Trabajo de Fin de Grado

Desarrollo de Algoritmos Dirigido por Retos

Challenge-driven algorithms development.

Alejandro Marrero Díaz

La Laguna, 1 de julio de 2017

D. **Eduardo Manuel Segredo González**, con N.I.F. 78.564.242-Z Profesor Asociado adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

D. **Carlos Segura González**, con N.I.F. 78.704.244-S Investigador Asociado adscrito al Departamento de Computación del Centro de Investigación en Matemáticas (CIMAT), Guanajuato, México, como cotutor.

C E R T I F I C A N

Que la presente memoria titulada:

"*Desarrollo de Algoritmos Dirigido por Retos.*"

ha sido realizada bajo su dirección por D. **Alejandro Marrero Díaz**, con N.I.F. 78.649.404-F.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 1 de julio de 2017

Agradecimientos

A mi familia y amigos que
me han apoyado incondicionalmente estos cuatro años.

Agradecer también a los directores de este Trabajo de Fin de Grado,
Eduardo Segredo y Carlos Segura,
por la ayuda en el desarrollo del mismo
y por adentrarme en el mundo de la investigación.

Licencia

Resumen

Con este Trabajo de Fin de Grado se ha buscado adentrar al alumno en el mundo de la investigación, y concretamente, en el campo de las meta-heurísticas. Para ello, se han implementado diversas técnicas algorítmicas, incluyendo algunas técnicas pertenecientes a la familia de los algoritmos evolutivos, así como otras técnicas meta-heurísticas poblacionales y de trayectoria.

Para evaluar el rendimiento de los algoritmos implementados, se ha participado en una competición de optimización global continua denominada Generalization-based Contest in Global Optimization (GenOpt).

En concreto, se han implementado los algoritmos Opposition-Based-Learning Competitive-Particle Swarm Optimization, Covariance Matrix Adaptation Evolutionary Strategy y Simulated Annealing los cuales se han aplicado a los problemas propuestos por la competición GenOpt. La elección de estos algoritmos se basa en la demostración de su buen desempeño en problemas de optimización global continua. Además de los tres algoritmos mencionados, se ha desarrollado el conjunto de métricas, especificadas en el manifiesto de la competición, necesarias para evaluar el rendimiento de los algoritmos desarrollados.

Palabras clave: Optimización Global Continua, Meta-heurísticas, Computación Evolutiva, Ajuste de Parámetros.

Abstract

The main goal of this degree thesis is to introduce the student to the research field through the application of several meta-heuristic approaches to global continuous optimization problems. For doing that, we have developed several algorithmic techniques, including some methods belonging to the family of evolutionary algorithms, as well as other population and trajectory-based meta-heuristics.

In order to assess the performance of the implemented schemes, we have participated in a competition on global continuous optimization referred to as Generalization-based Contest in Global Optimization (GENOPT). The main aim of this competition is to diversify the number of approaches solving the proposed problems.

In particular, we have implemented the algorithms Opposition-based Learning Competitive Particle Swarm Optimization, Covariance Matrix Adaptation Evolution Strategy and Simulated Annealing, which have been applied to solve the problems proposed by the GENOPT. The aforementioned algorithms were chosen because they have shown to provide better competitive performance in comparison to other alternatives when dealing with global continuous optimization problems. In addition to the above, we have also implemented a set of metrics proposed by the competition organization with the aim of measuring the performance of our algorithmic proposals.

Keywords: *Global Continuos Optimization, Meta-heuristics, Evolutionary Computation, Parameter Setting.*

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Descripción del Trabajo de Fin de Grado

En este Trabajo de Fin de Grado, se propone el diseño y análisis de diversas técnicas algorítmicas para la resolución de problemas definidos en el ámbito de un concurso/competición de optimización. En la actualidad, muchos de los problemas del mundo real se pueden formular como problemas de optimización con variables de decisión cuyos valores varían en un dominio continuo. Es por eso que podemos encontrar una gran cantidad de competiciones en esta materia que fomentan la cooperación y la investigación en el campo de la computación. Aunque son muchas las técnicas que pueden ser empleadas para resolver este tipo de problemas, se ha probado que la utilización de computación evolutiva y otras meta-heurísticas presentan numerosas y exclusivas ventajas como robustez y fiabilidad, capacidad de búsqueda global y abstracción del dominio del problema a resolver.

Además de las ventajas anteriormente mencionadas, las técnicas de computación evolutiva nos proporcionan otras características como pueden ser la facilidad con la que pueden ser implementadas y la posibilidad de paralelizarlas de un modo relativamente sencillo. Por lo tanto, el principal objetivo del presente Trabajo de Fin de Grado es el diseño, desarrollo y análisis de la parametrización de diversos algoritmos evolutivos y otras técnicas metaheurísticas en el ámbito de una competición de optimización continua.

1.2. Antecedentes y Estado Actual del Tema

Como se ha indicado en el apartado anterior, actualmente existen una gran cantidad de competiciones de optimización organizadas en diferentes congresos como son:

- **Congress on Evolutionary Computation - CEC 2017**: es un congreso anual sobre computación evolutiva que además propone varias com-

peticiones como Real-Parameter Single Objective Optimization o Evolutionary Multi-task Optimization entre otros.

- **Genetic and Evolutionary Computation Conference - GECCO 2017**: propone varias competiciones como son Black Box Optimization Competition u Optimisation of Problems with Multiple Interdependent Components entre otros [1].

- **Global Trajectory Optimisation Competition - GTOC**: es una competición de optimización global de trayectoria en el cual se plantea un problema de optimización a resolver diferente en cada edición [2].

- **Generalization-based Contest in Global Optimization - GenOpt**: concurso de optimización global continua en el que cada año se propone un conjunto de funciones a optimizar.

En concreto, este Trabajo de Fin de Grado se basará en la resolución de los problemas de optimización global continua descritos en GenOpt. En esta competición se propone la minimización de 18 funciones con 10 o 30 variables de decisión, las cuales se dividen en tres familias diferentes: funciones GKLS, funciones de benchmark clásicas transformadas, y funciones compuestas. La primera familia de funciones se obtiene mediante un generador GKLS [6] que permite obtener tres clases de funciones con los valores mínimos locales y globales conocidos para realizar una optimización global multidimensional de tipo "box-constrained", dónde el espacio de búsqueda se encuentra definido por dos límites, un límite inferior y otro límite superior, para cada una de las variables de decisión. La segunda familia de funciones se basan en problemas de benchmark de optimización continua clásicos, de dificultad variable.
Por último, la última familia se obtiene realizando una composición de funciones pertenecientes a la segunda familia, generando seis tipos diferentes de problemas. Se puede obtener más información al respecto en el Manifesto GenOpt [3].
Por otra parte, como se comentó en el apartado de introducción, se estudiarán diversos algoritmos evolutivos y otro tipo de metaheurísticas para la resolución de los problemas planteados. En este caso, cabe destacar los algoritmos Particle Swarm Optimization (PSO) [21], Differential Evolution (DE) [3] y Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [10], entre otros. Otro posible objeto de estudio serán diversas técnicas de inicialización de algoritmos, principalmente, la familia de técnicas Opposition-Based Learning (OBL) [23].

[1] Para obtener más información acerca de GECCO http://gecco-2017.sigevo.org/index.html/HomePage

[2] Para obtener más información acerca de GTOC https://sophia.estec.esa.int/gtoc_portal/

[3] Dirección desde la que se puede obtener el Manifesto del concurso GenOpt: http://www.genopt.org/genopt.pdf.

Capítulo 2

Conceptos previos

2.1. Optimización Global

La Optimización Global trata de optimizar una función dentro de un intervalo especificado y teniendo en cuenta diversos criterios y/o restricciones, como puede ser el caso de optimización multiobjetivo. Por lo general, el objetivo de la optimización global es la de encontrar el óptimo (mínimo o máximo) u óptimos globales de la función f a optimizar, tratando de evitar posibles óptimos locales. La optimización global se basa en encontrar aquel valor mínimo o máximo global dentro del espacio de búsqueda de la función $\boldsymbol{f}$ a optimizar. La elección de la técnica empleada para encontrar el óptimo global resulta de gran importancia dado que la función $\boldsymbol{f}$ puede tener varios óptimos locales que pueden llevar al algoritmo a **converger prematuramente**, sin que luego pueda escapar de los mismos.
De manera formal, el objetivo de la optimización global, considerando un problema de minimización, es encontrar un vector $X* \in \Omega$ tal que $f(X*) \leq f(X)$ para todo $X \in \Omega$ [25]. En este caso particular, tratamos la optimización de tipo ***box-constrained***, donde **el espacio de búsqueda** Ω está definido por un límite inferior (a_i) y superior (b_i) para cada una de las variables de decisión de la función, es decir: $\Omega = \prod_{i=1}^{D}[a_i, bi]$, siendo D el número de variables de decisión del problema a optimizar [25].

2.2. Técnicas Meta-heurísticas

Las meta-heurísticas son un sub-conjunto de métodos de optimización que se encuentran incluidas dentro del conjunto de métodos heurísticos que a su vez forman parte de los métodos aproximados.

A diferencia de los métodos de optimización exactos, las meta-heurísticas nos permiten obtener soluciones factibles a problemas complejos en un tiempo aceptable, aunque no nos garantizan obtener la solución óptima global [27]. En los últimos años estás técnicas algorítmicas han ganado una gran popularidad

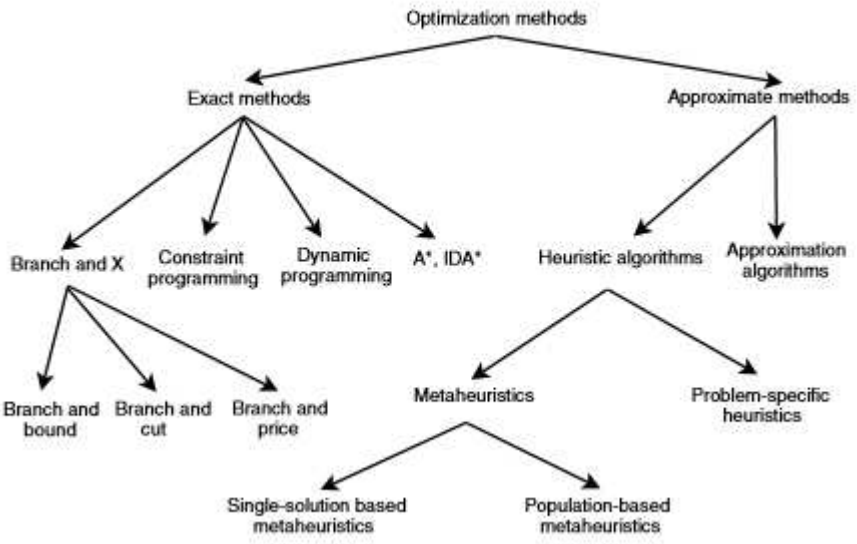

Figura 2.1: Métodos de optimización.

dado que han demostrado su efectividad y eficiencia en muchos campos y con una gran cantidad de problemas distintos.

En cuanto a las distintas meta-heurísticas que podemos encontrar hoy en día, podemos diferenciar las siguientes categorías [27]:

- **Búsquedas Locales**: Greedy Randomized Adaptive Search Procedure (GRASP) [4], Variable Neighborhood Search (VNS) [14].
- **Heurísticas Voraces**: Simulated Annealing (SA) [7].
- **Algoritmos Evolutivos**: Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) [10], Differential Evolution (DE) [30, 5, 28], Coevolutionary Algorithms (CEA) [2, 11, 9].

Esta gran variedad de técnicas se debe principalmente al gran abanico de criterios que podemos definir a la hora de diseñar una meta-heurística. Generalmente, las meta-heurísticas, y en especial, los algoritmos evolutivos, buscan un balance entre las propiedades de intensificación y diversificación. El concepto de intensificación significa aprovechar una solución prometedora obtenida y tratar de explotar la región del espacio de búsqueda cercana a la misma en busca de posibles mejores soluciones, mientras que la diversificación se caracteriza por priorizar la búsqueda por las diferentes áreas no exploradas del espacio de búsqueda. Sin embargo, algunas técnicas como la búsqueda local, se centran principalmente en intensificar. Otra clasificación agrupa las diferentes meta-heurísticas en las siguientes clases [3]:

- **Métodos inspirados en la naturaleza**: es común inspirarse en el comportamiento de animales o bien de procesos físicos para diseñar un método

meta-heurístico que emule ese comportamiento. Por ejemplo: Ant Bee Colonies [19] y Simulated Annealing [7].

- **Determinísticos o Estocásticos**: podemos optar por tomar decisiones deterministas o emplear reglas aleatorias durante la búsqueda de nuevas soluciones.
- **Métodos basados en poblaciones o basados en una única solución**: por un lado nos podemos basar en un conjunto de soluciones factibles que combinaremos entre ellas aplicando diversos operadores para obtener un nuevo conjunto de soluciones potencialmente mejores. O bien, podemos optar por utilizar una única solución al problema que transformaremos en cada iteración del algoritmo.
- **Iterativos o voraces**: en este aspecto podemos contemplar comenzar con un conjunto de soluciones factibles al problema y transformar dichas soluciones en cada iteración del algoritmo para obtener nuevas soluciones con mejor evaluación. Al contrario, en una estrategia voraz (Greedy) comenzamos sin una solución factible al problema y en cada iteración incluiremos una variable de decisión hasta obtener una solución completa al problema [3].

2.2.1. Representación

En las técnicas algorítmicas en general nos podemos encontrar con diversas maneras de representar la solución S al problema y en las meta-heuristicas las siguientes son las más comunes [27]:

- **Cadena binaria**: utilizada en problemas de decisiones.

Figura 2.2: Ejemplo de representación en cadena binaria.

- **Vector de valores naturales**: utilizada en problemas de optimización combinatoria.
- **Vector de números reales**: utilizada en problemas de optimización continua.

Figura 2.3: Ejemplo de representación en vector de números naturales.

Figura 2.4: Ejemplo de representación en vector de números reales.

Figura 2.5: Ejemplo de representación en permutaciones.

- **Permutaciones**: empleada, por ejemplo en problemas de rutas y de planificación de tareas.

Dadas las características del concurso seleccionado para este Trabajo de Fin de Grado, la representación empleada fue **vector de valores reales** dado que cada elemento del vector denota un valor dentro del dominio de la función para cada una de las ***D variables*** que posee.

2.2.2. Condición de Parada

Generalmente, las técnicas meta-heurísticas finalizan su ejecución según un criterio de finalización o **condición de parada**. Las más comunes son:

- **Iteraciones**: el algoritmo se detiene al alcanzar un determinado número de iteraciones.
- **Evaluaciones**: cuando el número de evaluaciones de la función objetivo a optimizar realizadas por el algoritmo llega a un valor prefijado, el algoritmo finaliza su ejecución.
- **Factor de error**: al alcanzar un factor de error lo suficientemente bajo como para considerar la solución óptima, el algoritmo detiene su ejecución.

En nuestro trabajo, el criterio de parada utilizado por todos los algoritmos desarrollados es 10^6 **evaluaciones**, criterio prefijado por la organización del concurso GenOpt.

Capítulo 3

Técnicas Algorítmicas desarrolladas

3.1. Opposition-based Learning (OBL)

Opposition-based Learning (OBL) [29, 23, 15, 20] es un concepto en computación que ha demostrado gran efectividad a la hora de mejorar diversas técnicas de optimización. Cuando evaluamos una solución X, perteneciente al conjunto de soluciones S, candidata para un problema dado, simultáneamente calcularemos la solución opuesta $\overline{X}$, consiguiendo así una mayor exploración del espacio de búsqueda Ω en busca del óptimo global [29]. Asimismo, podemos encontrar diversas variantes a la estrategia tradicional de OBL como pueden ser Quasi-opposition-based Learning (QOBL) o Quasi-reflected Opposition-based Learning (QROBL) [25].

Sea $x \in \Re$ un número real definido dentro de un cierto intervalo: $x \in [a, b]$. El número opuesto de x denotado como $\overline{x}$ se define de la siguiente forma [29]:

$$\overline{x} = a + b - x \tag{3.1}$$

Análogamente, para el caso de optimizar una función con D dimensiones o variables de decisión, la solución opuesta se calcularía de la siguiente forma:

Sea $P(x_1, x_2, ..., x_D)$ un punto dentro de un sistema de coordenadas $D-$*dimensional* con $x_1, ..., x_D \in \Re$ y además $x_i \in [a_i, bi]$ [29]. El opuesto del punto P se define como las coordenadas $\overline{x_1}, ...\overline{x_D}$ donde:

$$\overline{x_i} = a_i + b_i - x_i \quad i = 1, ..., D \tag{3.2}$$

En nuestro desarrollo hemos empleado este concepto en varios procesos, como puede ser el proceso de inicialización de los individuos que formarán parte de la población inicial de soluciones S o bien, tras cada iteración del algoritmo como medida de diversificación.

3.2. Búsqueda Global

En ciencias de la computación, una búsqueda global (Global Search) es un método heurístico para resolver problemas complejos de optimización [26, 1, 22]. Toda búsqueda global debe tener un componente que permita explorar nuevas regiones del espacio de búsqueda de la función a optimizar, intentando así, escapar de los óptimos locales y manteniendo un equilibrio entre diversificación e intensificación.

Para nuestro trabajo, hemos diseñado una búsqueda global que **se aplicará como último paso en cada iteración de los algoritmos** que hemos desarrollado con el objetivo de obtener una mayor diversificación y escapar de los posibles óptimos locales. En primer lugar, el procedimiento calculará el individuo centroide de la población. El centroide de un conjunto de k elementos, tal que $k = |S|$, se define como:

$$C = \frac{x_1 + x_2 + ... + x_k}{k} \tag{3.3}$$

Para nuestro caso particular, debemos tener en cuenta que cada uno de nuestros elementos x_i representan una solución factible a nuestro problema con D variables. Es por ello que, el individuo centroide se calculará de la siguiente manera:

Algoritmo 1 Cálculo del centroide()

1: **para** $i \leftarrow 0$ hasta D **hacer**
2: $Suma = 0$;
3: **para** $j \leftarrow 0$ hasta $|S|$ **hacer**
4: $Suma = Suma + S[i][j]$;
5: **fin para**
6: Centroide[i] = $\frac{Suma}{|S|}$;
7: **fin para**
8: **return** Centroide

Seguidamente, pasaremos a explorar la población de individuos aplicando pequeñas modificaciones a cada uno de ellos para obtener nuevos individuos *hijos*. Si estos individuos mejoran a sus ancestros serán incluidos dentro del conjunto de soluciones S y continuaremos aplicando modificaciones a su ancestro hasta que no se produzca mejora en su descendencia. En otro caso, serán descartados. Finalmente, devolveremos una nueva población con los $|S|$ mejores individuos encontrados entre padres e hijos.
A continuación, podemos ver el pseudocódigo de nuestra aproximación de búsqueda global empleada en nuestros desarrollos:

Algoritmo 2 Búsqueda global()

1: NumIndividuos = $|S|$;
2: OrdenarPoblacion(S);
3: MarcarNoExplorados(S);
4: Centroide = CalcularCentroide();
5: NumeroMejora = 0;
6: NumeroExplorado = 0;
7: **mientras** $NumeroMejora > 0$ y $NumeroExplorado < |S|$ **hacer**
8: k = 0;
9: **mientras** $S[k] = explorado$ y $NumeroExplorado < |S|$ **hacer**
10: k = rand(0, $|S|$); Buscamos un individuo k no explorado
11: **fin mientras**
12: S[k] = explorado;
13: NumeroExplorado = NumeroExplorado + 1;
14: Mejora = true;
15: **mientras** $Mejora = true$ **hacer**
16: **mientras** $|a_1| + |a_2| + |a_3| \neq 1$ **hacer**
17: GenerarRand(a1, a2, a3); Obtenemos tres valores generados aleatoriamente de manera uniforme en el rango [0, 1] para realizar la modificación del individuo
18: **fin mientras**
19: **mientras** $r_1 < k$ **hacer**
20: r_1 = rand(0, $|S|$); Buscamos un individuo r_1 aleatoriamente para usar sus variables en la generación del nuevo individuo
21: **fin mientras**
22: NuevoInd = ModificarIndividuo(k, a1, a2, a3, Centroide, r_1);
23: **si** $NuevoInd < S[k]$ **entonces**
24: $Mejora = true$;
25: $S = S \cap NuevoInd$;
26: NumeroMejora = NumeroMejora + 1;
27: **en otro caso**
28: Mejora = false;
29: **fin si**
30: **fin mientras**
31: **fin mientras**
32: OrdenarPoblacion(S);
33: S = ObtenerMejores(0, NumIndividuos, S);
34: **return** $|S|$ mejores individuos encontrados

3.3. OBL Competitive Particle Swarm Optimization (OBL-CPSO)

3.3.1. Particle Swarm Optimization

Particle Swarm Optimization (PSO) [3, 16, 18, 21] es una estrategia de optimización que ha demostrado ser muy eficiente en problemas de optimización global continua. Se basa en simular el conjunto de soluciones candidatas a un problema dado como un enjambre de partículas que se mueven dentro de un espacio de búsqueda definido. Dado que las partículas se mueven, estas poseen una posición x y una velocidad $\overrightarrow{v}$ en cada instante t de ejecución del algoritmo. La posición x representa los valores que toman las diferentes variables dentro de una solución y, la velocidad $\overrightarrow{v}$, determinará la dirección y la rapidez con la que la partícula se desplazará por el espacio de búsqueda, es decir, el factor de modificación para cada una de las variables de decisión. Aparte de estas características, cada partícula tiene la capacidad de recordar su mejor posición alcanzada durante la ejecución del algoritmo, y para cada partícula p_i la denotaremos como pb_i [3]. Esto nos permite evaluar si el movimiento de una partícula mejora la solución actual o por el contrario es mejor permanecer en la posición actual.

Por otra parte, en el algoritmo PSO también se tiene en cuenta una partícula denominada *global best* (gb) en la que se almacena la mejor posición histórica alcanzada por una partícula dentro del enjambre. Estas dos posiciones pb_i y gb son usadas en cada iteración del algoritmo para atraer el movimiento de las partículas hacia esas posiciones, consiguiendo así que el algoritmo PSO converja rápidamente hacia las posibles soluciones óptimas.
A continuación, se muestra el esquema básico del algoritmo PSO:

Algoritmo 3 Particle Swarm Optimization()

1: **mientras** Condición de parada no satisfecha **hacer**
2: **for all** p_i en S **hacer**
3: Evaluar p_i;
4: Actualizar mejor posición pb_i;
5: Actualizar mejor global gb;
6: **fin para**
7: **for all** p_i en S **hacer**
8: **for all** d_i en D **hacer**
9: $v_{i,d} = v_{i,d} + C_1 * Rnd(0,1) * [pb_{i,d} - x_{i,d}] + C_2 + Rnd(0,1) * [gb_d - x_{i,d}]$; Rnd(0,1) devuelve un número generado aleatoriamente en el rango [0, 1] $x_{i,d} = x_{i,d} + v_{i,d}$;
10: **fin para**
11: **fin para**
12: **fin mientras**
13: **return** Mejor solución obtenida

3.3.2. OBL-CPSO

En esta ocasión, el algoritmo Opposition-based Learning Competitive Particle Swarm Optimization (OBL-CPSO) [31] desarrollado incluye dos modificaciones sobre el esquema básico del algoritmo PSO como son Opposition-based Learning, detallado en la Sección 3.1, y un procedimiento de competición entre las partículas que conforman el enjambre.

El procedimiento de competición se basa en escoger tres partículas dentro del enjambre aleatoriamente, hacerlas competir entre ellas, mediante su valor de función objetivo, obteniendo un partícula ganadora, otra perdedora y una partícula neutra para, posteriormente, eliminarlas del enjambre. Para un enjambre de tamaño N, en cada iteración del algoritmo se realizarán N/3 competiciones [31]. Tras cada competición, la partícula ganadora pasará directamente a la siguiente iteración del algoritmo y la partícula perdedora pasará también a la siguiente iteración tras aprender de la partícula ganadora, es decir, la partícula ganadora atrae a la partícula perdedora hacia su posición. La partícula neutral será la escogida para utilizarla en la estrategia de OBL y seguidamente pasar a la siguiente iteración, tratando de mejorar, de este modo, la capacidad de exploración del algoritmo [31].

La partícula perdedora y la partícula neutral van a modificar su posición y su velocidad siguiendo las siguientes ecuaciones:

$$V_{ld}^{k}(t+1) = R_{1d}^{k}(t) * V_{ld}^{k}(t) + R_{2d}^{k}(t) * (X_{wd}^{k}(t) - X_{ld}^{k}(t)) + \varphi * R_{3d}^{k}(t) * (\overline{X}_{ld}^{k}(t+1)) \tag{3.4}$$

$$X_{ld}^{k}(t+1) = X_{ld}^{k}(t) + V_{ld}^{k}(t+1) \tag{3.5}$$

$$X_{nd}^{k}(t+1) = ub_d + lb_d - X_{nd}^{k}(t) + R_{4d}^{k}(t) * X_{nd}^{k}(t) \tag{3.6}$$

Donde $X_{wd}^{k}(t)$, $X_{ld}^{k}(t)$ y $X_{nd}^{k}(t)$ son las posiciones d-ésimas de las partículas ganadora (winner), perdedora (looser) y neutral en la k-ésima ronda de competición dentro de la iteración t. V_{ld}^{k} es la velocidad de la partícula perdedora en la dimensión d-ésima en la k-ésima ronda de competición de la iteración t [31]. Y por último, R_{1d}^{k}, $R_{2d}^{k}(t)$, $R_{3d}^{k}(t)$ y $R_{4d}^{k}(t)$ son valores escogidos aleatoriamente dentro del intervalo [0,1], φ es un parámetro fijado manualmente, $\overline{X}_{ld}^{k}(t)$ representa el valor medio de las posiciones de las partículas dentro del enjambre y, ub_d y lb_d son las cotas superiores e inferiores del espacio de búsqueda en la dimensión d-ésima [31].

A la hora de actualizar una partícula del enjambre puede darse la posibilidad de que las variables de dicha partícula tomen valores fueran del rango $[a, b]$. Si esto ocurre la solución adoptada es la siguiente:

- $X_{id} > b$: cuando se supera el límite superior del espacio de búsqueda, X_{id} se define como dicho limite superior, $X_{id} = b$.
- $X_{id} < a$: en este caso, el nuevo valor de X_{id} es más pequeño que el límite inferior el espacio de búsqueda y se redefine $X_{id} = a$.

El pseudocódigo del algoritmo OBL-CPSO se muestra a continuación:

Algoritmo 4 OBL Competitive Particle Swarm Optimization()

1: Inicializar();
2: **mientras** Condición de parada no satisfecha **hacer**
3: **for all** $k = 1 : N/3$ **hacer**
4: $r_1 = S(k)$;
5: $r_2 = S(k + N/3)$;
6: $r_3 = S(k + 2N/3)$;
7: $(w, n, l) = competir(r_1, r_2, r_3)$;
8: Actualizar $X^k_{ld}(t)$; (Ec. 3.4 y Ec. 3.5)
9: Actualizar $X^k_{nd}(t)$; (Ec. 3.6)
10: Actualizar los valores de fitness para N y L;
11: **fin para**
12: BusquedaGlobal(); secc. 3.2
13: **fin mientras**
14: **return** Mejor solución obtenida

3.4. Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES)

Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) [12, 13, 10] es un algoritmo evolutivo diseñado para problemas de optimización continua no lineales. La característica principal del algoritmo es que en cada iteración se generan, a través de una Distribución Normal Multivariante en $\Re^D$, λ individuos nuevos, y a partir de estos individuos realizar una serie de cálculos intermedios para actualizar la matriz de covarianza asociada a la distribución [12].
El diseño del algoritmo CMA-ES no necesita un gran esfuerzo en la elección de los parámetros ya que es el algoritmo quien, internamente, se encarga de establecer los parámetros que utiliza, a excepción del tamaño de la población λ que debe ser definido por el usuario. Sin embargo, esta estrategia interna que define los parámetros sí forma parte del diseño del algoritmo y es uno de los pilares esenciales del rendimiento del mismo.

Las estrategias internas para definir los parámetros, estrategias de reinicio y los criterios de finalización hacen que encontremos una gran cantidad de variantes del algoritmo CMA-ES puro. La estructura básica del algoritmo CMA-ES se puede dividir en las etapas descritas en las siguientes secciones.

3.4.1. Inicialización de los Parámetros

El algoritmo CMA-ES requiere que el usuario únicamente indique el tamaño de población a emplear y, posteriormente es el propio algoritmo quien calcula los parámetros restantes a partir de dicho valor. Los parámetros comunes empleados en todas las implementaciones del algoritmo CMA-ES son los siguientes:

- λ: Número de individuos de la población.
- C: La matriz de covarianza C tendrá dimensión C_λ^D, donde λ es el número de individuos empleados por el algoritmo y D es la dimensión de la función a optimizar.
- σ: Parámetro para controlar el índice de variación entre cada generación de individuos.
- μ: En una selección elitista de individuos supervivientes, representa los mejores individuos a tener en cuenta dentro de la población.
- c_c y c_1: Ratio de aprendizaje para la actualización de la matriz de covarianza empleando una selección elitista de sólo un individuo.

- c_μ: Ratio de aprendizaje para la actualización de la matriz de covarianza empleando los μ mejores individuos.
- c_σ: Ratio de aprendizaje para el incremento de σ.
- d_σ: Parámetro de amortiguación para la actualización del paso de control.
- $m^g \in \Re^D$: Valor medio de la distribución en la generación g.
- p_σ: Valor de paso para σ.
- p_c: Valor de paso para C.

3.4.2. Muestreo

La etapa de muestreo del algoritmo CMA-ES es la encargada de generar λ nuevos individuos a partir de una Distribución Normal Multivariante de media cero y covarianza C. Estos individuos y_i para $i = 1, ..., \lambda$ son posteriormente multiplicados por σ y sumados con el valor medio de los individuos de la generación anterior. En la etapa inicial el valor medio es generado de manera aleatoria con valores dentro del dominio de la función a optimizar. Tras esta operación obtenemos los individuos x_i y finalizamos la etapa de muestreo. A continuación, se muestra la ecuación que describe este proceso:

$$x_i = m + \sigma y_i, \quad y_i \sim N_i(0, C), \quad para \quad i = 1, ..., \lambda \tag{3.7}$$

3.4.3. Actualizar Media

Tras obtener los nuevos individuos para la iteración t del algoritmo, es el momento de actualizar el valor medio de los individuos aplicando la siguiente ecuación:

$$m \leftarrow \sum_{i=1}^{\mu} w_i x_{i:\lambda} = m + \sigma y_w \quad donde \quad y_w = \sum_{i=1}^{\mu} w_i y_{i:\lambda} \tag{3.8}$$

Donde y_w representa la suma de los μ mejores individuos generados multiplicados por los pesos w_i correspondientes.

3.4.4. Paso para C

La siguiente fórmula muestra el cálculo que lleva a cabo el algoritmo para obtener el nuevo valor de paso para la matriz de covarianza C.

$$p_c \leftarrow (1 - c_c)p_c + 1 \cdot \left\{ \|p_\sigma\| < 1{,}5\sqrt{n} \right\} \sqrt{1 - (1 - c_c)^2}\sqrt{\mu_w} y_w \tag{3.9}$$

3.4.5. Paso para σ

Seguidamente, el algoritmo debe calcular el valor de paso para el paŕametro de control σ. Para ello, aplicamos la siguiente fórmula:

$$p_\sigma \leftarrow (1 - c_\sigma)p_\sigma \sqrt{1 - (1 - c_\sigma)^2}\sqrt{\mu_w} \cdot C^{-1/2} y_w \tag{3.10}$$

3.4.6. Actualizar C

Una vez calculados los tamaños de paso, es el momento de actualizar la matriz de covarianza para poder realizar un nuevo muestreo en la siguiente iteración del algoritmo. Para ello, empleamos la siguiente fórmula:

$$C \leftarrow (1 - c_1 - c_\mu) \cdot C + c_1 p_c p_c^T + c_\mu \sum_{i=1}^{\mu} w_i y_{i:\lambda} y_{i:\lambda}^T \tag{3.11}$$

3.4.7. Actualizar σ

Por último, sólo queda actualizar el valor de σ para la siguiente iteración aplicando la fórmula:

$$\sigma \leftarrow \sigma \times \exp(\frac{c_\sigma}{d_\sigma}(\frac{\|p_\sigma\|}{E\,\|N(0, I)\|}) - 1) \tag{3.12}$$

3.4.8. Pseudocódigo

En nuestra implementación hemos partido de la base del algoritmo CMA-ES detallada en los apartados anteriores y hemos decidido añadir una fase de reinicio. La fase de reiniciar puede ser necesaria en los casos en que el paso de control σ haya aumentado bruscamente o bien no se hayan producido mejoras en las soluciones candidatas, o bien, el algoritmo se haya estancado en un óptimo local.

Durante la fase de reinicio recalculamos todos los parámetros de la misma manera que se realiza en la fase de inicialización de parámetros a excepción del tamaño de población que en este caso lo incrementaremos para poder escapar de ese óptimo local. En este caso, el nuevo valor λ se establecerá a 100.

Finalmente, podemos ver en la siguiente figura el pseudocódigo de nuestra aproximación al algoritmo CMA-ES:

Algoritmo 5 Covariance Matrix Adaptation Evolutionary Strategy ()

Require:

$$m \in R^n, \sigma \in R_+, \lambda$$

1: Inicialización

$$C = I, p_c = 0, p_\sigma = 0$$

$$c_c \approx 4/n, c_\sigma \approx 4/n, c_1 \approx 2/n^2, c_\mu \approx \mu_w/n^2, c_1 + c_\mu \leq 1$$

$$d_\sigma \approx 1 + \sqrt{\frac{\mu_w}{n}}, w_i = 1...\lambda \quad tal \quad que \quad \mu_w = \frac{1}{\sum_{i=1}^{\mu} w_i^2} \approx 0{,}3\lambda$$

2: **mientras** Condición de parada no satisfecha **hacer**
3: Muestreo (Ec.3.7);
4: Actualizar el valor medio (Ec. 3.8);
5: Incremento de C (Ec. 3.9);
6: Incremento de σ (Ec. 3.10);
7: Actualizar C (Ec. 3.11);
8: Actualizar σ (Ec. 3.12);
9: BusquedaGlobal(); secc. 3.2;
10: **si** Reinicio necesario **entonces**
11: Reiniciar();
12: **fin si**
13: **fin mientras**
14: **return** Mejor solución obtenida

3.5. Hybrid Simulated Annealing with Global Search (HSAGS)

Simulated Annealing (SA) [24, 17, 8] es una meta-heurística que se basa en una búsqueda probabilística con una única solución. SA está inspirada por el proceso de recocido en la metalurgia [3]. El recocido es un proceso físico donde un sólido es enfriado lentamente hasta que su estructura se congela en una configuración de mínima energía [3].

En SA, con una alta temperatura T, el sistema ignora los pequeños cambios en la energía y se aproxima a un equilibrio térmico rápidamente, es decir, se realiza una búsqueda global del óptimo dentro del espacio de búsqueda. En cambio, cuando la temperatura es baja, el sistema realiza una búsqueda local en el *"vecindario"* del óptimo encontrado en busca de una solución mejor [3].

Inicialmente el algoritmo SA calcula una solución factible preliminar aleatoriamente para comenzar el proceso de búsqueda. Además, se define T con un valor indicado por el usuario, generalmente un valor muy elevado. Durante el proceso de búsqueda del óptimo global, la solución aleatoria generada inicial-

mente sufre ciertas variaciones obteniendo una nueva solución x' a partir de la siguiente ecuación:

$$x = x + \Delta x \tag{3.13}$$

Seguidamente, se evalua la diferencia de energía (E) entre ambas soluciones como muestra la siguiente ecuación:

$$\Delta E(x) = E(x + \Delta x) - E(x) \tag{3.14}$$

La solución x' será directamente aceptada como nueva solución factible sí $\Delta E(x) < 0$. En caso contrario, x' se aceptará con probabilidad:

$$P = e^{-\frac{\Delta E}{T}} \tag{3.15}$$

Por último, debemos decrementar el valor de la temperatura con un valor prefijado:

$$T = T - \Delta T \tag{3.16}$$

Debido a que SA es un algoritmo de trayectoria que emplea un único individuo en la población, la búsqueda global descrita en la Sección 3.2 debe adaptarse a esta peculiaridad. En esta nueva aproximación, se omite el cálculo del centroide y simplemente se realizan modificaciones a la única solución que conforma S mientras se produzcan mejoras en la misma. Todas estas nuevas soluciones se incluyen en el conjunto S para, finalmente, obtener la mejor de todas ellas.

Algoritmo 6 BúsquedaGlobalSA()

```
1: Individuo = S[0];
2: NumeroMejora = 0;
3: mientras NumeroMejora > 0 hacer
4:    Mejora = true;
5:    mientras Mejora = true hacer
6:       mientras |a1| + |a2| + |a3| ≠ 1 hacer
7:          GenerarRand(a1, a2, a3); Obtenemos tres valores generados aleatoriamente de manera uniforme en el rango [0, 1] para realizar la modificación del individuo
8:       fin mientras
9:       NuevoInd = ModificarIndividuo(Individuo, a1, a2, a3);
10:      si NuevoInd < Individuo entonces
11:         Mejora = true;
12:         S = S ∩ NuevoInd;
13:         NumeroMejora = NumeroMejora + 1;
14:      en otro caso
15:         Mejora = false;
16:      fin si
17:   fin mientras
18: fin mientras
19: OrdenarPoblacion(S);
20: S = ObtenerMejorIndividuo(S);
21: return Mejor individuo encontrado
```

El siguiente pseudocódigo presenta el algoritmo HSAGS, una modificación sobre el esquema general del algoritmo SA [24] en el que se incluye la búsqueda global descrita en la Sección 3.2.
El método *AplicarPerturbacionAleatoria* es el encargado de modificar la solución actual haciendo uso de la Ecuación 3.13. Seguidamente, la nueva solución **S'** es comparada con su ancestro *S* mediante el método *EvaluarDiferencia* el cual, calcula la diferencia entre ambas mediante la Ecuación 3.14.
A continuación, se define la nueva solución factible actual. Para ello, el método *ActualizarSolucion* recibe ambas soluciones (S y S') y la diferencia entre ellas para determinar la nueva solución factible al problema y, si fuera necesario aplicar de la ecuación 3.15 para realizar esta labor.
Finalmente, se decrementa la temperatura actual aplicando la fórmula 3.16 y se aplica la búsqueda global explicada en la Sección 3.5.

Algoritmo 7 Hybrid Simulated Annealing with Global Search()

```
1: S = GenerarSolucionAleatoria();
2: T = InicializarTemperatura();
3: mientras Condición de parada no satisfecha hacer
4:    S' = AplicarPerturbacionAleatoria(S); 3.13
5:    Dif = EvaluarDiferencia(S, S'); 3.14
6:    S = ActualizarSolucion(S, S', Dif); 3.15
7:    T = ActualizarTemperatura(T); 3.16
8:    S = BusquedaGlobalSA(); secc. 3.2
9: fin mientras
10: return Mejor solución obtenida
```

Capítulo 4

Evaluación experimental

4.1. Descripción de las Funciones Propuestas por el GenOpt

El concurso *GenOpt* ha propuesto un total de **18 funciones** de dimensiones $D = 10, 30$ a optimizar, realizando **cien ejecuciones independientes** para cada una de ellas. A partir de sus características, estas funciones se pueden agrupar en tres familias diferentes.

4.1.1. Funciones GKLS

Las funciones GKLS [6] son obtenidas mediante un generador de funciones de tres tipos (no-diferenciable, continuamente diferenciable y dos veces continuamente diferenciable) con mínimos locales y globales conocidos.

4.1.2. Funciones clásicas transformadas

Esta familia de funciones se obtienen realizando una transformación a aquellas funciones clásicas para probar métodos de optimización global continua como son:

- Rastrigin, $D = 10, 30$
- Rosenbrock, $D = 10, 30$
- Zakharov, $D = 10, 30$

Cada función x es transformada en cada instancia de la siguiente manera:

$$x' = Mx + x_0 \tag{4.1}$$

Dónde $x_0 \in [-0{,}1, 0{,}1]^D$ es un translación aleatoria y M es una matriz ortogonal con número de condición igual a 100.

4.1.3. Funciones compuestas

Por último, en esta familia encontraremos funciones obtenidas a partir de la composición de las funciones de la segunda familia. Estas funciones se construyen seleccionando aleatoriamente n funciones clásicas $f_1, ..., f_n$ del siguiente conjunto:

- Goldstein-Price, $D_f = 2$;
- Hartmann, $D_f = 3; 6$;
- Rosenbrock, $D_f = rand(3, D/2)$;
- Rastrigin, $D_f = rand(3, D/2)$;
- Sphere, $D_f = rand(3, D/2)$;
- Zakharov, $D_f = rand(3, D/2)$;

La suma de las dimension $D_{f_1}, ..., D_{f_n}$ debe cumplir que $\sum_i D_{f_i} = D$. Una función f es sujeta a una roto-translación tal que, para cada instancia:

$$x' = Ux + X_0 \tag{4.2}$$

Dónde U es una matriz ortogonal aleatoria y $x_0 \in [-0{,}1, 0{,}1]^D$ es una pequeña translación aleatoria. El valor de cada instancia es calculado como:

$$f(x) = c + \sum_{i=1}^{n} f_i(x'_{b_i}, ..., x'_{b_i} + D_{f_i} - 1) \tag{4.3}$$

Dónde $b_i = \sum_{j=1}^{i-1} D_{f_i}$.
En la siguiente figura podemos observar un esquema de este proceso:

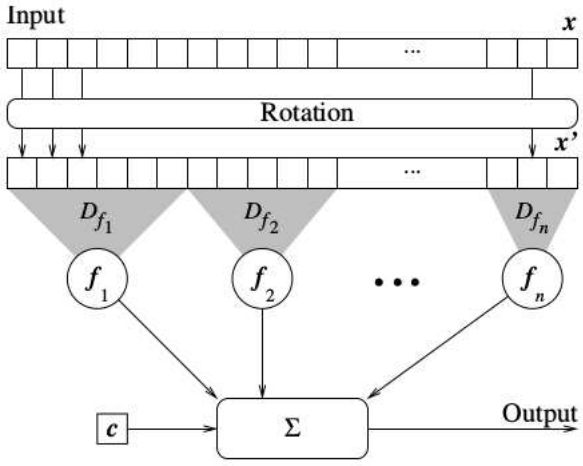

Figura 4.1: Composición de funciones.

4.2. Estudio de la Parametrización y Rendimiento

En esta sección trataremos el estudio de los parámetros empleados por cada uno de los algoritmos y como afectan al desempeño de los mismos.

Un parámetro común a todos los algoritmos desarrollados es el tamaño de la población, y es por ello que, se emplearán los mismos tamaños de población, $popsize = 20, 50, 75, 100$, para estudiar el rendimiento de cada algoritmo, a excepción del algoritmo SA (Sección 3.5) que debido a que se trata de una meta-heurística de trayectoria se empleará únicamente un individuo.

4.2.1. OBL-CPSO

En el diseño del algoritmo OBL-CPSO (Sección 3.3) sólo se contempla la utilización de un parámetro, el tamaño de población. Es por ello, que los experimentos realizados con este algoritmo buscan encontrar el tamaño de población más adecuado para su rendimiento. A continuación, podemos ver una comparativa de los experimentos realizados con $popsize = 20, 50, 75, 100$ en la que se presentan los valores de media (μ), mediana ($\tilde{x}$) y desviación típica (σ) para las 18 funciones propuestas por el concurso GenOpt.

Como podemos observar, en negrita se muestran los mejores valores obtenidos para cada una de las funciones a optimizar

	OBL-CPSO-20			OBL-CPSO-50			OBL-CPSO-75			OBL-CPSO-100		
	μ	$\tilde{x}$	σ	μ	$\tilde{x}$	σ	μ	$\tilde{x}$	σ	μ	$\tilde{x}$	σ
f_0	$4{,}361e-01$	$2{,}041e-01$	$4{,}492e-01$	$\mathbf{3{,}937e-01}$	$\mathbf{1{,}255e-01}$	$4{,}193e-01$	$4{,}095e-01$	$1{,}364e-01$	$4{,}224e-01$	$1{,}228e+00$	$1{,}218e+00$	$2{,}404e-01$
f_1	$1{,}776e+00$	$1{,}681e+00$	$4{,}699e-01$	$1{,}464e+00$	$1{,}401e+00$	$2{,}359e-01$	$\mathbf{1{,}403e+00}$	$\mathbf{1{,}351e+00}$	$2{,}047e-01$	$3{,}706e+00$	$3{,}572e+00$	$7{,}136e-01$
f_2	$\mathbf{7{,}417e-01}$	$1{,}002e+00$	$4{,}173e-01$	$7{,}657e-01$	$\mathbf{1{,}002e+00}$	$3{,}618e-01$	$7{,}968e-01$	$1{,}004e+00$	$3{,}407e-01$	$1{,}250e+00$	$1{,}227e+00$	$1{,}530e-01$
f_3	$1{,}779e+00$	$1{,}699e+00$	$4{,}513e-01$	$1{,}455e+00$	$1{,}402e+00$	$2{,}441e-01$	$\mathbf{1{,}373e+00}$	$\mathbf{1{,}350e+00}$	$1{,}722e-01$	$3{,}708e+00$	$3{,}572e+00$	$7{,}161e-01$
f_4	$7{,}822e-01$	$\mathbf{1{,}003e+00}$	$4{,}026e-01$	$\mathbf{7{,}764e-01}$	$1{,}003e+00$	$3{,}803e-01$	$8{,}439e-01$	$1{,}004e+00$	$3{,}219e-01$	$1{,}248e+00$	$1{,}216e+00$	$1{,}645e-01$
f_5	$1{,}790e+00$	$1{,}698e+00$	$5{,}522e-01$	$1{,}451e+00$	$1{,}406e+00$	$2{,}479e-01$	$\mathbf{1{,}423e+00}$	$\mathbf{1{,}357e+00}$	$2{,}284e-01$	$3{,}707e+00$	$3{,}572e+00$	$7{,}162e-01$
f_6	$9{,}041e-01$	$3{,}609e-01$	$1{,}297e+00$	$5{,}852e-01$	$3{,}130e-01$	$7{,}651e-01$	$\mathbf{4{,}691e-01}$	$\mathbf{2{,}570e-01}$	$4{,}897e-01$	$3{,}368e+00$	$2{,}636e+00$	$2{,}445e+00$
f_7	$1{,}057e+01$	$1{,}014e+01$	$4{,}274e+00$	$\mathbf{8{,}992e+00}$	$8{,}667e+00$	$3{,}394e+00$	$9{,}134e+00$	$\mathbf{8{,}482e+00}$	$3{,}717e+00$	$2{,}545e+01$	$2{,}466e+01$	$6{,}987e+00$
f_8	$5{,}837e-01$	$3{,}580e-01$	$6{,}216e-01$	$3{,}908e-01$	$\mathbf{2{,}501e-01}$	$4{,}752e-01$	$\mathbf{3{,}449e-01}$	$2{,}537e-01$	$3{,}115e-01$	$2{,}202e+00$	$1{,}999e+00$	$1{,}283e+00$
f_9	$5{,}574e+00$	$5{,}209e+00$	$2{,}245e+00$	$4{,}881e+00$	$4{,}840e+00$	$1{,}817e+00$	$\mathbf{4{,}720e+00}$	$\mathbf{4{,}588e+00}$	$1{,}643e+00$	$1{,}276e+01$	$1{,}279e+01$	$3{,}019e+00$
f_{10}	$6{,}175e-03$	$4{,}830e-03$	$5{,}254e-03$	$4{,}564e-03$	$3{,}010e-03$	$3{,}711e-03$	$\mathbf{3{,}874e-03}$	$\mathbf{2{,}688e-03}$	$3{,}495e-03$	$1{,}661e-02$	$1{,}440e-02$	$1{,}151e-02$
f_{11}	$4{,}735e-02$	$4{,}723e-02$	$1{,}483e-02$	$4{,}155e-02$	$3{,}961e-02$	$1{,}488e-02$	$\mathbf{4{,}151e-02}$	$\mathbf{3{,}904e-02}$	$1{,}285e-02$	$8{,}903e-02$	$8{,}608e-02$	$2{,}672e-02$
f_{12}	$4{,}103e-02$	$2{,}806e-02$	$4{,}074e-02$	$\mathbf{2{,}463e-02}$	$\mathbf{1{,}685e-02}$	$2{,}296e-02$	$3{,}093e-02$	$2{,}092e-02$	$3{,}662e-02$	$4{,}394e-01$	$3{,}230e-01$	$3{,}904e-01$
f_{13}	$8{,}880e-02$	$8{,}295e-02$	$3{,}394e-02$	$8{,}708e-02$	$8{,}431e-02$	$2{,}912e-02$	$\mathbf{8{,}200e-02}$	$\mathbf{8{,}032e-02}$	$2{,}643e-02$	$2{,}611e-01$	$2{,}452e-01$	$1{,}296e-01$
f_{14}	$1{,}026e-01$	$3{,}233e-02$	$1{,}907e-01$	$3{,}723e-02$	$2{,}042e-02$	$4{,}753e-02$	$\mathbf{3{,}049e-02}$	$\mathbf{1{,}543e-02}$	$4{,}289e-02$	$1{,}381e+00$	$1{,}147e+00$	$1{,}006e+00$
f_{15}	$1{,}317e-01$	$1{,}180e-01$	$5{,}010e-02$	$1{,}198e-01$	$1{,}121e-01$	$3{,}953e-02$	$\mathbf{1{,}196e-01}$	$\mathbf{1{,}115e-01}$	$4{,}406e-02$	$4{,}437e-01$	$3{,}095e-01$	$4{,}113e-01$
f_{16}	$1{,}516e-02$	$1{,}252e-02$	$1{,}090e-02$	$1{,}380e-02$	$1{,}075e-02$	$1{,}142e-02$	$\mathbf{1{,}201e-02}$	$\mathbf{1{,}052e-02}$	$8{,}260e-03$	$4{,}439e-02$	$3{,}137e-02$	$4{,}109e-02$
f_{17}	$2{,}666e-01$	$2{,}286e-01$	$1{,}587e-01$	$2{,}049e-01$	$1{,}744e-01$	$1{,}091e-01$	$\mathbf{1{,}845e-01}$	$\mathbf{1{,}687e-01}$	$8{,}040e-02$	$3{,}330e+00$	$2{,}985e+00$	$1{,}709e+00$

Tabla 4.1: Comparativa del algoritmo OBL-CPSO con diferentes tamaños de población.

4.2.2. CMA-ES

El diseño del algoritmo CMA-ES, como detallamos en la sección 3.4, sólo necesita la definición de dos parámetros iniciales para comenzar su ejecución, el tamaño de la población a emplear y el valor de sigma (σ) inicial.
Para evaluar el rendimiento del algoritmo variando estos dos parámetros, hemos realizado un total de 12 experimentos.

El tamaño de la población, como especificamos en el inicio de la sección, tomará los siguientes valores $popsize = 20, 50, 75, 100$.
En cuanto al parámetro σ, los valores que tomará serán $\sigma = 0{,}3; 0{,}8; 2{,}0$. La elección de estos valores se basa en que el parámetro σ determina la variación incluida en cada nueva generación de la población y teniendo en cuenta esto, unos valores de σ pequeños como 0.3 y 0.8 harán que el algoritmo se comporte como una búsqueda local, y al contrario, como una búsqueda global, con valores grandes de σ como 2.0 [12].

A continuación podemos observar una tabla en la muestran los resultados de la evaluación el rendimiento de las mejores configuraciones obtenidas de las 12 configuraciones totales. Estas configuraciones son CMA_ES-$2-50$, CMA_ES-$0{,}3-50$ y CMA_ES-$0{,}3-100$, mostrando para cada configuración los valores de media (μ), mediana ($\tilde{x}$) y desviación típica (σ) para las 18 funciones propuestas por el concurso GenOpt. La nomenclatura utilizada es la siguiente: CMA_ES-$\sigma - popsize$.

Como podemos observar, los resultados destacados en negrita muestran la mejor configuración, CMA_ES-$2-50$, de las tres presentadas aunque, estos resultados son idénticos a los obtenidos con la configuración CMA_ES-$0{,}8-50$. Esto es así debido a la estratégia de reinicio empleada en nuestro algoritmo, la cual define $\sigma = 2$ cuando es empleada.

	CMA_ES-2 − 50			CMA_ES-0,3 − 50			CMA_ES-0,8 − 50		
	μ	$\tilde{x}$	σ	μ	$\tilde{x}$	σ	μ	$\tilde{x}$	σ
f_0	**8,503e − 01**	**1,000e + 00**	2,677e − 01	6,503e + 00	1,089e + 00	2,677e − 01	8,503e − 01	1,000e + 00	2,677e − 01
f_1	**1,145e + 00**	**1,121e + 00**	9,401e − 02	1,145e + 00	1,331e + 00	9,401e − 02	1,145e + 00	1,121e + 00	9,401e − 02
f_2	**9,650e − 01**	**1,000e + 00**	1,102e − 01	6,890e − 01	1,120e + 00	1,102e − 01	9,650e − 01	1,000e + 00	1,102e − 01
f_3	**1,145e + 00**	**1,127e + 00**	9,699e − 02	1,175e + 00	1,197e + 00	9,699e − 02	1,145e + 00	1,127e + 00	9,699e − 02
f_4	**9,568e − 01**	**1,000e + 00**	1,375e − 01	7,544e − 01	1,000e + 00	1,375e − 01	9,568e − 01	1,000e + 00	1,375e − 01
f_5	**1,139e + 00**	**1,107e + 00**	9,068e − 02	2,177e + 00	1,107e + 00	9,068e − 02	1,139e + 00	1,107e + 00	9,068e − 02
f_6	**7,098e − 03**	**2,675e − 03**	1,102e − 02	6,068e − 03	2,895e − 03	1,102e − 02	7,098e − 03	2,675e − 03	1,102e − 02
f_7	**3,485e + 00**	**3,206e + 00**	1,737e + 00	4,785e + 00	3,206e + 00	1,737e + 00	3,485e + 00	3,206e + 00	1,737e + 00
f_8	**1,114e − 02**	**3,920e − 03**	2,502e − 02	1,184e − 02	3,500e − 03	2,502e − 02	1,114e − 02	3,920e − 03	2,502e − 02
f_9	**1,801e + 00**	**1,713e + 00**	8,352e − 01	2,881e + 00	1,503e + 00	8,352e − 01	1,801e + 00	1,713e + 00	8,352e − 01
f_{10}	**2,323e − 02**	**2,214e − 02**	2,532e − 02	1,393e − 02	2,124e − 02	2,532e − 02	2,323e − 02	2,214e − 02	2,532e − 02
f_{11}	**5,283e − 02**	**5,061e − 02**	1,399e − 02	3,883e − 02	5,031e − 02	1,399e − 02	5,283e − 02	5,061e − 02	1,399e − 02
f_{12}	**2,242e − 04**	**1,147e − 04**	5,236e − 04	4,442e − 04	1,147e − 04	5,236e − 04	2,242e − 04	1,147e − 04	5,236e − 04
f_{13}	**3,018e − 02**	**2,998e − 02**	7,994e − 03	2,018e − 02	2,998e − 02	7,994e − 03	3,018e − 02	2,998e − 02	7,994e − 03
f_{14}	**1,443e − 04**	**2,946e − 05**	3,504e − 04	1,443e − 04	2,946e − 05	3,504e − 04	1,443e − 04	2,946e − 05	3,504e − 04
f_{15}	**4,870e − 02**	**4,838e − 02**	1,111e − 02	4,870e − 02	4,838e − 02	1,111e − 02	4,870e − 02	4,838e − 02	1,111e − 02
f_{16}	**9,820e − 04**	**2,520e − 04**	5,633e − 03	9,820e − 04	2,520e − 04	5,633e − 03	9,820e − 04	2,520e − 04	5,633e − 03
f_{17}	**3,485e − 02**	**3,375e − 02**	9,540e − 03	3,485e − 02	3,375e − 02	9,540e − 03	3,485e − 02	3,375e − 02	9,540e − 03

Tabla 4.2: Comparativa de algunas de las instancias CMA_ES-σ − *popsize*.

4.2.3. HSAGS

Por último, el algoritmo HSAGS (sección 3.5) presenta únicamente un parámetro, la temperatura inicial. Según la literatura [3], inicialmente este parámetro debe tener un valor elevado, y es por ello, por lo que los valores con los que hemos evaluado el rendimiento de este algoritmo han sido $Temp_0 = 500, 1000, 10000$. A continuación, podemos ver una comparativa del **valor de error** obtenido por cada experimento en cada una de las funciones propuestas por el concurso GenOpt. Como se puede observar, los valores resaltados en negrita muestran el valor de error mínimo alcanzado para cada una de las funciones propuestas por el concurso GenOpt y qué configuración del algoritmo HSAGS la obtuvo. Además, las flechas bidireccionales muestran que los valores de error obtenidos por las configuraciones apuntadas por dichas flechas son muy similares. La nomenclatura empleada es la siguiente: HSAGS-$TEMP_0$.

	HSAGS-500		**HSAGS-1000**		**HSAGS-10000**
f_0	$8{,}060e-01$	$\leftrightarrow$	**9.338e-01**	$\leftrightarrow$	$7{,}384e-01$
f_1	**0,000e+00**		$9{,}732e-01$	$\leftrightarrow$	$9{,}732e-01$
f_2	$9{,}241e-01$	$\leftrightarrow$	$9{,}270e-01$	$\leftrightarrow$	**9.711e-01**
f_3	**0,000e+00**		**0,000e+00**		**0,000e+00**
f_4	$7{,}369e-01$	$\leftrightarrow$	**8.174e-01**	$\leftrightarrow$	$5{,}917e-01$
f_5	$9{,}481e-01$	$\leftrightarrow$	$9{,}460e-01$	$\leftrightarrow$	**9.978e-01**
f_6	$9{,}961e-01$	$\leftrightarrow$	$9{,}971e-01$	$\leftrightarrow$	**9.990e-01**
f_7	**9.990e-01**	$\leftrightarrow$	$9{,}971e-01$	$\leftrightarrow$	$9{,}981e-01$
f_8	**0,000e+00**		$1{,}000e+00$	$\leftrightarrow$	$1{,}000e+00$
f_9	$1{,}000e+00$	$\leftrightarrow$	$1{,}000e+00$	$\leftrightarrow$	**0,000e+00**
f_{10}	**9.990e-01**	$\leftrightarrow$	$9{,}971e-01$	$\leftrightarrow$	$9{,}981e-01$
f_{11}	$9{,}942e-01$	$\leftrightarrow$	$9{,}961e-01$	$\leftrightarrow$	**9.990e-01**
f_{12}	**1.000e+00**	$\leftrightarrow$	$1{,}000e+00$	$\leftrightarrow$	$1{,}000e+00$
f_{13}	**9.990e-01**	$\leftrightarrow$	$9{,}990e-01$	$\leftrightarrow$	$1{,}000e+00$
f_{14}	**1.000e+00**	$\leftrightarrow$	$1{,}000e+00$	$\leftrightarrow$	$1{,}000e+00$
f_{15}	**1.000e+00**	$\leftrightarrow$	$1{,}000e+00$	$\leftrightarrow$	$1{,}000e+00$
f_{16}	$1{,}000e+00$	$\leftrightarrow$	**9.990e-01**	$\leftrightarrow$	$9{,}990e-01$
f_{17}	**1.000e+00**	$\leftrightarrow$	$1{,}000e+00$	$\leftrightarrow$	$1{,}000e+00$

Tabla 4.3: Comparativa del algoritmo HSAGS con diferentes valores de temperatura inicial.

4.2.4. Comparativa de Rendimiento entre Algoritmos

En la siguiente tabla, se presenta una comparativa de los valores media (μ), mediana ($\tilde{x}$) y desviación típica (σ) de las mejores configuraciones conseguidas con cada uno de los algoritmos implementados. Como podemos apreciar, el algoritmo CMA-ES [10] (Sección 3.4) consigue los mejores resultados para las 18 funciones propuestas por el concurso GenOpt. Debido a estos resultados, el algoritmo CMA-ES fue escogido para competir en la fase final del concurso.

	CMA_ES-2 − 50			OBL-CPSO-75			HSAGS-500		
	μ	$\tilde{x}$	σ	μ	$\tilde{x}$	σ	μ	$\tilde{x}$	σ
f_0	$8{,}503e-01$	$1{,}000e+00$	$2{,}677e-01$	**4,095e−01**	**1,364e−01**	$4{,}224e-01$	$3{,}875e+00$	$3{,}731e+00$	$8{,}967e-01$
f_1	**1,145e+00**	**1,121e+00**	$9{,}401e-02$	$1{,}403e+00$	$1{,}351e+00$	$2{,}047e-01$	$1{,}065e+01$	$1{,}052e+01$	$1{,}439e+00$
f_2	**9,650e−01**	**1,000e+00**	$1{,}102e-01$	$7{,}968e-01$	$1{,}004e+00$	$3{,}407e-01$	$3{,}878e+00$	$3{,}731e+00$	$8{,}982e-01$
f_3	**1,145e+00**	**1,127e+00**	$9{,}699e-02$	$1{,}373e+00$	$1{,}350e+00$	$1{,}722e-01$	$1{,}065e+01$	$1{,}052e+01$	$1{,}439e+00$
f_4	**9,568e−01**	**1,000e+00**	$1{,}375e-01$	$8{,}439e-01$	$1{,}004e+00$	$3{,}219e-01$	$9{,}568e-01$	$1{,}000e+00$	$1{,}375e-01$
f_5	**1,139e+00**	**1,107e+00**	$9{,}068e-02$	$1{,}423e+00$	$1{,}357e+00$	$2{,}284e-01$	$1{,}064e+01$	$1{,}050e+01$	$1{,}435e+00$
f_6	**7,098e−03**	**2,675e−03**	$1{,}102e-02$	$4{,}691e-01$	$2{,}570e-01$	$4{,}897e-01$	$1{,}373e+01$	$1{,}227e+01$	$6{,}326e+00$
f_7	**3,485e+00**	**3,206e+00**	$1{,}737e+00$	$9{,}134e+00$	$8{,}482e+00$	$3{,}717e+00$	$4{,}560e+01$	$4{,}423e+01$	$1{,}053e+01$
f_8	**1,114e−02**	**3,920e−03**	$2{,}502e-02$	$3{,}449e-01$	$2{,}537e-01$	$3{,}115e-01$	$6{,}038e+00$	$6{,}183e+00$	$1{,}809e+00$
f_9	**1,801e+00**	**1,713e+00**	$8{,}352e-01$	$4{,}720e+00$	$4{,}588e+00$	$1{,}643e+00$	$1{,}903e+01$	$1{,}891e+01$	$3{,}118e+00$
f_{10}	**2,323e−02**	**2,214e−02**	$2{,}532e-02$	$3{,}874e-03$	$2{,}688e-03$	$3{,}495e-03$	$1{,}270e-01$	$5{,}255e-02$	$2{,}558e-01$
f_{11}	**5,283e−02**	**5,061e−02**	$1{,}399e-02$	$4{,}151e-02$	$3{,}904e-02$	$1{,}285e-02$	$4{,}264e+00$	$1{,}808e-01$	$2{,}459e+01$
f_{12}	**2,242e−04**	**1,147e−04**	$5{,}236e-04$	$3{,}093e-02$	$2{,}092e-02$	$3{,}662e-02$	$5{,}330e+00$	$5{,}016e+00$	$2{,}499e+00$
f_{13}	**3,018e−02**	**2,998e−02**	$7{,}994e-03$	$8{,}200e-02$	$8{,}032e-02$	$2{,}643e-02$	$2{,}689e+00$	$2{,}567e+00$	$1{,}265e+00$
f_{14}	**1,443e−04**	**2,946e−05**	$3{,}504e-04$	$3{,}049e-02$	$1{,}543e-02$	$4{,}289e-02$	$8{,}964e+00$	$8{,}338e+00$	$4{,}235e+00$
f_{15}	**4,870e−02**	**4,838e−02**	$1{,}111e-02$	$1{,}196e-01$	$1{,}115e-01$	$4{,}406e-02$	$1{,}009e+01$	$8{,}802e+00$	$5{,}327e+00$
f_{16}	**9,820e−04**	**2,520e−04**	$5{,}633e-03$	$1{,}201e-02$	$1{,}052e-02$	$8{,}260e-03$	$4{,}049e+00$	$3{,}340e+00$	$3{,}172e+00$
f_{17}	**3,485e−02**	**3,375e−02**	$9{,}540e-03$	$1{,}845e-01$	$1{,}687e-01$	$8{,}040e-02$	$1{,}603e+01$	$1{,}465e+01$	$5{,}702e+00$

Tabla 4.4: Comparativa de las mejores configuraciones de cada algoritmo implementado.

4.3. Clasificación en el Concurso GenOpt

Desde la organización del concurso GenOpt proponen varios criterios para clasificar los algoritmos enviados por los participantes. Estos criterios pueden ser:

- **High Jump**: Mejor valor obtenido en los puntos de control.
- **Target Shooting**: Éxito a la hora de alcanzar el óptimo global de la función.
- **Biathlon Score**: Media entre el High Jump y Target Shooting.

Para obtener más detalles sobre como se calculan estas métricas, se recomienda la lectura del Manifesto del GenOpt[1].

A la vista de los resultados obtenidos en las diversas pruebas llevadas a cabo en el análisis de rendimiento 4.2.4, decidimos emplear el algoritmo CMA-ES (Sección 3.4) para participar en la fase final del concurso.
Con este algoritmo, el cual denominamos *Hybrid Continuous Optimiser based on CMA-ES and a Global Neighbourhood Path Search (HCO-CMA-G)*, conseguimos la **tercera posición** en el ranking final según el parámetro **High Jump**, como podemos ver en la siguiente figura:

FINAL LEADERBOARD

Position	Submission Name	High Jump	Target Shooting	Biathlon Score	Submission Date
1	F. Romito, GABRLS	1.13889	1.2222	1.18056	Apr, 7th (71 day(s) passed)
2	E. Segredo, E. Lalla-Ruiz, E. Hart, B. Paechter, S. Voß, HOCO	1.41667	1.7778	1.59722	Apr, 6th (72 day(s) passed)
4	A. Marrero, E. Segredo, C. Segura, HCO-CMA-G	3.29167	3.3889	3.34028	Apr, 7th (71 day(s) passed)
3	A. Mariello, RRASH	3.41667	3.1667	3.29167	Apr, 6th (72 day(s) passed)

Figura 4.2: Clasificación en el concurso GenOpt según High Jump.

[1]Dirección desde la que se puede obtener el Manifesto del concurso GenOpt: http://www.genopt.org/genopt.pdf.

A continuación, podemos observar una comparativa del rendimiento entre los finalistas del concurso GenOpt basado en el criterio High Jump.

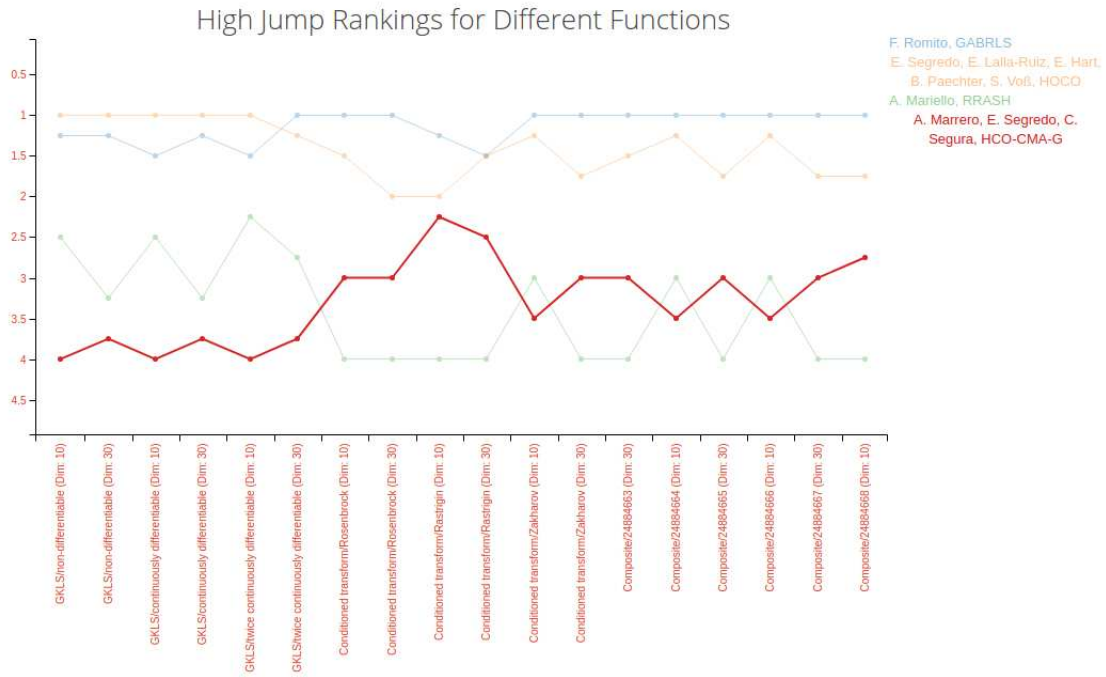

Figura 4.3: Clasificación en el concurso GenOpt según High Jump.

Capítulo 5

Conclusiones y líneas de trabajo futuras

5.1. Conclusiones

Durante la elaboración de este Trabajo de Fin de Grado hemos obtenido varias conclusiones que destacaremos a continuación.

En primer lugar, trabajar con algoritmos que requieren una gran cantidad de parámetros, como es el caso del algoritmo CMA-ES u OBL-CPSO, hace que la evaluación del rendimiento del algoritmo sea más compleja dado que por cada parámetro se incrementa significativamente la cantidad de experimentos a realizar para probar el rendimiento del mismo. Aún así, en el caso del algoritmo CMA-ES, la naturaleza auto-reguladora del algoritmo facilita en cierta medida esta tarea.

Hemos de destacar además que, debido al gran número de funciones propuestas por el concurso GenOpt, las modificaciones realizadas a los algoritmos resultaron difíciles de evaluar dado que en bastantes ocasiones, la mejora era simplemente en funciones concretas y en conjunto, los resultados no variaban significativamente para considerar dicha modificación.

Merece la pena mencionar que el algoritmo CMA-ES (Sección 3.4) consiguió el **tercer mejor puesto** según el criterio **High Jump** en la clasificación final del concurso GenOpt.

Finalmente, y considerando los resultados obtenidos por las técnicas implementadas, podemos concluir que el algoritmo CMA-ES es el más eficaz resolviendo el conjunto de problemas propuesto por GenOpt.

5.2. Líneas de Trabajo Futuras

El trabajo desarrollado en este proyecto puede servir como base para comprobar el rendimiento de diferentes técnicas meta-heurísticas a la hora de resolver

problemas de optimización continua.
Además, como líneas de trabajo futuras se prevé la mejora algunas de las técnicas implementadas, así como un estudio más intensivo de los parámetros, con vistas a participar en la siguiente edición del concurso GenOpt.

Capítulo 6

Summary and conclusions

6.1. Conclusions

Through the development of this degree thesis we have reached the following conclusions:

First of all, the fact of working with algorithms which use a high amount of parameters, like CMA-ES or OBL-CPSO, increases the complexity to evaluate the performance of the algorithms due to the fact that the number of possible tests increases with each parameter. Even though, the self-regulation nature of the algorithms like CMA-ES facilitates this task.

Furthermore, as a result of the large amount of proposed functions by the GenOpt contest, the task of assessing a new modification was really difficult. In many times, results were only improved for a low number of the proposed functions.

It is worth to be mentioned that the CMA-ES algorithm (Section 3.4) accomplished the **third place** in the final leaderboard of the GenOpt contest considering the High Jump criterion.

Finally, taking into account the obtained results by the developed algorithms, we can conclude that the algorithm CMA-ES was the most effective approach solving the functions proposed by the GenOpt contest.

6.2. Future work

The work developed in this project could be a starting point to test the performance of different meta-heuristic algorithms for solving continuos optimization problems.

A potential line of future work may be the improvement of the different tested algorithms, with the aim of participating in future editions of the GenOpt contest.

Capítulo 7

Presupuesto

En este capítulo se va a detallar el presupuesto para la realización de este Trabajo de Fin de Grado.

7.1. Ingeniero Informático

En la siguiente tabla se detallan todas las actividades llevadas a cabo durante el presente Trabajo de Fin de Grado indicando la duración en horas de cada una de ellas y el coste en Euros. Se ha establecido un precio base de 20 Euros para cada hora de trabajo. El coste total del desarrollo de la investigación es de 6000 Euros.

Actividad	Duración(horas)	Coste
Investigación de las técnicas algorítmicas existentes	20	400
Selección de los algoritmos a desarrollar	15	300
Desarrollo del algoritmo OBL-CPSO	40	800
Estudio de los parámetros del algoritmo OBL-CPSO	20	400
Desarrollo del algoritmo CMA-ES	55	1100
Estudio de los parámetros del algoritmo CMA-ES	20	400
Desarrollo del algoritmo Simulated Annealing	20	400
Estudio de los parámetros de Simualted Annealing	40	800
Desarrollo de las métricas propuestas por GENOPT	20	400
Obtención de las gráficas con los resultados del estudio	20	400
Redacción de la memoria	30	600

Tabla 7.1: Actividades

Coste total	6000 Euros

Tabla 7.2: Coste total de las actividades

7.2. Materiales

Por otra parte, también se ha usado el siguiente equipamiento durante el desarrollo de este Trabajo de Fin de Grado

Equipo	Coste(Euros)
Acer Aspire V3-772g	800
Monitor LG	120
Escritorio Skarsta	219
Silla de oficina Jules	50

Tabla 7.3: Equipamiento

Coste total	1189 Euros

Tabla 7.4: Coste del equipamiento

7.3. Coste Total

Finalmente, el coste total del Trabajo de Fin de Grado sumando el coste de las actividades llevadas a cabo y el equipamiento utilizado para dicha labor es de 7189 Euros.

Bibliografía

[1] Sigrún Andradóttir. A global search method for discrete stochastic optimization. *SIAM Journal on Optimization*, 6(2):513–530, 1996.

[2] A. Atashpendar, B. Dorronsoro, G. Danoy, and P. Bouvry. A parallel cooperative coevolutionary smpso algorithm for multi-objective optimization. pages 713–720, 2016.

[3] Ke-Lin Du and M.Ñ. S. Swamy. *Search and Optimization by Metaheuristics*. 2016.

[4] J.A. Díaz, D.E. Luna, J.-F. Camacho-Vallejo, and M.-S. Casas-Ramírez. Grasp and hybrid grasp-tabu heuristics to solve a maximal covering location problem with customer preference ordering. *Expert Systems with Applications*, 82:67–76, 2017.

[5] C.M. Fu, C. Jiang, G.S. Chen, and Q.M. Liu. An adaptive differential evolution algorithm with an aging leader and challengers mechanism. *Applied Soft Computing Journal*, 57:60–73, 2017.

[6] Marco Gaviano, Dmitri E. Kvasov, Daniela Lera, and Yaroslav D. Sergeyev. Algorithm 829: Software for generation of classes of test functions with known local and global minima for global optimization. *ACM Trans. Math. Softw.*, 29(4):469–480, December 2003.

[7] M. Gerber and L. Bornn. Improving simulated annealing through derandomization. *Journal of Global Optimization*, 68(1):189–217, 2017.

[8] M. Gerber and L. Bornn. Improving simulated annealing through derandomization. *Journal of Global Optimization*, 68(1):189–217, 2017.

[9] E. Glorieux, B. Svensson, F. Danielsson, and B. Lennartson. Improved constructive cooperative coevolutionary differential evolution for large-scale optimisation. pages 1703–1710, 2016.

[10] M. Hajebi, A. Hoorfar, and E. Bou-Daher. Inverse profiling of inhomogenous buried cylinders with arbitrary cross sections using cma-es. pages 863–864, 2016.

[11] K.H. Hajikolaei, G.H. Cheng, and G.G. Wang. Optimization on metamodeling-supported iterative decomposition. *Journal of Mechanical Design, Transactions of the ASME*, 138(2), 2016.

[12] N. Hansen and A. Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 312–317, May 1996.

[13] Nikolaus Hansen. Benchmarking a BI-Population CMA-ES on the BBOB-2009 Function Testbed. In *ACM-GECCO Genetic and Evolutionary Computation Conference*, Montreal, Canada, July 2009.

[14] Pierre Hansen, Nenad Mladenović, and José A. Moreno Pérez. Variable neighbourhood search: methods and applications. *Annals of Operations Research*, 175(1):367–407, 2010.

[15] Hajira Jabeen, Zunera Jalil, and Abdul Rauf Baig. Opposition based initialization in particle swarm optimization (O-PSO). *Proceedings of the 11th annual conference companion on Genetic and evolutionary computation conference - GECCO '09*, page 2047, 2009.

[16] Voratas Kachitvichyanukul. GA/PSO/DE..Comparison of Three Evolutionary Algorithms: GA, PSO, and DE. *Industrial Engineering and Management Systems*, 11(3):215–223, 2012.

[17] G. Karagiannis, B.A. Konomi, G. Lin, and F. Liang. Parallel and interacting stochastic approximation annealing algorithms for global optimisation. *Statistics and Computing*, 27(4):927–945, 2017.

[18] K. Kawakami and Zhi Qi Meng. Improvement of Particle Swarm Optimization. *PIERS Online*, 5(2):261–264, 2009.

[19] Palvinder Singh Mann and Satvir Singh. Improved artificial bee colony metaheuristic for energy-efficient clustering in wireless sensor networks. *Artificial Intelligence Review*, pages 1–26, 2017.

[20] Mahamed G. H. Omran and Salah Al-Sharhan. Using opposition-based learning to improve the performance of particle swarm optimization. *2008 IEEE Swarm Intelligence Symposium*, (1):1–6, 2008.

[21] Srinivas Pasupuleti and Roberto Battiti. The gregarious particle swarm optimizer (G-PSO). *Proceedings of the 8th annual conference on Genetic and evolutionary computation - GECCO '06*, 1(0):67, 2006.

[22] W. L. Price. Global optimization by controlled random search. *Journal of Optimization Theory and Applications*, 40(3):333–348, 1983.

[23] Shahryar Rahnamayan, Hamid R. Tizhoosh, and Magdy M. Salama. Opposition-based differential evolution. *Studies in Computational Intelligence*, 143(1):155–171, 2008.

[24] M.C. Robini, M. Ozon, C. Frindel, F. Yang, and Y. Zhu. Global diffusion tractography by simulated annealing. *IEEE Transactions on Biomedical Engineering*, 64(3):649–660, 2017.

[25] Eduardo Segredo, Ben Paechter, Carlos Segura, and Carlos I. González-Vila. On the comparison of initialisation strategies in differential evolution for large scale optimisation. *Optimization Letters*, pages 1–14, 2017.

[26] Jun Sun, Wenbo Xu, and Bin Feng. A global search strategy of quantum-behaved particle swarm optimization. In *IEEE Conference on Cybernetics and Intelligent Systems, 2004.*, volume 1, pages 111–116 vol.1, Dec 2004.

[27] El-Ghazali Talbi. *Metaheuristics*. Wiley, 1 edition, 2009.

[28] M. Tian, X. Gao, and C. Dai. Differential evolution with improved individual-based parameter setting and selection strategy. *Applied Soft Computing Journal*, 56:286–297, 2017.

[29] H R Tizhoosh. Opposition-Based Learning: A New Scheme for Machine Intelligence. *Computational Intelligence for Modelling, Control and Automation, 2005 and International Conference on Intelligent Agents, Web Technologies and Internet Commerce, International Conference on*, 1:695–701, 2005.

[30] L.M. Zheng, S.X. Zhang, K.S. Tang, and S.Y. Zheng. Differential evolution powered by collective information. *Information Sciences*, 399:13–29, 2017.

[31] Jianhong Zhou, Wei Fang, Xiaojun Wu, Jun Sun, and Shi Cheng. An Opposition-Based Learning Competitive Particle Swarm Optimizer. pages 515–521, 2016.