TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

# Comparación de metaheurísticas avanzadas aplicadas sobre un problema médico real de optimización de miles de variables


**Autor**
Arthur Rodríguez Nesterenko

**Director**
Daniel Molina Cabrera




Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación


—

Granada, septiembre de 2018


# Comparación de metaheurísticas avanzadas aplicadas sobre un problema médico real de optimización de miles de variables

**Autor**
Arthur Rodríguez Nesterenko

**Director**
Daniel Molina Cabrera


Departamento de Ciencias de la Computación e Inteligencia Artificial

# Comparación de metaheurísticas avanzadas aplicadas sobre un problema médico real de optimización de miles de variables

Arthur Rodríguez Nesterenko

**Palabras clave**: big optimization, electroencefalograma, metaheurísticas, algoritmos, big data, benchmark, LSGO, MAGA, MOS, SHADEILS, MLSHADE-SPA, DG2

## Resumen

Desde sus albores, la computación evolutiva ha contribuido significativamente a la resolución de problemas del mundo real representados a través de funciones que requieren optimización. Sin embargo, el rendimiento de estas técnicas disminuye considerablemente cuando se enfrentan a un problema con miles de variables. En este proyecto se estudia la efectividad de las técnicas metaheurísticas actuales más avanzadas en el área de optimización de alta dimensionalidad frente a la formulación teórica de un problema médico real, como es la optimización de los datos de un electroencefalograma. Se desarrolla un estudio experimental exhaustivo que conforma la base del objetivo final, que consiste en evaluar la utilidad real de las propuestas más modernas del panorama actual en áreas de interés como la medicina, donde la decodificación de un electroencefalograma de forma eficaz y eficiente permitiría mejorar las interfaces cerebro-ordenador actuales e incluso potenciar su uso en entornos del mundo real donde se requiera la toma de decisiones críticas y respuestas en tiempo real.

# Advanced metaheuristics comparison applied to a real medical optimization problem with thousands of variables

Arthur Rodríguez Nesterenko

**Keywords**: big optimization, electroencephalography, metaheuristics, algorithms, big data, benchmark, LSGO, MAGA, MOS, SHADEILS, MLSHADE-SPA, DG2

**Abstract**

Since early days, evolutionary computation has undoubtedly contributed to solve several real-world problems represented as optimization functions. However, the performance of these techniques decreases significantly when applied over problems containing thousands of variables. In this proposal, the capabilities of the most advanced metaheuristic techniques of large scale global optimization are studied, facing the theoretical formulation of a medical real-world problem such as the optimization of electroencephalography data. A comprehensive experimental study is conducted in order to comply the final goal, which is to evaluate the actual uses of current means over the area of intereset such as medicine, where decoding an electroencephalography, both efficiently and effectively, could enhance the available brain-computer interfaces and utilize their potential in the real world environments that requiere critical decision making and real-time responses.

Yo, **Arthur Rodríguez Nesterenko** , alumno de la titulación Grado en Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación** , con DNI Y1680851W, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Arthur Rodríguez Nesterenko

Granada a 7 de septiembre de 2018.

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

**Informa:**

Que el presente trabajo, titulado ***Comparación de metaheurísticas avanzadas aplicadas sobre un problema médico real de optimización de miles de variables***, ha sido realizado bajo su supervisión por **Arthur Rodríguez Nesterenko** , y autoriza la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expide y firma el presente informe en Granada a 7 de septiembre de 2018.

**El director:**

**Daniel Molina Cabrera**

# Agradecimientos

A mi padre, por ser mi padre particular de la informática. A mi madre, por su apoyo incondicional desde siempre. A mi hermano, por alentarme a superarme día a día. A mis más cercanos amigos y compañeros de carrera y profesión, por proveer constante ánimo durante el camino. A mi tutor, por su atención y ayuda a lo largo de todo el desarrollo de este trabajo.

# Índice general

# Índice de figuras

# Índice de tablas

# Capítulo 1

# Introducción

Desde sus inicios, la computación evolutiva ha sido capaz de proponer soluciones efectivas a una gran cantidad de problemas de optimización a través de algoritmos bio-inspirados, aquellos que se basan en comportamientos o procesos puramente naturales. Algoritmos Evolutivos (EA, Evolutionary Algorithms) como los algoritmos genéticos (GA)[4], de Nubes de Partículas (PSO, Particle Swarm Optimization)[5], de Evolución Diferencial (DE, Differential Evolution)[6] y de Colonias de Hormigas (ACO, Ant Colony Optimization)[7], responden de manera satisfactoria al ser aplicados a problemas de escalas pequeñas-medianas.

Sin embargo, cuando el tamaño del problema se situa en el rango de los cientos o miles de variables, el espacio de soluciones aumenta de manera exponencial conforme crece el número de éstas, lo que repercute directamente en un aumento significativo de la complejidad del problema. Estas condiciones hacen que para una técnica, anteriormente considerada como efectiva, sea mucho más difícil el encontrar una solución óptima para el problema en cuestión.

Con miras a solventar estas dificultades, surge una nueva vertiente de propuestas dedicadas a la resolución de problemas de optimización de escalas igual o superiores a las mil variables, lo que se conoce como *Large Scale Global Optimization* o **LSGO**[8]. Estas nuevas técnicas son, principalmente, producto de los más importantes congresos en computación evolutiva del mundo, como el Congress of Evolutionary Computation (IEEE CEC) o el World Congress on Computational Intelligence (WCCI).

## 1.1. Motivación

Las propuestas disponibles actualmente muestran resultados satisfactorios al ser evaluadas en función de benchmarks preestablecidos y bien defi-

nidos. No obstante, su aplicación real normalmente no trasciende más allá de la propia competición para la que fueron diseñados. Este hecho supone la principal motivación de este trabajo, que consiste en **verificar** si los algoritmos que ofrecen resultados competitivos en los benchmarks de LSGO actuales se comportan de igual manera en términos de efectividad y eficiencia cuando son aplicados frente a la formulación de un problema médico real como lo es, en este caso, **la optimización de los datos de un electroencefalograma**, EEG por sus siglas en inglés.

La optimización de un electroencefalograma es un problema que no guarda relación con ninguno de los benchmarks conocidos hasta la fecha, por ser este un problema cuya **complejidad** es **superior tanto a nivel conceptual como en tiempo y en espacio**. No obstante, existen resultados contrastados de distintos algoritmos y técnicas frente a un **benchmark parcialmente artificial** cuya descripción matemática permite representar a grandes rasgos el problema de la **decodificación real de un EEG**. A partir de estos resultados se pueden extraer conclusiones preliminares que sean determinantes en el futuro del procesamiento de las señales de un EEG en un entorno médico real.

Este benchmark, propuesto en 2015[9] será la principal herramienta utilizada para evaluar la potencia de las propuestas más modernas y potentes del panorama LSGO. Los resultados que se obtengan podrían sentar una base sólida en cuanto a soluciones a corto-medio plazo que reportasen importantes avances en cuanto al diseño e implementación de **interfaces cerebro-ordenador** (BCI, *brain-computer interfaces*)[10] más potentes, fiables y eficientes.

Este tipo de dispositivos podrían servir de ayuda en actividades y tareas del día a día donde la decodificación de un EEG cualitativo es crucial a la hora de reconocer estados cognitivos de orden superior, tales como emociones, memoria o planificación, aspectos que influyen directamente en la toma de decisiones críticas[2] en distintos entornos donde es necesaria una interacción y respuesta en tiempo real para garantizar el correcto desempeño de la actividad.

## 1.2. Objetivos

El objetivo principal de este Trabajo de Fin de Grado consiste en **realizar un estudio comparativo** de los algoritmos dedicados a resolver problemas de optimización con miles de variables, problemas del tipo **LSGO**, a través del benchmark mencionado anteriormente, de forma que se pueda plantear un posible curso de acción o propuesta, ya sea a corto-medio o largo plazo, que sea de utilidad para resolver el problema médico real de

la optimización de un electroencefalograma (EEG), con el objetivo final de mejorar el rendimiento de todas aquellas actividades derivadas de su uso.

Los siguientes capítulos contendrán el estudio completo en cuestión: desde la definición y representación del problema hasta la selección de posibles técnicas y procedimientos candidatos a estudio, así como del proceso de experimentación, la interpretación y valoración de los resultados obtenidos. Los objetivos principales se encuentran recogidos en los siguientes enunciados:

- **Realizar la descripción y representación del problema del EEG en su totalidad.**
- **Analizar en profundidad los algoritmos y técnicas más prometedoras para la resolución del problema.**
- **Diseñar e implementar un proceso de experimentación riguroso y completo para cumplir los requisitos del estudio.**
- **Evaluar los resultados obtenidos y enunciar la propuesta de solución más adecuada en función de éstos.**

Una vez enunciados los objetivos, obtener una perspectiva clara y concisa del proceso que se llevará a cabo para desarrollar este trabajo es crucial para comprender la magnitud del mismo. En el siguiente capítulo se detallan las principales tareas a desarrollar en este trabajo y la planificación que se seguirá para completar las mismas.

# Capítulo 2

# Tareas y Planificación del trabajo

En este capítulo se propone el curso de acción para el desarrollo del trabajo. Este incluye las principales **tareas a desarrollar** por el alumno en cuanto a los **objetivos** y la **propuesta**, así como la **planificación** de las mismas, entendida como la previsión en relación a recursos necesarios, sus costes derivados, optimización del tiempo de proceso, necesidades de potencia y espacio a nivel computacional, la estructuración de la carga de trabajo y por último la **metodología** idónea para elaborar este análisis

## 2.1. Tareas del proceso de investigación

La naturaleza investigativa de este trabajo implica el cumplimiento de una serie de tareas de cara a establecer la base sobre la que se sustenta el mismo; éstas se encuentran representadas en los siguientes enunciados y **componen las tareas más fundamentales** a llevar a cabo para completar satisfactoriamente el estudio. Como norma, se definirá una jornada de trabajo de entre 6 y 8 horas diarias para definir el tiempo total necesario para completar cada tarea.

1. **Estudiar de forma exhaustiva el problema del EEG**: **comprender** la naturaleza más primitiva de la proposición en todos los niveles, estudiar su **utilidad a escala real** y los **beneficios** que trae consigo su resolución para la sociedad. Esta tarea se satisface a lo largo del capítulo 3 y se estima alrededor de 2 semanas en completar esta fase de forma adecuada, allanando el camino para posteriores tareas, por lo que la estimación de tiempo total no supera las 120 horas.

2. **Revisar la literatura actual**: una vez conocido el problema en profundidad, el siguiente paso consiste en hacer una **revisión de la literatura del panorama LSGO**. Conocer en mayor medida las propuestas anteriormente publicadas en cuanto a la resolución del problema descrito, con el objetivo de identificar aquellas técnicas más prometedoras de cara a obtener resultados satisfactorios aplicando el benchmark EEG. La revisión, motivada por sugerencias del tutor y su experiencia en este campo, **no será mayor de 60h** y se desarrolla en el capítulo 4

3. **Seleccionar los principales algoritmos y técnicas a incluir en el trabajo**: teniendo en cuenta la revisión efectuada en la tarea anterior, junto con recomendaciones del tutor, se **eligen las tecnologías más prometedoras** de la literatura que formarán parte del estudio que se desarrolla a lo largo de este trabajo. El capítulo 5 recoge esta tarea junto con la siguiente, donde la actual tarea no ocupará mas de 1 o 2 días de trabajo, menos de 16h.

4. **Analizar las propuestas elegidas con el mayor nivel de detalle posible**: identificar los **puntos fuertes de cada implementación** y de las técnicas empleadas, con el objetivo de construir **juicios preliminares** en relación a la satisfacción de la propuesta, que serán finalmente validados en el proceso de experimentación. Se recoge este análisis en el capítulo 5, donde se espera una gran carga de trabajo que, al alza, ocuparía entre un mes y medio a dos, cerca de 480h.

5. **Obtener las implementaciones de las técnicas seleccionadas**: debido a las restricciones de eficacia y eficiencia que impone el problema, se requieren implementaciones sin fisuras y lo más testadas y completas posibles, por lo que se optará por **obtener una copia de las implementaciones** de los algoritmos a través de sus autores y/o sus respectivos repositorios, así como a través del tutor de este trabajo. El proceso de obtención de estas técnicas está implícito en el capítulo 6, donde se responderá de forma conjunta a ésta y la siguiente tarea. La estimación es que no ocupe más de 15h en total, aunque está claramente supeditado a la respuesta de los autores o la disponibilidad de las implementaciones en los repositorios.

6. **Implementar la adaptación de las técnicas al problema del EEG**: obtenidas las implementaciones particulares de cada técnica y teniendo en cuenta las herramientas utilizadas por los autores, es necesaria una **adaptación de la función objetivo** para que pueda ser procesada por los algoritmos de forma correcta. Una **configuración personalizada** será necesaria para acoplar esta función con las exigencias a nivel de código fuente de cada técnica. Se documenta esta

tarea en el capítulo 6, donde se estima cerca de una semana (60h) para cada técnica elegida y la validación de resultados preliminares, previo al proceso experimental completo. Total: 240-300h.

7. **Diseñar un proceso de experimentación completo**: para dotar de validez a las propuestas y obtener resultados representativos, se hace indispensable **planificar una fase de experimentación** robusta y **acotar la salida** que se quiere obtener para posteriormente realizar un análisis sólido y objetivo de los resultados. Se desarrolla a lo largo del capítulo 7 y, tomando como referencia la literatura de los benchmarks y como se conducen normalmente los experimentos, 50h deberían ser suficientes para completar esta tarea.

8. **Realizar los experimentos propuestos**: llevar a cabo los experimentos es crucial para obtener los resultados que motivarán las conclusiones expresadas posteriormente. Dado que depende directamente del tiempo de respuesta de cada algoritmo, la experimentación efectiva (hasta obtener los resultados) no debería ocupar más de una semana, cerca de 60h. Esta tarea no requiere documentación expresa, dado que es en la siguiente tarea donde se mostrarán los resultados obtenidos.

9. **Crear las tablas y gráficas de los resultados obtenidos**: un proceso indispensable que permitirá tener los **datos ordenados** y que sea **fácil identificar** aquellas técnicas con mejores rendimientos. Se estima poco mas de 30h para tener completada esta fase, siendo el preámbulo de la posterior tarea que, aunque se desarrollen ambas en el capítulo 8, se añade como una tarea por separado

10. **Analizar los resultados obtenidos de forma objetiva**: de acuerdo con las soluciones alcanzadas, utilizar todas las herramientas de representación de datos que sean necesarias para **decretar un veredicto con el mayor grado de precisión posible**, así como valerse de la información resumida de la tarea anterior. En 24h se podría tener un veredicto preliminar que precisará de consultas con el tutor. El capítulo 8 recoge la realización de esta tarea.

11. **Destacar las conclusiones y extensiones futuras**: enunciar las conclusiones de este trabajo, en forma de resumen de todo el proceso realizado, así como contemplar posibles extensiones en materia de investigación dentro de la actual disciplina. El capítulo 9 recoge estos enunciados finales y no deberían superarse las 24h de trabajo.

12. **Documentar el trabajo realizado**: esta es la única tarea que será realizada, de forma incremental, a lo largo de todo el desarrollo de este trabajo, por lo que se estima un tiempo de finalización total similar

al que ocupa la suma del tiempo de cada tarea, dado que durante el desarrollo de cada una se documentará el proceso de forma paralela.

Una vez especificadas las tareas a desarrollar a lo largo del trabajo, el siguiente paso consiste en planificar el proceso de estudio de forma que se contemplen, aunque sea de forma preliminar, los recursos y necesidades indispensables a las que habrá que hacer frente a lo largo del desarrollo, para tener una primera perspectiva de lo que implica llevar a cabo la investigación actual.

## 2.2. Planificación del trabajo

Los aspectos principales a considerar en este apartado están relacionados con los **recursos** disponibles y aquellos que son necesarios, así como la **distribución de la carga de trabajo** en términos, principalmente, de tiempo y potencia de cómputo. Considerar ambos factores implica realizar una **estimación de tiempos, costes y requerimientos** para la realización del trabajo.

### 2.2.1. Recursos disponibles y necesarios

El alumno hará uso de su propio ordenador portátil MacBook Pro del año 2012 con procesador Intel Core i7 a 2,9 GHz, con 16 GB de memoria RAM DDR3 y una tarjeta gráfica Intel HD Graphics de 1,5 GB. Este equipo será más que suficiente para afrontar las tareas derivadas de la **investigación y documentación**, para las cuales también será necesario tener acceso a una **red de internet** de banda ancha para consultar toda la bibliografía y documentación requerida, red de la que el alumno dispone en su domicilio particular.

Acceder a **documentación, artículos científicos y libros**, ya sea a nivel de Internet o de bibliotecas virtuales, será posible gracias a los convenios que tiene la Universidad de Granada con bases de datos tales como Scopus[11] o IEEE, que proporcionan bibliografía fiable y contrastada, imprescindible en este trabajo, y se accederá a ellas a través de una conexión VPN a la red de la Universidad. El alumno se valdrá también de una libreta donde documentará de forma escrita aquellas fases o hitos que considere necesarios.

Para la **fase de implementación**, donde se validará la adaptación de los algoritmos, bastará con el ordenador portátil disponible y las principales herramientas software como distintas IDEs, procesadores de texto y demás tecnologías, así como los recursos energéticos que proporciona el lugar donde

reside el alumno, medios que serán igualmente suficientes para los **procesos finales de evaluación y conclusión**.

Sin embargo, dada la **dimensionalidad del problema y la complejidad de los experimentos**, será necesaria una mayor potencia de cómputo para llevar a cabo las pruebas pertinentes. Para ello, el alumno se servirá del **Clúster Hercules del CITIC de la UGR** [12], que *"posee 46 nodos, cada uno de ellos equipado con un procesador Intel Core i7 930 a 2.8 GHz, 24 GB de RAM y HDD SATA2 de 1TB"*. Funciona a través de un sistema de colas SLURM[13] que facilita la creación y ejecución de trabajos en el clúster. El acceso a este clúster se realizará a través de un usuario y contraseña proporcionados por el equipo de SysOp de la UGR.

El alumno por tanto asumirá únicamente los costes derivados de la investigación cuando utilice los **recursos propios** y las herramientas de las que dispone, por lo que no se tienen en consideración aquellos que asume la propia Universidad, como cuando se utiliza el Cluster Hércules.

### 2.2.2. Estimaciones y presupuesto

Teniendo en cuenta el tiempo estimado para cada tarea que se detalla en la sección 2.1 se estima un total de **6 meses** en llevar a cabo el trabajo, donde serán las tareas de documentación las que en un principio requerirán mayor tiempo, sobre todo las relacionadas con las **tareas 1, 2, 3 y4**, dado que son las que sustentan toda la base del estudio y requieren ser precisadas de forma exhaustiva para descartar la existencia de cualquier tipo de incongruencia o error.

El proceso de **adaptación de las técnicas** al problema del EGG (tarea 6) está subordinado a las particularidades de cada implementación y puede complicarse por situaciones que actualmente escapan al conocimiento del alumno, pero es evidente que resolver las tareas 3 y 4 despeja el camino a seguir y una vez realizada la adaptación sobre uno de los candidatos, se asume que el proceso será similar para los algoritmos restantes.

El **diseño e implementación experimental** (tareas 7 y 8) quizá sean la tareas con mayor incertidumbre en cuanto a estimaciones, dado que está ligada de forma directa a los **procesos de validación previos** donde se evaluarán principalmente los tiempos de ejecución de los algoritmos. Cuando finalmente sean sometidos a test, estos tiempos supondrán la gran parte del tiempo total que se empleará en esta fase. Al contar con recursos como el Clúster Hércules, los tiempos de respuesta de los algoritmos frente a los experimentos no deberían suponer un gran problema, independientemente de lo que se tarde en ellos, por lo que las estimaciones anteriormente mencionadas deben ser suficientes. El resto de tareas se desarrollarán de acuerdo con las estimaciones temporales mencionadas.

En resumen, la tabla siguiente muestra como se distribuye el presupuesto en relación a las tareas mencionadas en la sección anterior, tomando un precio por hora de trabajo de **ocho euros**; en base al número de horas y el precio por hora, se puede estimar un presupuesto aproximado de lo que conlleva realizar este trabajo. Para las tareas que ocupen tiempos más variables, se tomará el mayor número de horas para evitar subestimaciones.

| **Tareas** | **Tiempo (h.)** | **Presupuesto (euros)** |
|---|---|---|
| Estudio problema | 120 | 960 |
| Rev. Literatura | 60 | 480 |
| Selección técnicas | 16 | 128 |
| Análisis propuestas | 480 | 3840 |
| Obtener impl. | 15 | 120 |
| Adaptar técnicas | 300 | 2400 |
| Diseño experimental | 50 | 400 |
| Realizar exps. | 60 | 480 |
| Tablas y gráficas | 30 | 240 |
| Analisis resultados | 24 | 192 |
| Conclusiones | 24 | 192 |
| **Total** | 1179 | **9432** |

Tabla 2.1: Presupuesto estimado del trabajo

### 2.2.3. Metodología

La metodología a seguir para llevar a cabo este trabajo será la de un desarrollo iterativo e incremental donde cada ciclo estará formado por una fase similar: la especificación de requisitos se realizará **a nivel de cada una de las tareas** que se esten desarrollando, el **diseño** se limitará a la **fase experimental**, la **implementación** casa con la adaptación de los algoritmos, la puesta en marcha de los experimentos y con el proceso de creación, modificación y adaptación de **código fuente**. La **documentación estará íntimamente ligada a cada uno de los procesos**, por lo que también se desarrolla de forma iterativa e incremental.

A lo largo de todo el proceso de desarrollo de la investigación, la **comunicación directa con el tutor** del trabajo será vital si se quiere consultar y/o validar cada una de las iteraciones realizadas y llevar un seguimiento del desarrollo del trabajo. Dada las situaciones particulares tanto del alumno como del tutor, se primará el uso del **correo electrónico** para el seguimiento del trabajo, siendo indispensables al menos **dos o tres reuniones más**,

sin contar la primera, de cara a la culminación del proyecto.

Sin más aspectos que añadir en esta fase, en el capítulo siguiente se procederá al estudio en profundidad del problema del EEG, tarea que junto con la revisión de la literatura, supone el punto de partida del resto del trabajo, por lo que se precisa conocer con la mayor exactitud posible todos los componentes de la actual formulación.

# Capítulo 3

# Introducción: Optimización de los datos de un EEG

## 3.1. Panorama actual

La optimización de los datos de un electroencefalograma, a partir de ahora EEG por sus siglas en inglés, está dentro de la categoría de problemas de Big Data, problemas donde no sólo el tamaño del mismo supone de por si una dificultad, sino que también influyen aspectos como el ruido en los datos, la no existencia de patrones fácilmente reconocibles o restricciones de tiempo inherentes al problema.

En el año 2015 se presenta este problema como candidato a conformar la base del **Optimization of Big Data Competition - CEC 2015**[2], donde se realiza su estudio a través de dos algoritmos multiobjetivo muy potentes, cuyos resultados, a pesar de catalogarse como satisfactorios, demostraron que era necesario disponer de mejores propuestas y métodos que aportasen un rendimiento superior en términos de tiempo y calidad de las soluciones.

Para entender mejor los orígenes del problema hace falta hacer un breve repaso por la utilidad real que tienen los EEGs en distintas áreas de conocimiento, principalmente para comprender porqué es tan importante disponer de técnicas de resolución eficientes. En ámbitos médicos como la neurociencia se utilizan dispositivos denominados BCI (Brain-computer Interfaces)[10] que se encargan de **capturar la actividad cerebral** a través de electrodos, con el fin de analizar estos datos, procesarlos y traducirlos en acciones o estados cognitivos que puedan utilizarse para actuar en consecuencia ante una determinada situación, como puede ser para diagnosticar distintas enfermedades del sistema nervioso central.

Los BCIs hacen uso principalmente de los **electroencefalogramas** (EEG),

que son exploraciones neurofisiológicas que registran la actividad bioeléctrica cerebral, concretamente de las neuronas, a través de electrodos (componentes de los BCIs), con el objetivo principal de detectar y diagnosticar enfermedades o trastornos del sistema nervioso central[14], tales como epilepsia, daños cerebrales de distintos tipos, trastornos psiquiátricos, encefalopatías y demás afecciones[15], así como para potenciar las capacidades cognitivas frente a determinadas situaciones[16].

Ciertos campos de estudio y aplicaciones en concreto utilizan lo que se conoce como **electroencefalografía cuantitativa**, QEEG [17], que por medio de una malla de electrodos registra de **forma simultánea** los impulsos eléctricos de **múltiples partes del cerebro**. Decodificar de forma efectiva y eficiente estos QEEG en **estados cognitivos de orden superior**[2], como pueden ser emociones, recuerdos o estados cerebrales, promovería la creación de BCIs más avanzados que sustenten el uso de estos sistemas tanto en el tratamiento de pacientes con distintos trastornos del sistema nervioso como para las actividades del día a día, sobre todo de aquellas que conlleven la toma de decisiones críticas en tiempo real, como por ejemplo, el control del tráfico aéreo [18].

Sin embargo, el correcto funcionamiento de los BCIs se ve normalmente truncado por dos principales aspectos: la cantidad de **información cerebral *real*** que es captada y la **distorsión que producen las señales eléctricas no cerebrales**, lo que se denominan **artifacts**, y que quedan plasmadas en el EEG cuantitativo, empañando los resultados obtenidos y elevando la complejidad del proceso de decodificación. Es aquí donde entran en juego técnicas como el **Análisis de Componentes Independientes, ICA**, y otras técnicas que se encargan de separar la señales obtenidas de distintas fuentes.

Finalmente, tras un proceso de inspección visual llevado a cabo actualmente por un ser humano, se eliminan los artifacts de las componentes resultantes y se recompone el EEG original. Naturalmente, esta tarea se vuelve **totalmente inviable** de cara a los procesos de obtención de QEEG actuales, donde se requiere inspeccionar grandes cantidades de datos, lo que consume mucho tiempo y reduce la eficiencia del proceso. Es por esta principal razón por la que se requieren de técnicas eficaces que permitan **automatizar el proceso** y **reducir la intervención humana**, en aras de una optimización mucho mas eficiente. La figura 3.1 muestra un ejemplo de EEG, en una propuesta que contempla la eliminación de artifacts oculares a través del procedimiento ICA.

La propuesta recogida en [2] provee una forma de representar el problema de la decodificación de un QEEG, de forma que cualquier técnica, método o algoritmo preparado para el procesamiento de grandes cantidades de datos y la optimización global de miles de variables, pueda ser sometido

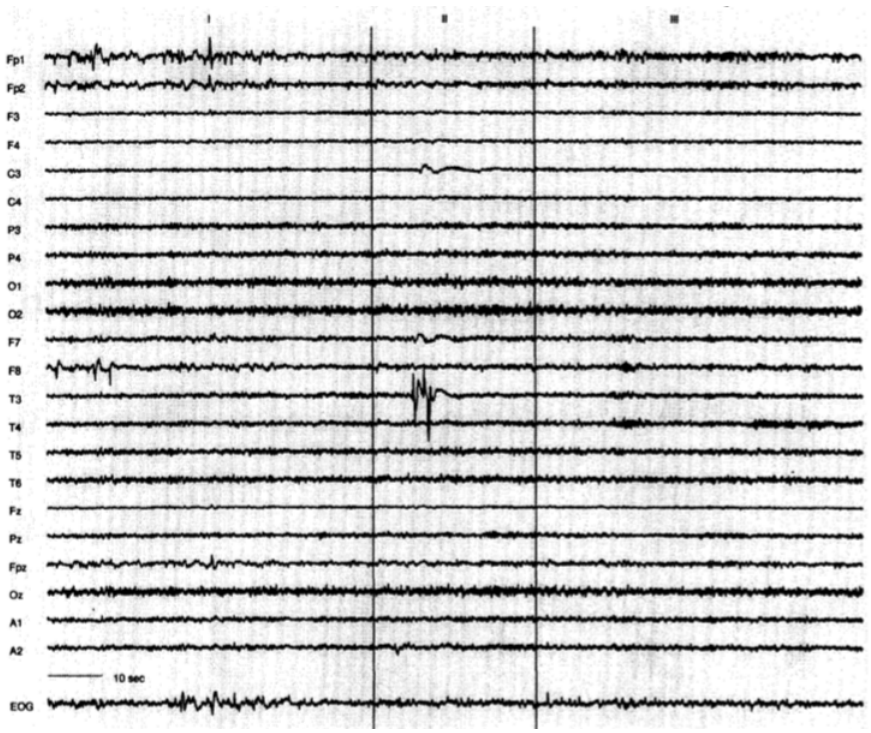


Figura 3.1: EEG: Estudio de eliminación de artifacts oculares a través de ICA. Fuente [1]

a pruebas exhaustivas frente a este problema real, y finalmente sugerir, si cabe, un posible candidato que sustituya al actual método de decodificación, descrito en el párrafo anterior, con el último fin de dar un salto importante en el diseño de sistemas BCIs para la mejora de la calidad de vida de todas aquellas personas que lo requieran.

Como el estudio se basa completamente en la clara definición del problema en cuestión, los siguientes párrafos contendrán toda la información referente a éste, desde la representación elegida, pasando por las bases de datos disponibles y llegando hasta la definición de la función objetivo que marcará el camino a seguir en este estudio. Como aclaración, se habla de QEEG como un tipo de EEG utilizado en algunos campos, aunque a efectos prácticos para entender y describir el problema, se les considera equivalentes.

## 3.2. Representación del problema: EEG

Para definir el problema de la optimización de un EEG se ha empleado una representación que divide al problema en 3 subproblemas con la misma naturaleza que el original, donde únicamente cambian la dimensión del mis-

mo y la existencia de ruido en los datos. De esta forma, se ha generado una base de datos sintéticos (generados de forma artificial)[19] que se reúnen en tres Datasets **A, B y C** tal y como se muestra en la siguiente tabla.

| Dataset | Nº Fuentes | Nº Artifacts | Sin ruido | Con ruido |
|---|---|---|---|---|
| **A** | 4 | 2 | D4 | D4N |
| **B** | 12 | 6 | D12 | D12N |
| **C** | 19 | 6 | D19 | D19N |

Tabla 3.1: Grupos del problema del EEG.

Cada dataset propone una dificultad distinta en términos del número de fuentes de señales reales y de ruido con una **varianza de 0.1**. Las fuentes de señales efectivas son la suma del **número de fuentes de datos reales** y el **número de artifacts**, donde estos últimos operan en altas frecuencias y amplitudes.

Las señales de los artifacts **simulan impulsos electromagnéticos** que se generan de forma involuntaria con movimientos de cualquier parte de nuestro cuerpo durante el tiempo en el que se recogen los datos en el EEG, y al ser señales eléctricas, son captadas por éste mezclándose con las señales reales de nuestro cerebro y empañando el EGG. Cada señal se muestrea a una frecuencia de **256Hz,** y los artifacts son activados en intervalos que oscilan entre los últimos **250ms-500ms** de cada segundo (ver [19]).

Para el **Dataset A**, las 6 señales totales se mezclan en **4 señales de datos**, $\vec{x_1}, \vec{x_2}, \vec{x_3}$ y $\vec{x_4}$ compuestas según las siguientes ecuaciones, donde las señales 5 y 6 son artifacts.

$$\begin{aligned} \vec{x_1} &= \vec{s_1} + 0.9\vec{s_5} \\ \vec{x_2} &= \vec{s_2} + 0.9\vec{s_6} \\ \vec{x_3} &= \vec{s_3} + \vec{s_5} \\ \vec{x_4} &= \vec{s_4} + \vec{s_6} \end{aligned} \tag{3.1}$$

Para los **Datasets B y C**, las 6 señales de artifacts se disparan en posiciones determinadas de la cabeza de una persona, indicadas en la figura 3.2. De forma análoga, cada fuente de datos real de las 12 (Dataset B) o 19 (Dataset C) se mezcla con cada una de las $k$ fuentes de artifacts siguiendo la ecuación:

$$\begin{aligned} \vec{x_i} &= \vec{s_i} + \sum_{k=1}^{N} w_{ik} s_k \\ w_{ik} &= exp(-r^2) \end{aligned} \tag{3.2}$$

donde $w_{ik}$ representa el exponencial de la distancia Euclídea al cuadrado con signo negativo de la señal *i-ésima* y el artifact *k-ésimo*. Esta distancia se calcula teniendo en cuenta la posición donde se colocan los electrodos (fuentes de datos reales) y desde donde se disparan los artifacts en una cabeza de radio 0,5dm, tal y como se puede ver en la figura.

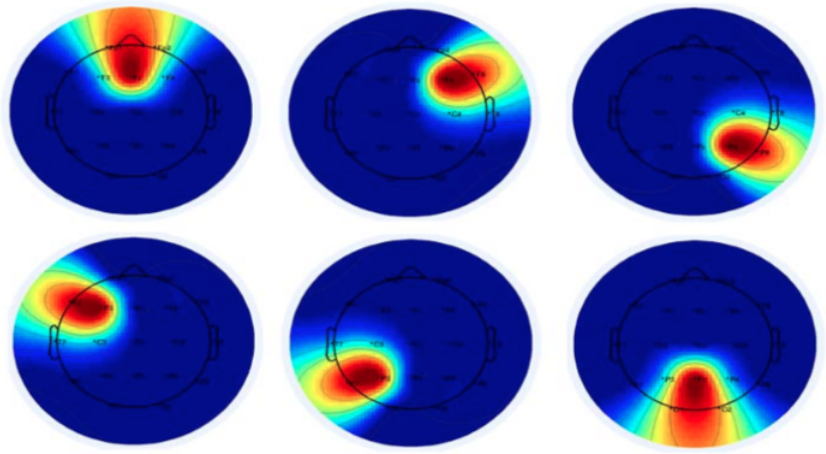

Figura 3.2: Disposición de los 6 artifacts y la contribución de cada uno en todos los electrodos. Fuente: [2]

En resumen, un total de **6 problemas distintos**, **D4, D4N, D12, D12N, D19 y D19N**, donde cada señal se muestrea a 256Hz, da lugar a **tres valores de dimensionalidad del problema** que repercuten directamente en la complejidad en cuanto a espacio de soluciones, donde cada dimensión se calcula como el **número de señales de entrada** ($N$) de cada problema multiplicada por la **frecuencia de muestreo (256Hz)**. Con esta definición se busca facilitar la representación interna de las soluciones como un único conjunto de variables a optimizar; así, el **Dataset A** tiene **1024** variables, el **Dataset B** cuenta con **3072** y el **Dataset C** concretamente dispone de **4864** variables.

Esta elección conceptual conforma la base de este estudio a nivel de representación de los datos, lo que permitirá evaluar el grado de desempeño de un algoritmo frente a la optimización de los datos simulados de un EEG, cuyos resultados son extrapolables a una aplicación médica real si se tiene en cuenta el proceso matemático y estadístico sobre el que se sustenta la decodificación de las señales medidas y que se describe a continuación.

## 3.3. ICA: Independent Component Analysis

Un Análisis de Componentes Independientes, **ICA** en inglés, es un método de procesamiento de señales utilizado principalmente para eliminar artifacts [19]. Permite **separar en fuentes de datos independientes** aquellas fuentes que han sufrido transformaciones lineales y se encuentran mezcladas, a través de una maximización de la independencia estadística de las componentes resultantes[16].

A grandes rasgos un ICA consta de dos pasos, un primer paso donde se elimina toda la correlación de los datos, llamado *data whitening*, y un segundo paso donde se aplica la matriz de rotación inversa a la transformación aplicada, con el fin de obtener los datos originales. La correlación de los datos se debe principalmente a que un electrodo localizado en una parte determinada de la cabeza, no sólo medirá los impulsos eléctricos de esa zona en particular, sino que también se verá afectado por las señales cercanas que estén siendo sensadas por otros electrodos.

Expresado de forma matemática, un ICA busca una **transformación lineal V** de los **datos D** de forma tal que $P = V \cdot D$ y que $Cov(P) = I$, siendo $Cov$ la matriz de covarianza e $I$ la matriz de identidad, lo que indica que las variables no tienen correlación alguna (paso 1, **data whitening**). Tras encontrar $V$, se procede a realizar la rotación de la matriz a través de la **minimización de la Gaussianidad** de la matriz en cuestión.

Por tanto, el problema de la optimización del EEG se puede formular análogamente a la aplicación de un ICA de la siguiente manera:

$$X = A \cdot S + N \tag{3.3}$$

donde $X$ es la matriz combinada de señales (debido a la naturaleza de la medición), A es la matriz de combinación que representa la transformación lineal $V$ del ICA al combinarse las señales entre sí, $N$ es el ruido y $S$ las fuentes de datos originales; así al disponer de $A$ y unas fuentes estimadas $\hat{S}$, hay que encontrar $W$ tal que $\hat{S} = W \cdot X$. Finalmente se reconstruyen las señales limpias a través de las componentes útiles, tal que : $\hat{X} = W^{-1}\hat{S}$

## 3.4. Formulación del problema: enfoque uniobjetivo

La formulación original del problema sigue un enfoque multiobjetivo pero adoptar un **enfoque uniobjetivo** reduce la complejidad del problema sin incurrir en pérdida alguna de generalidad. Por tanto, se utilizará el enfoque multiobjetivo para formular el problema pero experimentalmente se

generalizará a uno uniobjetivo.

Sea $X$ (matriz combinada de señales) una matriz de dimensión $n \times m$ donde $n$ es el número de **series de tiempo interdependientes** y $m$ la **longitud** de cada serie. Sea a su vez $S$ (matriz de fuentes reales) una matriz $n \times m$ con $n$ series de tiempo **independientes** de longitud $m$ y $A$ una **matriz de transformación lineal** $n \times n$. Se sabe que:

$$X = A \times S \tag{3.4}$$

Por tanto el problema consiste en descomponer S en $S_1$ y $S_2$ tal que $S = S_1 + S_2$ y que además $X = A \times S_1 + A \times S_2$. Y sea $C$ la matriz de **coeficientes de correlación de Pearson** entre $X$ y $A \times S_1$ definida como:

$$C = \frac{covar(X, AS_1)}{\sigma(X) \cdot \sigma(A \cdot S_1)} \tag{3.5}$$

donde $covar()$ representa la matriz de covarianzas y $\sigma$ la varianza. El objetivo consiste en **maximizar los elementos diagonales de C** a la vez que se minimizan los elementos de fuera de la diagonal, y conseguir que la **distancia entre $S$ y $S_1$** sea la mínima posible, lo que a su vez maximice la similitud entre estas [2]. Para ello, se debe encontrar una matriz $S_1$ que minimice el valor de las siguientes funciones:

$$Minimize\ f_1 = \frac{1}{N^2 - N} \sum_i \sum_{j \neq i} C_{ij}^2 + \frac{1}{N} \sum_i (1 - C_{ii})^2 \tag{3.6}$$

$$Minimize\ f_2 = \frac{1}{N \times M} \sum_i \sum_j (S_{ij} - S_{1ij})^2 \tag{3.7}$$

donde se busca **eliminar la mayor cantidad de artifacts posibles** a la vez que se **minimiza la pérdida de información real**. A través de estas dos funciones se propone el **enfoque uniobjetivo** donde la función objetivo a optimizar es la siguiente:

$$f = Minimize(f_1 + f_2) \tag{3.8}$$

Queda definida por tanto la base sobre la que se construye este estudio, donde la función objetivo juega un papel principal en cuanto al problema de la optimización de los datos de un EEG. En el siguiente capítulo será revisada la literatura del problema en cuestión, además del conjunto de problemas más utilizados actualmente para la evaluación de los algoritmos, los denominados benchmarks, y por último los principales algoritmos y técnicas utilizados para problemas de tipo **LSGO**.

# Capítulo 4

# Revisión de la literatura

En este capítulo se revisa el estado actual de la literatura en cuanto a problemas de Big Optimization se refiere, en concreto los **benchmarks** más utilizados en esta rama. Análogamente, una revisión del la literatura relacionada con el benchmark del EEG se llevará a cabo, prestando especial atención a las dos vertientes sobre las que se basa este estudio: las **técnicas previamente aplicadas** para resolver el problema del EEG y las propuestas más relevantes de la literatura actual en cuanto a **algoritmos de optimización de miles de variables** se refiere, donde intervienen técnicas de distinta naturaleza, con el objetivo final de establecer el símil sobre el que se basa el desarrollo de este Trabajo Fin de Grado.

## 4.1. CEC y Benchmarks

Los problemas de optimización han estado presentes desde los inicios de la computación evolutiva y una vertiente de estos, los **problemas de optimización de miles de variables (LSGO)**, se encuentra actualmente en auge dada la necesidad de resolución de problemas del mundo real donde la formulación de estos incurre en una representación con miles de variables que precisa de una optimización efectiva y eficiente.

**IEEE**, la organización profesional para el avance tecnológico más grande del mundo, ha sido desde 1999 la encargada de organizar el **Congress on Evolutionary Computation**, uno de las mayores y más importantes convenciones de computación evolutiva a nivel mundial [20]; en ella se exponen numerosas técnicas para resolver diversos problemas de optimización, siendo estas técnicas evaluadas a través de distintas competiciones, y donde la dimensionalidad del problema normalmente no supone un impedimento adicional a la hora de resolver los problemas que en ella se plantean.

Sin embargo, no es hasta el año 2008 cuando se propone la primera

**Special Session & Competition on Large-Scale Global Optimization**[21], con el objetivo de establecer una base sobre la que evaluar las distintas propuestas, algoritmos o técnicas destinadas a resolver **problemas de optimización de alta dimensionalidad**, donde las técnicas disponibles en ese momento sufrían una considerable reducción de su rendimiento conforme aumentaba el tamaño del espacio de soluciones.

A partir de este momento entran en escena un **conjunto de problemas de optimización representados por funciones acotadas** que, a diferencia de años anteriores, contenía un elevado número de variables a optimizar. Este conjunto de funciones se agrupa dentro de lo que se denomina un **benchmark** o **test suite** que sirve para evaluar el desempeño de las técnicas de tipo LSGO. El concepto de benchmark, tanto la propuesta original como el de alta dimensionalidad, será ampliado posteriormente en el capítulo 7 donde se explicará en profundidad todo lo relacionado con el método para evaluar los algoritmos y técnicas que se proponen en las competiciones.

## 4.2. *EEG Problem* y propuestas de solución

En el año 2015, durante el IEEE Congress on Evolutionary Computation llevado a cabo en Sendai, Japón[22], se presenta la **Optimization of Big Data 2015 Competition**[9] donde, a diferencia de otros años, no se propone un conjunto de funciones benchmark de alta dimensionalidad sino un problema completamente distinto, el de la **optimización de los datos de un electroencefalograma**, que ha sido descrito en el capítulo anterior.

No se conocen con exactitud todas las técnicas propuestas para esta competición, pero tras ésta los organizadores publican el artículo *Evolutionary Big Optimization (Big Opt) of Signals*[2], en el que se recogen los resultados obtenidos al aplicar el problema sobre dos técnicas evolutivas multiobjetivo consideradas como las más potentes: MOEA/D y NSGA-II.

**MOEA/D**[23] emplea la estrategia de descomposición para hacer frente a los problemas multiobjetivo: descompone el problema MO en un subconjunto de problemas y los optimiza de forma separada utilizando únicamente la información de los subproblemas vecinos, reduciendo considerablemente la complejidad del problema en cada generación si lo comparamos con el algoritmo **Nondominated Sorting GA II** (NSGA-II)[24], que posee una complejidad de orden $O(MN^2)$ siendo $M$ la cantidad de objetivos y $N$ el tamaño de la población; el algoritmo crea una *mating pool* donde la población de padres y offsprings se combinan, eligiendo posteriormente el mejor de estos, dando solución a los impedimentos típicos de los algoritmos MO que utilizan clasificación no-dominada.

En el artículo mencionado anteriormente, se propone el enfoque **multiobjetivo**, optimizando por separado ambas funciones 3.6 y 3.7. Se elige optimizar dos representaciones distintas, frente al **dominio del tiempo** y al **de la frecuencia**; este último para intentar superar los inconvenientes derivados de la dimensionalidad del problema.

Los resultados muestran un mejor desempeño de **MOEA/D** frente a NSGA-II en la representación frente al dominio de la frecuencia, dado que esta, a través de una **Fast Fourier Transform** aplicada sobre los componentes principales, es capaz de reducir **hasta la mitad la dimensionalidad** del problema (con una frecuencia de 1Hz). La **reducción de la dimensionalidad** y los **buenos resultados** conseguidos son las principales bazas a tener en cuenta de cara a una aplicación real donde los datos de un EEG sean procesados y posteriormente "blanqueados"por algoritmos evolutivos multiobjetivo.

### 4.2.1. MAGA-BigOpt: Algoritmo Genético Multi-Agente para optimización uniobjetivo

**MAGA-BigOpt**, ***A Multi-Agent Genetic Algorithm for Big Optimization Problems***[25], es la principal propuesta de solución para el problema del EEG formulado en la Optimization of Big Data 2015 Competition, con enfoque uniobjetivo. Esta técnica, basada en el framework MAGA[26], "*reestructura los operadores de competición y autoaprendizaje, que finalmente combina con operadores de cruce y mutación para* ***simular la cooperación, competición y comportamientos de aprendizaje de los agentes***".

MAGA, en primera instancia propuesto para optimización numérica global a gran escala en[26], evolucionó para resolver el problema en cuestión, EEG, con 4 operadores genéticos donde un operador de autoaprendizaje con subgradiente es su principal activo, obteniendo resultados que superan con creces los propuestos como base en la competición mencionada.

Este algoritmo se basa en la idea de **agente**, que no es más que una solución candidata, y la **energía de un agente** computada como el valor negativo de la función $f$ definida en la ecuacion 3.8. Los agentes se encuentran en un entorno similar a una rejilla $L$, que se llama la **rejilla de agente**, con tamaño $L_{size} \times L_{size}$. La representación vectorial de un agente (solución) permite situarlo en una posición $L_{i,j}$ donde tendrá 4 vecinos espaciales subyacentes: arriba, abajo, izquierda o derecha según donde se encuentre en la rejilla.

El objetivo consiste por tanto en maximizar la energía de los agentes a través de los operadores evolutivos propuestos, donde los procesos de competición y cooperación son llevados a cabo por los **operadores de com-**

**petición y cruce de vecinos** respectivamente, mientras que el uso del conocimiento del contexto se basa en los operadores de cruce y autoaprendizaje. La descripción completa de cada uno de los operadores y su aplicación se puede consultar en [25].

Atendiendo a los resultados obtenidos tras aplicar MAGA-BigOpt sobre el problema del EEG, tanto en problemas con ruido como sin este, se observan valores superiores a los propuestos como base en la competición, alcanzando valores muy cercanos a cero y con una varianza prácticamente despreciable, hecho que junto a la potencia de los operadores propuestos, reafirma la candidatura de esta técnica a ser considerada como **punto de referencia** en este estudio tanto a niveles de calidad de la solución como a restricciones de tiempo, al elegir utilizar el mismo enfoque, uniobjetivo, por el que se ha optado en este trabajo.

## 4.3. Algoritmos Large Scale Global Optimization

Los computación evolutiva a través de algoritmos dedicados a la resolución de **problemas de optimización global continua con miles de variables** es un campo que, como anteriormente ha sido mencionado, ha sufrido una reciente evolución en cuanto al número de propuestas debido a la necesidad de plantear soluciones efectivas y sobre todo eficientes a una gran cantidad de problemas reales que pueden ser representados mediante un problema de Big Optimization.

La *Special Session and Competition on Large-Scale Global Optimization* del WCCI 2018[3] indica que no es hasta el año 2015 cuando se experimenta un cambio significativo entre la cantidad de artículos publicados en esta materia y los propuestos únicamente para conferencias; así lo refleja el gráfico de la figura 4.1, donde se pueden ver los resultados que arroja la búsqueda del término "*Large Scale Global Optimization*" en la base de datos Scopus[11], sitio desde el que se han obtenido la gran mayoría de referencias bibliográficas que se encuentran en este trabajo.

Un estudio publicado en 2014 bajo el nombre de ***A comprehensive comparison of large scale global optimizers***[27] recoge la comparativa entre las **tres mejores propuestas** de las sesiones del CEC 2010, CEC 2012, CEC 2013 y SOCO 2011 con el objetivo de **analizar el desempeño global** de aquellas técnicas cuando son evaluadas con respecto a **tres distintos benchmarks** con distintos tipos de funciones, para determinar no solo la efectividad y eficiencia de forma individual sino también de manera global. Resaltar que los algoritmos que sean seleccionados para formar parte de este estudio serán detallados en profundidad en el capítulo 5.

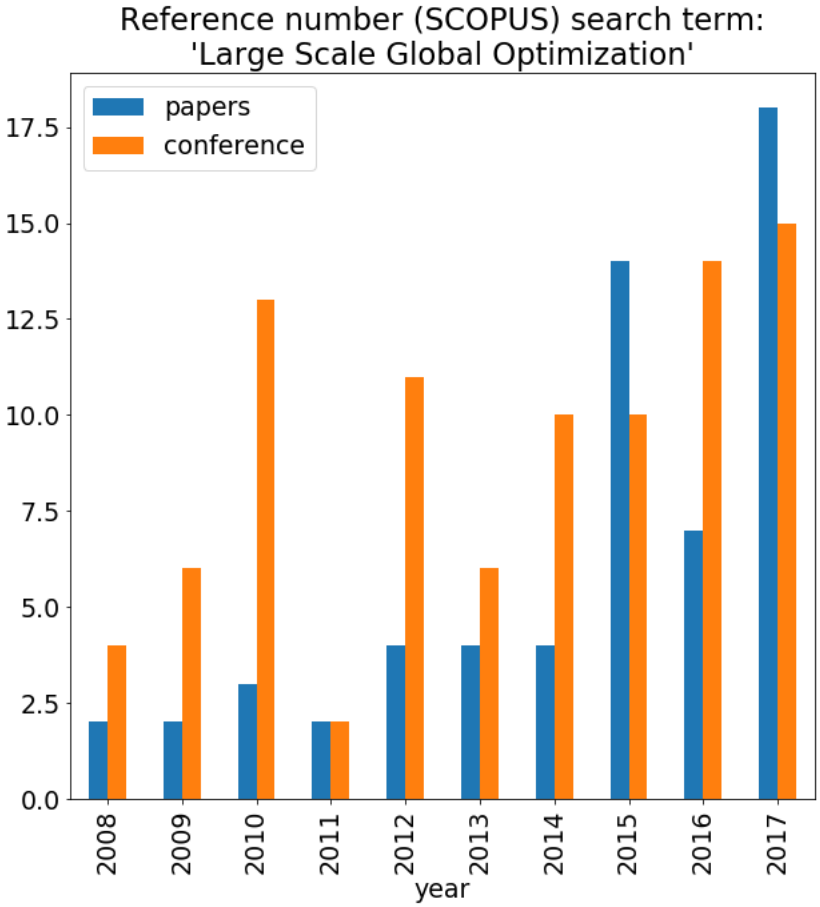


Figura 4.1: Evolución de las referencias de propuestas de algoritmos LSGO. Fuente: [3]

### 4.3.1. SOCO 2011 Special Issue

La *SOCO 2011 Special Issue* cuenta con fuertes propuestas como lo son ***A MOS-based dynamic memetic differential evolution algorithm for continuous optimization: a scalability test***[28], o MOS-SOCO2011, un algoritmo que basándose en la utilización del framework MOS (Multiple Offspring Sampling), es capaz de **combinar distintas estrategias de búsqueda** sin tener ninguna fuga; se combinan en este caso un algoritmo de **evolución diferencial (DE)** y la primera de las tres búsquedas locales del algoritmo MTS[29], la **MTS-LS1**.

Ajustando la participación de cada técnica en base a una función de participación híbrida que favorece la utilización de aquella técnica que ha

desempeñado mejor en base a dos medidas distintas, el algoritmo es capaz de obtener resultados muy competitivos, logrando vencer a los demás contrincantes en la competición.

Teniendo en cuenta que la evolución diferencial (DE)[6] es considerada actualmente una de las técnicas más potentes en cuanto a optimización continua, otra propuesta interesante a tener en cuenta de esta misma competición es ***Scalability of generalized adaptive differential evolution for large-scale continuous optimization (GaDE)***[30], una generalización del DE adaptativo en el que a través de una distribución de probabilidad es capaz de **estimar el valor de cada uno de los parámetros para cada individuo** de la población. Esto se consigue haciendo que la distribución de probabilidad oscile en cada generación en función de las mejoras en fitness de cada individuo, confiriéndole una gran potencia al algoritmo a la vez que lo situa como posible técnica candidata a representar una futura solución.

### 4.3.2. CEC 2013 y CEC 2015

Aunque las propuestas del CEC 2012 no llamen significativamente la atención, por tratarse de técnicas parecidas a las del SOCO 2011 pero con ligeros cambios, en las del **CEC 2013** se encuentran candidatos muy interesantes como lo son el *Large Scale Global Optimization: experimental results with MOS-based hybrid algorithms*[31], técnica que combina un algoritmo genético con dos búsquedas locales muy conocidas y potentes, otra vez, a través del framework MOS: la del algoritmo **Solis Wets**[32] y la **MTS-LS1 Reduced**, una modificación de la MTS-LS1[29] desarrollada exclusivamente para esta competición donde son optimizadas aquellas variables que contribuyen en mayor medida a una mejora en la calidad de las soluciones.

En esta misma sesión disponemos de una entrada que llama la atención en particular. A pesar de quedar en tercer lugar dentro de esta competición, *Scaling up Covariance Matrix Adaptation Evolution Strategy using cooperative coevolution* o CC-CMA-ES[33] es interesante debido a la formulación de la propuesta: dividir el problema en subproblemas más pequeños que serán optimizados utilizando la técnica evolutiva de adaptación por matrices de covarianza CMA-ES. Considerando la dimensionalidad del problema presentado en este trabajo, estrategias de **descomposición o subdivisión en problemas más pequeños** se postulan como plausibles alternativas a tener en cuenta de cara al estudio.

Durante el **CEC2015** tuvo lugar una competición de Large-Scale Global Optimization, cuyos resultados se encuentran disponibles en [34] y donde algunas técnicas como CC-CMA-ES o MOS repiten aparición, siendo finalmente **MOS** el algoritmo que obtiene mejores resultados de forma global, hito que también logró durante el CEC 2013. De esta forma, se refuerza

la concepción que se tiene de estos algoritmos como fuertes candidatos a proponer soluciones efectivas a los problemas de optimización de alta dimensionalidad.

### 4.3.3. WCCI 2018 Competition on LSGO

Finalmente, observando los resultados de la más reciente competición en materia de LSGO,la **Special Session and Competition on Large-Scale Global Optimization, WCCI 2018** se proponen dos algoritmos que, dada las características y los resultados que obtienen, llaman la atención de manera significativa. Estas propuestas tienen por nombre respectivamente *SHADE with Iterative Local Search for Large-Scale Global Optimization*[35] o **SHADEILS** y *LSHADE-SPA Memetic Framework for Solving Large Scale Problems*[36], que el autor denomina **MLSHADE-SPA**.

El primero, SHADEILS, es un algoritmo con una construcción muy similar a una propuesta anterior denominada IHDELS[37] pero con tres principales diferencias que lo sitúan por encima de su antecesor: mecanismo de **elección de LS mejorada**, mecanismo de **reinicio de la población** y el uso de **SHADE**[38] en vez de SaDE[39]. Se elige una LS de entre MTS-LS1[29] y L-BFGS-S[40], siendo ambas complementarias: mientras que la primera es rápida pero sensible a rotaciones del sistema de coordenadas, la segunda es mas lenta pero insensible a este tipo de rotaciones. Además, al utilizar SHADE, autoajusta sus parámetros basado en la *historia* y manteniendo la población entre iteraciones, consigue intensificar el proceso de exploración.

Por otra parte el segundo, ML-SHADE-SPA, es un framework que utiliza **tres algoritmos basados en poblaciones** para el proceso de **exploración**, LSHADE-SPA, EADE y ANDE, mientras que el proceso de **explotación** es llevado a cabo por una **versión modificada del algoritmo MTS**. Introduce además la idea de **divide y vencerás**: se mezclan de forma aleatoria las dimensiones y se resuelven de forma separada. Este proceso es completamente independiente de cualquier tipo de información, esto es, las particiones se realizan sin tener en cuenta ninguna suposición acerca de la estructura del problema, añadiendo robustez al algoritmo y confiriéndole un grado de generalización que le permite hacer frente a disintos problemas siguiendo el mismo enfoque.

Cabe destacar que estos dos algoritmos han logrado superar al algoritmo que hasta este año habia sido considerado como *estado del arte*, el mencionado anteriormente **MOS**, por lo que son los más fuertes candidatos a tener en cuenta en este estudio. Para finalizar, se revisarán las propuestas de algoritmos de descomposición de variables y las posibles ventajas que puede aportar a favor de la resolución del problema tratado en este trabajo.

## 4.4. Algoritmos de descomposición de variables

La interacción de las variables de un problema de optimización juega un papel fundamental en el proceso de obtención de soluciones óptimas. Es por ello que surgen técnicas de **descomposición** y **co-evolución cooperativa** como alternativas para hacer frente al problema inherente de la interacción. De forma análoga, la descomposición del problema en **subproblemas más pequeños** favorece la reducción de dimensionalidad global del problema y por consiguiente disminuye la complejidad del espacio de soluciones.

**Cooperative Co-Evolution (CC)**[41] es el framework por excelencia para guiar el proceso de optimización una vez creadas las descomposiciones, el cual requiere de un **agente** (un algoritmo evolutivo, principalmente) para coordinar como se aplica este agente a las distintas componentes fruto de la descomposición; no obstante, la **descomposición del problema** sigue siendo un impedimento considerable. La descomposición óptima es aquella que **minimiza la interacción** entre las componentes descompuestas y aquí entran en juego técnicas de agrupación como **Random Grouping o Delta Grouping**, donde es **Differential Grouping (DG)-2014** [42] la propuesta que ha alcanzado mejores cotas de éxito, utilizando conceptos como **función aditivamente separable** y **forward difference** para detectar las interacciones.

Existen una familia completa de este tipo de propuestas, donde están incluidas **Global Differential Grouping**[43], **eXtended Differential Grouping**[44] o la más reciente incorporación **Differential Grouping 2** (DG2)[45]. Cada una de estas propuestas aborda el problema de la descomposición de variables siguiendo el mismo precepto pero centrándose en características particulares, como la detección directa o indirecta de interacciones.

Esta grupo de técnicas conforman un bloque con fuerte base matemática que ha mostado resultados muy competitivos, sobretodo cuando las componentes individuales son optimizadas con el **framework CC** a través agentes robustos tales como **MOS, CMA-ES o MA-SW-Chains**, por lo que dada la dimensionalidad del problema y los beneficios que implica el uso de este tipo de técnicas, si se satisfacen las condiciones necesarias, aspiran a formar parte de una solución efectiva para el problema del EEG.

Tras analizar las actuales técnicas de resolución de problemas en el ámbito de LSGO, el siguiente punto a tratar tiene que ver con la **selección de las técnicas** que formarán parte de este estudio, basando las decisiones en la **identificación de posibles ventajas** de las que dispongan algunas técnicas con respecto a otras y las diferencias fundamentales entre ellas.

## 4.5. Selección de algoritmos

El estado actual de la literatura abre una posibilidad de estudio sobre un problema real tomando como punto de referencia el problema del EEG. El estudio se centrará principalmente en llevar a cabo una **adaptación híbrida de los algoritmos** seleccionados de forma que sean capaces de procesar la función objetivo que representa el problema del EEG y de proveer resultados que solucionarán, en caso tal, las deficiencias anteriormente expuestas.

Mediante un estudio experimental lo suficientemente completo, se pretende identificar técnicas potentes en las que **prime una calidad de soluciones alta**, mejor o igual a la propuesta por el algoritmo **MAGA** (dado el enfoque uniobjetivo elegido), que será el algoritmo de referencia con el que comparar, utilizando para ello, los algoritmos más adecuados según este criterio pero que además sean capaces de cumplir con las **restricciones temporales** inherentes al enfoque de este estudio y cuya **complejidad interna** no sea demasiado elevada.

Aquel algoritmo que muestre el desempeño más adecuado de acuerdo a las anteriores consideraciones de complejidad, restricciones temporales y calidad de las soluciones, se someterá a estudio bajo un **algoritmo de descomposición de variables** para comprobar si la **optimización por separado de grupos de variables** implica realmente una **reducción de la dimensionalidad** a nivel individual del problema

La implementación de este proceso está sujeta al tipo de función con la que se enfrenta el algoritmo y la factibilidad de la aplicación de técnicas de cooperación co-evolutiva depende directamente de los factores derivados de la naturaleza de la función, principalmente su dimensionalidad, lo que influirá de forma determinante en el **tiempo de respuesta**; estas son las condiciones que se deben cumplir para poder llevar a cabo una hibridación que tenga la capacidad de obtener mejores resultados sin dejar de ser útil a efectos prácticos.

Posteriormente, se emitirá un juicio en cuanto a si se consigue mejorar los resultados que proporciona el algoritmo base elegido. En caso favorable, se propondrá una solución lo suficientemente respaldada de cara a ser implementada en sistemas BCIs donde se pueda estudiar de forma mucho mas eficiente y efectiva un EEG cuantitativo, incluso en entornos donde se requiera respuesta en tiempo real.

El siguiente capítulo analiza en detalle todas las técnicas que han sido elegidas para formar parte de este estudio, así como de las razones que motivaron la elección final de las mismas.

# Capítulo 5

# Análisis de las Propuestas: Algoritmos LSGO

La finalidad de esta sección consiste en dar respuesta a las tareas 3 y 4 planteadas en el capítulo 2 donde se propone, principalmente, **analizar de forma exhaustiva los distintos algoritmos** seleccionados como fuente de estudio en este trabajo.

El análisis está organizado de forma que se expongan todos los aspectos relevantes del algoritmo en cuestión tales como la **representación matemática** de su composición, **estructura interna** de la técnica, **elementos esenciales**, características particulares de la **implementación**, así como de la motivación para su **elección**, que estará basada esencialmente en los elementos anteriores, y en general, cualquier detalle significativo que aporte información relevante para este trabajo.

Al término de este capítulo se habrán completado las tareas necesarias para iniciar el proceso de **adaptación de las técnicas** al problema del EEG, donde se repasarán **aspectos fundamentales de la propuesta de problema** relacionados con la representación e implementación en código fuente y que sentará las bases del **proceso experimental** que finalmente conduzca a la generación de resultados, que serán en última instancia evaluados e interpretados y conformarán la base de las conclusiones pertinentes.

## 5.1. Multiple Offspring Sampling (2011)

La primera propuesta elegida para este estudio es la conocida como **MOS2010**, dado que fue formulada en ese año y utilizada en la posterior competición del CEC 2011[27]. La publicación llega bajo el nombre de ***A MOS-based dynamic memetic differential evolution algorithm***

***for continuous optimization: a scalability test*** [28].

A grandes rasgos, en la publicación se propone un innovador **algoritmo memético híbrido** que a través del framework MOS (Multiple Offspring Sampling)[46] **combina distintas estrategias de búsqueda** con el objetivo de alcanzar un balance exploración-explotación óptimo, en concreto, dos técnicas heurísticas que aplicadas por separado han mostrado resultados altamente competitivos.

La elección de este algoritmo como primer candidato está motivada por las siguientes razones: es la **primera formulación de MOS** que se propone para la conferencia del **SOCO 2011**, donde su diseño totalmente innovador se rige por las exigencias típicas de problemas de Big Optimization. Además, fue la técnica ganadora de esta edición, superando a otras que a pesar de utilizar algoritmos de evolución diferencial (DE), no consiguieron superar el rendimiento de MOS, y dónde algunas de estas técnicas eran estructuralmente más complejas.

La incorporación de la búsqueda local **MTS-LS1** de [29] ha sido otro de los principales motivos de su elección; tras la primera *Special Session on Large-Scale Global Optimization* de 2008, el algoritmo MTS fue el que obtuvo mejores resultados para problemas de alta dimensionalidad, por lo que su uso en este algoritmo híbrido presenta una ventaja sustancial de partida frente a otras técnicas dada las capacidades de exploración que MTS añade durante el proceso de explotación.

Finalmente, MOS2011 **sentó la base del algoritmo** que durante los últimos 5 años ha sido considerado como *state-of-the-art* en problemas LSGO, donde la formulación matemática subyacente relacionada con la **calidad de las soluciones** y la **participación** de cada técnica en el proceso tienen un rol fundamental, dado que estas contemplan dos tipos de mediciones que implican que cada técnica no solo debe mejorar sustancialmente las soluciones, sino además un buen porcentaje de éstas, lo que otorga más capacidad de respuesta ante el problema del estancamiento. Revisada la motivación de la elección, una descripción detallada de todo el algoritmo se expone a continuación.

El algoritmo enunciado propone dar solución al inconveniente que surge del aumento de la dimensionalidad sobre un problema, incluso cuando la naturaleza del mismo no cambia. Es aquí donde entra en juego el framework MOS, que permite combinar dos potentes técnicas: un algoritmo de **evolución diferencial (DE)** y la primera de las **búsquedas locales** (LS) del algoritmo MTS, la conocida como **MTS-LS1**[29].

Este framework permite combinar distintas metaheurísticas siguiendo distintos tipos de enfoque, donde ha sido el **HRH** (High-level Relay Hybrid) el que se ha elegido; mediante esta perspectiva en particular, se **ajusta de**

**forma dinámica** el número de **evaluaciones de la función objetivo** que cada algoritmo puede realizar. Para comprender la naturaleza de este enfoque es imprescindible conocer el origen de la terminología y de los demás enfoques.

En el artículo ***A taxonomy of hybrid metaheuristics (2002)*** [47] se propone una taxonomía de los algoritmos híbridos, con el fin de establecer una terminología común a estos mecanismos, donde se combinan **esquemas jerárquicos**, para reducir la cantidad de clases, y **planos**, para cuando las clases que definen cada algoritmo se eligen de forma arbitraria. De esta taxonomía surgen 4 tipos de estrategias principales que detallamos a continuación:

1. **LRH - Low-level Relay Hybrid**: una técnica metaheurística se incrusta en otra de **una sola solución**. Algoritmos de Simmulated Annealing (SA) combinados con LS son ejemplos de este enfoque.

2. **LTH - Low-level Teamwork Hybrid**: la técnica es **incrustada en otra metaheuristica basada en poblaciones**. Un claro ejemplo de este tipo de enfoques es un algoritmo genético **GA** cuyo operador de mutación o cruce introduzca comportamientos que aumenten la capacidad de explotación del algoritmo.

3. **HRH - High-level Relay Hybrid**: este enfoque ejecuta las metaheurísticas de **forma secuencial**, una a continuación de la otra. Es el enfoque elegido en esta propuesta, por lo que se requiere un mecanismo de control para organizar las técnicas y su ejecución.

4. **HTH - High-level Teamwork Hybrid**: distintas técnicas son ejecutadas en paralelo, donde cada una se sirve de la información de las demás, cooperando en conjunto para encontrar soluciones óptimas. Aquí se pueden mencionar las técnicas basadas en CC (Cooperative Co-evolution).

La técnica que se propone en esta publicación se basa en la hibridación de distintas técnicas, donde los algoritmos de evolución diferencial (DE) han sido los más utilizados a lo largo de los años. De forma complementaria, el algoritmo MTS[29] fue capaz de resolver problemas de hasta 1000 variables en el CEC 2008, valiéndose de **tres poderosas búsqueda locales**, razón por la cual se presenta como integrante de esta propuesta en particular. Así, el algoritmo DE dejaría lugar a que la LS encontrase regiones mas prometedoras a la vez que se reducen las evaluaciones a la función objetivo y por otra parte, la propia búsqueda local intentaría paliar los efectos del estancamiento en este tipo de algoritmos.

En esta propuesta en particular, el término **técnica de descendencia** hace referencia a **cualquier mecanismo para crear soluciones candidatas**, donde además son necesarios otros cuatro elementos: un modelo de algoritmo evolutivo en particular, una codificación de las soluciones, operadores específicos y parámetros necesarios.

Esta característica implica que se pueden utilizar de **forma simultánea** varias técnicas para generar descendientes, lo que a su vez conlleva la existencia de un mecanismo que controle el uso de estas. El framework MOS propone dos grupos de funciones para solventar este inconveniente, lo que le permite **ajustar de forma dinámica el grado de participación de cada técnica** durante el proceso de búsqueda:

- **Funciones de Calidad**: evalúan el fitness de los individuos en función de alguna característica deseable

- **Funciones de Participación**: asigna la cantidad de descendientes que genera cada técnica en función de la calidad de las soluciones.

El autor de la propuesta considera dos tipos de algoritmos según la taxonomía formulada en [47], estos son, HTH y HRH, sin embargo, se opta finalmente por el algoritmo de tipo **HRH**. En este tipo de algoritmos, las técnicas elegidas, en nuestro caso DE y MTS-LS1, son aplicadas en secuencia una a continuación de la otra, y cada una de cierta forma reutiliza la población resultante de la anterior. Este enfoque es más adecuado debido a que se utiliza en este caso una técnica no poblacional, como es la búsqueda local en cuestión.

El proceso de búsqueda se establece al principio de la ejecución, dividiendo el mismo en un **número fijo de pasos**. A cada paso se le asigna un número **fijo** de evaluaciones de la función objetivo, que son administradas de forma interna por cada técnica a través de la **función de participación** de la misma. De forma aclaratoria se expone el pseudocódigo de la propuesta HRH y se remite al lector a [28] si se quiere profundizar en la propuesta HTH.

**Algorithm 1** : HRH MOS

1: Crear población de soluciones candidatas $P_0$
2: Distribuir la participación uniformemente entre las $n$ tecnicas usadas $\rightarrow$ $\forall j\ \Pi_0^{(j)} = \frac{FE_{s_0}}{n}$. Cada tecnica produce un subconjunto de individuos de acuerdo con su participacion $\Pi_0^{(j)}$
3: Evaluar $P_0$
4: **while** no se supere el numero de pasos **do**
5: Actualizar la Calidad de la técnica $T^{(j)}$ como la media de la calidad de los individuos que ha creado en el paso anterior
6: Actualizar los ratios de participación en función de los valores de calidad del paso 5 $\rightarrow \forall j\ \Pi_{i+1}^{(j)} = PF(Q_i^{(j)})$
7: Actualizar el número de evaluaciones de la función objetivo de cada técnica $\rightarrow \forall j\ FE_{s_i}^{(j)} = \Pi_{i+1}^{(j)} \cdot FE_{s_i}$
8: **for** cada tecnica $T^{(j)}$ **do**
9: **while** no se supere $FE_{s_i}^{(j)}$ **do**
10: Evolucionar
11: **end while**
12: **end for**
13: **end while**

Como se puede observar, la participación de cada técnica depende de una función de calidad $Q$ que toma en consideración dos características: el **incremento medio del fitness** de los individuos tras la evaluación y el **número de veces** que se ha producido esta mejora. Se representa en la siguiente ecuación:

$$Q_i^{(j)} = \begin{cases} \Sigma_{i-1}^{(j)} \textbf{ if } \forall k,l \in [1,n] : \Sigma_{i-1}^{(k)} > \Sigma_{i-1}^{(l)} \implies \Gamma_{i-1}^{(k)} > \Gamma_{i-1}^{(l)} \\ \\ \Gamma_{i-1}^{(j)} \textbf{ otherwise} \end{cases} \tag{5.1}$$

donde

$Q_i^{(j)} \equiv$ Calidad de la técnica $T^{(j)}$ en el paso $i$
$\Sigma_i^{(j)} \equiv$ Incremento medio del fitness de $T^{(j)}$ en el paso $i$
$\Gamma_i^{(j)} \equiv$ Num. de mejoras de fitness de $T^{(j)}$ en el paso $i$

En esta función de calidad $Q$ se utiliza $\Sigma_i^{(j)}$ si y sólo si hay consenso entre el incremento medio de fitness y el número de mejoras de una técnica a otra. En caso contrario se usa solo el número de incrementos del fitness. Esto es así debido a que una técnica que no este explorando el espacio de soluciones de forma adecuada, puede introducir grandes mejoras en el

fitness, lo que puede no ser representativo. Estas acciones se penalizan en favor de comportamientos más adecuados, tales como mejoras relativamente pequeñas sobre buenos individuos de la población.

Una **función de participación dinámica** utiliza estos valores de $Q$ para ajustar el número de evaluaciones de la función objetivo asignado a cada técnica en cada paso. La función de participación $PF$ calcula, para cada paso del algoritmo, un **factor de compensación** $\Delta_i^{(j)}$ para cada técnica, lo que representa el decrecimiento de participación para cada técnica excepto para la mejor. Cada una aumentará su participación en proporción al cociente $\frac{\Sigma\Delta_i^{(j)}}{n^o tecnicas}$. La $PF$ se describe en la siguiente ecuación:

$$PF_{dyn}(Q_i^{(j)}) = \begin{cases} \Pi_{i-1}^{(j)} + \eta \textbf{ if } j \in best \\ \\ \Pi_{i-1}^{(j)} - \Delta_i^{(j)} \textbf{ otherwise} \end{cases} \tag{5.2}$$

$$\eta = \frac{\Sigma_{k \notin best} \Delta_i^{(k)}}{|best|}$$

$$best = l / Q_i^{(l)} \geq Q_i^{(m)} \forall l, m \in [1, n]$$

donde $\Pi_i^{(j)}$ es el **porcentaje de individuos que genera de la población actual**. El factor de compensación se calcula como se puede ver en la siguiente ecuación, donde $\xi$ es el ratio de transferencia de una técnica a otra, con valor 0,05.

$$\Delta_i^{(j)} = \xi \cdot \frac{Q_i^{(best)} - Q_i^{(j)}}{Q_i^{(best)}} \cdot \Pi_{i-1}^{(j)} \ \forall j \in [1, n] / j \neq best \tag{5.3}$$

En resumen, MOS2011 es una **técnica memética híbrida de clase HRH** que combina un DE y una búsqueda local MTS-LS1. Al ser HRH, una técnica es ejecutada tras la anterior y además, cada una de estas técnicas participa de forma distinta a las demás ya que ejecuta un número de evaluaciones de la función objetivo que cambia de forma dinámica según una función de participación, la cual basa sus cálculos en las mejoras del fitness que ha llevado a cabo esa técnica en el paso anterior.

Cabe destacar que se puede establecer un **ratio mínimo de participación** de forma que una técnica siempre contribuya a la convergencia del algoritmo. Si se diese el caso de que toda la población converge en un mismo punto del espacio de soluciones, la mejor de las soluciones se guarda y el resto de la población se reinicia de manera uniforme. Estas son las principales características **MOS2011**, primera técnica candidata de estudio en este trabajo.

## 5.2. Multiple Offspring Sampling (2013)

La segunda de las técnicas seleccionadas para estudiar en este trabajo se trata de una muy similar a la anterior pero con cambios lo suficientemente sustanciales como para superar en rendimiento a su antecesora. La propuesta publicada en el **2013 IEEE Congress on Evolutionary Computation** que lleva por nombre ***Large Scale Global Optimization: Experimental Results with MOS-based Hybrid Algorithms* (2013)**[31] ha sido considerado desde ese año como el estado del arte en cuanto a algoritmos de LSGO.

Basado nuevamente en el framework MOS, la creación de este algoritmo memético híbrido pasa previamente por una etapa de estudio intensivo donde se ponen a prueba **ocho técnicas de optimización** sobre las que se realiza una estimación de parámetros lo más precisa posible, donde finalmente aquellas que mejores resultados arrojen serán las que se incluirán como agentes en esta propuesta.

Las ocho técnicas de partida en este estudio suponen la principal razón que ha motivado la elección de este algoritmo para este trabajo, y son: un algoritmo genético (**GA**), un algoritmo de evolución diferencial (**DE**[6]), **SaDE**[39] (Self-Adaptive DE), **GODE**[48] (Generalized Opposition-based DE), **SaGODE** (como combinación de las dos anteriores y propuesto únicamente para este estudio) y las búsquedas locales **Solis Wets**[32], MTS-LS1 y MTS-LS1-Reduced, estas dos últimas diseñadas a partir de [29] específicamente para este estudio.

Si se quiere conocer los parámetros de cada técnica en concreto y los valores de los mismos tras el proceso de estimación de parámetros, se remite al lector a [31]. En esta propuesta en concreto, tras los pertinentes estudios de estimación de parámetros, han sido elegidas tres técnicas para formar el algoritmo memético híbrido basado en el framework MOS, nuevamente con taxonomía HRH[47]: GA, Solis Wets y MTS-LS1-Reduced.

Elegir este algoritmo como candidato para el estudio **frente a otras alternativas** se hace indispensable debido al propio estudio que se realiza en la publicación, previo a la elección de los componentes del algoritmo, donde se **descartan las técnicas menos prometedoras** en función de un sistema de puntos que utiliza el autor. Por tanto, técnicas como SaDE o GaDE quedan descartadas por su bajo rendimiento en este trabajo, al igual que CC-CMA-ES puesto que no consigue acercarse a los resultados de MOS2013, o IHDELS, que aun tras quedar en segundo puesto en el CEC2015, no consiguió acercarse a los resultados que esta hibridación era capaz de ofrecer.

Como ya ocurrió con MOS2011, una propuesta que permita **ejecutar**

**varias técnicas en secuencia** a través del enfoque **MOS HRH**, obtiene **mejores resultados que las técnicas por separado**, por lo que cualquier otra que no haya sido finalmente utilizada para formar parte de esta hibridación memética, también se puede descartar sin pérdida alguna de generalidad. Esta idea se ve reforzada por el hecho de que esta propuesta, **MOS2013**, se ha mantenido en la cima desde los últimos cinco años, sin que ninguna técnica de años anteriores haya conseguido superar su gran potencia y robustez.

A pesar de que la publicación no destaca importantes cambios con respecto a su predecesor en cuanto al funcionamiento interno, derivado de las funciones de calidad y participación, si que remarca algunos aspectos de implementación que, a primera vista, le otorgan un mejor balance exploración-explotación, y por ende un rendimiento superior frente a otras técnicas.

La modificación que se realiza sobre la MTS-LS1, diseñada específicamente para este estudio, transforma la misma en lo que el autor denomina **MTS-LS1-Reduced**. Se dice *reducida* porque intenta optimizar al máximo el número de evaluaciones que le corresponden. En vez de realizar la exploración sobre la totalidad de las dimensiones, lo que hace es optimizar **las regiones (variables) más prometedoras**.

Para ello, en cada paso del algoritmo se almacena, para cada dimensión, la **mejora de la calidad de la solución** conseguida al explorar esa variable. Esta información es la que guía el proceso de exploración en el siguiente paso, sirviéndose de dos variables porcentuales para determinar cuántas dimensiones serán exploradas en el siguiente paso: uno indica el porcentaje de **variables mejoradas a explorar** y otro, el porcentaje **mínimo del resto de variables** que serán seleccionadas de forma aleatoria para ser optimizadas, de forma que se eviten estancamientos o comportamientos que conduzcan a una sobreexplotación del espacio de soluciones.

Nuevamente, cada uno de los algoritmos de la propuesta aporta sus soluciones particulares siguiendo una **función de participación**, que basa su funcionamiento en una **función de calidad**, tal y como lo hacía su sucesor MOS2011. En MOS2013, se han utilizado las mismas funciones, tanto de calidad como de participación, las cuales aparecen respectivamente en las ecuaciones 5.1 y 5.2. El cálculo del factor de compensación en la función de participación sigue la misma ecuación descrita en 5.3.

Finalmente, los algoritmos que forman parte de la propuesta de MOS 2013 se rigen por dos variables principales: el **ratio de participación mínimo** que se confiere a cada técnica, que es de un 20 %, lo que quiere decir que una técnica no puede ejecutar menos de ese **porcentaje de las evaluaciones** durante el paso *i-ésimo*. La **cantidad de evaluaciones de cada paso**, la segunda variable, tiene un valor de 36000.

En el algoritmo 1 se detalla con claridad el pseudocódigo de la propuesta de algoritmo **MOS HRH**, donde se pueden ver, tras la explicación de la propuesta **MOS2013**, que los cambios introducidos frente a su antecesor incurren principalmente en la **variedad de técnicas** elegidas para ejecutarse de forma secuencial a través del enfoque HRH y las **variables** que rigen el tamaño de cada paso del algoritmo, así como la participación mínima de cada subtécnica en el proceso de optimización.

Este hecho refuerza la decisión de elegir este algoritmo para formar parte de este trabajo, dado que aparte de mantenerse en la cima durante los últimos cinco años, se requiere comprobar si realmente existe una **mejora sustancial** frente al algoritmo **MOS2011** que catapulte los resultados a nivel de eficacia y eficiencia de la actual propuesta. A continuación, se enuncia la tercera propuesta a estudiar en este trabajo.

## 5.3. SHADEILS

La tercera propuesta elegida para estudiar en este trabajo se trata de un algoritmo muy reciente, presentado en la **IEEE WCCI 2018: Special Session and Competition on Large-Scale Global Optimization**[3], donde se busca dar una solución sencilla pero eficaz al problema de la dimensionalidad y al aumento de la complejidad del espacio de soluciones que esta produce. Así es ***SHADE with Iterative Local Search for Large-Scale Global Optimization***[35] (SHADEILS), el algoritmo que se puede considerar actualmente como el **estado del arte** a partir de este año, arrebatándole el puesto a un MOS que llevaba ganando desde 2011 y considerado estado del arte desde 2013.

Se trata de una evolución sustancialmente más sencilla y completa que su antecesor, propuesto para el CEC2015 ***Iterative hybridization of DE with local search for the CEC'2015, special session on large scale global optimization***(IHDELS)[37]. Este algoritmo entra en la categoría de **meméticos**, dado que combina de forma iterativa un algoritmo evolutivo como lo es el SHADE[38] y una búsqueda local de entre dos posibles candidatas, MTS-LS1[29] y L-BFGS-B[40], que son además complementarias. Durante este proceso iterativo, a diferencia de otras técnicas de agrupación de variables, se exploran **todas las dimensiones a la vez**. Son tres las diferencias principales con su predecesor: el mecanismo mejorado de **selección de la búsqueda local** a aplicar, el renovado **mecanismo de reinicio de población** y la decisión de utilizar SHADE frente a SaDE[39].

El hecho de seleccionar esta reciente propuesta, diseñada y evaluada hace apenas unos pocos meses, esta motivada por una recomendación del tutor de este Trabajo Fin de Grado. Además, tras analizar la técnica predecesora, se

hace latente la **imposibilidad de IHDELS de vencer a MOS** en 2015, por lo que se requeriría de una renovación más adecuada pero sobretodo con mejor capacidad de adaptación y, porque no, más sencilla. Por tanto, todos los algoritmos basados en SaDE serían **poco útiles** en este estudio, donde se busca una propuesta lo más robusta, eficaz y eficiente posible.

Los elementos principales que se destacan al inicio de la propuesta hacen que aumente el interés por esta de forma inmediata: el combinar dos búsqueda locales, una **rápida pero sensible a las rotaciones** del espacio de coordenadas y otra **más lenta pero muy robusta y sin sensibilidad alguna** a las rotaciones, y que además posee un **mecanismo de selección** de la más adecuada según la *historia* reciente de la **mejora en la calidad de soluciones**, le confieren gran capacidad para hacer frente a la complejidad del espacio de soluciones pero sobre todo para intentar sobreponerse a las necesidades de eficiencia que se busca en este trabajo.

Como añadido, el renovado mecanismo de **reinicio de la población** contempla un pequeño umbral de mejora tras el que se reinicia la población cuando no se consigue superar este umbral tras 3 iteraciones consecutivas, lo que permite que el algoritmo siga **avanzando aunque los pasos sean pequeños**. Pero no solo la población se reinicia, sino que también lo hacen los **parámetros que rigen la aplicación de la LS**, lo que atribuye una mayor capacidad de exploración para la siguiente iteración, lo que resulta crucial para resolver problemas de estancamiento. A continuación, se procede a introducir el esquema general del algoritmo, prestando atención a las tres diferencias principales con IHDELS.

El primer paso para entender el algoritmo pasa por presentar el pseudocódigo general del mismo, donde se destacarán las tres partes principales que se resaltan en la publicación, que serán posteriormente extendidas en favor de una mejor comprensión y para reforzar las ideas que han motivado la decisión de utilizar este algoritmo para el estudio conducido.

1. **Algoritmo de Exploración SHADE**: este sencillo algoritmo de DE autoajusta sus parámetros (CR y F) **basándose en el concepto de *historia***[38], una memoria de tamaño $k$ que contiene valores $F$ y $Cr$ que han generado individuos mejores tras el proceso de mutación. Además, la población se mantiene de una iteración a otra con lo que se **intensifica el proceso de exploración**, dejando lugar a las LS para ocupar la explotación. Con las versiones reducidas como L-SHADE[49], que reducen la población de forma lineal, se perdería la característica exploratoria del algoritmo de la que se precisa en este ámbito y es por ello que se opta por SHADE frente a L-SHADE en esta propuesta.

   Cabe destacar las tres estrategias de SHADE, **mutación**, **cruce** y **selección**, donde cada una se desarrolla como sigue:

**Algorithm 2** : SHADEILS

```
poblacion ← rand(dimension, tam_poblacion)
sol_inicial ← (ub + lb)/2
mejor_actual ← LS(sol_inicial)
mejor_solucion ← mejor_actual
while (totalevals < maxevals) do
   mejor_actual ← SHADE(poblacion, mejor_actual) [1]
   anterior ← mejor_actual . fitness
   mejora ← anterior - mejor_actual . fitness
   Elegir el método de LS a aplicar en esta iteracion [2]
   mejor_actual ← LS(poblacion,mejor_actual)
   Actualizar probabilidad de aplicación de LS para la siguiente iteracion

   // Actualizar la mejor solucion
   if mejor_actual es mejor que mejor_solucion then
      mejor_solucion ← mejor_actual
   end if
   // Mecanismo de reinicio
   if Es necesario reiniciar then
      Reiniciar y actualizar mejor_actual [3]
   end if
end while
```

- **Mutación**: descrita por la ecuación

$$v_i = x_i + F_i \cdot (x_{pbest} - x_i) + F_i \cdot (x_r - a_{r2}) \tag{5.4}$$

  donde $F_i$ se obtiene de forma aleatoria de una distribución normal de media $F_{meanK}$, $x_{pbest}$ es un solución elegido aleatoriamente de los $p$ mejores, $x_{r1}$ es un individuo aleatorio de la población y $a_{r2}$ es un individuo aleatorio de la Poblacion $\cup$ A, donde A es un archivo poblacional externo con una series de **vectores padre** que han producido mejores **vectores soluciones**[38].

- **Cruce**: donde la probabilidad de cruce $CR$ se obtiene al igual que F, de una distribución normal de media $CR_{meanK}$, siendo ésta una de las probabilidades de cruce *alojadas en memoria* de forma interna por el algoritmo según un parámetro que se especifica al inicio del mismo, lo que añade más diversidad al algoritmo; se muestra en la siguiente ecuación.

$$u_i = \begin{cases} v_i \textbf{ if } rand[0,1] < CR_i \\ x_i \textbf{ otherwise} \end{cases} \tag{5.5}$$

- **Selección**: el gen que se selecciona es $u_i$ si mejora el valor fitness de $x_i^t$, o el mismo en caso contrario.

2. **Selección de la búsqueda local a aplicar**: la elección de qué LS ejecutar se basa en una primera ejecución de **ambas búsquedas locales**, tras las cuales se calcula un **ratio de mejora** de cada una con respecto a la mejor solución encontrada hasta el momento siguiendo la ecuación:

$$Ratio_{LS_M} = \frac{Fitness_{antes_{LS}} - Fitness_{despues_{LS}}}{Fitness_{antes_{LS}}} \tag{5.6}$$

   A partir de aquí, en cada iteración se aplicará la LS que **mayor ratio anterior** tenga. A diferencia de IHDELS, donde se selecciona la LS con **mejor valor medio tras las iteraciones**, elegir la LS con **mayor ratio es muy fácil y mucho más rápido** para detectar el bajo rendimiento de una técnica, dado que de la otra manera pueden pasar muchas iteraciones antes de descubrir que la técnica no optimiza de forma adecuada. Así, por ejemplo, si el espacio de soluciones sufre rotaciones, MTS-LS1 gastaría demasiadas iteraciones antes que el algoritmo se de cuenta de que **debería** aplicar L-BFGS-B.

3. **Mecanismo de reinicio de la población**: teniendo en cuenta los problemas de optimización continua, donde es común que un algoritmo mejore las soluciones de una iteración a otra aunque sea por **factores muy pequeños**, los mecanismos de reinicio habituales son aplicados cuando no se consigue **ninguna mejora** durante toda la iteración, hecho que no es muy frecuente que ocurra, lo que tampoco aporta superiores resultados.

   SHADEILS opta por un mecanismo que contemple estos casos, de forma que el reinicio se hará efectivo únicamente cuando transcurran **tres (3) iteraciones completas** donde el **ratio de mejora** tras aplicar DE y LS **no supere un 5 %**. Se selecciona una solución *sol* aleatoriamente y se le introduce una **alteración aleatoria** a sus componentes con una media $\in [0, 0.1 \cdot (b - a)]$, donde $(a, b)$ representa las cotas del espacio de búsqueda. Posteriormente se reinicia toda la población del DE y los parámetros de las LS vuelven a sus valores iniciales.

Queda descrito el algoritmo SHADEILS, con todas sus componentes principales y todos los aspectos relativos a su implementación. En el siguiente capítulo, cuando se proponga el diseño experimental, los valores utilizados en la experimentación referentes a cada algoritmo serán detallados con claridad para garantizar que los experimentos sean completos, claros y repetibles.

## 5.4. MLSHADE-SPA

La cuarta propuesta elegida como fuente de estudio se trata de una técnica peculiar, propuesta durante el reciente **IEEE WCCI 2018: Special Session and Competition on Large-Scale Global Optimization**[3], que combina **cuatro algoritmos distintos** a través del framework CC[41] para conseguir un balance exploración-explotación flexible y robusto cuyo funcionamiento sea independiente de la naturaleza del problema. Así, ***LSHADE-SPA Memetic Framework for Solving Large Scale Problems*** o **MLSHADE-SPA**[36] obtiene resultados muy competitivos en la **Special Session and Competition on LSGO**, lo que le otorga la segunda posición en la misma.

A grandes rasgos, se trata de una propuesta innovadora que utiliza la Cooperación Co-evolutiva[41] para organizar y dirigir cuatro técnicas que se encargan de mantener un balance adecuado entre exploración y explotación, sin realizar ninguna suposición previa acerca de la naturaleza del problema y realizando las particiones de forma aleatoria. El proceso de exploración está liderado por tres algoritmos DE, cada uno con su particular funcionamiento, y son **LSHADE-SPA**[50], **ANDE**[51] y **EADE**[52]. La explotación es llevada a cabo principalmente por una versión modificada del algoritmo **MTS**[29], diseñada específicamente para esta publicación.

La selección de esta cuarta técnica está principalmente respaldada por el uso del enfoque **divide y vencerás**, dado que ninguna técnica elegida anteriormente utilizaba este planteamiento. La potencia que ofrece el framework CC[41] cuando tiene a su disposición agentes robustos y eficaces, puede llegar a obtener **resultados tanto o incluso más competitivos** que los principales algoritmos de referencia aplicados de forma separada. Naturalmente, se precisa de una descomposición que, en este caso, al ser **aleatoria y distinta para cada generación**, consigue aumentar las capacidades individuales de cada técnica, potenciando aún más los resultados.

Adicionalmente, el empleo de técnicas *novel* de **mutación y adaptación de parámetros** en cada uno de los tres algoritmos que se encargan de la exploración, hace que la riqueza de este proceso se vea beneficiada directamente por las **posibilidades de explotación que se añaden de forma paralela durante la exploración** que conducen los operadores. Finalmente, el empleo de una versión modificada de la búsqueda local por múltiples trayectorias (MTS) permite utilizar **toda la capacidad de la técnica** utilizando todos los subprocesos de búsqueda.

A continuación, se detallan las 3 componentes utilizadas en el proceso de exploración, así como las modificaciones realizadas en la técnica de explotación. La combinación de todos estos agentes se verá retratado en el pseudocódigo del algoritmo para facilitar su comprensión estructural.

Este framework memético está compuesto por dos partes bien diferenciadas: las técnicas evolutivas encargadas de la exploración y la técnica de múltiples trayectorias para la explotación. Estos agentes están coordinados por el framework de cooperación co-evolutiva, conformando la totalidad de la propuesta principal. Se hace imprescindible por tanto desgranar el funcionamiento de todas las técnicas para clarificar los atributos que aporta cada una de cara a la obtención de soluciones. Se detalla el pseudocódigo del algoritmo en paralelo con las técnicas.

1. **LSHADE-SPA**: desarrollada para el **CEC2017**[50] es una técnica que combina la variante de **SHADE**[38] que implementa una r**educción lineal del tamaño de la población** llamada **LSHADE**[49] con una **semi-adaptación en dos etapas** de los parámetros del algoritmo, principalmente el factor de escalado **F** y la probabilidad de cruce **Cr**. Esta descripción se centrará en los elementos particulares de la implementación de LSHADE-SPA, donde se describen los procesos que conforman la propuesta.

   La población se inicializa de forma aleatoria entre las cotas inferior y superior del espacio de búsqueda. Posteriormente, se utiliza el operador *current-to-pbest/1* propuesto en [53] para generar el vector de mutación, estando guiado mayoritariamente por una de las $p$ mejores soluciones, donde $p \in (0, 1]$ y otros dos individuos seleccionados aleatoriamente. El *trial vector* se genera mezclando el *target vector* con el **vector mutación** siguiendo una probabilidad de cruce **Cr** y la selección es guiada por la calidad de la solución obtenida.

   A partir de este punto, entra en escena la modificación introducida por LSHADE: se utiliza una **reducción lineal del tamaño de la población** para mejorar el rendimiento de SHADE. Esta reducción se dice lineal debido a que es una función del número de evaluaciones de la función objetivo, y sigue la siguiente ecuación donde $N^{init}$ representa el tamaño inicial de la población, $N^{min}$ es el mínimo de individuos con el que puede trabajar un DE, $NFE$ representa la cantidad actual de evaluaciones de la función objetivo y $MAX_{NFE}$ el máximo de evaluaciones:

   $$N_{G+1} = round[(\frac{N^{min} - N^{init}}{MAX_{NFE}}) \cdot NFE + N^{init}] \tag{5.7}$$

   El proceso de semi-adaptación de parámetros también juega un papel fundamental en este algoritmo. Se aplica en dos etapas que se detallan a continuación, y segun se indica en [50], obtiene mejores resultados que las técnicas **aleatorias**, que no son capaces de resolver eficientemente una gran variedad de problemas al **no estar acotada**

**la dirección** seguida en el espacio de búsqueda, y que las técnicas de **adaptación completa o auto-adaptadas**, dado que no existen garantías de prevenir estancamientos u óptimos locales debido a la tendencia de mantenerse en un espacio de búsqueda limitado durante muchas generaciones. Para conocer en mayor profundidad este algoritmo, se remite al lector a [50].

- **Etapa 1**: aplicada durante la primera mitad de la búsqueda, $nfes < max_{nfes}/2$, se adapta la probabilidad de cruce **Cr** mientras que $F$ se genera siguiendo una distribución uniforme. Durante la fase de **SPA** (Semi-Parameter Adaptation), cada individuo tiene su $F_i$ y $C_r$ donde se generan siguiendo las ecuaciones:

$$\begin{aligned} F_i &= 0.45 + 0.1 \cdot rand\ ; \\ Cr_i &= randn(Mcr_i, 0.1) \end{aligned} \tag{5.8}$$

  donde $Mcr_i$ es una posición aleatoria de una *memoria* de tamaño $h$ donde se encuentran las medias de $Crs$ de anteriores generaciones que han producido buenos individuos, donde inicialmente tienen valor 0.5 y al final de cada generación una posición aleatoria se actualiza con el valor medio de las $Cr_i$ que han producido mejores individuos.
- **Etapa 2**: se concentrarán los esfuerzos en adaptar **F**, que se genera siguiendo una distribución de Cauchy.

$$F_i = randc(MF_i, \sigma) \tag{5.9}$$

  donde $\sigma = 0.1$ es la desviación estándar de la distribución y $MF_i$ representa el mismo concepto que $Cr_i$ pero para el factor de escalado. La memoria de $MF_i$ comiena con los valores de $F_i$ de las últimas 5 generaciones de la etapa 1 y se actualiza nuevamente una posición aleatoria mediante la media Lehmer.

2. **EADE**: este *enhanced adaptive DE algorithm*[52] introduce un **operador de mutación novel** basado en **tres vectores solución seleccionados de forma aleatoria** de porciones de la población total. Este operador se mezcla con el **DE/rand/1/bin** clásico con probabilidad de aplicación 0.5 para cada uno, ayudando a mantener un equilibrio entre la exploración global y la explotación local del algoritmo. Además, se propone un **esquema auto-adaptativo del parámetro CR** que basa su funcionamiento en la historia evolutiva reciente.

   - El esquema del nuevo operador se basa en utilizar **tres vectores** para generar el vector mutación, donde se utiliza **dos vectores**

$x_{p\_best}$ **y** $x_{p\_worst}$ **elegidos aleatoriamente** de la parte **superior e inferior** del 100p % de los individuos de la población de tamaño NP, mientras que un **tercer vector** $x_r$ se selecciona de la **mitad restante de individuos** (NP-2(100p %)), con $p \in (0, 1]$. La generación del vector de mutación se rige por la ecuación:

$$v_i^{G+1} = x_r^G + F1 \cdot (x_{p\_best}^G - x_r^G) + F2 \cdot (x_r^G - x_{p\_worst}^G) \quad (5.10)$$

Tanto F1 como F2 son los factores de escalado de la mutación que se generan siguiendo una **distribución uniforme en (0,1)**. La interpretación de este operador es simple: cada vector mutación, de cierta forma, **aprende de ambas vertientes buena y mala** de los individuos disponibles. Siguiendo estas pautas, se pueden evitar **estancamientos prematuros en óptimos locales** ya que no se sigue siempre el mismo camino de la mejor solución y además, concentrar los **esfuerzos de explotación en subespacios prometedores** es posible evitando las malas soluciones, factor al que contribuye el tercer sumando de la ecuación anterior.

- En cuanto al esquema de adaptación de la probabilidad de cruce **Cr**, para cada individuo se calcula un $CR_i$ de una *pool* **A** de valores CR que cambian durante un **periodo de aprendizaje LP**, donde $LP = 10\,\%$ de GEN, donde G es la generación actual y GEN el total de generaciones. *CR_Flag_List[i]* representa una lista con valores 0 y 1, donde 0 indica que no se ha mejorado un individuo y 1 el caso contrario; otra *failure_counter_list[i]* cuenta la cantidad de veces que no se ha mejorado ese individuo en concreto tras el proceso de aprendizaje, hasta un máximo de 20. Una última *CR_Ratio_List[k]* aloja la mejora relativa entre los vectores **target y trial** con respecto a cada valor k de la pool A en la generación G. A grandes rasgos, según la **generación G** en la que se encuentre el algoritmo y si ***f(target vector)*** > ***f(trial vector)*** , **el valor CR se elegirá de una pool A cada vez mayor** (empezando con un único valor 0.05 y añadiendo más valores en incrementos de 0.05 la primera vez y 0.1 a partir de entonces, hasta un máximo de 0.95) si se cumple $G <= LP$ y la segunda condición, o se elegirá el valor de **CR** con mayor ratio de la lista anterior si el trial vector es mejor que el target vector. Se remite finalmente al lector a [51] si se quiere profundizar en este operador.

3. **ANDE**: esta técnica introduce nuevamente una modificación en el **operador de mutación**, donde se genera una **mutación triangular**

**basada en una combinación convexa de una tripleta de vectores** elegidos aleatoriamente. Como en la técnica anterior, también se combina con la estrategia **DE/rand/1/bin** pero con una probabilidad de 2/3, al poseer ambas capacidades de exploración y explotación. También introduce un esquema de adaptación de la probabilidad de cruce **Cr** muy similar a la de la técnica anterior que se discutirá brevemente en los siguientes puntos.

- **Esquema de mutación triangular**: con el objetivo de agilizar la convergencia del algoritmo manteniendo el balance exploración-explotación se introduce este enfoque, donde se eligen tres vectores aleatoriamente que serán clasificados según su fitness como *best*, *better* y *worst*, formando la combinación convexa que deriva en la generación de las mutaciones siguiendo la ecuacion:

$$v_i^{G+1} = \overline{x}_c^G + F1 \cdot (x_{best}^G - x_{better}^G) + F2 \cdot (x_{best}^G - x_{worst}^G) \\ +F3 \cdot (x_{better}^G - x_{worst}^G) \tag{5.11}$$

  donde $\overline{x}_c^G$ representa el **vector de combinación convexa** y donde los factores de escalado se obtienen de una distribución uniforme en (0,1). Este vector de combinación se rige por la ecuación:

$$\begin{gathered} \overline{x}_c^G = \omega_1 \cdot x_{best} + \omega_2 \cdot x_{better} + \omega_2 \cdot x_{worst} \\ | \ \forall \omega_i \geq 0 \ y \ \sum_{i=1}^{3} \omega_i = 1 \ ; \ ademas \\ \omega_i = p_i / \sum_{i=1}^{3} p_i \mid i = [1,2,3], p_1 = 1, p_2 = rand(0.75, 1), \\ p_3 = rand(0.75, p_2) \end{gathered} \tag{5.12}$$

  Mediante este enfoque se quiere dotar de un **comportamiento de gradiente** al operador, dirigiendo las perturbaciones en la dirección (mayoritariamente) de la zona más prometedora, al realizarse desde la **peor a la mejor solución** aleatoria, aumentando la capacidad exploratoria global del algoritmo a la vez que las subregiones prometedoras del espacio de búsqueda son explotadas. Para terminar, los factores de escalado se generan a partir de distribuciones uniformes en (0,1) y la adaptación del parámetro **Cr** se realiza de la misma forma que en **EADE**, por lo que se remite al lector a [51] si se quiere profundizar en el algoritmo o la adaptación en cuestión.

4. **MMTS**: versión modificada del algoritmo MTS donde la principal diferencia radica en la **selección de los agentes de partida**. En vez

de generarse tal y como se propone en [29], se eligen estos agentes de **la población de entrada** tras la ejecución iterativa de los 3 anteriores algoritmos poblaciones, donde se seleccionan por un procedimiento de *limpieza*.

A continuación, se detalla el pseudocódigo del algoritmo en cuestión.

**Algorithm 3** : ***Evolve_MLSHADESPA()***

[Pop, Fit] = SPA(SPA_nfes,Pop, Fit)
nfes = nfes+SPA_nfes; Grupo_num = 3
CC_Ind = dividir_dimensiones(Grupo_num, D)
Alg_Ind = find(CC_Ind==1)
[Pop, Fit, imp(1)] = SPA_CC(CC_nfes(1), Pop, Fit, Alg_Ind); nfes+=CC_nfes(1)
Alg_Ind = find(CC_Ind==2)
[Pop, Fit, imp(2)] = ANDE_CC(CC_nfes(2), Pop, Fit, Alg_Ind); nfes+=CC_nfes(2)
Alg_Ind = find(CC_Ind==3)
[Pop, Fit, imp(3)] = EADE_CC(CC_nfes(3), Pop, Fit, Alg_Ind); nfes+=CC_nfes(3)
[Pop, Fit] = MMTS(MMTS_nfes, Pop, Fit); nfes+=MMTS_nfes
Calcular imp y $N_{r+1}$

**Algorithm 4** : MLSHADE-SPA

$N_r = N^{init}$ ; $nfes = 0$ ; $cc\_n = 50$
Inicializar poblacion $P_r$ aleatoriamente
Inicializar parametros de SPA, EADE, ANDE y MMTS
round_nfes = max_nfes/cc_n ; flag = 0
**while** $nfes < max\_nfes$ **do**
  EA_nfes = MMTS_nfes= round(round_nfes/2)
  SPA_nfes = round(EA_nfes/2)
  **if** $flag == 0$ **then**
    SPA_CC_nfes=ANDE_CC_nfes=EADE_CC_nfes=round(EA_nfes/6); flag = 1
  **else**
    Recalcular CC_nfes
  **end if**
  ***Evolve_MLSHADESPA()***
  **if** $N_r < N_{r+1}$ **then**
    Ordenar individuos de P basado en fitness y eliminar los $N_r - N_{r+1}$ individuos más abajo
  **end if**
**end while**

## 5.5. Differential Grouping 2

La quinta y última propuesta de algoritmo sometida a estudio en este trabajo se trata de una técnica particularmente distinta a las anteriores. Esta técnica denominada ***DG2: A Faster and More Accurate Differential Grouping for Large-Scale Black-Box Optimization***[45] es una versión mejorada del algoritmo de descomposición de variables **DG** de la publicación ***Cooperative Co-Evolution With Differential Grouping for Large Scale Optimization*** [42], donde se intenta solventar una serie problemas relacionados con la forma en la que se aborda el problema de la **dimensionalidad** y de la **interacción de variables**, inconvenientes típicos de los problemas de *large-scale global optimization*.

La optimalidad de las soluciones que un algoritmo obtiene están intimamente relacionadas con la **interacción de las variables en el problema**. Es por esta razón que surgen este tipo de técnicas, donde se pretende identificar las variables con interacción a través de un estudio exhaustivo de **pares de variables** que es incluso capaz de detectar **componentes solapadas** de una función objetivo (representación de un problema de optimización), para que un enfoque de tipo **divide y vencerás** pueda resolver de la mejor forma posible el problema, centrándose en aquellas conjuntos de variables que, de cierta forma, aportan más a la **calidad del fitness** que otras.

Elegir esta última propuesta se justifica debido a la posibilidad que ofrece de atacar el **problema de la dimensionalidad** con una herramienta distinta a las anteriores meteheurísticas. Descomponer el problema en subproblemas optimales más pequeños que son optimizados de forma iterativa disminuye considerablemente la **complejidad del espacio de soluciones**. A través de la **cooperación co-evolutiva**, una descomposición óptima del problema que se aplica de forma iterativa a una metaheurística potente puede arrojar resultados mucho más competitivos que de forma individual.

Este tipo de técnica en particular, dados sus orígenes y la familia de algoritmos a la que pertenece, se encuentra actualmente en la cúspide debido a que no sólo es capaz de descomponer **problemas de dimensionalidad superior a 1000** con bastante precisión sino que además **resuelve todos los inconvenientes y vulnerabilidades de su generación**: tiene un mecanismo **renovado para identificar la interacción** que requiere un número considerablemente menor de evaluaciones de la función objetivo que sus predecesores y además es completamente *parameter-free*, ya que el único **parámetro** que necesita **especificación previa** en otras propuestas de su familia y que es decisivo para una descomposición óptima, DG2 lo calcula de forma **automática e independiente para cada variable**, lo que naturalmente, aumenta la efectividad y permite un ajuste más adecuado a cualquier tipo de problema.

Partiendo de estos preceptos se procede a especificar el algoritmo en su totalidad y detallar sus componentes principales, destacando los aspectos relevantes que respaldan la elección de esta técnica. Al término de esta ilustración se precisarán ciertas condiciones a cumplir con el algoritmo que a su vez determinarán el grado de implicación de éste a lo largo del proceso de experimentación.

DG2, como técnica de tipo **divide y vencerás**, descompone el problema en una serie de subproblemas más pequeños y simples, con el objetivo de optimizarlos posteriormente de forma iterativa mediante un algoritmo de co-evolución cooperativa y distintos agentes. Para llevar a cabo esta tarea, una función objetivo $f$ se debe descomponer de tal forma que se **minimice la interacción entre las componentes resultantes**; esto es una tarea realmente compleja para funciones de tipo black-box, donde no se tiene **información acerca de la interacción de variables**.

La propuesta elegida está destinada a solventar los inconvenientes que tenía su predecesora DG, principalmente: alto coste computacional para funciones separables, incapacidad de detectar componentes solapadas, sensibilidad a errores de redondeo y dependencia de un parámetro umbral. DG2 **mejora la eficiencia**, requiriendo de tan sólo la **mitad de las evaluaciones de la función objetivo que DG** para funciones separables, que es el caso que más consume. Además, aumenta la capacidad de agrupamiento debido a la **estimación automática del umbral para cada variable**, lo que otorga mayor sensibilidad frente a interacciones débiles.

Para que la descomposición de variables sea óptima, se requiere **minimizar la interacción entre las componentes resultantes**, donde el grado de separabilidad de la función juega un papel fundamental. Se dice que una variable $x_i$ es **separable** o que **no interactúa** con ninguna otra variable[42] sí y sólo si:

$$arg\ minf(x_1, \dots, x_n) = (arg\ minf(x_1, \dots),\ \dots\ arg\ minf(\dots, x_n)) \quad (5.13)$$

Por tanto, una función será **separable** si es posible alcanzar su óptimo global únicamente optimizando **una dimensión a la vez**, independientemente del valor que tomen las demás variables. A partir de esta definición surgen otras definiciones de separabilidad: una función se dice **parcialmente separable** con $m$ componentes independientes sí y sólo si:

$$arg\ minf(x) = (arg\ minf(x_1, \dots),\ \dots\ arg\ minf(\dots, x_m)) \quad (5.14)$$

donde $\mathbf{x} = (x_1, \dots, x_n)^T$, $x_1, \dots, x_m$ son vectores solución de $\mathbf{x}$ y $2 \leq m \leq n$

Finalmente se define una función **parcialmente aditivamente separable** como sigue:

$$f(x) = \sum_{i=1}^{m} f_i(x_i), m > 1 \tag{5.15}$$

donde $f_i(\cdot)$ es una subfunción no-separable y $m$ el número de subcomponentes independientes de $f$ como se describe en la ecuación 5.14.

Partiendo de estas definiciones se proponen distintos tipos de algoritmos de descomposición que mezclan diversas heurísticas para llevar a cabo la agrupación, sin embargo obtienen bajos índices de agrupación. Dentro de estos algoritmos está DG[42], la base de DG2, donde su rendimiento de agrupación es muy superior a las demás propuestas, y esto se debe al teorema enunciado a continuación.

**Teorema 1**: sea $f(x)$ una función parcialmente aditivamente separable (ecuación 5.15). $\forall a,\ b_1 \neq b_2,\ \delta \neq 0$, las variables $x_p$ y $x_q$ interactúan si se cumple la siguiente condición:

$$\Delta_{\delta,x_p}[f](x) \mid_{x_p=a,x_q=b_1} \neq \Delta_{\delta,x_p}[f](x) \mid_{x_p=a,x_q=b_2} \tag{5.16}$$

donde $\Delta_{\delta,x_p}[f](x) = f(..., x_p + \delta, ...) - f(..., x_p, ...)$ hace referencia a la *forward difference*[42] de $f$ con respecto a $x_p$ y un intervalo $\delta$. Si se evalúa la ecuación 5.16 con dos valores distintos de $x_q$ dando distintos resultados, entonces se dice que $x_p$ **y** $x_q$ **interactúan**. Se denota a la parte izquierda de la ecuación anterior como $\Delta^{(1)}$ y a la derecha como $\Delta^{(2)}$, con el único objetivo de abreviar.

Sin embargo, la comprobación $\Delta^{(1)} \neq \Delta^{(2)} \implies \mid \Delta^{(1)} - \Delta^{(2)} \mid \neq 0$ es prácticamente imposible de realizar a nivel computacional debido a la precisión de los número expresados en coma flotante. Por esta razón se introduce un término $\epsilon$ para restringir estos errores, quedando finalmente la ecuación: $\lambda = \mid \Delta^{(1)} - \Delta^{(2)} \mid > \epsilon$. Este es el parámetro que DG2 calcula de forma automática para cada par de variables y que sus antecesores DG definen de forma global para todas las variables. Para diferenciar DG2 de sus predecesores se muestra el pseudocódigo del algoritmo, donde las tres partes principales se discuten seguidamente.

1. **Formación de *raw interaction structure matrix***: la matriz $\Lambda$ contiene la cantidad $\mid \Delta^{(1)} - \Delta^{(2)} \mid$ para cada par de variables y es computada en la función **ISM**. Aquí es imprescindible prestar atención a la **formulación del algoritmo *ISM*** que se puede ver en [45], dado que para una función *n-dimensional* el número total de interacciones es $\binom{n}{2}$.

**Algorithm 5** : $(g, x_1, ..., x_g, x_{sep}, \Gamma) = DG2(f, n, \overline{x}, \underline{x})$

$(\Lambda, F, \hat{f}, f_{base}, \Gamma) = ISM(f, n, \overline{x}, \underline{x})$
$\Theta = DSM(\Lambda, F, \hat{f}, f_{base}, \Gamma)$
$(k, y_1, ..., y_k) = ConnComp(\Theta)$
$x_{sep} =$ , $g = 0$
**for** $i = 1 \rightarrow k$ **do**
  **if** $|y_i| = 1$ **then**
    $x_{sep} = x_{sep} \cup y_i$
  **else**
    $g = g + 1$; $x_g = y_i$
  **end if**
**end for**

Según el teorema 1, se requieren de 4 evaluaciones de la función objetivo, lo que implica un total de $4 \cdot \binom{n}{2} = 2n(n-1)$ evaluaciones de la función objetivo. Para reducir de forma considerable el número de evaluaciones de la función objetivo y por consiguiente el tiempo de respuesta, se realiza una **agrupación de los 4 puntos necesarios** para el cálculo de $| \Delta^{(1)} - \Delta^{(2)} |$ mediante un proceso de **generalización para cualquier número de variables**, lo que finalmente, tras eliminar las evaluaciones redundantes de la interacción $x_i - x_j$ descrita en [45], resulta en un número inferior de evaluaciones: $\frac{n(n+1)}{2} + 1$.

2. **Determinación del parámetro** $\epsilon$: parámetro para transformar $\Lambda$ en $\Theta$, la *design structure matrix* donde $\Theta = 1 \iff \Lambda_{i,j} > \epsilon$, 0 en otro caso. El cálculo de $\epsilon$ es informado: se obtiene a partir de estimar la magnitud de la función $f$ y de los valores de $\Lambda$.

   Establecer con exactitud el parámetro $\epsilon$ es complejo: si se establece a 0, cualquier diferencia positiva entre $\Delta^{(1)}$ y $\Delta^{(2)}$ implicaría interacción, pero asumir esto no es correcto debido a los **errores de redondeo y de escala** de la representación de los puntos flotantes, que pueden arrojar valores $\lambda$ que indiquen interacción en variables separables, lo cual no es deseable.

   Se estiman por tanto dos cotas ($e_{inf}$),la cota inferior más grande, y ($e_{sup}$) - la cota superior más pequeña - de $\epsilon$ para cada par de variables, nuevamente utilizando la información de $\lambda = | \Delta^{(1)} - \Delta^{(2)} |$, por lo que ahora se considera **interacción** si $\lambda > e_{sup}$ y **separable** si $\lambda < e_{inf}$.

   El algoritmo se suple del error de representación computacional de un número $x$ a través de una función $f(x) = x(1 + \delta)$, que indica que el error de representación crece con x. Se basa además en los errores que acarrean las operaciones en coma flotante y la cantidad de estas para calcular las cotas anteriormente mencionadas.

El cálculo de ($e_{inf}$) estará guiado por los **errores que se cometen al aplicar DG** en el cálculo de la diferencia $\lambda = \mid \Delta^{(1)} - \Delta^{(2)} \mid$, siguiendo la ecuación descrita en [45], que será obviada para no extender innecesariamente este documento. La mínima cota superior ($e_{sup}$), se calcula a partir del error que acarrean las **operaciones de coma flotante y el número de éstas**, estimando este último parámetro asumiendo **complejidad lineal en el número de operaciones de *coma flotante***, asegurando un ajuste más estricto para esta cota. Se remite al lector nuevamente a [45] si se quiere conocer en profundidad las ecuaciones utilizadas para el cálculo de las cotas.

3. **Descomposición de variables en grupos no-separables**: identificar las componentes conexas del grafo a través de la matriz $\Theta$ de adyacencia.

Las implementaciones concretas de los algoritmos **ISM** y **DSM** se recogen en la publicación anteriormente mencionada [45] y se remite al lector a ésta para evitar engrosar esta descripción sin necesidad alguna. Queda por tanto detallada la última propuesta de algoritmo perteneciente a este estudio. En el siguiente capítulo se lleva a cabo la implementación en cuanto a adaptación de los algoritmos se refiere, atendiendo y resolviendo todos los inconvenientes que se derivan de esta tarea.

# Capítulo 6

# Implementación: adaptación al problema EEG

Las tareas que componen este capítulo incluyen la obtención de las **implementaciones de las propuestas elegidas**, derivadas de las necesidades anteriormente expuestas: garantías de implementaciones *seamless* (sin fisuras), optimizadas lo mejor posible y probadas en distintos entornos frente a una gran variedad de problemas. El proceso de implementación que da nombre a este capítulo hace referencia a la **adaptación de los algoritmos**, tarea indispensable para que estos sean capaces de procesar y trabajar con la función objetivo del problema del EEG.

La primera sección de este capítulo detalla la especificación a nivel de pseudocódigo de la función objetivo del problema del EEG. Conocer la especificación de la misma permitirá una adaptación más precisa con cada algoritmo, dado que se tienen en cuenta las operaciones necesarias a realizar y se puede ajustar más adecuadamente a cada implementación en particular.

En la segunda sección, se mostrará el **proceso principal de adaptación** de la función objetivo a una implementación en concreto. Esta adaptación ocurre en **dos vertientes**, donde una **primera adaptación completa** se realiza en el **ordenador portátil del alumno** para comprobar su **estado de funcionamiento** y descartar la existencia de errores derivados del proceso.

La segunda vertiente, de extensión previsiblemente más reducida, comprende los procedimientos llevados a cabo para acondicionar aquellas propuestas que lo requieran de cara al proceso experimental que será conducido en los nodos del **Cluster Hércules**, donde la arquitectura del clúster puede ser determinante en cuanto a la facilidad de incrustar las implementaciones en el mismo y que funcionen correctamente. Se detallan a continuación ambas vertientes en una única sección.

## 6.1. Función objetivo del problema EEG

La función objetivo del problema propuesto se detalla a continuación, y será el capítulo siguiente donde se discutan las diferencias esenciales entre las funciones que normalmente incluyen los benchmarks de LSGO y la función objetivo de este problema, en aras de facilitar la especificación del proceso experimental que se detallará también en el siguiente capítulo.

Se detallan, en este orden, el cálculo del **coeficiente de correlación de Pearson** de la ecuación 3.5 y de la función $f_1$ representada por la ecuación 3.6. Estos pseudocódigos incluyen las operaciones básicas derivadas de las ecuaciones anteriores, operaciones que no es posible identificar a simple vista a partir de las ecuaciones, por lo que el autor del benchmark proporciona el código fuente en C++ en [9] de una implementación particular de esta función y que será en la que se base todo el proceso de adaptación de los algoritmos.

**Algorithm 6** : $COR = Covar(A, B)$

1: $partial = null$ , $COR = null$
2: **for** cada elemento $A_i$ de $A$ **do**
3:  **for** cada elemento $B_i$ de $B$ **do**
4:  $A1 \leftarrow [mean(A_i), stdev(A_i)]$ ; $B1 \leftarrow [mean(B_i), stdev(B_i)]$
5:  $sum = 0$ ; $\sigma = A1(1) \cdot B1(1)$
6:  **if** $\sigma > 0.00001$ **then**
7:  **for** cada elemento $i$ de $A$ **do**
8:  $temp1 = A(i) - A1(0)$ ; $temp2 = B(i) - B1(0)$
9:  $sum = sum + (temp1 \cdot temp2)$
10:  **end for**
11:  $sum = \frac{sum}{size(A_i) \cdot \sigma}$ ; $partial \leftarrow push(sum)$
12:  **else**
13:  $partial \leftarrow push(0)$
14:  **end if**
15:  **end for**
16:  $COR \leftarrow push(partial)$ , $partial = null$
17: **end for**
18: $return\ COR$

Finalmente, la función objetivo del algoritmo 8 se vale de estos cálculos intermedios para efectuar el cálculo de la función conjunta $f1 + f2$ de la ecuación 3.8.

**Algorithm 7** : $f1 = Diagonals(COR)$

$diag = noDiag = 0$
**for** $\forall i \mid i < size(COR)$ **do**
  **for** $\forall j \mid j < size(COR)$ **do**
    **if** $i == j$ **then**
      $diag = diag + (1 - C_{ii})^2$
    **else**
      $noDiag = noDiag + (C_{ij})^2$
    **end if**
  **end for**
**end for**
$f1 = (diag/size(COR)) + (noDiag/size(COR)/size(COR) - 1)$
return $f1$

**Algorithm 8** : $fitness = EEG\_Objective(X_i)$

// Sea $X_i$ un vector solución de tamaño $(n \times m)$, donde $n$ es el número
// de series de tiempo (fuentes de entrada)y $m$ el tamaño de cada serie.

$S1 = matrix(X_i) \mid size(X_i) = n \times m$ y $dim(S1) = n \times m$
$X1 = A \times S1 \mid A$ es la matriz de combinación de la ecuación 3.3
$COR1 = Covar(X1, X) \mid X$ es la matriz combinada de señales de 3.3
$f2 = \sum_i \sum_j (S_{ij} - S1_{ij})^2 \mid S$ es la matriz de señales original
$fitness = Diagonals(COR1) + (f2/N \times M)$

Representada la función objetivo, la siguiente sección se encarga de la adaptación de los algoritmos al procesamiento de esta función, según las particularidades propias del diseño elegida.

## 6.2. Adaptación de los algoritmos

Esta sección contiene toda la información relevante acerca de la **adaptación** llevada a cabo para cada uno de los algoritmos seleccionados para este estudio. Se destacan únicamente los elementos de implementación que aportan información del proceso a alto nivel de abstracción, donde detalles como pueden ser las órdenes de configuración específicas o la creación y/o modificación de ficheros varios y su contenido explícito se descartan porque, a efectos prácticos, no aportan información adicional en cuanto a lo que se desarrolla en esta sección.

Naturalmente, el proceso de adaptación tiene que considerarse superado con éxito antes de realizar cualquier tipo de experimento. Para garantizar la correcta adaptación de los algoritmos, el alumno se vale de una herramien-

ta que autodenomina **Validador**, que no es más que la **función objetivo pura**, sin ningun tipo de capa ni utilizando librerías externas, incrustada en un pequeño programa en C++ que permite comprobar que la solución ofrecida por el algoritmo en cuestión tras el proceso de adaptación, es realmente correcta en términos de **fitness**. Por tanto, cuando se enuncie que una adaptación ha sido **validada**, implicará que ha superado satisfactoriamente el **Validador**. Se indica a continuación el proceso de adaptación de cada algoritmo.

1. **Multiple Offspring Sampling (2011):**

   La implementación del algoritmo **MOS2011**[28, 54] está basada en el framework de **Multiple Offspring Sampling** (MOS)[46] y la librería C++ para componentes de Algoritmos Genéticos **GAlib**[55]. Este algoritmo fue propuesto para el *2011 Special Issue of the Soft Computing Journal* y la implementación disponible está preparada para procesar distintos benchmarks, como los del SOCO2010, CEC2012 o CEC2013. La copia de esta implementación corresponde a un repositorio[54] al que tiene acceso el tutor de este trabajo y que ha compartido con el alumno, con el consentimiento previo del autor de la propuesta.

   A continuación se describe el proceso de adaptación seguido para este algoritmo en particular.

   *a*) La estructura de directorios sobre la que se sustenta el proyecto contiene por separado los ficheros del conjunto de problemas de tipo **benchmark** y los referentes a la implementación **MOS+GAlib**. Los ficheros de funciones están divididos en ficheros fuente (**.cc**) y ficheros de cabeceras (**.h**), siendo el fichero de cabeceras donde se **implementa** el código de la función objetivo, ya que el fichero **.cc** contiene una estructura particular derivada de GAlib que actúa como intermediario entre la función objetivo del fichero **(.h)** y el framework MOS.

   *b*) Conocida la implementación particular, se crean sendos ficheros **eeg_problem** con extensión **.h** y **.cc**. El fichero **eeg_problem.h** contiene la implementación de los algoritmos 6, 7 y 8 y además incluye otro fichero de cabeceras donde se implementa la carga de los ficheros de texto que contienen las matrices **A, S y X**, necesarios para la evaluación de un vector solución, así como de una matriz **S1** proporcionada por el autor como *baseline* que sirve como criterio de parada, aunque en esta propuesta no es utilizada a efectos prácticos. La implementación de la función objetivo sigue todos los estándares que exige GAlib, prestando especial cuidado a los tipos de datos utilizados para no incurrir en problemas de compatibilidad con la implementación.

*c*) El fichero **eeg_problem.cc** envuelve la función objetivo a través de la definición de una función ***objective*** con la sintaxis particular que precisa GAlib, en especial atendiendo a la definición particular de **individuo**, modelizado a través de un tipo de dato concreto propio de **GAlib**. Posee además una función que **define el problema**, donde se indican los valores de cotas del espacio de soluciones, así como la especificación completa del problema en términos de dimensión, función objetivo y elementos poblacionales como el tamaño de la misma y otros factores que se obtienen principalmente de un **fichero de configuración** especificado al lanzar el algoritmo.

*d*) Solventados los problemas derivados de la implementación particular de la propuesta se procede a compilar el proyecto. Se utiliza la herramienta **CMake** para generar los **Makefile** de forma automática, por lo que es preciso añadir los ficheros fuente y de cabeceras al fichero de objetivos **CMakeLists.txt**. Para evitar sobrecargas no deseadas, se compila únicamente el problema del EEG como si fuese parte del benchmark de SOCO2010, pero sin ninguno de los demás problemas de este. Previa a esta compilación se precisa resolver un pequeño porcentaje de errores de la implementación particular, concretamente generados a partir de conflictos con una **sobrecarga** de operadores.

*e*) Compilada la función objetivo y enlazada a la implementación de MOS2011, el siguiente paso consiste en comprobar que el algoritmo es realmente capaz de ejecutar un proceso de optimización guiada por la función objetivo. El alumno se vale de un **script bash** que incluye el autor para lanzar las diferentes ejecuciones. Ajustando la dimensión de cada problema, se comprueba que es posible ejecutar el algoritmo frente al problema del EEG, por lo que el último paso consiste en generar tantos scripts como problemas distintos se tenga, esto es, uno para cada dimensión del problema tanto con ruido como sin este, haciendo un total de 6 scripts, que serán útiles de cara a la fase experimental.

Finalmente, algunas librerías como **Boost**, **OpenMPI** o **Libconfig** precisan de instalación para una ejecución exitosa. A partir de este momento y tras comprobar que se obtiene un resultado sensato y validado, se considera a **MOS2011** preparado para la segunda fase de adaptación. No obstante, se quiere dejar plasmada la circunstancia de que el proceso de adaptación de este algoritmo en particular ha sido sumamente complicado y los pasos que aquí se reflejan tan solo denotan las acciones llevadas a cabo a gran escala, sin entrar en los detalles que son donde han surgido la mayor parte de complicaciones.

2. **Multiple Offspring Sampling (2013)**

   La propuesta e implementación de **MOS2013**[31, 54] vuelve a estar nuevamente basada en el framework MOS[46] y la librería GAlib[55]. La implementación utilizada en este estudio está disponible en el mismo repositorio que la anterior, por lo que se puede consultar en [54]. La estructura de la implementación es muy similar a la de su predecesor, por lo que se precisan de los siguientes pasos para llevar a cabo su adaptación.

   *a*) Se comprueba si existen los mismos errores de implementación que en la versión anterior, para resolverlos si es preciso. Una vez solucionados, se generan nuevamente los ficheros **eeg_problem** esta vez en un directorio propio y separado del conjunto de demás benchmarks. Dada la implementación particular (aunque similar a la anterior), la única diferencia con su predecesor radica en la utilización del tipo de dato **double** frente al **long double** del **MOS2011**, manteniendo el resto de componentes intactas, esto es, tal y como fueron propuestas para la adaptación del algoritmo **MOS2011**.

   *b*) Generados los ficheros del problema EEG, se crea el fichero de objetivos **CMakeLists.txt** para compilar y enlazar la especificación del problema con el algoritmo MOS. Es preciso especificar la versión mínima de **CMake** requerida para evitar errores de compilación o enlazado; se establece a la versión 2.8. Con esta última directiva se consigue compilar con éxito el algoritmo y todas sus componentes.

   *c*) El penúltimo paso es generar los **scripts bash** que se encargarán posteriormente de llevar a cabo la fase experimental en su totalidad, ajustando los parámetros indispensables para la ejecución de cada tipo de problema. Finalmente, el validador ratifica los resultados preliminares obtenidos por el algoritmo, por lo que se considera finalizada la primera fase de adaptación de **MOS2013**.

3. **SHADEILS**

   El tercer prospecto de estudio es **SHADEILS**[35, 56], una implementación donde la totalidad del algoritmo está desarrollada en **Python** y el conjunto de problemas benchmark que ejecuta se encuentra implementado en **C++** pero con **wrappers de Python**, lo que permite casar ambas implementaciones sin ningún tipo de fuga. El código fuente utilizado se encuentra disponible en la página web de la competición del **WCCI CEC 2018**[3]. Los paquetes, módulos y aspectos relevantes de la implementación se describen en el proceso de adaptación que se detalla seguidamente:

*a*) El proyecto en su totalidad está preparado para ejecutar una serie de comandos de un script **install.sh**, donde se crea inicialmente un **entorno virtual** donde se instalarán todos los paquetes y dependencias que el proyecto necesita para funcionar. Esto incluye descargar una versión del software de gestión de paquetes de Python llamado **Pip**[57]. Posteriormente se instalan a través de este gestor los paquetes de *data science* que utiliza el proyecto, tales como **Numpy** o **Scipy**; otro de los requeridos es **Cython**[58], una herramienta que sirve para escribir extensiones del lenguaje **C** o **C++** que sean soportadas por Python, concretamente para el uso de las funciones benchmark.

*b*) Tras comprobar que la versión actual de Python y la del paquete de creación de entornos virtuales **venv** de la que dispone el alumno en su portátil es la adecuada, se procede a instalar todas las dependencias y poner en marcha el entorno virtual para poder ejecutar los scripts de Python correctamente. Debido a errores de incompatibilidad de librerías entre Windows, Linux y MacOS, se requiere modificar el script que **compila las funciones benchmark** indicando que la librería estándar que debe utilizar en este proceso es **libc++** (MacOS) en vez de **libstdc++** (Windows y Linux).

*c*) Como inciso, remarcar que esta incompatibilidad afecta directamente a la puesta en marcha del algoritmo en sistemas operativos MacOS. Por ello, se ha eliminado el paquete de funciones benchmark del CEC2013-LSGO de la lista de **paquetes requeridos** por la implementación, ya que al ejecutar el script de instalación, no se contempla el sistema operativo y surgen errores de compilación. Por esta razón, se ha optado por descargar manualmente el paquete y modificar el script como se indica en el paso anterior, para garantizar la compilación del conjunto de problemas.

*d*) Resueltos una serie de pequeños errores de sintaxis se procede a realizar la adaptación requerida de la función objetivo. El script principal (**shadeils.py**) se duplicará en otro fichero denominado **main_eeg_problem.py** para añadir al mismo los parámetros particulares del problema del EEG, como la dimensionalidad, la existencia de ruido, el total de funciones del problema y el número de evaluaciones a ejecutar.

*e*) El siguiente y último paso consiste en la adaptación de la función objetivo como tal. Para ello, se generan dos ficheros **EEGProblem**, uno de cabecera y el otro fuente. El fichero de cabeceras contiene la definición de las funciones intermedias que se corresponden con los algoritmos 6 y 7, así como la de la función objetivo del algoritmo 8 y las que permiten cargar los ficheros de datos,

y esta vez con la sintaxis nativa de C++ y sus tipos de datos y librerías estándar, sin valerse de ninguna otra librería como GAlib. El fichero fuente implementa las funciones en cuestión y define las restricciones de umbrales superior e inferior, necesarias para acotar el espacio de búsqueda.

*f*) Finalmente, se modifica el formato del fichero donde se redirige la salida del algoritmo, lo que incluye el proceso de convergencia, solución, fitness y la marca de tiempo. Para comprobar que la adaptación ha resultado efectiva, se comprueba con el **validador**, obteniendo un resultado favorable que permite concluir con esta primera fase de adaptación.

4. **MLSHADE-SPA**

   La técnica **MLSHADE-SPA**[36, 56] es la cuarta propuesta de estudio en este trabajo. Los autores de la misma publican su implementación en el lenguaje **MATLAB**, disponible también en la web del WCCI 2018[3], por lo que el alumno hace uso de una licencia de prueba para poder realizar la adaptación pertinente. En este caso concreto, la adaptación requiere implementar la función objetivo en este lenguaje, buscando aprovechar al máximo la representación de los tipos de datos como matrices y las operaciones matriciales. El proceso se describe a continuación.

   *a*) La adaptación del algoritmo al problema del EEG requiere generar la función objetivo en un **fichero por separado** y que a la vez se acople con la definición del conjunto de funciones benchmarks de la implementación del algoritmo. Lo primero que se requiere es establecer un **código numérico** para hacer referencia a la función objetivo, donde se elige **256** - frecuencia de muestreo de las señales - para este indicador. El código servirá para hacer efectiva la llamada a la función objetivo entre las demás del benchmark. Recordar que MATLAB, al ser un lenguaje interpretado, no requiere de compilación, por lo que **mantener el resto de funciones benchmark** no supone sobrecarga alguna.

   *b*) Se añade el código necesario dentro del script que controla la ejecución de las distintas funciones benchmark para que procese el código asignado a la función del EEG y se procede con la implementación del código de la función objetivo. En esta adaptación, el alumno ha intentando aprovechar toda la **potencia** que ofrece MATLAB para el procesamiento eficiente de datos mediante su representación matricial; la implementación es ligeramente distinta a las realizadas con el lenguaje C++ precisamente debido a este factor.

*c*) El último paso es el mismo que en los anteriores algoritmos, donde se comprueba que la solución obtenida casa con la del **validador**. Completar este proceso lleva a la última adaptación que se realizará en esta fase inicial, previo a poner en marcha el clúster Hércules para lanzar los experimentos finales.

5. **Differential Grouping 2:**

   El algoritmo de descomposición de variables DG2[45, 59] es la última propuesta sometida a estudio en este trabajo. La implementación en este caso se encuentra tanto en **C++** como en **MATLAB** y está disponible en el repositorio del autor [59]. Se opta por la implementación de **C++** al no estar la de MATLAB completamente testada, pero para la evaluación de los resultados obtenidos por el algoritmo, se emplea el código MATLAB proporcionado por el autor para este fin. El proceso de adaptación, dada la particular implementación del algoritmo, ha sido muy natural y sin complicaciones reseñables.

   El único paso llevado a cabo en esta adaptación consiste en implementar la función objetivo a través de un fichero de cabeceras **EEG.h** y otro fuente **EEG.cpp** con una estructura a nivel de implementación similar las anteriores, para que pase a formar parte finalmente del conjunto de problemas benchmark total donde, naturalmente, solo se compilará este de cara a los procedimientos experimentales.

   La única modificación destacable consiste en una alteración de la clase **wrapper** del benchmark de forma que, según el tipo de función, se pueda crear un **tipo de dato adecuado con los parámetros que requiere el problema del EEG**, tal y como lo son la dimensión y la existencia de ruido. El fichero **main.cpp** también precisa de modificación para incorporar la actual adaptación, optando por duplicar el mismo y utilizar uno distinto expresamente para este fin.

   Dado que el algoritmo no produce directamente un resultado en forma de **vector solución** como los anteriores, en la **fase experimental** se explicarán las consecuencias derivadas de aplicar este algoritmo frente a la función objetivo del EEG, consecuencias que determinarán en gran medida los pasos efectuados en lo largo de la misma. Finalizado esta subsección, se detallan brevemente los procedimientos de adaptación que sufren los algoritmos de cara a la experimentación en el clúster que, a efectos prácticos, se considera común para todas las técnicas elegidas en este estudio.

Para esta fase, se requiere que la adaptación conducida anteriormente permita a los algoritmos funcionar correctamente en el clúster Hércules. Debido a la cualidad privativa de MATLAB, que precisa de una licencia

para poder utilizarse, los experimentos con este algoritmo **no podrán ser ejecutados en el clúster**, razón por la cual el alumno opta por utilizar su propio ordenador portátil; esta condición será expuesta con detalle en el siguiente capítulo. Esta fase afecta por tanto a las implementaciones de **MOS2011** y **2013**, como **SHADEILS** y **DG2**. Se detallan a continuación los procedimientos llevados a cabos en esta parte de la adaptación de forma general y sin profundizar en detalles sin importancia.

1. El primer inconveniente a resolver en esta adaptación está relacionado con lo **limitada** que está la **imagen del SO de los nodos del cluster**, donde se tienen únicamente los componentes indispensables para funcionar. Algunos elementos como compiladores C y C++, y algunas librerías básicas de estos lenguajes si están disponibles, pero la mayoría de aquellas herramientas y aplicaciones que precisan las implementaciones no están presentes, por lo que es preciso instalarlas en el **directorio raíz** de la cuenta local que tiene el alumno.

2. En total, se ha requerido de una serie de herramientas, librerías y aplicaciones para poner en marcha todos los algoritmos. La distribución **Anaconda3** permite ejecutar el código Python de **SHADEILS** sin incurrir en ningún error de compatibilidad con las versiones disponibles en el nodo cabecera. Otras herramientas y librerías como **Bison, Boost, CMake, flex y libcofig** son indispensables para poner en marcha las dos versiones de MOS; **OpenMPI**, otra librería indispensable para ejecutar estos algoritmos sí se encuentra disponible en el nodo cabecera. Todas estas herramientas han sido instaladas en un directorio **bin** en el directorio raíz local.

3. La necesidad de instalar toda esta gama de herramientas, librerías y aplicaciones hace que para cada algoritmo en particular, los procesos de compilación y/o ejecución requieran acceder a las direcciones donde se encuentran las mismas, incidiendo muchas veces en **conflictos** derivados de una **incorrecta definición** de la variable de entorno **$PATH**, la cual necesita poseer de toda la información relativa a la ubicación de estas librerías y herramientas en la cuenta local.

4. Otro tipo de *path* que precisa especificación previa es el de la ubicación de los **ficheros de datos DATAin** que la función objetivo exige para poder evaluar una solución. Para evitar replicar innecesariamente estos datos, se proporciona un **directorio común** donde accederán todos los algoritmos para recuperar esta información, evitando así problemas locales de la ubicación de cada algoritmo en el árbol de directorios al ser este un **path** absoluto.

5. En particular, para ambas implementaciones de MOS, se requiere especificar ciertas líneas de comandos en los ficheros de **CMakeLists.txt**

que generan los **makefiles** respectivos. Estos comandos están relacionados con el **enlazado de las bibliotecas estáticas**, que se han instalado en local, que se requieren en tiempo de compilación, principalmente las librerías de Boost y OpenMPI. Este es el último compendio de pasos requeridos para conseguir una ejecución exitosa del **proceso experimental**, ya que tanto SHADEILS como DG2 han sido acoplados al clúster sin más incidencias reseñables.

En el siguiente capítulo se muestra el diseño experimental conducido por el alumno para obtener resultados representativos que puedan ser interpretados y analizados en profundidad para extraer las conclusiones pertinentes.

# Capítulo 7

# Diseño experimental

La finalidad de este capítulo consiste en recoger todos los detalles referentes al **diseño experimental** en su totalidad. Este procedimiento está compuesto por una serie de tareas que el alumno ha de desarrollar de acuerdo a la planificación del capítulo 2. Estas tareas tienen como objetivo final permitir al alumno **obtener los resultados** que serán la base de los procesos de evaluación y conclusión posteriores, por lo que resultará fundamental desarrollar esta tarea con alto grado de precisión.

Con respecto al proceso experimental, se realizará en primera instancia un **estudio de escalabilidad** que permita comprobar el comportamiento de los algoritmos de forma preliminar, donde su mayor implicación será la de tomar decisiones que determinen la continuidad de su utilización de cara a la **experimentación completa**, que será la que ofrezca los resultados definitivos con los que se trabajará en la interpretación y el análisis de los mismos.

La estructura de este capítulo comprende dos secciones, donde la primera se centra en una **comparativa a nivel de complejidad** entre las funciones que conforman los benchmarks utilizados en el campo de LSGO actualmente y la proposición de benchmark particular que recoge la definición del problema del EEG a través de la función objetivo, cuyo pseudocódigo se incluye también en esta parte.

La segunda sección estará dedicada a mostrar la configuración de parámetros que se utilizarán tanto en el **estudio de escalabilidad** como en la **experimentación completa**, ya que algunos de los parámetros utilizados en esta última dependerán en gran medida de los resultados de la primera. A continuación, se procede con la comparativa de benchmarks.

## 7.1. Diseño experimental: benchmarks y EEG

Los propuestas elegidas para formar parte de este estudio han sido seleccionadas por el alumno siguiendo unos determinados criterios. La información relativa a las mismas se encuentra disponible a lo largo y ancho de todas las bases de datos de publicaciones científicas más importantes del mundo, como Scopus[11] o IEEE Explorer[20], que han sido utilizadas directamente para los procesos de documentación.

Una vez han sido elegidas las técnicas, el siguiente paso más adecuado en esta situación en particular consiste en obtener una copia original de las implementaciones de los algoritmos elegidos, tarea que se detalla en el capítulo anterior. Sin embargo, estas propuestas están normalmente preparadas para ejecutar un conjunto de funciones que sirven para evaluar el rendimiento de estas propuestas. Este conjunto de funciones se agrupan en **benchmarks**, que ya han sido introducidos en el capítulo 4, y se procede a precisar una visión de conjunto del estado actual, tomando como referencia las propuestas elegidas.

Atendiendo a la definición de [8], las funciones benchmark se agrupan formando distintas clases según el **grado de interacción de las variables**; así tenemos, principalmente, grupos de funciones **separables**, **parcialmente separables**, funciones con **subcomponentes solapadas** y funciones **completamente no separables**. Como ya se describe en el capítulo 5, la **interacción de las variables** de un problema juega un papel fundamental a la hora de llevar a cabo una optimización efectiva, sobre todo en problemas de alta dimensionalidad, dado que ésta determina el **potencial o la capacidad** que tiene una variable para **influir en el efecto de otra u otras variables** y por tanto ser determinante en la calidad de la solución obtenida.

Los benchmarks propuestos para las distintas competiciones de LSGO llevadas a cabo desde el año 2008 contienen habitualmente **entre 15 y 20 funciones** de distinto tipo como las funciones **Shifted Elliptic, Shifted Sphere, Rastrigin, Schwefel, Rosenbrock**, funciones híbridas variadas y otros tipos de problemas de elevada complejidad[27]. La dimensionalidad de estos problemas ronda las 1000 variables, por lo que supone un punto de partida robusto en cuanto a evaluación de las capacidades de un algoritmo se refiere.

Actualmente, los benchmarks que sirven como principal referencia para los algoritmos dedicados a LSGO son los propuestos en las competiciones del **CEC 2010** y, mayoritariamente, los del **CEC 2013**[60], utilizados en la más reciente competición del WCCI 2018[3]. No obstante, como ya se menciona en la revisión de la literatura del capítulo 4, las actuales funciones benchmark **no tienen similitud alguna** con el benchmark que se estudia

este trabajo, el cual contiene una formulación de un problema real como lo es el del EEG.

Como se puede apreciar en los algoritmos 6 y 7 que describen las componentes (operaciones) de la función objetivo del algoritmo 8, la complejidad de la misma es muy alta debido a la cantidad de operaciones de multiplicación que realiza cada vez que requiere evaluar un vector solución. La complejidad algorítmica también aumenta conforme lo hace la dimensión del problema, como se expresa en el capítulo 3. Además, este tipo de funciones se puede catalogar dentro del tipo de funciones **black-box**, funciones donde no se tiene conocimiento acerca de la interacción de sus variables, información de la que **si se dispone** para las funciones de los benchmarks de LSGO utilizados en las competiciones del CEC, lo que ratifica aún más la complejidad que tiene ofrecer una solución escalable y eficiente para el problema del EEG.

Es por ello que se opta por realizar un **estudio de escalabilidad** previo a la experimentación completa, dado que esto permitirá descartar aquellas técnicas que, dado su diseño y/o su implementación particular, no ofrezcan resultados lo suficientemente competitivos como para ser considerados en el posterior análisis. En la siguiente sección por tanto se detalla la **configuración de parámetros** que se utilizarán a lo largo del proceso experimental, así como los principales objetivos que se quieren alcanzar mediante este proceso, que serán detallados en profundidad en el siguiente capítulo, donde se analizarán los resultados obtenidos tanto del estudio de escalabilidad como del estudio completo.

## 7.2. Parámetros de la experimentación

En esta sección se detallan los **ajustes paramétricos** del proceso experimental, donde se explicará la **naturaleza** de cada experimento, el objetivo a conseguir y, naturalmente, los parámetros experimentales de cada algoritmo para garantizar la repetibilidad de los experimentos.

El **estudio de escalabilidad** está destinado a obtener una primera impresión de los algoritmos y técnicas elegidas. Se procederá a ejecutar cada algoritmo con su configuración **de fábrica**, esto es, con la especificación de parámetros que por defecto definen los autores en su implementación, con el objetivo de comprobar las capacidades de **generalización**, desde el punto de vista de un problema benchmark completamente distinto a cualquiera de los anteriores, y de **especificación**, en cuanto a capacidad de convergencia.

Para ello, cada algoritmo dispondrá de **cien mil (100000) o 100K** evaluaciones de la función objetivo. De esta manera se busca comprobar cómo responde el algoritmo cuando la función objetivo es compleja, requiere de mucho cómputo y además está limitado en cuanto a sus capacidades de ex-

ploración y explotación. Se analizarán los comportamientos de **convergencia** del algoritmo, prestando especial atención al problema del estancamiento de cara a calibrar la siguiente fase experimental

Estos parámetros de experimentación sólo afectan a las 4 primeras técnicas, siendo **DG2** la técnica que sufrirá un proceso experimental distinto. Debido a las necesidades de **reducir la dimensionalidad**, obtener descomposiciones óptimas - aquellas que **minimicen la interacción** entre componentes - resulta crucial en problemas de alta dimensionalidad. Las descomposiciones dependen de la naturaleza de la función objetivo, por lo que mediante este experimento se busca **descartar el peor escenario posible**: una función no-separable.

El algoritmo DG2 será puesto a prueba con la función objetivo frente a los problemas **D4, D4N y D12** para comprobar si existe algún tipo de descomposición que pueda derivar en una hibridación de DG2 con el mejor de los algoritmos del estudio experimental completo. Sin embargo, DG2 no seguirá las mismas condiciones que los demás algoritmos en cuanto a evaluaciones de la función objetivo, sino que según la dimensión del problema el algoritmo calcula internamente la cantidad de evaluaciones que necesita realizar para identificar la interacción de las variables (ver capítulo 5).

DG2 por tanto no sigue las mismas reglas que los demás algoritmos, pero en el análisis del estudio de escalabilidad se comprobará si esta propuesta cumple con las condiciones necesarias (que serán detalladas en el propio análisis) para continuar con la experimentación completa. Si estas condiciones **no se cumpliesen**, no se podría realizar esta hibridación, y los resultados de DG2 quedarán limitados a este estudio de escalabildad.

Los resultados del estudio de escalabilidad serán detallados y analizados en la sección 8.1 del capítulo siguiente, por lo que a continuación se procede a especificar la **batería de parámetros** de cada algoritmo según las implementaciones correspondientes. Recordar la condición *parameter-free* de DG2, por lo que no se mostrará ninguna tabla con respecto a esta técnica.

1. **Multiple Offspring Sampling (2011)**: los parámetros de la técnica MTS-LS1 son los utilizados originalmente en el paper [29], por lo que se enumeran los del algoritmo en cuestión. Estos parámetros se especifican en dos ficheros de configuración distintos donde el principal contiene aquellos parámetros **comunes** al algoritmo en su totalidad y el segundo los parámetros **particulares** de cada una de las técnicas que lo componen.

| **Param.** | **Valor** | **Param.** | **Valor** |
|---|---|---|---|
| **Tam. Pobl.** | 15 | **DE CR** | 0.5 |
| **DE F** | 0.5 | **DE op. cruce** | Exponencial |
| **DE op. selección** | Torneo 2 | **DE Modelo** | Clasico |
| **Ratio min. part.** | 5 % | **Tam. paso** | 35715 |

Tabla 7.1: Parámetros de MOS 2011

2. **Multiple Offspring Sampling (2013)**: se muestran los parámetros globales del algoritmo MOS2013 así como los de sus algoritmos componentes. Aquí ocurre algo similar con el anterior algoritmo, disponiendo de dos ficheros de configuración para detallar los valores que toma cada parámetro.

| **Param.** | **Valor** | **Param.** | **Valor** | **Param.** | **Valor** |
|---|---|---|---|---|---|
| **Tam. Pobl** | 400 | **Op. selección** | Torneo 2 | **pcx** | 0.9 |
| **pmut** | 0.01 | **Min. part.** | 20 % | **Tam.paso** | 36000 |

Tabla 7.2: Parámetros de MOS 2013

| **Param.** | **Valor** | **Param.** | **Valor** | **Param.** | **Valor** |
|---|---|---|---|---|---|
| **maxSuccess** | 5 | **maxfailed** | 5 | **adjustSuccess** | 4 |
| **adjustFailed** | 0.75 | **delta** | 2.4 | | |

Tabla 7.3: Parámetros de MOS 2013 - Solis Wets LS

| **Param.** | **Valor** | **Param.** | **Valor** | **Param.** | **Valor** |
|---|---|---|---|---|---|
| **adjustFailed** | 2 | **adjustMin** | 10 | **moveLeft** | 0.25 |
| **moveRight** | 0.5 | **searchProb** | 0.9 | **minProb** | 0.025 |

Tabla 7.4: Parámetros de MOS 2013: MTS-LS1-Reduced)

3. **SHADEILS**: los parámetros que propone el autor de esta implementación son los que se muestran a continuación. Estos parámetros, aunque modificables a través de la posibilidad de que el usuario introduzca los valores de cada uno, toman una serie de **valores por defecto** en caso de que no se indique ninguno en específico, y son estos los que se muestran en la siguiente tabla.

| Param. | Valor | Param. | Valor |
|---|---|---|---|
| **Tam. Pobl.** | 100 | **Evals. DE** | 25000 |
| **Evals. LS** | 25000 | **MTS Step** | 20 |
| **Umbral** | 0.001 | **SHADE Tam. Historia** | 3 |
| **Reinicio pobl. fallido** | 3 | **Min. ratio mejora** | 5 % |

Tabla 7.5: Parámetros de SHADEILS

4. **MLSHADE-SPA**: se muestran los parámetros globales del algoritmo, pero debido a la gran cantidad de parámetros de los demás algoritmos componentes, se describirán únicamente los más importantes, remitiendo al lector a [52, 51] para los de cada pool de valores **CR** de los algoritmos EADE y ANDE.

| Param. | Valor | Param. | Valor |
|---|---|---|---|
| **Tam. Pobl.** | 250 | **Min. Poblacion** | 20 |
| **Evals. CC** | Evals/50 | **Tam. memoria** | 5 |
| **Pos. Mem. inicial** | 1 | **Ratio PBest** | 0.1 |

Tabla 7.6: Parámetros de MLSHADE-SPA

El objetivo del **estudio experimental completo** es comprobar la **capacidad de convergencia** de las técnicas sin sobrepasar las restricciones temporales implícitas en la dimensionalidad del problema, debido a que, como se expresa anteriormente, el objetivo es proponer una solución lo **más factible posible** de cara a una implementación en el mundo real. Se tienen que considerar otros factores, como la especificación de semillas, que permitirán llevar a cabo un proceso repetible y comprobable, siempre que se den las mismas condiciones.

Para el estudio completo, en cuanto a los cuatro primeros algoritmos se utilizarán nuevamente los parámetros especificados en las **tablas anteriores**. Con el fin de obtener **mediciones más representativas** de las capacidades de cada técnica, cada algoritmo se ejecutará **10 veces** sobre cada uno de los **6 problemas** D4, D4N, D12, D12N, D19 y D19N. Para garantizar que los experimentos sean repetibles, se utilizará un conjunto de **10 valores de semillas aleatorias** que fijarán, para cada algoritmo y problema, un punto de partida distinto entre ejecuciones de un mismo problema.

La utilización de DG2 en esta fase de experimentación está supeditada a los resultados del estudio de escalabilidad, por lo que de ser satisfactorios,

se procederá a realizar una hibridación de esta técnica con el mejor de los algoritmos de la experimentación completa. Se mostrarán sus resultados en los mismos términos con los que se detallan en el estudio de escalabiliadad, para que el posterior análisis de los resultados de esta fase sean comparables con los del estudio de escalabilidad anterior.

Cabe destacar que para los cuartro algoritmos, **MOS2011**, **MOS2013**, **SHADEILS** y **DG2**, la experimentación se realizará utilizando el clúster **Hercules**, que tiene la configuración expresada en la tabla 7.7, mientras que para el algorimto **MLSHADE-SPA**, se utilizará el portátil del alumno, que posee unas características similares al del clúster en cuanto a recursos; se detallan sus especificaciones en la tabla 7.8.

| **Especificaciones.** | **Valor** | **Especificaciones.** | **Valor** |
|---|---|---|---|
| **Nodos** | 46 | **S.O** | Ubuntu 18.04.1 LTS |
| **Procesador** | Intel Core i7 930 2.8 GHz | **RAM** | 24GB |

Tabla 7.7: Especificaciones del Clúster Hercules

| **Especificaciones.** | **Valor** | **Especificaciones.** | **Valor** |
|---|---|---|---|
| **Nodo** | MacBook Pro (2012) | **S.O** | MacOS HighSierra 10.13.6 |
| **Procesador** | Intel Core i7 2.9 GHz | **RAM** | 16GB |

Tabla 7.8: Especificaciones del portátil del alumno

Finalmente, se realizará un acopio de todos estos datos y se presentarán en forma de tablas que incluyan mediciones relevantes como los valores **mínimos** y **máximos** de fitness obtenido, así como una media de los 10 experimentos para cada problema y, naturalmente, el **tiempo medio** empleado para resolver el mismo. También se proporcionarán **gráficas de convergencia** que permitan identificar hasta qué punto cada técnica es capaz de responder con suficiente precisión sin dejar de lado las restricciones temporales inherentes al problema.

# Capítulo 8

# Pruebas y análisis de resultados

Este capítulo ofrece los resultados del estudio experimental conducida en este trabajo. Es importante destacar que este capítulo está dividido en dos secciones, un **estudio de escalabilidad** donde se comprueba el comportamiento de los algoritmos en términos de convergencia cuando disponen de **100 mil** de evaluaciones de la función objetivo; a la vez, comprueba como evoluciona la convergencia en función del tiempo, de cara a establecer las evaluaciones de la función objetivo que serán conducidas en el **estudio experimental completo**, donde se obtendrán resultados más representativos al permitir que cada algoritmo ejecute **10 veces** cada problema.

Una sección dedicada al **análisis exhaustivo de los resultados** será conducida al final de la experimentación completa, donde se interpretan las soluciones de cada una de las técnicas y donde primará un enfoque de implementación real, lo que implica revisar la calidad de las soluciones obtenidas en términos de eficacia y eficiencia, teniendo en mente el objetivo que se busca desde el principio, que es trasladar el uso de las técnicas más potentes del campo LSGO a un entorno real.

## 8.1. Estudio de escalabilidad

Este estudio sirve para establecer de forma preliminar el comportamiento de los algoritmos frente a cada problema del EEG. Se resumen los primeros resultados en tablas temporales donde se miden los **tiempos de ejecución** de cada algoritmo en relación al número de **evaluaciones de la función objetivo** que va ejecutando. De esta forma se identifican aquellos comportamientos que determinan la **posible validez** de un algoritmo de cara a formar parte de una solución real del problema del EEG.

En esta fase, para cada algoritmo y subproblema del EEG, esto es, para cada dimensión tanto con ruido como sin éste, se exponen las tablas de resultados de tiempos junto con la **gráfica de convergencia resumida**. Se obvian mediciones del fitness debido a que no son lo suficientemente representativas, al depender, naturalmente, de la cantidad de evaluaciones de la función objetivo.

Finalmente, se muestra un total de cinco tablas con las marcas temporales de cada algoritmo para cada problema y una tabla temporal resumen con las marcas temporales de cada problema en particular. A pesar de que no se detalla en este estudio de escalabilidad ninguna información acerca de la **convergencia de valores fitness**, en el primer anexo se encuentran 6 gráficas de convergencia, una para cada problema, contemplando los cuatro primeros algoritmos: MOS2011, MOS2013, SHADEILS y MLSHADE-SPA.

- Los resultados de **MOS2011** frente a cada subproblema del EEG se disponen en la tabla a continuación. Recordar que los problemas D4(N) contienen 1024 variables, los D12(N) son de 3072 variables y D19(N) crece hasta las 4864, donde **(N)** indica que los problemas con ruido manejan la misma cantidad de variables.

| **Problema** | **Tiempo(s)** |
|---|---|
| D4 | 144.405 |
| D4N | 144.224 |
| D12 | 1099.725 |
| D12N | 1109.071 |
| D19 | 2666.309 |
| D19N | 2672.099 |

Tabla 8.1: **MOS2011** - Escalabilidad **100K Evals**

- La siguiente tabla muestra los primeros resultados obtenidos por **MOS 2013** tras ejecutar 100K evaluaciones de la función objetivo para cada uno de los problemas del benchmark.

| **Problema** | **Tiempo(s)** |
|---|---|
| D4 | 140.144 |
| D4N | 139.702 |
| D12 | 1043.632 |
| D12N | 1050.612 |
| D19 | 2553.400 |
| D19N | 2566.802 |

Tabla 8.2: **MOS2013** - Escalabilidad **100K Evals**

- Esta tabla muestra los resultados del estudio de escalabilidad de **SHADEILS**, resultados que serán discutidos al final de esta sección.

| **Problema** | **Tiempo(s)** |
|---|---|
| D4 | 37.576 |
| D4N | 37.220 |
| D12 | 121.785 |
| D12N | 120.492 |
| D19 | 184.093 |
| D19N | 186.080 |

Tabla 8.3: **SHADEILS** - Escalabilidad: **100K Evals**

- Los resultados que se han obtenido con **MLSHADE-SPA** durante la primera fase de experimentación se muestran a continuación, así como sus gráficas de convergencia.

| **Problema** | **Tiempo(s)** |
|---|---|
| D4 | 63.490 |
| D4N | 74.310 |
| D12 | 398.180 |
| D12N | 399.780 |
| D19 | 1045.690 |
| D19N | 1054.130 |

Tabla 8.4: **MLSHADE-SPA** - Escalabilidad: **100K Evals**

- Finalmente, se muestran los resultados del estudio de escalabilidad del algoritmo **DG2** y posteriormente se procede a mostrar las gráficas de convergencia en términos temporales, previo al análisis de este fase experimental.

| Problema | Tiempo(s) |
|---|---|
| D4 | 670.1 |
| D4N | 670.3 |
| D12 | 47549.9 |
| D12N | - |
| D19 | - |
| D19N | - |

Tabla 8.5: **DG2** - Escalabilidad

La salida del algoritmo para los problemas D4, D4N y D12 se muestran a continuación.

```
Function F: D4
Number of separables: 0
Number of non-separable groups: 1
Expected sizes    |   1024
Size of G01: 1024  |   1024
============================
Function F: D4N
Number of separables: 0
Number of non-separable groups: 1
Expected sizes    |   1024
Size of G01: 1024  |   1024
============================
Function F: D12
Number of separables: 0
Number of non-separable groups: 1
Expected sizes    |   3072
Size of G01: 3072  |   3072
```

Listing 8.1: Análisis de la matriz DSM tras la ejecución de DG2 para los problemas D4, D4N y D12N

Una vez comprobada la escalabilidad temporal de los algoritmos, el siguiente paso es obtener la **gráfica de convergencia** resumida para cada dimensionalidad del problema y algoritmo, y que se puede ver en la figura 8.1. En esta gráfica se puede apreciar de forma preliminar el comportamiento de convergencia que exhiben los algoritmos, donde se pueden destacar tres grupos en cuanto a los tiempos de respuesta: primero SHADEILS, MLSHADE-SPA en segunda instancia y ambas técnicas MOS en el tercer grupo.

Esta conducta será analizada en profundidad en la sección de comparativa global tras la experimentación completa, aunque ya se puedan intuir grandes diferencias entre los tiempos de respuesta de estos tres grupos. Un cuarto grupo donde estaría DG2 **no aparece** en esta gráfica por razones que es posible advertir a pesar de no haber comenzado el análisis pertinente, el cual se expresa en mayor detalle justo después de mostrar la gráfica.

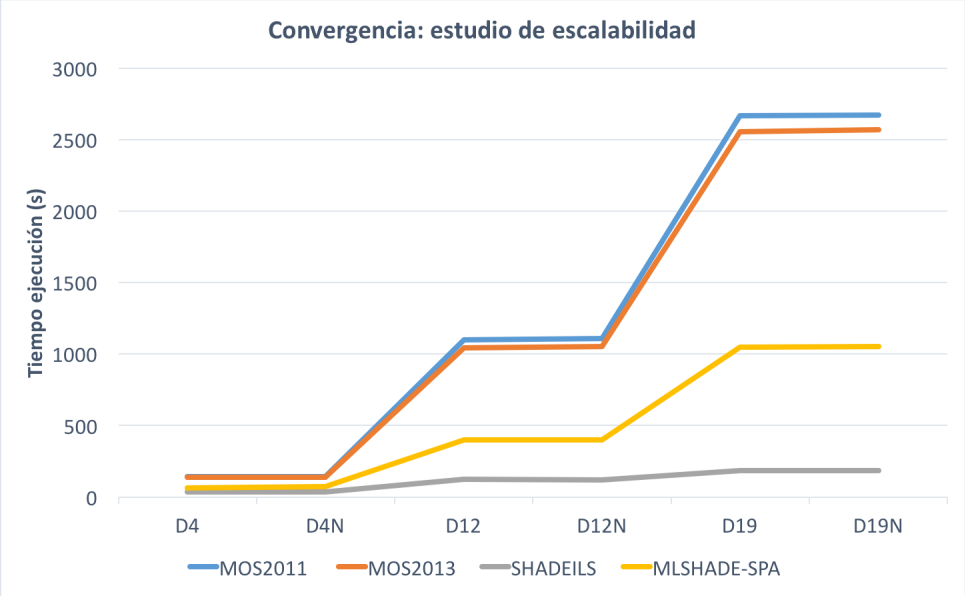


Figura 8.1: Convergencia temporal de los 4 algoritmos para cada problema

Atendiendo a los resultados obtenidos para los tres experimentos conducidos con DG2, se puede comprobar que el escenario obtenido no es el más indicado como para abordar el problema mediante una descomposición de variables: la salida del algoritmo, además de mostrar el excesivo tiempo de computación que emplea, muestra que no es posible realizar una descomposición del problema debido a que la función es no separable. Este escenario puede tener como explicación una razón, donde la propia naturaleza del problema puede marcar el camino del algoritmo sin necesidad de una ejecución posterior.

Como se explica en la introducción al problema, en un EEG real las señales se combinan debido a *interferencias y ruido* propios de la medición, lo que hace que un electrodo **i** pueda medir información tanto local a su zona como la que se produce en otras zonas que **también** están siendo medidas por otros electrodos **k**. De esta forma, la interacción de las variables es máxima, como se puede ver en este caso, lo que limita considerablemente las capacidades del algoritmo, al no ser capaz de obtener una descomposición óptima que **minimice** la interacción entre componentes, por el simple y llano hecho de que no existe descomposición posible.

Teniendo en cuenta estos resultados y la primitiva que se expresa en [42], la técnicas de Cooperación Co-evolutiva dedicadas a optimizar funciones no-separables de alta dimensionalidad no han mostrado resultados suficientemente competentes como para ser considerados aplicables en un ámbito real donde no se tiene información alguna acerca de la interacción

de las variables de la función. A esto hay que añadirle el hecho de que los más recientes estudios de CC con descomposición de variables han sido probados con funciones separables o parcialmente separables, por lo que en gran medida se desconoce la escalabilidad de estas propuestas en funciones no-separables[42].

Estas circunstancias motivan la decisión de **no continuar con DG2** para la fase de experimentación completa, dado que el **resultado** se torna **incierto** desde un principio y que además, las restricciones temporales inherentes al problema no pueden contemplar que una técnica híbrida que se pretende aplicar en un campo u entorno real, tenga un tiempo de respuesta superior a las **doce horas**, como es el caso de la función del EEG en el problema D12.

Teniendo en cuenta esta situación, se hace impensable la aplicación de este tipo de técnicas para optimizar un encefalograma actual que puede disponer de más de **20 señales de entrada e innumerable artifacts**, con una frecuencia de muestreo superior a la de este benchmark sintético. A partir de este punto, queda totalmente descartada la aplicación de este algoritmo en los posteriores experimentos de este estudio, por lo que tampoco se tendrá en consideración en las conclusiones finales, en cuanto a propuesta de solución real se refiere. Se procede con la fase de experimentación completa.

## 8.2. Estudio experimental completo

El estudio experimental completo muestra los resultados de 10 ejecuciones de cada algoritmo para cada uno de los 6 problemas de dimensionalidad que proponer el benchmark del EEG. En esta fase, se miden características como los valores **mínimos y máximos** de fitness alcanzados por cada algoritmo a lo largo de las 10 ejecuciones, así como una **media del fitness global** y donde, en esta ocasión, se muestra el **tiempo medio** transcurrido teniendo en cuenta la totalidad de las ejecuciones realizadas.

En este estudio se ha utilizado un número de **evaluaciones de la función objetivo**, que es el criterio de parada, igual a 1 millón (1M); se ha elegido este valor en función de los resultados de la gráfica de convergencia destacada con anterioridad y por la escalabilidad en cuanto al tiempo de ejecución de los algoritmos. Para cada una de las técnicas, se muestra una tabla con valores **mínimos**, **máximos** y **medios de fitness** tras las 10 ejecuciones, así como el **tiempo de ejecución medio**.

Resaltar que teniendo en cuenta que muchas de las **diferencias de valores fitness** entre los distintos algoritmos se producen más allá de las 10 cifras decimales, es posible encontrarse valores iguales tanto para mínimo, máximo y media para distintos algoritmos, fenómeno que no ocurre para las

medias temporales. Posteriormente se mostrará una tabla resumen de fitness medio y tiempo medio, junto con el algoritmo de referencia, momento a partir del cual se comenzará el análisis de los resultados.

1. **MOS2011**:

| **Problema** | **Min.** | **Max.** | **Media** | **Tiempo** |
|---|---|---|---|---|
| D4 | 0.06103 | 0.06103 | 0.06103 | 1519.903 |
| D4N | 0.05897 | 0.05897 | 0.05897 | 1567.199 |
| D12 | 0.00194 | 0.00200 | 0.00198 | 12634.289 |
| D12N | 0.00187 | 0.00190 | 0.00188 | 12593.868 |
| D19 | 0.00830 | 0.00992 | 0.00894 | 28033.663 |
| D19N | 0.00824 | 0.00978 | 0.00918 | 28586.774 |

Tabla 8.6: Resultados MOS2011: Experimentación completa

2. **MOS2013**:

| **Problema** | **Min.** | **Max.** | **Media** | **Tiempo** |
|---|---|---|---|---|
| D4 | 0.06103 | 0.06103 | 0.06103 | 1265.065 |
| D4N | 0.05897 | 0.05897 | 0.05897 | 1221.990 |
| D12 | 0.00194 | 0.00194 | 0.00194 | 9597.190 |
| D12N | 0.00183 | 0.00183 | 0.00183 | 9520.520 |
| D19 | 0.00251 | 0.00251 | 0.00251 | 23524.796 |
| D19N | 0.00256 | 0.00256 | 0.00256 | 23037.743 |

Tabla 8.7: Resultados MOS2013: Experimentación completa

3. **SHADEILS**:

| Problema | Min. | Max. | Media | Tiempo |
|---|---|---|---|---|
| D4 | 0.06103 | 0.06103 | 0.06103 | 251.759 |
| D4N | 0.05897 | 0.05897 | 0.05897 | 250.908 |
| D12 | 0.00194 | 0.00194 | 0.00194 | 802.331 |
| D12N | 0.00183 | 0.00183 | 0.00183 | 786.965 |
| D19 | 0.00251 | 0.00252 | 0.00252 | 1578.034 |
| D19N | 0.00256 | 0.00256 | 0.00256 | 1573.507 |

Tabla 8.8: Resultados SHADEILS: Experimentación completa

4. **MLSHADE-SPA**:

| Problema | Min. | Max. | Media | Tiempo |
|---|---|---|---|---|
| D4 | 0.06103 | 0.06103 | 0.06103 | 490.294 |
| D4N | 0.05897 | 0.05897 | 0.05897 | 487.606 |
| D12 | 0.00194 | 0.00194 | 0.00194 | 3918.041 |
| D12N | 0.00183 | 0.00183 | 0.00183 | 3936.233 |
| D19 | 0.00252 | 0.00252 | 0.00252 | 9229.300 |
| D19N | 0.00256 | 0.00256 | 0.00256 | 8595.423 |

Tabla 8.9: Resultados MLSHADE-SPA: Experimentación completa

## 8.3. Comparativa Global

La tabla resumen que se muestra a continuación permite obtener una visión global de los resultados, además de permitir comparar con el algoritmo de referencia **MAGA**[25]. En esta se aprecia el valor de fitness medio tras las 10 ejecuciones del algoritmo para cada problema. Tras discutir los resultados obtenidos en cuanto a valores fitness, se procederá con al análisis teniendo en cuenta el tiempo de ejecución.

| Algoritmo | D4 | D4N | D12 | D12N | D19 | D19N |
|---|---|---|---|---|---|---|
| MOS2011 | 0.06103 | 0.05897 | 0.00198 | 0.00188 | 0.00894 | 0.00918 |
| MOS2013 | 0.06103 | 0.05897 | 0.00194 | 0.00183 | 0.00251 | 0.00256 |
| MLSHADE-SPA | 0.06103 | 0.05897 | 0.00194 | 0.00183 | 0.00252 | 0.00256 |
| SHADEILS | 0.06103 | 0.05897 | 0.00194 | 0.00183 | 0.00252 | 0.00256 |
| MAGA* | 0.0610 | 0.0590 | 0.0019 | 0.0018 | 0.0025 | 0.0026 |

Tabla 8.10: Valores fitness medio. MAGA*: algoritmo de referencia.

A partir de los resultados obtenidos podemos deducir las siguientes conclusiones generales:

- El problema **D4** (1024 variables) puede parecer más complejo de resolver a priori, en cuanto a valor fitness se refiere. Sin embargo lo que realmente ocurre es que se dispone de **menos información** - menor cantidad de señales mezcladas - de las que el algoritmo necesita para identificar la correlación entre las señales y separar los artifacts de la misma. Si además se tiene en cuenta que en un problema médico real en el que intervenga un electroencefalograma es común encontrar al menos **20 electrodos**, que actuarían como fuentes de señales, este problema no es representativo de cara a un entorno real.

- El problema **D12** (3072 variables) presenta los mejores resultados y eso se debe al equilibrio entre cantidad de información disponible y capacidad de las técnicas actuales. A pesar de crecer en términos de dimensionalidad, la calidad de las soluciones es mejor porque precisamente se dispone de mayor cantidad de información para que el algoritmo calcule la matriz **S1** con una precisión mayor que para el problema anterior.

  Además cabe destacar en este punto la flexibilidad y robustez de las técnicas elegidas, debido a que a pesar de haber sido diseñadas para un benchmark cuya dimensionalidad es de 1000 variables, no pierden capacidad de convergencia frente a una dimensionalidad 3 veces mayor, llegando incluso a mostrar ajustes mejores que con menor cantidad de información. En cualquier caso, sigue siendo precipitado considerar este problema como *suficientemente representativo* de un problema de optimización de un EEG real, a pesar de si ser mas representativo que el anterior.

- El problema **D19** (4864 variables) muestra como afecta la dimensionalidad del problema en la calidad de las soluciones. A pesar de alcanzar resultados bastante más satisfactorios en comparación con D4, la elevada dimensionalidad hace que a pesar de contener más información

representativa para identificar la correlación entre las señales, el algoritmo no sea capaz (en términos de potencia) de obtener mejores soluciones que con menos variables. Sin embargo, se dispone de argumentos suficientes como para afirmar que las propuestas de algoritmos elegidas son **escalables**: al cambiar de benchmark han respondido de forma satisfactoria; un indicador más de la robustez de las técnicas empleadas.

- El algoritmo de referencia **MAGA** que se utiliza en esta comparativa es una técnica que, como anteriormente se ha expuesto, ha evolucionado de su antecesora y ha sido diseñada exclusivamente para la resolución del benchmark del EEG. Tomando en consideración este precepto, podemos comprobar como todos los algoritmos menos **MOS2011** (en problemas de dimensionalidad mayor, principalmente) muestran resultados prácticamente idénticos que el algoritmo de referencia, lo que es un buen indicador, nuevamente, de la potencia de estos algoritmos y las capacidades de hacer frente a problemas muy distintos.

- Finalmente, destacar que la introducción de ruido en los problemas (de varianza 0.1) no es lo suficientemente relevante como para causar un detrimento en la calidad de las soluciones, donde tampoco se ve afectado de forma significativa el tiempo de ejecución medio de los algoritmos, por lo que es importante destacar que la formulación del benchmark elegida no es la más adecuada en términos de ruido añadido, ya que no añade suficiente dificultad durante el proceso de optimización.

Tomando como referencia la tabla resumen de fitness medio (tabla 8.10), se puede comprobar como **MOS2013** introduce una mejora sustancial al cambiar el diseño del algoritmo, mejora que se hace más notable conforme crece la dimensión del problema, llegando a reducir el error obtenido en los problemas D19(N) en alrededor de un 25 % . A pesar de que la implementación de MOS2011 utiliza un algoritmo de evolución diferencial (DE) que es más escalable que el algoritmo genético clasico de MOS2013, los métodos de búsqueda local empleados por MOS2013 son mucho más potentes y rápidos, lo que le otorga a este una velocidad de convergencia superior a la de su predecesor.

Destacar en este punto la implementación de la **MTS-LS1-Reduced** utilizada en MOS2013, donde en vez de explorar todas las variables a la vez, se centra en aquellas que más aportan a la mejora de la calidad de las soluciones, optimizando al máximo la cantidad de evaluaciones de la función objetivo a la vez que reduce la dimensión de las operaciones que realiza, otorgándole a MOS2013 mayor velocidad de convergencia, lo que se nota de forma indiscutible en las dimensiones más altas.

Si se consideran las implementaciones de **MOS2013**, **SHADEILS** y **MLSHADE-SPA** es fácilmente reconocible la potencia de las tres técnicas en comparación con MOS2011, sobretodo conforme crece la dimensionalidad del problema. Partiendo de la base de que se disponen de técnicas muy robustas y escalables, que además han mostrado rendimientos prácticamente idénticos, tanto entre ellas como con el algoritmo de referencia MAGA, se torna indispensable utilizar otro enfoque de comparativa: el **enfoque temporal**. Teniendo en cuenta que los resultados están supeditados a los tiempos de respuesta efectiva de los algoritmos, se procede a comparar las propuestas de este estudio utilizando la característica temporal, detallando en primera instancia

| **Algoritmo** | **D4** | **D4N** | **D12** | **D12N** | **D19** | **D19N** |
|---|---|---|---|---|---|---|
| MOS2011 | 1519 | 1567 | 12634 | 12593 | 28033 | 28586 |
| MOS2013 | 1265 | 1221 | 9597 | 9520 | 23524 | 23037 |
| MLSHADE-SPA | 490 | 487 | 3918 | 3936 | 9229 | 8595 |
| SHADEILS | 251 | 250 | 802 | 786 | 1578 | 1573 |

Tabla 8.11: Tiempo medio (s)

Se parte de la idea de que, como es natural, los tiempos de ejecución están determinados por el número de variables: a mayor tamaño del espacio de soluciones, mayor cantidad de operaciones hay que realizar y más costosas se tornan estas operaciones. El ruido introducido en este benchmark, de forma sintética, no afecta de forma significativa alguna al tiempo de respuesta, llegando en ocasiones incluso a *facilitar* (véase la tabla 8.11) la convergencia del algoritmo en términos temporales. Por último, destacar el **aumento no lineal** de los tiempos de ejecución en función del número de variables, lo que reafirma la incapacidad de obtener información alguna acerca de la correlación entre las variables de cara a una implementación híbrida de técnicas que requieran descomposiciones lo más óptimas posibles para alcanzar resultados competitivos.

La tabla de porcentajes de tiempos que se muestra a continuación, que no es más que un resumen en representación porcentual de la tabla 8.11, muestra el porcentaje de reducción temporal de cada una de las técnicas con respecto a la propuesta más lenta, en este caso, MOS2011.

A través de esta tabla se puede comprobar como **MOS2013** tiene un mejor tiempo de respuesta que su antecesor MOS2011. Las técnicas de búsqueda locales empleadas por este son más potentes que las de su predecesor, lo que se traduce también en un tiempo de respuesta menor: MOS2013 consigue reducir los tiempos de ejecución entre un 20 % y 25 % con respecto a MOS2011 para la totalidad de los problemas.

| Algoritmo | D4 | D4N | D12 | D12N | D19 | D19N |
|---|---|---|---|---|---|---|
| MOS2011 | 100 | 100 | 100 | 100 | 100 | 100 |
| MOS2013 | 83 | 77 | 75 | 75 | 83 | 80 |
| MLSHADE-SPA | 32 | 31 | 31 | 31 | 32 | 30 |
| SHADEILS | 16 | 15 | 6 | 6 | 6 | 5 |

Tabla 8.12: Porcentaje ( %) de reducción temporal

En cuanto al segundo mejor, en términos temporales, **MLSHADE-SPA** tiene un rendimiento destacable frente a las dos propuestas MOS. Emplea alrededor de un 30 % del tiempo que necesita MOS2011 y menos de la mitad del que requiere MOS2013, llegando a resolver el problema de dimensión D12 en una hora en vez de 3, al igual que emplea de media algo menos de 3 horas en resolver el problema D19, tiempo que en MOS2013 se va hasta rozar las 8 horas de ejecución.

A grandes rasgos, queda muy claro que **SHADEILS** es el mejor de todos los algoritmos en relación calidad de la solución - tiempo de ejecución, donde se mantiene a menos de un 10 % del tiempo de ejecución que emplea MOS2011 para los problemas D12(N) y D19(N). La diferencia con MOS2013 es un poco menos significativa, en torno a un 12.5 % de media para todos los problemas, y cerca de un 20 % del tiempo con respecto a MLSHADE-SPA para dimensiones D12 y D19, situándose como claro vencedor en esta comparativa dado que es **único algoritmo** que ha conseguido resolver todos los problemas en menos de media hora.

Sin embargo, estos resultados pueden parecer contradictorios a priori debido a la **naturaleza de las implementaciones** y de las **tecnologías empleadas**. Es bien sabido que **C++ y C** son los lenguajes de programación **más rápidos** de la actualidad, dada su condición de *compilados*. Seguidamente, otros lenguajes como pueden ser en este caso **Python** o **MATLAB** quedan forzosamente por detrás de C++ al ser lenguajes interpretados. Surge entonces la pregunta de cómo una implementación en un lenguaje compilado, como las de MOS2011 y MOS2013, es ordenes de magnitud **más lenta** que una implementación en un lenguaje interpretado, como las de SHADEILS y MLSHADE-SPA.

La respuesta está en el diseño de cada una de las técnicas. Mientras que tanto MOS2011 como MOS2013 utilizan la librería **GAlib**, que a grandes rasgos no es más que un conjunto de plantillas para diseñar algoritmos evolutivos, pero que añade mucha complejidad estructural al algoritmo además de la propia que ya introduce el diseño del algoritmo en cuanto a parámetros propios y las operaciones que efectúa, SHADEILS hace un uso intensivo de librerías como **Numpy**, que permiten acceder desde Python a implemen-

taciones robustas y optimizadas de operaciones en C++, consiguiendo un resultado en cuanto a tiempo de respuesta muy similar al que se obtendría si se utilizase C++ de forma nativa.

Siguiendo con el diseño, MOS utiliza algoritmos evolutivos clásicos como el **GA**, cuya escalabilidad es muy baja, lo que aumenta considerablemente el tiempo de ejecución. Por otra parte, tanto SHADEILS como MLSHADE-SPA hacen uso intensivo de técnicas muy avanzadas de evolución diferencial, como lo es **SHADE** o **L-SHADE**, esta última presente en MLSHADE-SPA donde aplica una reducción gradual de la población. Aunque no es posible saber con certeza si un tamaño de población que se reduce constantemente pueda ser indicativo de una mejora en el tiempo de respuesta, como es el caso si se comparan MOS y MLSHADE-SPA, hay que tener en cuenta que SHADEILS utiliza apenas un cuarto de la población de MOS2013, lo que si puede ser indicativo de una mejora sustancial de un algoritmo a otro.

Dada la implementación particular de cada técnica y los resultados obtenidos al utilizar distintos lenguajes de programación, se puede afirmar que si se implementasen todas las técnicas en un mismo lenguaje, prevalecerían los resultados que se han obtenido en este estudio, consiguiendo reducir aún más los tiempos de ejecución de algoritmos como SHADEILS o MLSHADE-SPA.

No obstante, si ahora se compara a SHADEILS con el algoritmo de referencia **MAGA**, los resultados no parecen ser tan competitivos. MAGA es un algoritmo que ha sido diseñado expresamente para este benchmark, por lo que no se tienen registros algunos de resultados con otros benchmarks del panorama LSGO. Además, el autor de la publicación no refleja el criterio de parada del algoritmo ni la cantidad de evaluaciones de la función objetivo que ejecuta, por lo que se ha tomado la decisión de medir el tiempo que tarda **SHADEILS** en alcanzar una **precisión igual** a la que se muestra en el algoritmo de referencia en cuanto a valor fitness, para así poder comparar ambos algoritmos bajo condiciones similares.

Para ello, se ha ejecutado SHADEILS para cada problema con una semilla aleatoria y 1 millón de evaluaciones de la función objetivo. La tabla correspondiente se muestra a continuación, donde los valores fitness de referencia que se han utilizado para esta experimentación en concreto son los que aparecen en la tabla 8.10.

| **Algoritmo** | **D4** | **D4N** | **D12** | **D12N** | **D19** | **D19N** |
|---|---|---|---|---|---|---|
| SHADEILS | 1.152 | 0.934 | 47.175 | 47.759 | 472.229 | 472.010 |
| MAGA | 0.190 | 0.191 | 1.322 | 1.396 | 7.176 | 9.216 |

Tabla 8.13: Tiempo (s) de SHADEILS en conseguir un error igual a MAGA

Los datos muestran que SHADEILS tiene una velocidad de convergencia

alta, hecho que queda demostrado tras haber medido el tiempo transcurrido hasta que se alcanza por primera vez la solución optima de MAGA. Si se tiene en consideración que el autor de la publicación ha utilizado Visual Studio 2010 y Windows 7 para la implementación de la propuesta[25], es posible que también haya utilizado C++ para la implementación del algoritmo. Tomando en cuenta las consideraciones expresadas anteriormente en cuanto a la mejora de tiempo que se conseguiría al trasladar la implementación de los algoritmos de MATLAB y Python (MLSHADE-SPA y SHADEILS respectivamente) existe una alta probabilidad de que los tiempos de respuesta de estos algoritmos también mejoren, lo que reduciría las distancias actuales entre SHADEILS y MAGA, situando al primero de estos en una muy buena posición con respecto a sus demás competidores.

Llegados a este punto es apropiado afirmar que las propuestas elegidas en este estudio han mostrado resultados competitivos. Aún así se puede concluir que son necesarias técnicas no más potentes pero si **más rápidas** para permitir su aplicación sobre problemas de optimización en ámbitos reales. El campo LSGO es un área de conocimiento que ha nacido relativamente hace poco y hasta hace apenas tres años, la gran mayoría de propuestas se remitían únicamente a los congresos, conferencias y competiciones, siendo estudiadas y evaluadas sobre benchmarks poco representativos del panorama del LSGO del mundo real.

Sin embargo, durante los últimos tres años se han propuesto una mayor cantidad de técnicas y algoritmos en este campo de estudio, lo que indudablemente promoverá su crecimiento y evolución de forma notable. Además, permitirá que sea solo cuestión de tiempo que se propongan técnicas mucho más sofisticadas y potentes que lleguen a ser aplicables a un problema real, repercutiendo en una mejora de la calidad de los procesos que requieren optimización y por ende afectando de forma positiva al entorno social donde son implantadas.

# Capítulo 9

# Conclusiones y posibles extensiones

En este trabajo se ha estudiado cómo es el rendimiento y la escalabildad de las propuestas más novedosas y potentes del panorama de optimización en alta dimensionalidad. Para lograr este objetivo, se ha utilizado un benchmark con una formulación de un problema totalmente distinto a los que se proponen en los benchmarks más utilizados actualmente: la optimización de los datos de un electroencefalograma en un ámbito médico real.

Se han estudiado el diseño de estas propuestas y se han adaptado las mismas para ser capaces de procesar la función objetivo del problema propuesto, lo que derivaría en la posterior obtención de los resultados, a través de un proceso de experimentación exhaustivo, que servirían de base para sustentar la posible utilidad de estas técnicas de cara a una implantación en un entorno real.

Los resultados han sido analizados y se concluye que las técnicas elegidas del ámbito *large scale global optimization* son muy robustas, escalables y potentes. Sin embargo, aún queda un largo camino por recorrer en materia LSGO, donde el principal inconveniente sigue siendo los excesivos tiempos de ejecución de estos algoritmos, aunque como se ha visto, distintas implementaciones utilizando distintas tecnologías podrían ser capaces de resolver estas deficiencias.

A criterio del alumno, no se conoce en la literatura actual del panorama de optimización en alta dimensionalidad un trabajo como el que se presenta en estas páginas. Es este precepto el que motivó la realización del mismo, donde se propone una hoja de ruta a seguir en cuanto a la optimización del problema del electroencefalograma se refiere, lo que puede derivar, ya sea a corto-medio o largo plazo, en una mejora sustancial de las técnicas empleadas actualmente para su optimización, lo que se traduciría en un uso

mucho más extensivo y eficiente en determinadas áreas de conocimiento y para determinadas aplicaciones que así lo requieran.

Como posibles extensiones de este trabajo se propone un ajuste exhaustivo y extensivo de los parámetros de cada algoritmo siguiendo el enfoque del benchmark estudiado en este trabajo, con vistas a conseguir una posible mejora de las soluciones obtenidas. De forma análoga, la inclusión de más benchmarks formulados a partir de problemas reales aumentaría la diversidad de los estudios, lo que podría llevar a mejorar el rendimiento de las técnicas actuales.

Ampliar el presente estudio con más técnicas también puede formar parte de una extensión futura, donde se añadan implementaciones en lenguajes más eficientes de cara a preparar las propuestas a un entorno real donde las restricciones temporales juegan un papel clave. Realizar una mejora del propio benchmark del EEG con el fin de que sea más representativo de un problema médico real, añadiendo más variables y mayor ruido, para crear mayor incertidumbre y forzar a que las técnicas que busquen su resolución sean cada vez más sofisticadas, se considera también una posible extensión de este trabajo.

Concluye así por tanto este estudio, donde el alumno agradece toda la ayuda proporcionada por la institución en cuanto a los recursos para llevar a cabo este trabajo, así como al tutor del mismo por el apoyo condicional y por guiar el buen desarrollo de este Trabajo Fin de Grado.

# Bibliografía


[1] R. Vigário. Extraction of ocular artefacts from EEG using independent component analysis. *Electroencephalography and Clinical Neurophysiology*, 103(3):395–404, 1997.

[2] S.K. Goh, K.C. Tan, A. Al-Mamun, and H.A. Abbass. Evolutionary Big Optimization (BigOpt) of signals. *2015 IEEE Congress on Evolutionary Computation, CEC 2015 - Proceedings*, (7257307):3332–3339, 2015.

[3] D. Molina and A. LaTorre. Special Session and Competition on Large-Scale Global Optimization, June 2018. `http://www.tflsgo.org/special_sessions/cec2018.html#benchmark-competition`.

[4] A. Wright. Genetic Algorithms for Real Parameter Optimization. *Foundations of Genetic Algorithms*, 1:205–218, 1991.

[5] J. Kennedy and R. Eberhart. Particle Swarm Optimization. *Proceedings of ICNN'95 - International Conference on Neural Networks*, pages 1942–1948, 1995.

[6] R. Storn and V. Price. Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997.

[7] M. Dorigo and G Di Caro. Ant Algorithms for Discrete Optimization. *Artificial Life*, 5(2):137–172, 1999.

[8] M. N. Omidvar and X. Li. Evolutionary Large-Scale Global Optimization - An Introduction. In *GECCO '17 Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 807–827, 2017.

[9] S.K. Goh, H.A. Abbass, and K.C. Tan. IEEE Congress on Evolutionary Computation - Optimization of Big Data 2015 Competition, March 2018. `http://www.husseinabbass.net/BigOpt.html`.

[10] J.J. Shih, D.J. Krusienski, and J.R. Wolpaw. Brain-Computer Interfaces in Medicine. *Mayo Clinic Proceedings*, 87(3):268–279, 2012.

[11] Elsevier's Scopus Database, February 2018. https://www.scopus.com/home.uri.

[12] Hercules Cluster - UGR, July 2018. http://citic.ugr.es/pages/informacion_general/equipamiento/cluster.

[13] Université de Namur. Slurm Quick Start Tutorial, July 2018. https://support.ceci-hpc.be/doc/_contents/QuickStart/SubmittingJobs/SlurmTutorial.html.

[14] Inc. A.D.A.M. What is an EEG?, April 2018. https://medlineplus.gov/ency/article/003931.htm.

[15] D. Saceda Corralo. Electroencefalograma (EEG), March 2018. https://www.webconsultas.com/pruebas-medicas/electroencefalograma-eeg-12529.

[16] S.K. Goh, H.A. Abbass, K.C. Tan, and A. Al Mammun. Artifact Removal from EEG Using a Multi-objective Independent Component Analysis Model. *Neural Information Processing*, pages 570–577, 2014.

[17] P.A. de Madeiros Kanda, R. Anghinah, M. Taino Smidth, and J.M. Silva. The clinical use of quantitative EEG in cognitive disorders. *Dementia and Neuropsychologia*, 3(3):195–203, 2009.

[18] H.A. Abbass, J. Tang, R. Amin, M. Ellejmi, and S. Kirby. Augmented Cognition using Real-time EEG-based Adaptive Strategies for Air Traffic Control. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting. SAGE*, 8836, 2014.

[19] H.A Abbass. Calibrating Independent Component Analysis with Laplacian Reference for Real-Time EEG Artifact Removal. *Neural Information Processing*, pages 68–75, 2014.

[20] IEEE: Congress on Evolutionary Computation, August 2018. https://ieeexplore.ieee.org/servlet/opac?punumber=1000284.

[21] CEC: Congress on Evolutionary Computation Special Session / Competition, August 2018. http://www.ntu.edu.sg/home/epnsugan/index_files/cec-benchmarking.htm.

[22] IEEE. IEEE Congress on Evolutionary Computation 2015, Sendai, Japan, April 2018. http://sites.ieee.org/cec2015/.

[23] Q. Zhang and H. Li. MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition. *IEEE Transactions on Evolutionary Computation*, 11(6):712–731, 2007.

[24] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[25] Y. Zhang, M. Zhou, Z. Jiang, and J. Liu. A multi-agent Genetic Algorithm for Big Optimization Problems. *2015 IEEE Congress on Evolutionary Computation, CEC 2015 - Proceedings*, (7256959):703–707, 2015.

[26] W. Zhong, J. Liu, M. Xue, and L. Jiao. A Multiagent Genetic Algorithm for Global Numerical Optimization. *IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics*, 34(2):1128–1141, 2004.

[27] A. LaTorre, S. Muelas, and J.M. Peña. A comprehensive comparison of large scale global optimizers. *Information Sciences*, 316:517–549, 2015.

[28] A. LaTorre, S. Muelas, and J.-M. Peña. A MOS-based dynamic memetic differential evolution algorithm for continuous optimization: a scalability test. *Soft Computing*, 15(11):2187–2199, 2010.

[29] L.-Y. Tseng and C. Chen. Multiple Trajectory Search for Large Scale Global Optimization. *Proceedings of the 10th IEEE congress on evolutionary computation, CEC 2008 (IEEE World Congress on Computational Intelligence)*, (4631210):3052–3059, 2008.

[30] Z. Yang, K. Tang, and X. Yao. Scalability of generalized adaptive differential evolution for large-scale continuous optimization. *Soft Computing*, 15(11):2141–2155, 2011.

[31] A. LaTorre, S. Muelas, and J.M. Peña. Large Scale Global Optimization: Experimental Results with MOS-based Hybrid Algorithms. *2013 IEEE Congress on Evolutionary Computation, CEC 2013*, (6557901):2742–2149, 2013.

[32] F.J. Solis and R.J.B Wets. Minimization by Random Search Techniques. *Mathematics of Operations Research*, 1981.

[33] J. Liu and K. Tang. Scaling Up Covariance Matrix Adaptation Evolution Strategy Using Cooperative Coevolution. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, (8206):350–357, 2013.

[34] X. Li, K. Tang, Z. Yang, and D. Molina. 2015 IEEE Congress on Evolutionary Computation Competition on: Large Scale Global Optimization, August 2018. `http://titan.csit.rmit.edu.au/~e46507/lsgo-competition-cec15/`.

[35] D. Molina, A. LaTorre, and F. Herrera. SHADE with Iterative Local Search for Large-Scale Global Optimization. *2018 World Congress on Computational Intelligence (WCCI-2018), 2018 IEEE Conference on Evolutionary Computation (IEEE CEC'2018)*, pages 1252–1259, 2018.

[36] A.A. Hadi, A.W. Mohamed, and K.M. Jambi. LSHADE-SPA Memetic Framework for Solving Large Scale Problems. *2018 World Congress on Computational Intelligence (WCCI-2018), 2018 IEEE Conference on Evolutionary Computation (IEEE CEC'2018)*, 2018.

[37] D. Molina and F. Herrera. Iterative hybridization of DE with local search for the CEC'2015 special session on large scale global optimization. *2015 IEEE Congress on Evolutionary Computation, CEC 2015 - Proceedings*, (7257127):1974–1978, 2015.

[38] R. Tanabe and A. Fukunaga. Success-history based parameter adaptation for Differential Evolution. *2013 IEEE Congress on Evolutionary Computation, CEC 2013*, (6557555):71–78, 2013.

[39] J. Brest, S. Greiner, B. Boskovic, M. Mernik, and V. Zumer. Self-Adapting Control Parameters in Differential Evolution: a Comparative Study on Numerical Benchmark Problems. *IEEE Transactions on Evolutionary Computation*, 10(6):646–657, 2006.

[40] J.L. Morales and J. Nocedal. Remark on algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound constrained optimization. *ACM Transactions on Mathematical Software*, 38(1), 2011.

[41] M.A. Potter and K.A De Jong. A cooperative coevolutionary approach to function optimization. *Proc. Int. Conf. Parallel Problem Solving from Nature*, 2:249–257, 1994.

[42] M.N. Omidvar, X. Li, Y. Mei, and X. Yao. Cooperative Co-Evolution With Differential Grouping for Large Scale Optimization. *IEEE Transactions on Evolutionary Computation*, 18(3):378–393, 2014.

[43] M. Yi, L. Xiaodong, Y. Xin, and M.N. Omidvar. A competitive divide-and-conquer algorithm for unconstrained large-scale black-box optimization. *ACM Transactions on Mathematical Software*, 42(2), 2016.

[44] Y. Sun, M. Kirley, and S.K. Halgamuge. Extended differential grouping for large scale global optimization with direct and indirect variable interactions. *GECCO 2015 - Proceedings of the 2015 Genetic and Evolutionary Computation Conference*, pages 313–320, 2015.

[45] M.N. Omidvar, M. Yang, Y Mei, X. Li, and X. Yao. DG2: A Faster and More Accurate Differential Grouping for Large-Scale Black-Box Opti-

mization. *IEEE Transactions on Evolutionary Computation*, 21(6):929–942, 2017.

[46] A. LaTorre. A framework for hybrid dynamica evolutionary algorithms: Multiple Offspring Sampling. Ph.d. thesis, Universidad Politécnica de Madrid, Noviembre 2009.

[47] E.G. Talbi. A Taxonomy of Hybrid Metaheuristics. *Journal of Heuristics*, 8(5):541–564, 2002.

[48] H. Wang, Z. Wu, S. Rahnamayan, and L. Kang. A Scalability Test for Accelerated DE Using Generalized Opposition-Based Learning. *ISDA 2009 - 9th International Conference on Intelligent Systems Design and Applications*, (5364196):1090–1095, 2009.

[49] R. Tanabe and A.S. Fukunaga. Improving the search performance of SHADE using linear population size reduction. *Proceedings of the 2014 IEEE Congress on Evolutionary Computation, CEC 2014*, (6900380):1658–1665, 2014.

[50] A.W. Mohamed, A.A. Hadi, A.M. Fattouh, and K.M Jambi. LSHADE with Semi-Parameter Adaptation Hybrid with CMA-ES for Solving CEC 2017 Benchmark Problems. *2017 IEEE Congress on Evolutionary Computation, CEC 2017 - Proceedings*, (7969307):145–152, 2017.

[51] A.W. Mohamed and A.S. Almazyad. Differential Evolution with Novel Mutation and Adaptive Crossover Strategies for Solving Large Scale Global Optimization Problems. *Applied Computational Intelligence and Soft Computing*, (7974218), 2017.

[52] A.W. Mohamed. Solving large-scale global optimization problems using enhanced adaptive differential evolution algorithm. *Complex Intell. Syst.*, 3(4):205–231, 2017.

[53] J. Zhang and A.C. Sanderson. JADE: Adaptive differential evolution with optional external archive. *IEEE Transactions on Evolutionary Computation*, 13(5):945–958, 2009.

[54] D. Molina. Código fuente de las propuestas del estudio: A comprehensive comparison of large scale global optimizers, 2018. `https://www.dropbox.com/sh/1rz9l51dd6zhzd6/AACHO6zeAPi9YiYU1VLafa-La?dl=0`.

[55] M. Wall. A C++ Library of Genetic Algorithm Components, 1999. `http://lancet.mit.edu/ga/`.

[56] D. Molina, A. LaTorre, and F. Herrera. SHADE with Iterative Local Search for Large-Scale Global Optimization. `http://www.tflsgo.org/special_sessions/cec2018.html#benchmark-competition`.

[57] Python Software Foundation. Herramienta recomendada para instalar paquetes Python, 2018. `https://pypi.org/project/pip/`.

[58] Python Software Foundation. Lenguaje de creación de extensiones de C para Python, 2018. `https://pypi.org/project/Cython/`.

[59] M.N. Omidvar. Código fuente de la propuesta DG2, 2018. `https://bitbucket.org/mno/differential-grouping2/src/f2192aab33f2?at=master`.

[60] X. Li, K. Tang, M.N. Omidvar, Z. Yang, and K. Qin. Benchmark Functions for the CEC'2013 Special Session and Competition on Large-Scale Global Optimization. *IEEE CEC 2013*, 2013, Updated 2018.

# Apéndice A

# Gráficas de convergencia del estudio de escalabilidad

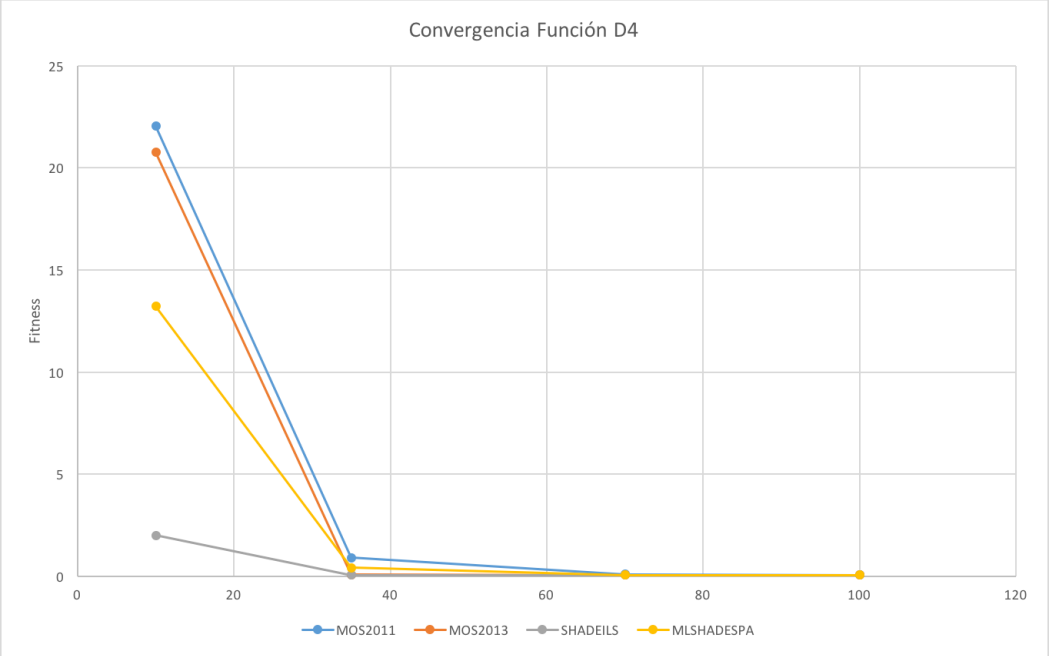


Figura A.1: Convergencia de los 4 algoritmos: problema D4

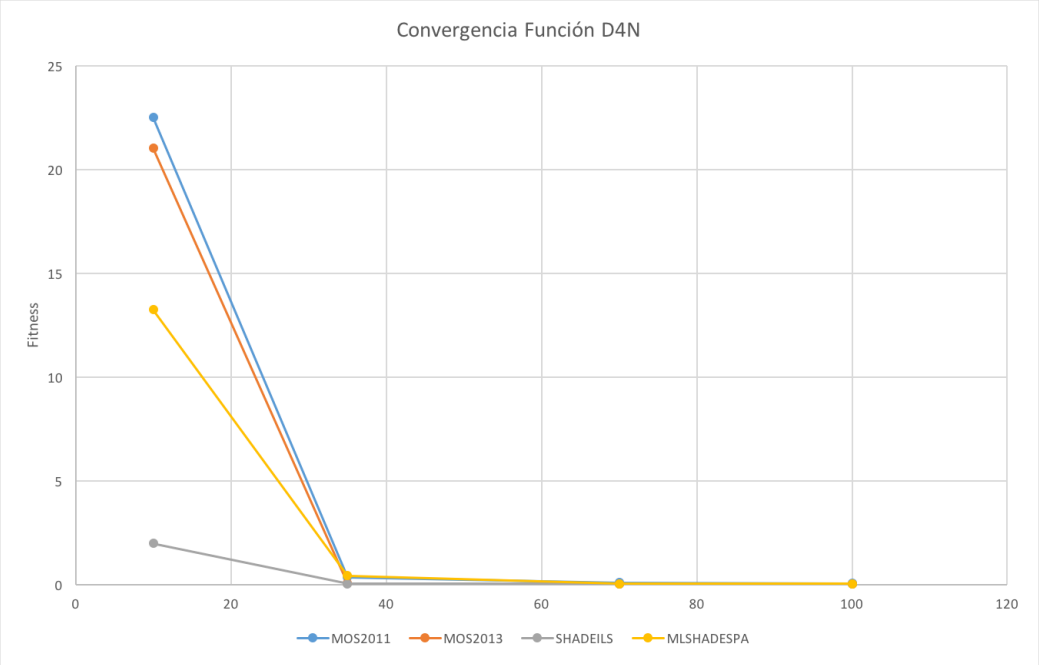


Figura A.2: Convergencia de los 4 algoritmos: problema D4N

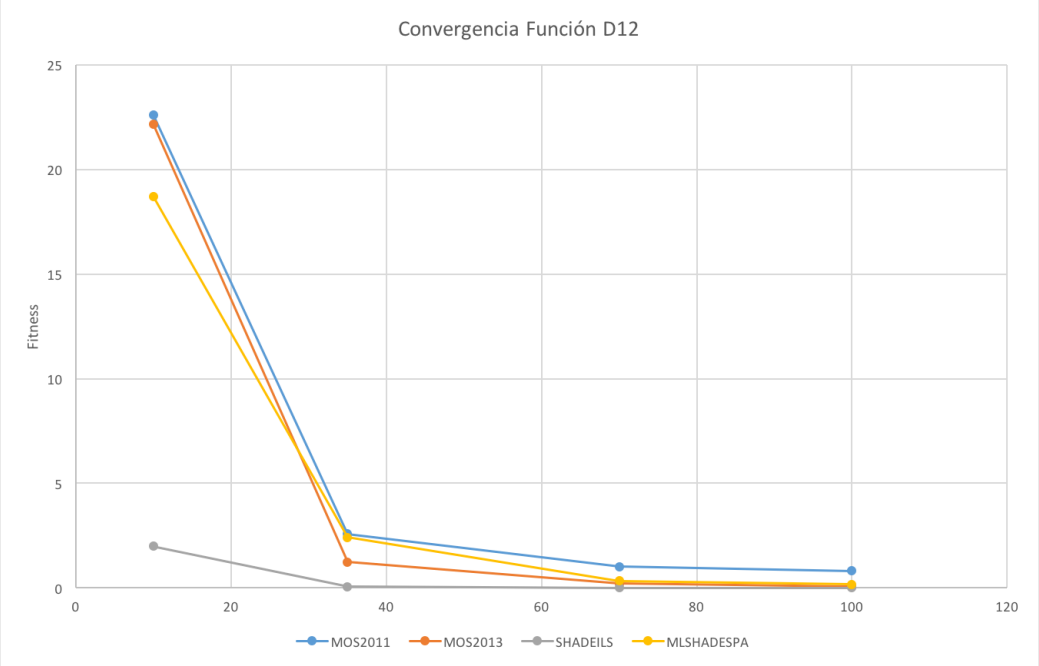


Figura A.3: Convergencia de los 4 algoritmos: problema D12

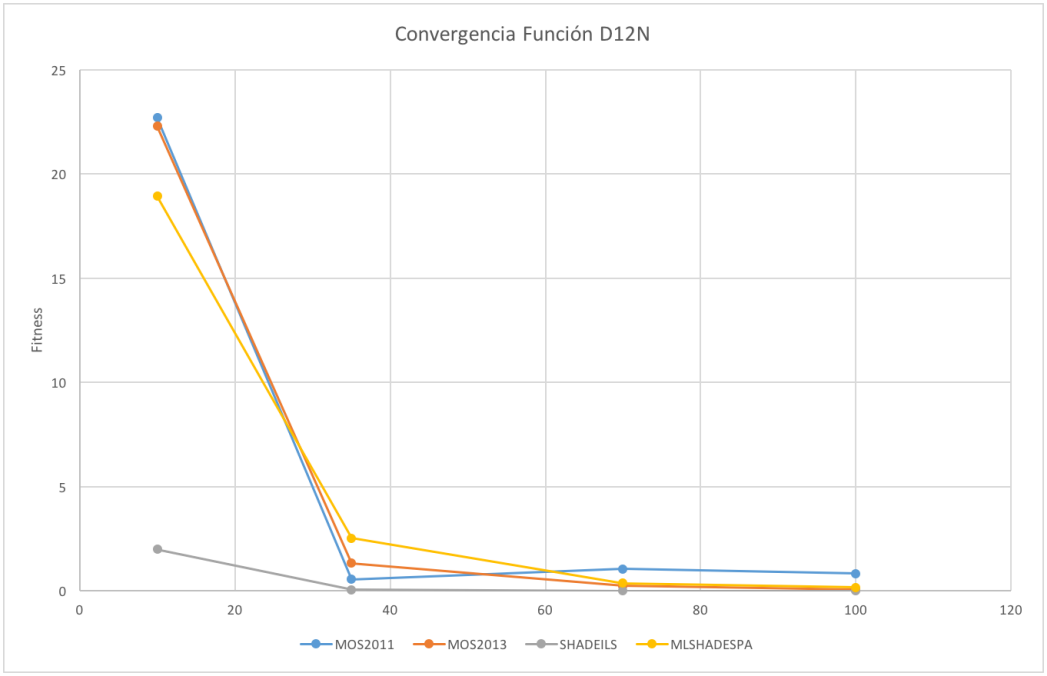


Figura A.4: Convergencia de los 4 algoritmos: problema D12N

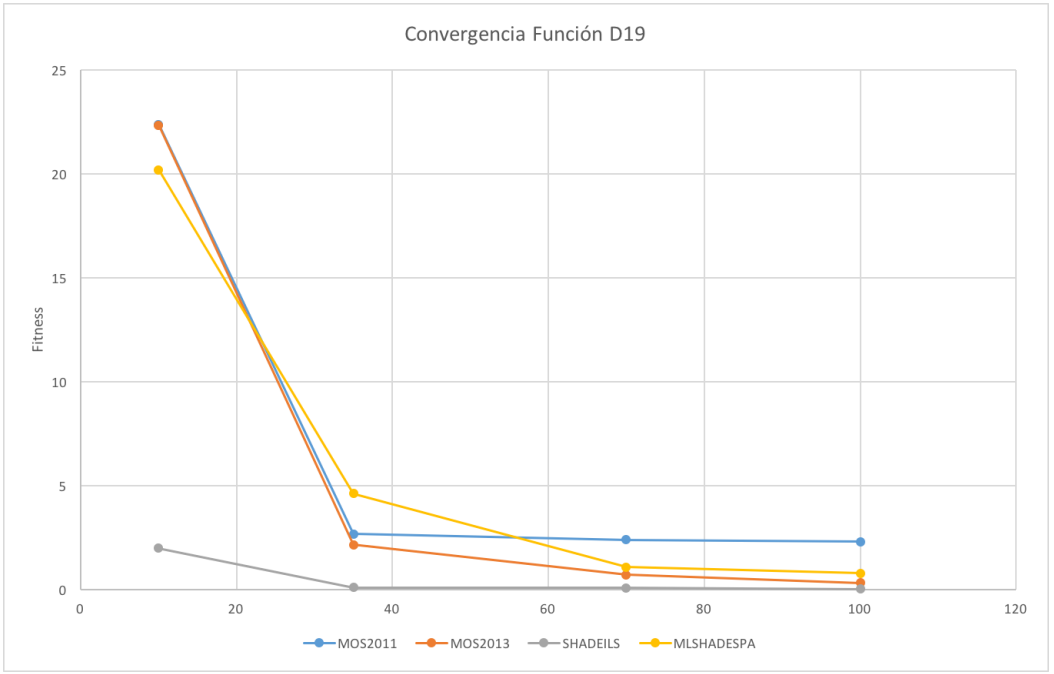


Figura A.5: Convergencia de los 4 algoritmos: problema D19

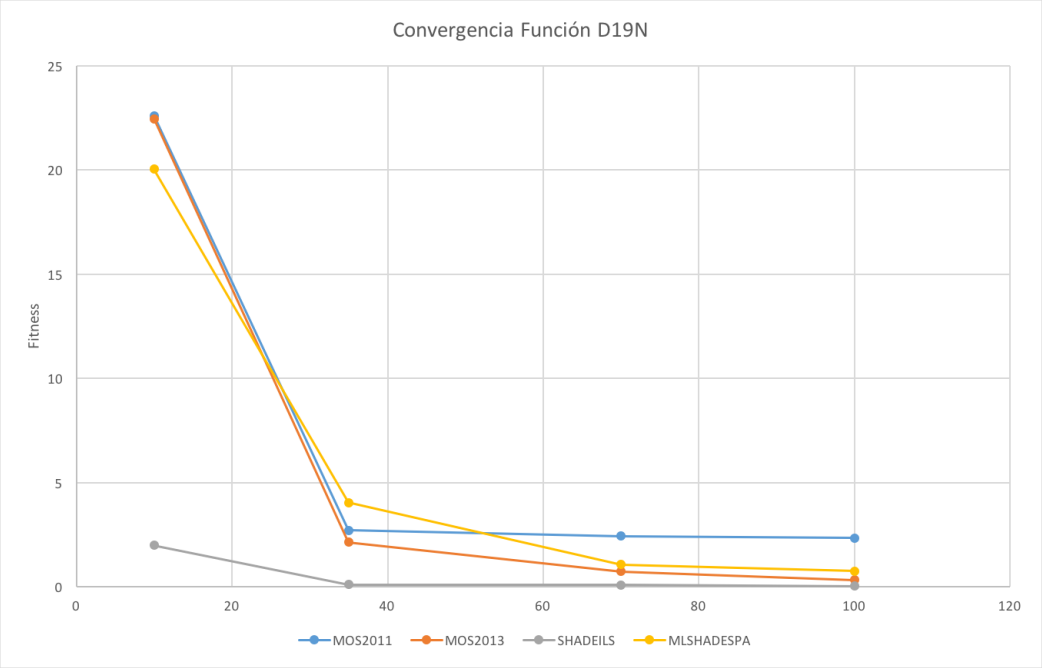


Figura A.6: Convergencia de los 4 algoritmos: problema D19N