Load Dispatch using Bacterial Foraging Technique with Particle Swarm Optimization Biased Evolution*”, IEEE Swarm Intelligence Symposium, St. Louis MO USA, Septiembre 21-23 año 2008, 8 págs. DOI:10.1109/sis.2008.4668291
- [5] Tridib Kumar Das, Ganesh Kumar Venayagamoorthy y Usman O. Aliyu. “*Bio-Inspired Algorithms for the Design of Multiple Optimal Power System Stabilizers: SPPSO and BFA*”, IEEE Transaction On Industry Applications VOL. 44, 13 págs, año 2008. DOI: 10.1109/ias.2006.256593
- [6] W. J. Tang, M. S. Li, Q. H. Wu, y J. R. Saunders, “*Bacterial Foraging Algorithm for Optimal Power Flow in Dynamic Environments*”, IEEE Transactions On Circuits And Systems—I: Regular Papers, VOL. 55, NO. 8, 10 págs, Septiembre de 2008. DOI:10.1109/tcsi.2008.918131
- [7] Jianfa Wu, Honglun Wang, Na Li, Peng Yao y Yu Huang, “*Distributed trajectory optimization for multiple solar-powered UAVs target tracking in urban environment by Adaptive Grasshopper Optimisation Algorithm*”, IEEE Transactions On Circuits And Systems-I: Regular Papers, VOL. 55, NO. 8, 10 págs Septiembre 2008. DOI: 10.1016/j.ast.2017.08.037
- [8] Baran Hekimoğlu y Serdar Ekinci, “*Grasshopper Optimization Algorithm for Automatic Voltage Regulator System*”, 5 págs, 5th International Conference on Electrical and Electronics Engineering, año 2008. DOI:10.1109/iceee2.2018.8391320.

- [9] T. Apostolopoulos y A. Vlachos, “*Application of the Firefly Algorithm for Solving the Economic Emissions Load Dispatch Problem*”, Hindawi Publishing Corporation International Journal of Combinatorics VOL. 2011, 23 págs, año 2010. DOI:10.1155/2011/523806
- [10] R.H.Bhesdadiya, Narottam Jangir, Mahesh H. Pandya, Pradeep Jangir, Indrajit N. Trivedi y Arvind Kumar, “*Price Penalty factors Based Approach for Combined Economic Emission Dispatch Problem Solution using Dragonfly Algorithm*”, International Conference on Energy Efficient Technologies for Sustainability, 6 págs, 7-8 Abril año 2016. DOI:10.1109/iceets.2016.7583794.
- [11] Seyedali Mirjalili. “*Dragonfly Algorithm: a new meta-heuristic optimization technique for solving single-objective, discrete and multiobjective problems.*” , The Natural Computing Applications Forum, 21 págs, año 2015. DOI:10.1007/s00521-015-1920-1.
- [12] Matthew Luckie.“ *CScamper: a scalable, extensible packet prober for active measurement of the internet*”, Internet Measurement Conference, 7 págs, año 2010.DOI:10.1145/1879141.1879171.
- [13] Y.Liu y K.M.Passino. “*Biomictry of Social Foraging Bacteria for Distribuded Optimization: Models, Principles and Emergent Behaviors*”,Journal Of Optimization Theory And Applications: VOL. 115, No. 3, págs 603–628, Diciembre año 2002. DOI:10.1023/a:1021207331209.
- [14] Ying Chu, Hua Mi, Huilian Liao, Zhen Ji y Q.H.Wu. “*A Fast Bacterial Swarming Algorithm for High-dimensional Function Optimization* ”, IEEE Congress on Evolutionary Computation,6 págs, año 2008, DOI:10.1109/cec.2008.4631222.
- [15] Raghavendra V. Kulkarni “*Bio-inspired Algorithms for Autonomous Deployment and Localization of Sensor Nodes*”,IEEE Transactions On Systems, Man, And Cybernetics—Part C: Applications And Reviews, VOL. 40, NO. 6, 13 págs, Noviembre año 2010, DOI:10.1109/tsmcc.2010.2049649.
- [16] Shahrzad Daremi, Seyedali Mirjalili y Andrew Lewis. “*Grasshopper Optimisation Algorithm: Theory and Application.*”, Advances in Engineering Software, VOL 105,pags 30–4, año 2017, DOI:10.1016/j.advengsoft.2017.01.004.

- [17] Ali Safari Mamaghani, Meysam Hajizadeh “*Software Modularization Using the Modified FireflyAlgorithm*”, 8th Malaysian Software Engineering Conference (MySEC),4 págs, 23-24 Sept. 2014, DOI:10.1109/mysec.2014.6986037.
- [18] J.J Liang, B.Y.Qu y P.N.Suganthan. “*Problem definitions and evaluation criteria for the CEC 2014 Special Session and Competition on Single Objective Real-Parameter numerical optimization*”, 32 págs, año 2013