METAHEURÍSTICAS

Basic Dynamics of the Universe - BDU

Pedro Gallego López

Algoritmo de optimización para el problema CEC 2017

28 de junio de 2021

Índice general

Dinámicas básicas del Universo Metaheurística

Esta memoria es el reflejo del desarrollo de una metaheurística basada en la participión que tiene la fuerza gravitatoria dentro de las dinámicas y los fenómenos más básicos del Universo. En particular, se va a intentar identificar la masa de un cuerpo con la calidad de la solución que aporta. De esta forma, intuitivamente, una buena solución aplicará una fuerza de atracción alta sobre otros cuerpos intentando mejorar la solución que representan estos otros.

1.1. Fundamentos

La metaheurística desarrollada utiliza una "física relajada" tanto para la simplificación de la misma como para unos resultados intuitivos, esto es: no todos los fundamentos físicos se han tomado al pie de la letra aunque hayan sido la guía de principio a fin de la elaboración de la metaheurística. Empezando por el ingrediente principal, la gravedad: el uso de la gravedad en nuestra metaheurística es de especial relevancia y es por ello por lo que hay que entender su principal función en nuestro trabajo.

La gravedad es una *fuerza central*. Decimos que una fuerza es central cuando siempre se dirige al mismo punto, independientemente del movimiento que tenga el cuerpo y cuando su valor depende exclusivamente de la distancia del cuerpo a dicho punto. Lo cierto es que la fuerza gravitatoria que ejerce el Sol sobre un planeta, independientemente de cómo sea la órbita de este, se dirige siempre desde el planeta hacia el propio Sol y su valor es inversamente proporcional a la distancia entre ellos. Esto es,

$$F_g = G \cdot \frac{M \cdot m}{r^2} = \frac{cte}{r^2}$$

Así, la fuerza que por ejemplo puede ejercer el Sol sobre los planetas del Sistema Solar es una *fuerza central* dirigida siempre hacia el propio Sol y cuyo valor es inversamente proporcional al cuadrado de la distancia entre este y el planeta. Esto se cumple tanto si la órbita es circular como elíptica.

Así, la gravedad tiene dos repercusiones directas que nos serán de gran utilidad:

1. **Las órbitas**: volviendo al ejemplo del Sistema Solar, la *fuerza central* ejercida por el Sol, es la que provoca el movimiento de traslación de los planetas, creando unas órbitas.

2. **La atracción**: Una gran masa ejerce una fuerza sobre otros cuerpos que provoca que estos sean atraídos.

En cuanto a los elementos que constituyen el Universo y que haremos uso están: los agujeros negros, las galaxias, los sistemas planetarios y los planetas y estrellas. El Universo está formado principalmente por galaxias. Las galaxias a su vez pueden contener sistemas planetarios (sistemas similares al Sistema Solar) en donde existe una estrella sobre la que orbitan un determinado número de planetas. Por otra parte tenemos los agujeros negros que se definen como una región finita del espacio en cuyo interior existe una concentración de masa lo suficientemente alta como para generar un campo gravitatorio tal que ninguna partícula material puede escapar de ella.

1.2. Descripción de la Metaheurística

Habiendo descrito los principales elementos que participan en nuestra Metaheurística vamos a describir su utilidad real dentro de esta. El espacio de nuestra metaheurística es el espacio de soluciones. Este espacio será considerado como *el Universo*. En nuestro Universo unificaremos la idea de *Galaxia* y *Sistema Planetario* de forma que una Galaxia contendrá únicamente un Sistema Planetario. Todas las Galaxias tendrán, por simplicidad, el mismo número de planetas y una única estrella. La *Estrella* de una galaxia se definirá como el planeta de su galaxia que mejor solución represente (intentando identificar que la Estrella va a ser el elemento con "más masa" y por lo tanto con mayor campo gravitatorio). Así, las galaxias serán en última instancia nuestros elementos explotadores del espacio.

Con esta definición, si nos damos cuenta, en una Galaxia es posible que en un momento un planeta sea nombrado *Estrella* pero en otro momento deje de serlo, teniendo la etiqueta de *Estrella* aquel planeta que es mejor. Los *Planetas* orbitarán alrededor de su *Estrella*, explorando el entorno local de la solución representada por la *Estrella*.

Por otro lado, identificaremos un *Agujero Negro* con la mejor solución obtenida hasta el momento. Este *Agujero Negro* ejercerá una fuerza de atracción alta sobre todas las galaxias, provocando un movimiento en estas. Las galaxias que se aproximen demasiado al *Agujero Negro* desaparecerán de nuestro Universo.

Algorithm 1: Metaheurística Universo

```
1  Definición de las Galaxias en el espacio del Universo
2  Mientras [Condición de parada] hacer:
3        Para cada galaxia G:
4              Para cada planeta P_G ∈ G con P_G ≠ Star_G:
5                    Orbitar(P_G, Star_G) y actualizar Star_G
6        AgujeroNegro = mejor_solucion
7        Para cada galaxia G:
8              Atraer(G, AgujeroNegro, fuerza_atraccion_G) y actualizamos el Universo
9              dist=distancia(G,mejor_solucion)
10             Si dist<umbral_1 and G ≠mejor_galaxia and num_galaxias> umbral_2:
11                   Eliminamos la galaxia G. (El AgujeroNegro la ha absorbido)
```

En Algorithm 1 vemos la idea básica de proceder con esta metaheurística. A grandes rasgos lo que se intenta es que las galaxias sean equipos de individuos explotadores. Las galaxias al final están repartidas por el espacio y cada una explotará su zona a través del operador *Orbitar*, que hará que los planetas giren alrededor de su estrella, explorando el espacio local de soluciones de la misma. Por otro lado, se fomenta la explotación del espacio de soluciones a través del Agujero Negro. Al final, el Agujero Negro es la mejor solución, y la explotación se consigue intentando atraer a las demás galaxias a ese punto. La atracción se hace de forma gradual, por lo que las galaxias que son atraídas avanzarán hasta el agujero negro a través de un camino, ¡un camino que explotarán! por lo que se provoca una clara exploración del espacio.

1.3. Descripción del problema CEC 2017

Nuestro problema se basa en resolver problemas de optimización de funciones con distinto nivel de complejidad y distinta dimensión. Todas las funciones son problemas de minimización definidas como:

$$\text{mín } f(\mathbf{x}), \qquad \mathbf{x} = [x_1, x_2, ..., x_D]^T$$

siendo D la dimensión.

El óptimo está aleatoriamente distribuido dentro de $[-80, 80]^D$ y nuestro rango de búsqueda es $[-100, 100]^D$. Como se ha dicho al principio son 30 funciones con distinto nivel de complejidad, las cuales se van a evaluar en distintas dimensiones: 10 y 30. La dimensión 50 y 100 queda fuera de este proyecto debido a su alto coste computacional.

Se sigue un criterio de parada común para cualquier algoritmo que intente solucionarlo: el número máximo de evaluaciones es $10.000 \cdot dimension$. Las funciones van numeradas del 1 al 30. Se va a medir el error con respecto su óptimo, que el óptimo es $fun_{id} \cdot 100$.

Codificación del problema

Las soluciones serán vectores $D - dimensionales$ donde cada posición del vector contendrá un valor real entre -100 y 100. Se utilizará como material auxiliar el framework ofrecido en

`https://github.com/dmolina/cec2017real/`

1.4. Algoritmos de comparación

Antes de nada haremos un breve resumen sobre el funcionamiento de estos tres algoritmos para poder hacer una comparación con criterio.

Differential Evolution - DE

Es un modelo evolutivo que enfatiza la mutación, utiliza un operador de cruce/recombinación a posteriori de la mutación. Fue propuesto para optimización con parámetros reales.

- Inicialización: una población $P_{x,0}$ de N_p vectores de parámetros D-dimensionales $x_{i,0} = [x_{1,i,0}, ..., x_{D,i,0}]$, $i = 1, ..., N_p$ se genera aleatoriamente dentro de unos límites inferiores y superiores previos $b_L = [b_{1,L}, ..., b_{D,L}]$ y $b_U = [b_{1,U}, ..., b_{D,U}]$.
- Generación: con respecto a cada vector $x_{i,g}$ en la población actual, llamado vector objetivo, se genera un vector mutado $v_{i,g}$ añadiendo un vector diferencia, escalado y aleatoriamente muestreado, a un vector base aleatoriamente seleccionado de la población actual. En la generación $g - esima$ se genera una población $P_{u,g}$ consistente de N_p vectores D-dimensionales $u_{i,g} = [u_{1,i,g}, ..., u_{D,i,g}]$ a través de operadores de mutación y recombinación aplicados a la población actual $P_{x,g}$.
- Recombinación: con respecto a cada vector objetivo $x_{i,g}$ en la población actual, un nuevo vector $u_{i,g}$ se genera cruzando el vector objetivo $x_{i,g}$ con el correspondiente vector mutado $v_{i,g}$ bajo un ratio predefinido de cruce $cr \in [0, 1]$
- Reemplazamiento: si el vectir $u_{i,g}$ tiene mejor valor de la función objetivo que su correspondiente vector objetivo $x_{i,g}$, sustituye el vector objetivo en la generación $(g+1)$; si esto no ocurre, el vector objetivo permanece en la generación $(g + 1)$

Particle Swarm Optimization - PSO

Es una metaheurística poblacional inspirada en el comportamiento social del vuelo de las bandadas de aves y el movimiento de los bancos de peces. La población se compone de varias partículas que se mueven por el espacio de búsqueda durante la ejecución del algoritmo.

El movimiento de cada partícula p depende de:

- Su mejor posición desde que comenzó el algoritmo: $pBest$
- La mejor posición de las partículas de su entorno desde que comenzó el algoritmo: $lBest$

En cada iteración se cambia aleatoriamente la velocidad de p para acercarla a las posiciones $pBest$ y $lBest$

Squirrel Search Algorithm - SSA

Algoritmo inspirado en la recolección de comida de las ardillas. El proceso de búsqueda comienza cuando las ardillas voladoras comienzan a buscar alimento.

Durante el clima cálido (otoño), las ardillas buscan recursos alimenticios planeándose de un árbol a otro. Mientras lo hacen, cambian de ubicación y exploran diferentes áreas del bosque. Como las condiciones climáticas son lo suficientemente cálidas, pueden satisfacer sus necesidades energéticas diarias más rápidamente con la dieta de bellotas disponibles en abundancia y, por lo tanto, las consumen inmediatamente después de encontrarlas. Después de cumplir con su requerimiento diario de energía, comienzan a buscar la fuente de alimento óptima para el invierno (nueces de nogal).

El almacenamiento de nueces de nogal les ayudará a mantener sus necesidades energéticas en condiciones climáticas extremas y reducirá los costosos viajes de alimentación y, por lo tanto, aumentará la probabilidad de supervivencia. Durante el invierno, la pérdida de la cobertura foliar en los bosques caducifolios aumenta el riesgo de depredación y, por lo tanto, se vuelven menos activos pero no hibernan en invierno. Al final de la temporada de invierno, las ardillas voladoras vuelven a activarse. Este es un proceso repetitivo y continúa hasta la vida útil de una ardilla voladora y forma la base de SSA. Así, los operadores a seguir serán

- Inicialización aleatoria, evaluación y ordenación
- Selección aleatoria
- Generación de nuevas localizaciones
- Aerodinámicas del planeo
- Monitorización de las condiciones de la estación del año
- Relocalización aleatoria al final del invierno

1.5. Algoritmo de optimización basado en dinámicas básicas del Universo para el CEC 2017

Vamos a ver una aplicación de la metaheurística vista en la sección 1.2 en el problema del CEC 2017. El interés de esta metaheurística está en su potencial capacidad tanto de exploración como explotación como bien se ha comentado.

Para adaptarlo a nuestra problema se han añadido algunas operaciones auxiliares para mejorar el proceso. Entre estas operaciones está la incorporación de una especie de `learning_rate` para el proceso de atracción que describiremos después de ver el algoritmo.

Algorithm 2: Algoritmo Universo para CEC 2017.

Leyenda: hiperparámetros, operadores no explicados, operadores ya explicados

```
1  Creamos n_0 soluciones aleatorias. Cada solución representará un planeta de una galaxia.
2  Creamos n_1−1 vecinos del planeta 0 creado en el paso 1 para cada galaxia
3  Para cada galaxia G establecemos fuerza_atraccion_G
4  Evaluamos (asignamos estrellas) y guardamos el mejor_fitness, la mejor_solucion, la
     mejor_galaxia

5  evals = 0
6  Mientras [evals< 10000·dimension] hacer:
7        Para cada galaxia G:
8              Para cada planeta P_G ∈ G con P_G ≠ Star_G:
9                    intentos=0
10                   Mientras No Mejora P_G or [intentos<max_intentos] hacer:
11                         intentos++
12                         Si distancia(P_G,Star_G)>=1 hacer:
13                               Orbitar(P_G, Star_G)
14                         En otro caso:
15                               P_G = nuevo vecino a distancia n_0
16                         Evaluamos y actualizamos la estrella si P_G es mejor que Star_G
17             Actualizamos el Universo: mejor_fitness, mejor_solucion y mejor_galaxia

18       AgujeroNegro = mejor_solucion
19       Para cada galaxia G:
20             Atraer(G, AgujeroNegro, fuerza_atraccion_G)
21             Actualizamos el Universo
22             Si mejora G:
23                   fuerza_atraccion_G+ = (1−fuerza_atraccion_G)· η
24             En otro caso:
25                   fuerza_atraccion_G− =fuerza_atraccion_G· η
26             dist=distancia(G,mejor_solucion)
27             Si dist<umbral_1 and G ≠mejor_galaxia and num_galaxias> umbral_2:
28                   Eliminamos la galaxia G. (El AgujeroNegro la ha absorbido)
```

En el Algorithm 2 podemos ver en pseudocódigo como sería nuestro algoritmo. Hay varios hiperparámetros que vamos a pasar a comentar. Los hiperparámetros n_0, n_1 hacen referencia al número de galaxias y de planetas por galaxia respectivamente. Con n_0, al aumentarlo vamos a aumentar la capacidad de exploración del propio método, ya que tendremos más "exploradores" repartidos por el espacio. Por otro lado con n_1 regulamos la capacidad de explotación dentro de un entorno de la galaxia.

Más hiperparámetros serían $fuerza\_atraccion_G$ y η, que son los que hemos dicho que están relacionados con el `learning_rate` en el proceso de atracción del Agujero Negro. Empezando por $fuerza\_atraccion_G$, este es un valor único de la galaxia G, este valor nos dice como de fuerte es la atracción del Agujero Negro a G, haciendo que cuanto mayor sea la fuerza de atracción, menor distancia resultante habrá entre el agujero negro y la galaxia G. Para regular dinámicamente esta fuerza está el parámetro η que tiene un comportamiento adaptativo sobre la calidad de la atracción. Vemos que esta regulación

- Aumenta más rápido conforme $fuerza\_atraccion_G$ se acerca a 1 y aumenta más lento conforme $fuerza\_atraccion_G$ se acerca a 0.
- Disminuye más rápido conforme $fuerza\_atraccion_G$ se acerca a 0 y disminuye más lento conforme se acerca a 1.

1.5.1. Operadores

Operador Vecino

Para generar vecinos se ha utilizado un operador que añade una pequeña perturbación a las coordenadas del planeta sobre el que se va a crear el vecindario. Se generan D números aleatorios comprendidos en el rango $r_i \in [-dist, dist]$, quedando el vecino determinado por sus coordenadas:

$$neighboor[i] = planeta[i] + r_i$$

en donde $dist$ es un parámetro que se le pasa al operador. Será por lo general un valor no muy grande, ya que tenemos que tener en cuenta que las galaxias se tienen que centrar en la explotación de una zona.

Orbitar

Para orbitar nos vamos a salir un poco de los fundamentos. Puesto que no tenemos capacidad infinita de memoria y queremos optimizar la explotación, se ha decidido que la órbita de los planetas sea un movimiento en espiral. Este movimiento en espiral implica que un planeta gire alrededor de su estrella mientras se acerca a su vez aplicando una $fuerza\_central$ la estrella. El problema viene cuando el planeta está muy cerca de la estrella y ya se confunden, en este momento devolvemos al planeta a una distancia lejana de la estrella provocando un "*reinicio de su búsqueda*" que se podría entender también como una fuerza repulsora de la estrella hacia el planeta. Esta acción no se representa en nuestra realidad física, y no sería necesaria en nuestro problema si dispusiesemos en cada Galaxia de varios sistemas planetarios con varios planetas, en donde la explotación se podría hacer con una órbita estacionaria (sin atracción ni repulsión, simplemente una trayectoria fija a una distancia fija) pero esto requeriría muchísima memoria, este es el motivo por lo que se ha optado por este método. Con este método barremos un área local importante, que nos acercaría a lo que sería el modelo de una galaxia normal en espiral.

Algorithm 3: Orbitar.

Leyenda: hiperparámetros, operadores no explicados, operadores ya explicados

Parámetros: 1: planeta, **2:** estrella

```
distancia_objetivo = C· distancia_euclidea(planeta,estrella)
dist_objetivo_cuadrado_restante = distancia_objetivo^2
Para cada coordenada i de planeta barajado:
    valor = random∈ [−1, 1]
    planeta[i]=estrella[i]+valor·√(dist_obj_cuadrado_restante)
    dist_obj_cuadrado_restante -= valor^2·dist_obj_cuadrado_restante
```

Luego si la distancia del planeta a la estrella es mayor a un $umbral_3$ el planeta orbitará. En otro caso, se mandará a una distancia fija de la estrella.

Podemos pensar que orbitar en un espacio $n-dimensional$ es algo complejo, y puesto que nuestra ejecución no simula un espacio continuo de tiempo, tenemos que simular iteraciones en saltos de tiempo. Esto nos facilita la órbita en cierto modo: sabemos que con nuestra órbita en espiral, tras cada iteración, el planeta tiene que estar en una posición más cercana a la estrella. Para ahorrar cómputo se ha decidido, mediante una constante C indicar cuánto se acerca el planeta respecto a la distancia que hay antes orbitar. De esta forma el planeta, tras cada iteración estará $1/C$ veces más cerca de la estrella. El ahorro de cómputo viene en el cálculo de la posición exacta del planeta. Se ha decidido que el planeta estará dentro de la $hiperesfera$ $\mathbb{S}^{n-1}$ en una posición aleatoria. Así, solo nos tenemos que preocupar de que sus componentes estén a una distancia concreta: $C \cdot distancia_{anterior}$.

Atraer

Algorithm 4: Atraer.

Leyenda: hiperparámetros, operadores no explicados, operadores ya explicados

Parámetros: 1: galaxia, **2:** AgujeroNegro, **3:** $fuerza\_atraccion_G$

```
n_cambios = fuerza_atraccion_G·dimension

Para cada j ∈ barajar([0, 1, ..., dimension]):
      Para cada planeta P ∈galaxia:
            Si aún no se han producido n_cambios hacer:
                  P[i][j]=AgujeroNegro[j], y sumamos +1 al número de cambios

Actualizamos el fitness y con ello la Galaxia
return True si ha mejorado el fitness de la estrella, False en otro caso.
```

Este operador se va a aplicar sobre las Galaxias hacia el Agujero Negro. Nuestro operador de acercar no va a ser el intuitivo de acercarse en línea recta, sino que nuestra visión de acercar va referida a disminuir la distancia entre la Galaxia y el Agujero Negro. Esto vamos a hacerlo con un operador parecido a un cruce uniforme en algoritmos genéticos, donde se cruzarán los propios planetas de la galaxia con el agujero negro. El resultado del cruce del planeta i-ésimo será el nuevo planeta i-ésimo. El acercamiento será más o menos fuerte dependiendo del hiperparámetro $fuerza\_atraccion_G$.

1.5.2. Análisis de los experimentos y resultados

DIM 10	*DE*	*PSO*	*SSA*	*UNIVERSE*	*DIM 30*	*DE*	*PSO*	*SSA*	*UNIVERSE*
F01	0.000e+00	5.255e+07	3.766e+03	4.314e+03	***F01***	4.909e+04	4.175e+09	3.765e+03	1.180e+04
F02	0.000e+00	1.000e+00	1.000e+00	2.500e+01	***F02***	1.309e+19	1.000e+00	1.000e+00	1.116e+05
F03	0.000e+00	1.989e+03	1.000e-10	2.336e+00	***F03***	3.481e+03	5.453e+04	2.416e-05	2.344e+03
F04	1.105e-04	4.684e+01	4.172e+00	1.582e+00	***F04***	8.430e+01	1.183e+03	8.388e+01	3.711e+01
F05	1.151e+02	3.212e+01	4.840e+01	3.064e+01	***F05***	2.015e+02	2.170e+02	2.649e+02	1.652e+02
F06	3.460e+01	1.001e+01	2.632e+01	2.856e+00	***F06***	6.320e+00	3.694e+01	6.384e+01	1.182e+00
F07	3.848e+01	4.275e+01	6.198e+01	5.386e+01	***F07***	2.334e+02	3.596e+02	4.609e+02	2.131e+02
F08	2.983e+01	2.203e+01	3.849e+01	2.895e+01	***F08***	1.894e+02	1.745e+02	2.205e+02	1.725e+02
F09	1.938e+02	5.686e+01	2.439e+02	9.539e+01	***F09***	6.530e+01	2.842e+03	6.212e+03	4.067e+03
F10	3.597e+02	1.077e+03	1.086e+03	6.958e+02	***F10***	3.764e+03	6.938e+03	4.844e+03	3.304e+03
F11	1.942e-02	3.843e+01	9.529e+01	3.332e+01	***F11***	7.958e+01	1.207e+03	1.537e+02	1.666e+02
F12	4.931e+00	2.517e+06	2.313e+04	7.083e+04	***F12***	3.258e+05	3.586e+08	2.528e+06	8.798e+05
F13	5.988e+00	8.409e+03	8.101e+03	4.537e+03	***F13***	1.536e+02	4.508e+07	1.926e+05	3.206e+04
F14	5.240e-02	9.993e+01	1.181e+02	2.994e+01	***F14***	7.100e+01	3.059e+05	3.286e+03	1.283e+04
F15	6.060e-02	2.066e+03	5.244e+02	1.002e+01	***F15***	6.256e+01	2.737e+05	8.852e+04	1.465e+04
F16	4.561e+02	1.415e+02	1.884e+02	1.925e+02	***F16***	1.319e+03	1.568e+03	1.658e+03	8.848e+02
F17	2.350e+01	6.497e+01	9.769e+01	6.709e+01	***F17***	4.809e+02	4.725e+02	8.471e+02	5.431e+02
F18	3.630e-02	1.484e+04	1.360e+04	1.207e+03	***F18***	6.122e+01	2.170e+06	8.855e+04	2.045e+05
F19	5.192e-03	3.220e+03	1.246e+02	8.248e+00	***F19***	3.572e+01	1.260e+06	1.510e+05	9.816e+03
F20	3.837e+02	8.444e+01	1.058e+02	3.278e+01	***F20***	2.751e+02	4.621e+02	6.450e+02	4.401e+02
F21	1.889e+02	1.320e+02	1.044e+02	1.926e+02	***F21***	3.255e+02	4.113e+02	4.502e+02	3.792e+02
F22	1.005e+02	7.740e+01	1.091e+02	1.581e+02	***F22***	1.002e+02	1.027e+03	4.196e+03	3.055e+03
F23	8.098e+02	3.304e+02	3.487e+02	3.338e+02	***F23***	5.346e+02	6.404e+02	8.365e+02	6.046e+02
F24	1.000e+02	1.810e+02	2.597e+02	3.098e+02	***F24***	6.059e+02	7.095e+02	9.264e+02	6.588e+02
F25	4.040e+02	4.478e+02	4.380e+02	3.926e+02	***F25***	3.870e+02	6.864e+02	4.176e+02	3.897e+02
F26	2.706e+02	3.729e+02	3.904e+02	4.389e+02	***F26***	4.038e+02	3.369e+03	5.522e+03	3.569e+03
F27	3.897e+02	4.134e+02	4.117e+02	4.068e+02	***F27***	4.925e+02	8.068e+02	6.550e+02	4.753e+02
F28	3.517e+02	4.698e+02	4.318e+02	3.339e+02	***F28***	3.943e+02	1.105e+03	4.896e+02	3.901e+02
F29	2.375e+02	3.193e+02	3.676e+02	3.088e+02	***F29***	1.025e+03	1.409e+03	1.934e+03	8.970e+02
F30	8.051e+04	6.352e+05	1.469e+06	1.562e+04	***F30***	3.657e+03	1.359e+07	2.308e+06	5.270e+03
Best	18	5	1	6	*Best*	16	2	3	10

Cuadro 1.1: Comparativa de resultados de los distintos algoritmos. A la izquierda está la comparativa de las funciones con dimensión 10. A la derecha está la comparativa de las funciones con dimensión 30.

Ejecutaremos el algoritmo 10 veces con 10 semillas distintas para cada dimensión (10 y 30). Con las medias de estas ejecuciones se comparará con los algoritmos DE, PSO y SSA. En el

Cuadro 1.1 se pueden ver los resultados finales de cada algoritmo.

Vemos que el algoritmo es cuanto menos prometedor, siendo el segundo mejor de entre los cuatro. A pesar de ello DE tiene un mejor desempeño en general. En los Cuadros 1.2 y 1.3 podemos ver un resumen de los resultados en los duelos 1 versus 1: Universe vs cada uno de los otros. En estos cuadros se ve como gana a PSO y SSA. Además podemos ver como al subir la dimensionalidad, el porcentaje de victorias es mayor en todo los casos, lo que nos puede decir que el algoritmo se comporta mejor que sus competidores con el aumento de dimensionalidad. En concreto, el número de victorias frente a PSO sube muchísimo, siendo tanto demérito de PSO como mérito de UNIVERSE.

DIM 10	**DE**	**PSO**	**SSA**
Número de victorias vs	10	19	21
Porcentaje de victorias vs	33.33 %	63.33 %	70 %

Cuadro 1.2: Número y porcentaje de victorias de Universe en 1vs1 con cada uno de los otros algoritmos en Dimensión 10.

DIM 30	**DE**	**PSO**	**SSA**
Número de victorias vs	13	25	22
Porcentaje de victorias vs	43.33 %	83.33 %	73.33 %

Cuadro 1.3: Número y porcentaje de victorias de Universe en 1vs1 con cada uno de los otros algoritmos en Dimensión 30.

Así, un algoritmo de evolución clásico como es DE es un algoritmo que tiene un desempeño mejor de lo que pueden tener algoritmos como PSO, SSA y el que hemos desarrollado. Al final nuestro algoritmo nace de una naturaleza que provoca la atracción hacia la mejor solución encontrada, fomentando la exploración con la atracción de los agujeros negros y haciendo que este algoritmo no tenga una convergencia clara de soluciones. Así, UNIVERSE es un algoritmo que explora y explota en igual medida durante todo el proceso. Sin embargo, los algoritmos evolutivos sabemos que de no ser por la mutación se tiene una convergencia, esto es que es más probable que encuentren los mejores mínimos, puesto que tienen la porción perfecta de exploración y explotación en función de lo avanzada que esté la ejecución.

Por otro lado, PSO y SSA son dos algoritmos que no han sido capaces de realizar un buen desempeño (respecto al resto). Así podemos concluir que DE es el mejor algoritmo entre los 4 para este problema, seguido del propuesto UNIVERSE.

1.6. Hibridación

En esta sección se va a proponer una mejora con una hibridación con un algoritmo de búsqueda Local. En concreto utilizaremos el método *Solis Wets*, este método viene detallado en los códigos 6 y 7.

Algorithm 5: Algoritmo Hibridación Universo con Solis Wet para CEC 2017.

Leyenda: hiperparámetros, operadores no explicados, operadores ya explicados

```
1  Creamos n_0 soluciones aleatorias. Cada solución representará un planeta de una galaxia.
2  Creamos n_1−1 vecinos del planeta 0 creado en el paso 1 para cada galaxia
3  Para cada galaxia G establecemos fuerza_atraccion_G
4  Evaluamos (asignamos estrellas) y guardamos el mejor_fitness, la mejor_solucion, la
     mejor_galaxia

5  evals = 0
6  Mientras [evals< 10000·dimension] hacer:
7        Para cada galaxia G:
8              Para cada planeta P_G ∈ G
9                    intentos=0
10                   Si P_G ≠ Star_G:
11                         Mientras No Mejora P_G or [intentos<max_intentos] hacer:
12                               intentos++
13                               Si distancia(P_G,Star_G)>=1 hacer:
14                                     Orbitar(P_G, Star_G)
15                               En otro caso:
16                                     P_G = nuevo vecino a distancia n_0
17                         Evaluamos y actualizamos la estrella si P_G es mejor que Star_G
18                   Si P_G = Star_G:
19                         soliswets(P_G,fitness_PG, 3, 5·max_intentos)
20             Actualizamos el Universo: mejor_fitness, mejor_solucion y mejor_galaxia

21       AgujeroNegro = mejor_solucion
22       Para cada galaxia G:
23             Atraer(G, AgujeroNegro, fuerza_atraccion_G)
24             Actualizamos el Universo
25             Si mejora G:
26                   fuerza_atraccion_G+ = (1−fuerza_atraccion_G)· η
27             En otro caso:
28                   fuerza_atraccion_G− =fuerza_atraccion_G· η
29             dist=distancia(G,mejor_solucion)
30             Si dist<umbral_1 and G ≠mejor_galaxia and num_galaxias> umbral_2:
31                   Eliminamos la galaxia G. (El AgujeroNegro la ha absorbido)
```

La explotación que nos puede ofrecer este método casa perfectamente con con el proceso de *Orbitar*. Tal y como hemos descrito nuestro algoritmo, al *Orbitar* solo intervienen los planetas, quedando la estrella a un lado y sin ningún papel más que ser el centro de las órbitas de la galaxia. Así, tiene sentido que la estrella tenga un papel de explotación, por lo tanto se ha decidido que la estrella explote la zona con el método *Solis Wets* mientras que los demás planetas Orbitan. Esto se puede ver reflejado en el código 5.

Con esta hibridación se espera una mucha mayor explotación de la zona de cada galaxia. Con esta explotación extra, quitamos evaluaciones del proceso de atracción del Agujero Negro, que era nuestro método de exploración. Sin embargo, si lo pensamos, las primeras iteraciones lo más importante es la exploración, para encontrar alguna zona que sea buena candidata a tener un óptimo local cercano al global. Visto así, las primeras iteraciones de exploración seguirán existiendo, con la ventaja de que explotaremos mejor los mínimos locales que encontremos.

Algorithm 6: Solis Wets.

Leyenda: hiperparámetros, operadores no explicados, operadores ya explicados

Parámetros: 1: planeta, **2:** fit, **3:** delta, **4:** max_evals

```
1  evals=1, bias=0, num_success=num_failed=0
2  Mientras [evals < max_evals] hacer:
3        dif = vector aleatorio (0,delta)
4        newsol = planeta + bias + dif
5        evals++

6        Si [fitness(newsol)<fitness(planeta)] hacer:
7              planeta=newsol
8              bias = 0.2bias+0.4(dif+bias)       (operación de vectores)
9              num_success++
10             num_failed=0
11       En otro caso:
12             newsol'=planeta-bias-dif
13             evals++
14             Si [fitness(newsol') < fitness(planeta)] hacer:
15                   planeta=newsol'
16                   bias=bias-0.4(dif+bias)       (operación de vectores)
17                   num_success++
18                   num_failed=0
19             En otro caso:
20                   bias=bias/2
21                   num_failed++
22                   num_success=0
23       update_step_size(num_failed,num_success)
```

Algorithm 7: Solis Wets. Procedimiento update_step_size.

Leyenda: hiperparámetros, operadores no explicados, operadores ya explicados

Parámetros: 1: num_failed, **2:** num_success

```
1  Si [num_success ≥ 5] hacer:
2        step_size*=2
3        num_success=0
4  Si [num_failed ≥ 3] hacer:
5        step_size/=2
6        num_failed=0
```

1.6.1. Análisis de los experimentos y resultados

Como hicimos en la sección anterior, ejecutaremos el algoritmo 10 veces con 10 semillas distintas para cada dimensión (10 y 30). Con las medias de estas ejecuciones se comparará con los algoritmos DE, PSO y SSA y por último se sacarán conclusiones de la hibridación.

En el Cuadro 1.4 se puede ver como con esta hibridación conseguimos ganar más. En concreto, si nos fijamos en los duelos individuales (se pueden ver en Cuadro 1.5 y Cuadro 1.6) vemos como mejoramos los porcentajes por lo general. Además, fijándonos otra vez en el Cuadro 1.4 se puede observar como las diferencias entre el resultado conseguido y el mejor de los 4 son menores que con el algoritmo original.

DIM 10	*DE*	*PSO*	*SSA*	*HYBRID*	*DIM 30*	*DE*	*PSO*	*SSA*	*HYBRID*
F01	0.000e+00	5.255e+07	3.766e+03	1.772e+03	***F01***	4.909e+04	4.175e+09	3.765e+03	4.024e+03
F02	0.000e+00	1.000e+00	1.000e+00	8.280e+01	***F02***	1.309e+19	1.000e+00	1.000e+00	6.375e+05
F03	0.000e+00	1.989e+03	1.000e-10	9.262e-01	***F03***	3.481e+03	5.453e+04	2.416e-05	1.291e+04
F04	1.105e-04	4.684e+01	4.172e+00	9.622e+00	***F04***	8.430e+01	1.183e+03	8.388e+01	5.845e+01
F05	1.151e+02	3.212e+01	4.840e+01	2.388e+01	***F05***	2.015e+02	2.170e+02	2.649e+02	1.809e+02
F06	3.460e+01	1.001e+01	2.632e+01	3.588e+00	***F06***	6.320e+00	3.694e+01	6.384e+01	2.798e+00
F07	3.848e+01	4.275e+01	6.198e+01	4.257e+01	***F07***	2.334e+02	3.596e+02	4.609e+02	2.269e+02
F08	2.983e+01	2.203e+01	3.849e+01	2.617e+01	***F08***	1.894e+02	1.745e+02	2.205e+02	1.937e+02
F09	1.938e+02	5.686e+01	2.439e+02	5.477e+01	***F09***	6.530e+01	2.842e+03	6.212e+03	4.227e+03
F10	3.597e+02	1.077e+03	1.086e+03	5.369e+02	***F10***	3.764e+03	6.938e+03	4.844e+03	3.541e+03
F11	1.942e-02	3.843e+01	9.529e+01	1.804e+01	***F11***	7.958e+01	1.207e+03	1.537e+02	1.517e+02
F12	4.931e+00	2.517e+06	2.313e+04	7.175e+04	***F12***	3.258e+05	3.586e+08	2.528e+06	1.357e+06
F13	5.988e+00	8.409e+03	8.101e+03	5.208e+03	***F13***	1.536e+02	4.508e+07	1.926e+05	9.595e+03
F14	5.240e-02	9.993e+01	1.181e+02	2.897e+01	***F14***	7.100e+01	3.059e+05	3.286e+03	1.502e+04
F15	6.060e-02	2.066e+03	5.244e+02	1.658e+01	***F15***	6.256e+01	2.737e+05	8.852e+04	1.114e+04
F16	4.561e+02	1.415e+02	1.884e+02	1.512e+02	***F16***	1.319e+03	1.568e+03	1.658e+03	8.123e+02
F17	2.350e+01	6.497e+01	9.769e+01	5.162e+01	***F17***	4.809e+02	4.725e+02	8.471e+02	4.736e+02
F18	3.630e-02	1.484e+04	1.360e+04	1.926e+02	***F18***	6.122e+01	2.170e+06	8.855e+04	2.125e+05
F19	5.192e-03	3.220e+03	1.246e+02	1.908e+01	***F19***	3.572e+01	1.260e+06	1.510e+05	5.831e+03
F20	3.837e+02	8.444e+01	1.058e+02	3.210e+01	***F20***	2.751e+02	4.621e+02	6.450e+02	3.501e+02
F21	1.889e+02	1.320e+02	1.044e+02	1.773e+02	***F21***	3.255e+02	4.113e+02	4.502e+02	3.716e+02
F22	1.005e+02	7.740e+01	1.091e+02	1.016e+02	***F22***	1.002e+02	1.027e+03	4.196e+03	3.534e+03
F23	8.098e+02	3.304e+02	3.487e+02	3.223e+02	***F23***	5.346e+02	6.404e+02	8.365e+02	6.047e+02
F24	1.000e+02	1.810e+02	2.597e+02	2.541e+02	***F24***	6.059e+02	7.095e+02	9.264e+02	6.008e+02
F25	4.040e+02	4.478e+02	4.380e+02	3.634e+02	***F25***	3.870e+02	6.864e+02	4.176e+02	3.786e+02
F26	2.706e+02	3.729e+02	3.904e+02	3.196e+02	***F26***	4.038e+02	3.369e+03	5.522e+03	3.562e+03
F27	3.897e+02	4.134e+02	4.117e+02	4.141e+02	***F27***	4.925e+02	8.068e+02	6.550e+02	4.721e+02
F28	3.517e+02	4.698e+02	4.318e+02	4.512e+02	***F28***	3.943e+02	1.105e+03	4.896e+02	4.146e+02
F29	2.375e+02	3.193e+02	3.676e+02	2.810e+02	***F29***	1.025e+03	1.409e+03	1.934e+03	7.893e+02
F30	8.051e+04	6.352e+05	1.469e+06	5.809e+03	***F30***	3.657e+03	1.359e+07	2.308e+06	2.843e+03
Best	19	3	1	7	*Best*	14	3	3	11

Cuadro 1.4: Comparativa de resultados de los distintos algoritmos con el algoritmo híbrido-universo. A la izquierda está la comparativa de las funciones con dimensión 10. A la derecha está la comparativa de las funciones con dimensión 30.

En la comparación con DE, PSO y SSA podemos concluir lo mismo que hicimos con UNIVERSE en la sección anterior. Por otro lado, está claro que ha habido una mejora en general con la hibridación, consiguiendo ganar más veces en los globales y en los 1vs1 (Cuadros 1.5 y 1.6). Concluimos por lo tanto que la explotación añadida en nuestra Estrella es un mecanismo fuerte y poderoso para llegar a mejores soluciones.

DIM 10	**DE**	**PSO**	**SSA**
Número de victorias vs	10	23	23
Porcentaje de victorias vs	33.33 %	76.67 %	76.67 %

Cuadro 1.5: Número y porcentaje de victorias del Híbrido en 1vs1 con cada uno de los otros algoritmos en Dimensión 10.

DIM 30	**DE**	**PSO**	**SSA**
Número de victorias vs	14	24	25
Porcentaje de victorias vs	46.67 %	80 %	83.33 %

Cuadro 1.6: Número y porcentaje de victorias del Híbrido en 1vs1 con cada uno de los otros algoritmos en Dimensión 30.

1.7. Propuesta de mejora

Procedemos en esta sección a proponer mejoras para el mejor funcionamiento del algoritmo. La primera mejora, ya aplicada, es la del uso de un método de búsqueda local como *Solis Wet* haciendo una hibridación con el algoritmo original. Esta mejora se ha visto como funcionaba en la sección anterior.

Ahora se propone una mejora sobre esta última. La mejora consiste en, conforme se elimina una Galaxia porque ha sido absorbida, se crea otra nueva, fomentando de nuevo la exploración del espacio. A priori parece una buena idea puesto que hasta ahora estamos atados a la inicialización de las galaxias y a los movimientos de atracción que se hacían entre ellas, dejando mucho espacio sin explorar. Con esta mejora añadimos búsqueda aleatoria por el espacio, en donde puede darse el caso que nos sirva para escapar de unos mínimos locales a otros mejores. En el Algorithm 8 podemos ver el algoritmo con esta mejora.

Algorithm 8: Algoritmo Hibridación Universo con Solis Wet para CEC 2017.

Leyenda: hiperparámetros, operadores no explicados, operadores ya explicados , novedad

```
Creamos n_0 soluciones aleatorias. Cada solución representará un planeta de una galaxia.
Creamos n_1−1 vecinos del planeta 0 creado en el paso 1 para cada galaxia
Para cada galaxia G establecemos fuerza_atraccion_G
Evaluamos (asignamos estrellas) y guardamos el mejor_fitness, la mejor_solucion, la
 mejor_galaxia

evals = 0
Mientras [evals< 10000·dimension] hacer:
      Para cada galaxia G:
            Para cada planeta P_G ∈ G
                  intentos=0
                  Si P_G ≠ Star_G:
                        Mientras No Mejora P_G or [intentos<max_intentos] hacer:
                              intentos++
                              Si distancia(P_G,Star_G)>=1 hacer:
                                    Orbitar(P_G, Star_G)
                              En otro caso:
                                    P_G = nuevo vecino a distancia n_0
                        Evaluamos y actualizamos la estrella si P_G es mejor que Star_G
                  Si P_G = Star_G:
                        soliswets(P_G,fitness_PG, 3, 5·max_intentos)
            Actualizamos el Universo: mejor_fitness, mejor_solucion y mejor_galaxia

      AgujeroNegro = mejor_solucion
      Para cada galaxia G:
            Atraer(G, AgujeroNegro, fuerza_atraccion_G)
            Actualizamos el Universo
            Si mejora G:
                  fuerza_atraccion_G+ = (1−fuerza_atraccion_G)· η
            En otro caso:
                  fuerza_atraccion_G− =fuerza_atraccion_G· η
            dist=distancia(G,mejor_solucion)
            Si dist<umbral_1 and G ≠mejor_galaxia and num_galaxias> umbral_2:
                  Eliminamos la galaxia G. (El AgujeroNegro la ha absorbido)
                  Creamos una galaxia en las condiciones de la inicializacion
```

1.7.1. Análisis de los experimentos y resultados

En el Cuadro 1.7 podemos ver el resultado de las ejecuciones con esta mejora. Observamos dos cambios fuertes:

1. En la dimensión 10 se consigue un incremento de victorias en el global (pasamos de 7 a 10).

2. En la dimensión 30 se reducen el número de victorias en el global (pasamos de 11 a 8).

Analicemos estos hechos. El primer punto era lo que esperábamos al aplicar esta mejora: al añadir estocasticidad y una exploración mayor, sería posible obtener mejores mínimos.

El segundo punto, sin embargo, choca con esta idea previa que teníamos. En realidad, tiene todo el sentido del mundo, puesto que en dimensión 30 el espacio crece una barbaridad, haciendo que las inicializaciones aleatorias sean un método de busca bastante malo debido al gran aumento de posibilidades. Es por ello que con dimensión 10, al ser un espacio más reducido, si que tienes un efecto a favor. Sin embargo, con dimensión 30 tenemos un efecto contrario: estamos buscando soluciones donde no las hay por un espacio muy grande, esto se traduce por lo tanto en malgastar evaluaciones de la función fitness, haciendo que nuestro método deje de ser tan bueno.

DIM 10	***DE***	***PSO***	***SSA***	***HYBRID***	***DIM 30***	***DE***	***PSO***	***SSA***	***HYBRID***
F01	0.000e+00	5.255e+07	3.766e+03	3.151e+05	***F01***	4.909e+04	4.175e+09	3.765e+03	4.484e+03
F02	0.000e+00	1.000e+00	1.000e+00	5.210e+01	***F02***	1.309e+19	1.000e+00	1.000e+00	5.540e+17
F03	0.000e+00	1.989e+03	1.000e-10	1.098e+01	***F03***	3.481e+03	5.453e+04	2.416e-05	1.255e+04
F04	1.105e-04	4.684e+01	4.172e+00	4.456e+00	***F04***	8.430e+01	1.183e+03	8.388e+01	1.735e+02
F05	1.151e+02	3.212e+01	4.840e+01	1.363e+01	***F05***	2.015e+02	2.170e+02	2.649e+02	1.702e+02
F06	3.460e+01	1.001e+01	2.632e+01	1.518e-01	***F06***	6.320e+00	3.694e+01	6.384e+01	3.597e+00
F07	3.848e+01	4.275e+01	6.198e+01	2.769e+01	***F07***	2.334e+02	3.596e+02	4.609e+02	2.299e+02
F08	2.983e+01	2.203e+01	3.849e+01	1.483e+01	***F08***	1.894e+02	1.745e+02	2.205e+02	1.507e+02
F09	1.938e+02	5.686e+01	2.439e+02	2.740e+01	***F09***	6.530e+01	2.842e+03	6.212e+03	3.109e+03
F10	3.597e+02	1.077e+03	1.086e+03	2.839e+02	***F10***	3.764e+03	6.938e+03	4.844e+03	3.301e+03
F11	1.942e-02	3.843e+01	9.529e+01	1.118e+01	***F11***	7.958e+01	1.207e+03	1.537e+02	1.572e+02
F12	4.931e+00	2.517e+06	2.313e+04	1.744e+04	***F12***	3.258e+05	3.586e+08	2.528e+06	9.367e+06
F13	5.988e+00	8.409e+03	8.101e+03	7.371e+02	***F13***	1.536e+02	4.508e+07	1.926e+05	5.980e+03
F14	5.240e-02	9.993e+01	1.181e+02	2.145e+01	***F14***	7.100e+01	3.059e+05	3.286e+03	9.108e+03
F15	6.060e-02	2.066e+03	5.244e+02	1.456e+01	***F15***	6.256e+01	2.737e+05	8.852e+04	1.001e+04
F16	4.561e+02	1.415e+02	1.884e+02	1.048e+02	***F16***	1.319e+03	1.568e+03	1.658e+03	1.145e+03
F17	2.350e+01	6.497e+01	9.769e+01	3.016e+01	***F17***	4.809e+02	4.725e+02	8.471e+02	2.216e+02
F18	3.630e-02	1.484e+04	1.360e+04	4.164e+01	***F18***	6.122e+01	2.170e+06	8.855e+04	3.022e+05
F19	5.192e-03	3.220e+03	1.246e+02	3.664e+00	***F19***	3.572e+01	1.260e+06	1.510e+05	3.438e+03
F20	3.837e+02	8.444e+01	1.058e+02	1.467e+01	***F20***	2.751e+02	4.621e+02	6.450e+02	2.524e+02
F21	1.889e+02	1.320e+02	1.044e+02	1.156e+02	***F21***	3.255e+02	4.113e+02	4.502e+02	3.422e+02
F22	1.005e+02	7.740e+01	1.091e+02	1.021e+02	***F22***	1.002e+02	1.027e+03	4.196e+03	3.468e+03
F23	8.098e+02	3.304e+02	3.487e+02	3.198e+02	***F23***	5.346e+02	6.404e+02	8.365e+02	5.950e+02
F24	1.000e+02	1.810e+02	2.597e+02	2.115e+02	***F24***	6.059e+02	7.095e+02	9.264e+02	7.228e+02
F25	4.040e+02	4.478e+02	4.380e+02	4.272e+02	***F25***	3.870e+02	6.864e+02	4.176e+02	4.193e+02
F26	2.706e+02	3.729e+02	3.904e+02	3.001e+02	***F26***	4.038e+02	3.369e+03	5.522e+03	3.221e+03
F27	3.897e+02	4.134e+02	4.117e+02	3.985e+02	***F27***	4.925e+02	8.068e+02	6.550e+02	6.145e+02
F28	3.517e+02	4.698e+02	4.318e+02	3.868e+02	***F28***	3.943e+02	1.105e+03	4.896e+02	4.595e+02
F29	2.375e+02	3.193e+02	3.676e+02	2.776e+02	***F29***	1.025e+03	1.409e+03	1.934e+03	1.039e+03
F30	8.051e+04	6.352e+05	1.469e+06	6.012e+04	***F30***	3.657e+03	1.359e+07	2.308e+06	4.326e+05
Best	18	1	1	10	*Best*	18	1	4	8

Cuadro 1.7: Comparativa de resultados de los distintos algoritmos con el algoritmo híbrido-universo. A la izquierda está la comparativa de las funciones con dimensión 10. A la derecha está la comparativa de las funciones con dimensión 30.

En las tablas de 1vs1 (Cuadro 1.8 y Cuadro 1.9) podemos ver como se cumple esta pérdida de mejora en dimensión alta. Para dimensión 10 si que vemos como se incrementa el número de victorias frente a cada algoritmo. Pero sin embargo, para dimensión 30 no pasa eso, empeorando en DE y SSA y mejorando en PSO, fruto probablemente de que la búsqueda aleatoria en espacios de dimensionalidad tan grande no son tan efectivos.

Por lo tanto, la mejora hasta la hibridación nos mejoraba siempre. Ahora con la nueva mejora, tenemos un mejor algoritmo para problemas con menos complejidad, como el de la dimensión 10, pero sin embargo, tenemos un algoritmo con peor desempeño en complejidades más altas.x

DIM 10	**DE**	**PSO**	**SSA**
Número de victorias vs	11	27	25
Porcentaje de victorias vs	36.67 %	90 %	83.33 %

Cuadro 1.8: Número y porcentaje de victorias de las Mejoras en 1vs1 con cada uno de los otros algoritmos en Dimensión 10.

DIM 30	**DE**	**PSO**	**SSA**
Número de victorias vs	10	26	21
Porcentaje de victorias vs	33.33 %	86.67 %	70 %

Cuadro 1.9: Número y porcentaje de victorias de las Mejoras en 1vs1 con cada uno de los otros algoritmos en Dimensión 30.

Conclusión, la mejora propuesta provoca una gran mejora en problemas de baja complejidad donde el espacio de búsqueda no es muy grande. Una vez que se empieza a aumentar la complejidad del espacio, esta mejora deja de ser tan útil y se vuelve incluso contraproducente por el gasto de evaluaciones.

Bibliografía

[GRVT] F. Gravedad: https://www.fisicalab.com/apartado/gravedad-orbitas. Recurso online.

[TLAB] TACOLAB, resultados de otros algoritmos: https://tacolab.org/bench. Recurso online.

[SSA] Squirrel Search Algorithm: https://dx.doi.org/10.1016/j.swevo.2018.02.013. Recurso online.

[MHUGR] Asignatura de Metaheurísticas. Recurso de la Universidad de Granada.