Metaheurísticas

Departamento de Ciencias de la Computación e Inteligencia Artificial

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Práctica Alternativa: General Relativity Search Algorithm (GRSA)

Álvaro Rodríguez Gallardo. 77034155W.

alvaro155w@correo.ugr.es

Grupo 2 (17.30-19.30)

Curso 2023-2024

Índice general

Introducción y descripción de GRSA

La relatividad general, una de las teorías más revolucionarias en la física moderna propuesta por Albert Einstein, describe cómo la materia y la energía interactúan con la geometría del espacio-tiempo. En este marco, las partículas se mueven siguiendo trayectorias llamadas **geodésicas**, que son los caminos de menor acción física en un espacio-tiempo curvado. La curvatura del espacio-tiempo, a su vez, es determinada por la distribución de la masa y la energía. Este concepto fundamental del movimiento en un campo gravitatorio proporciona una rica analogía en el diseño de metaheurísticas.

Inspirados en los principios de la relatividad general, los autores proponen en **Beiranvand H, Rokrok E. General Relativity Search Algorithm: A Global Optimization Approach. International Journal of Computational Intelligence and Applications. 2015;14(3); p. 1–29.** el Algoritmo de Búsqueda basado en Relatividad General. En GRSA, se genera inicialmente una población aleatoria de partículas, donde las componentes de posición de estas partículas se representan como un tensor. La posición donde las partículas encuentran la menor acción física (óptimo de la función) se considera mejor posición. Las partículas se mueven hacia esta mejor posición siguiendo las trayectorias dadas por las geodésicas en función de la energía-momento en un espacio curvo.

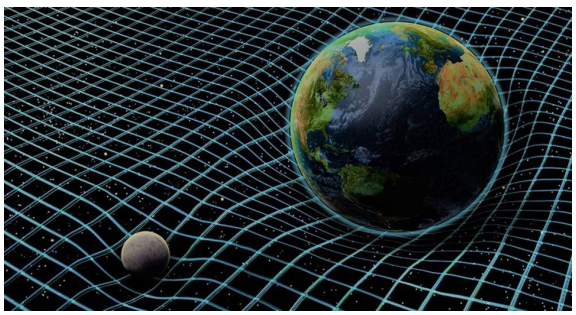

Figura 1.1: Curvatura del espacio-tiempo. Caso particular de órbita de la Luna en torno a la Tierra. Imagen extraída de la siguienteweb

Para comenzar el algoritmo, se debe inicializar un tensor (población) de partículas. Para ello, se usa una distribución uniforme en $[0, 1]$, y se aplica la siguiente fórmula, habiendo prefijado las constantes de valor mínimo y valor máximo por característica (o componente del vector de posición).

$T_{i,j}(1) = T_j^{min} + rand() \cdot (T_j^{max} - T_j^{min})$

Se inicializa entonces la componente de posición j de la partícula i, siendo $rand()$ una variable aleatoria que sigue $\mathcal{U}(0,1)$. Además, se inicializa como población anterior, que se irá actualizando a lo largo del algoritmo, un tensor cuyas partículas están en el origen.

A continuación, se subdivide el tensor en h subtensores, sobre los que se calculará

la velocidad en el movimiento de las partículas. La idea detrás de esto es la fuerza de la interacción gravitacional entre partículas, siendo más fuerte si estas se encuentran más cercanas (por ejemplo, la Luna se ve más influenciada por el campo gravitacional de la Tierra que por el de Neptuno). Ahora, para cada subdivisión se calcula el radio de energía cinética por partícula, que teóricamente tiene la expresión

$$\zeta = \frac{E_k}{mC^2}$$

donde E_k es la energía de la partícula, m su masa y C la velocidad de la luz. No obstante, dado que no se conoce la masa de la partícula, se obtiene con la siguiente aproximación:

$$\zeta_i(t) = \frac{I_{rand} - n}{n-1} GM_1 + \frac{1 - I_{rand}}{n-1} GM_2$$

donde GM_1 y GM_2 son constantes geométricas tales que imponen la satisfacción de la ley de conservación ($GM_1 + GM_2 = 1$), n el número de partículas e I_{rand} un índice aleatorio entre 0 y $n-1$.

Con estos valores se calcula la tranformada de Lorentz ($\gamma_{0,i} = 1 + \zeta_i(t)$) y se transforma en vector ($\gamma_i(t) = \gamma_{0,i}(t) + (1 - \gamma_{0,i}(t))K_g$ para K_g vector de dimensión la cantidad de características cuyas componentes siguen una uniforme $[0, 1]$).

Dados los resultados anteriores, se obtiene la velocidad de acuerdo a la igualdad teórica, $\gamma = \sqrt{\frac{g_{tt}}{g_{tt} + g_{ss}v^2}}$, la cual se implementa como $V_i(t) = K_{V,i}(t)\sqrt{\frac{1}{\gamma_i^2(t)} - 1}$, y finalmente la longitud a recorrer en la iteración t se expresa como

$$\lambda_i(t) = \omega(t) V_i(t)$$

donde ω es un valor de peso que desciende de 0,9 a 0,1 linealmente. La razón de su uso es que a mayor valor de iteración, más cerca se estará del óptimo y menos se querrá mover la partícula. Gráficamente, para 100 iteraciones, se vería como aparece en la figura Fig 1.2.

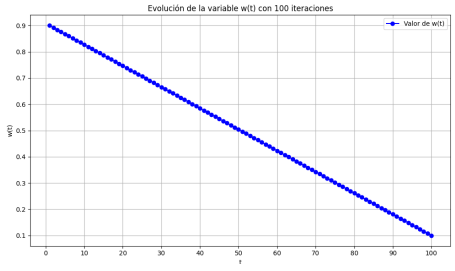

Figura 1.2: Gráfica asociada al peso

Las trayectorias geodésicas en el algoritmo se forma de acuerdo a la geometría del espacio curvo y ofrecen la mínima resistencia en contra del movimiento de las partículas bajo campos gravitatorios. Como la energía o masa está relacionada con el campo gravitatorio, a mayor energía-momento se genera un campo gravitatoio más fuerte y, por tanto, tiene mayor impacto en la creación de las geodésicas. Generalmente, para cada componente de posición de una partícula, se calcula el sentido y dirección de la trayectoria con la siguiente aproximación para la iteración t:

$$\delta_{i,j} = -sgn(\delta_{time,i,j}(t) + K_{f,j}\delta_{space,i,j}(t) + (1 - K_{f,j})\delta_{null,i,j}(t)$$

donde $\delta_{time,i}$ es la geodésica del tiempo de la partícula i (trayectoria de las partículas con masa que no alcanza la velocidad de la luz, se movería .$^{\text{en}}$ el espacio"), $\delta_{space,i}$ es la geodésica del espacio (trayectoria realizada para partículas que alcanzan

la velocidad de la luz, se movería .$^{\text{en}}$ el tiempo") y $\delta_{null,i}$ es la geodésica nula (trayectoria de partículas sin masa, como fotones, y aparece en lugares como los horizontes de sucesos de los agujeros negros). Además, K_f se obtiene aleatoriamente como 0 o 1, forma en la que se incluye tanto a partículas con masa y sin masa (aunque la aleatoriedad puede introducir sesgo).

Finalmente, teniendo el sentido del movimiento y la velocidad de cada partícula (calculada a partir de las transformadas de Lorentz, que son en este algoritmo completamente aleatorias), se calcula la posición que debe moverse la partícula.

Por cada iteración, el tensor de partículas cambia, obteniendo cada una diferente masa y energía respecto a la anterior iteración. Al cumplirse la condición de parada (máximo de iteraciones), se obtiene la partícula con mejor posición (mayor fitness) y sería la solución óptima. Teóricamente, el algoritmo converge a un óptimo global. Sin embargo, dada la limitación de recursos computacionales existentes, las iteraciones son menores a las recomendadas y debido al gran número de suposiciones para poder reducir complejidad, la solución es un óptimo local y la convergencia no sigue una función estrictamente creciente.

A grandes rasgos, la metaheurística busca siempre converger a un óptimo, permitiendo a veces algunos cambios para que la solución empeore. Por ello, se considera que el algoritmo siempre busca mejorar de acuerdo a las posiciones con mejor fitness, por lo que prevalece la explotación (es fácil que caiga en óptimos locales, aunque teóricamente intente aproximar a óptimos globales). La mejora incluida por el alumno añade cierta exploración dada la existencia de soluciones peores que otras, no siendo útiles que permanezcan en dicha posición si otra partícula convergerá antes.

Aspectos generales de GRSA

2.1. Esquema y representación de las soluciones

Se ha buscado representar de manera consistente todos y cada uno de los datos presentes en los conjuntos usados. Por ello, se declaran las siguientes estructuras de datos.

```
struct Atributo {
    string nombre;
    string tipo;
};
```

Con lo anterior, se busca representar un atributo, que consta tanto del nombre como del tipo de datos.

```
struct Muestra {
    vector<double> caracteristicas;
    string clase;
};
```

Representación de una muestra para los algoritmos de aprendizaje automático supervisado en clasificación. Consta tanto del vector de características (numérico en este caso) y la clase asociada a la muestra.

```
struct Dataset {
    vector<Atributo> atributos;
    vector<Muestra> muestras;
};
```

Implementación de un conjunto de muestras de un conjunto de datos, junto con el atributo correspondiente por posición en el vector de características.

Habiendo conseguido representar los datos de forma concisa, se vuelve al objetivo del problema, el cual es conseguir un vector de pesos a partir de una metaheurística para un problema de clasificación 1-NN. Por ello, se representa un vector solución (de pesos) al problema de la siguiente manera.

```
struct Pesos {
    vector<double> valores;
};
```

Dado que la metaheurística implementada es GRSA, de acuerdo a la terminología física, **las soluciones del problema se consideran partículas, y un conjunto de estas en un tensor**. Por ello, la solución del problema, apoyado en las representaciones anteriores, es la siguiente.

```
struct Particula{
    Pesos posicion;
    Pesos mejor_posicion;
    double fitness;
};
```

Es necesario para realizar la mutación propuesta en el artículo almacenar la mejor posición (mayor valor de fitness) a lo largo de la ejecución del algoritmo.

Por último, la representación de un tensor aparece en la siguiente estructura.

```
struct EspacioTiempo{
    vector<Particula> T;
};
```

Este vector de partículas representa la posición de cada una en un espacio de tantas dimensiones como características, aunque de acuerdo a la literatura, representaría el espacio-tiempo sobre el que se disponen las partículas e interaccionan entre ellas.

2.2. Operaciones auxiliares

De acuerdo a las indicaciones para inicializar el tensor, se implementa la siguiente función para crear un tensor de forma aleatoria.

Para calcular las geodésicas, es necesario obtener una población anterior. Dado que no se especifica nada en el artículo, se supone que la pobación anterior a la inicial se encuentra con todas las partículas en el origen (supóngase ha ocurrido una especie de Big Bang).

Además, dada la necesidad de particionar en subtensores por la propiedad del campo gravitatorio, se implementa la siguiente función que hace particiones aleatorias según el valor de h.

Algorithm 1 inicializarTensor

```
1: procedure INICIALIZARTENSOR(EspacioTiempo & tensor, int & n_evals)
2:     distribution ← uniform_real_distribution(0.0, 1.0)
3:     valor, T_max, T_min
4:     for i ← 0 to DIMENSION do
5:         T_max ← Random::get(distribution)
6:         if T_max == 0.0 then
7:             T_max ← 1.0
8:         end if
9:         T_min ← T_max - 1
10:        if T_min <0.0 then
11:            T_min ← 0.0
12:        end if
13:        T_maxs.push_back(T_max)
14:        T_mins.push_back(T_min)
15:    end for
16:    for i ← 0 to TAM_POBLACION do
17:        Particula part
18:        for j ← 0 to DIMENSION do
19:            valor ← T_mins[j] + Random::get(distribution) * (T_maxs[j] -
   T_mins[j])
20:            if valor <0.0 then
21:                valor ← 0.0
22:            end if
23:            if valor >1.0 then
24:                valor ← 1.0
25:            end if
26:            part.posicion.valores.push_back(valor)
27:            part.mejor_posicion.valores.push_back(valor)
28:        end for
29:        part.fitness ← cec17_fitness(part.posicion.valores[0])
30:        n_evals ← n_evals + 1
31:        tensor.T.push_back(part)
32:    end for
33: end procedure
```

Algorithm 2 inicializacionDistinta

```
1: procedure INICIALIZACIONDISTINTA(EspacioTiempo & tensor, int & n_evals)
2:     for i ← 0 to TAM_POBLACION do
3:         Particula part
4:         for j ← 0 to DIMENSION do
5:             valor ← 0.0
6:             part.posicion.valores.push_back(valor)
7:             part.mejor_posicion.valores.push_back(valor)
8:         end for
9:         part.fitness ← cec17_fitness(part.posicion.valores[0])
10:        n_evals ← n_evals + 1
11:        tensor.T.push_back(part)
12:    end for
13: end procedure
```

Algorithm 3 Partición de Subespacios

Require: `TAM_POBLACION` ▷ Número total de elementos en la población
Require: `S` ▷ Número de subespacios a crear
Require: `h` ▷ Tamaño de cada subespacio
Ensure: `subtensores` ▷ Lista de listas que contiene los subespacios

```
indexes ← lista vacía                          ▷ Inicializa la lista de índices
subtensores ← lista vacía                      ▷ Inicializa la lista de subespacios
num_index ← 0                                  ▷ Índice de seguimiento
for i ← 0 to TAM_POBLACION-1 do
    indexes.push_back(i)
end for
mezclarAleatoriamente(indexes)                 ▷ Mezcla la lista de índices
for i ← 0 to S-1 do
    subtensor ← lista vacía                    ▷ Inicializa un nuevo subespacio
    for j ← 0 to h-1 do
        subtensor.push_back(indexes[num_index])
        num_index ← num_index + 1
    end for
    subtensores.push_back(subtensor)
end for
assert (num_index = TAM_POBLACION)   ▷ Verifica que todos los índices se han utilizado
    return subtensores
```

Para programar la función que obtiene las geodésicas no se ha encontrado una librería con la función signo, por lo que se ha implementado con el siguiente pseudocódigo.

Algorithm 4 Función de Signo

Require: `valor` ▷ Número real del cual se quiere determinar el signo
Ensure: `signo` ▷ Retorna 1 si `valor` es positivo, -1 si es negativo, 0 si es cero

```
function SGN(valor)
    if valor > 0 then
        return 1
    else if valor < 0 then
        return -1
    else
        return 0
    end if
end function
```

Las próximas funciones auxiliares se han usado para implementar la mutación propuesta por los autores. Primero, la función que busca las S peores partículas en el espacio-tiempo es

Algorithm 5 Buscar Partículas con Peor Fitness en Todo el Espacio

Require: `tensor` ▷ EspacioTiempo que contiene las partículas
Ensure: `indexes` ▷ Índices de las partículas con peor fitness

```
1: function BUSCARPARTICULASPEORFITNESSTODOESPACIO(tensor)
2:     copia ← tensor.T                          ▷ Copia de las partículas del tensor
                                                 ▷ Ordena la copia en orden creciente de fitness
3:     stable_sort(copia.begin(), copia.end(),
   compararFitnessOrdenarMenorMayorAlternativa)
4:     indexes ← lista vacía                     ▷ Inicializa la lista de índices
       ▷ Encuentra los índices de las primeras S partículas en la copia ordenada
5:     for i ← 0 to S-1 do
6:         indice ← distancia(tensor.T.begin(),
   encontrar(tensor.T.begin(), tensor.T.end(), copia[i]))
7:         indexes.push_back(indice)
8:     end for
       return indexes
9: end function
```

Las dos siguientes funciones se encargan de obtener las S mejores partículas, con la diferencia de que no son las mejores globales, sino que son las mejores de su subtensor. La primera devuelve las partículas y la segunda sus posiciones en el tensor.

Algorithm 6 Obtener Mejor Partícula por Subespacio

Require: `tensor` ▷ EspacioTiempo que contiene las partículas
Require: `subespacios` ▷ Lista de subespacios representada como una lista de listas de índices
Ensure: `mejores` ▷ Lista de las mejores partículas por cada subespacio

```
1: function OBTENERMEJORPARTICULAPORSUBESPACIO(tensor, subespacios)
2:     mejores ← lista vacía                     ▷ Inicializa la lista de mejores partículas
3:     for i ← 0 to subespacios.size()-1 do
4:         subtensor ← devolverSubtensor(tensor, subespacios[i])   ▷ Obtiene el subespacio correspondiente
5:         mejor_particula ← obtenerMejorParticula(subtensor).first   ▷ Obtiene la mejor partícula del subespacio
6:         mejores.push_back(mejor_particula)   ▷ Agrega la mejor partícula a la lista
7:     end for
       return mejores
8: end function
```

Algorithm 7 Obtener Posiciones de la Mejor Partícula por Subespacio

Require: `tensor` ▷ EspacioTiempo que contiene las partículas
Require: `subespacios` ▷ Lista de subespacios representada como una lista de listas de índices
Ensure: `mejores` ▷ Lista de índices de las mejores partículas por cada subespacio

```
function obtenerPosicionesMejorParticulaPorSubespacion(tensor, subespacios)
    mejores ← lista vacía                ▷ Inicializa la lista de mejores posiciones
    for i ← 0 to subespacios.size()-1 do
        subtensor ← devolverSubtensor(tensor, subespacios[i])   ▷ Obtiene el subespacio correspondiente
        mejor_posicion ← obtenerMejorParticula(subtensor).second   ▷ Obtiene la posición de la mejor partícula del subespacio
        mejores.push_back(mejor_posicion)   ▷ Agrega la posición a la lista
    end for
     return mejores
end function
```

Para facilitar algunas implementaciones, aunque por eficiencia se haya optado por usar una lista de índices para representar las partículas, es necesario obtener el subtensor explícitamente.

Algorithm 8 Devolver Subtensor

Require: `tensor` ▷ EspacioTiempo que contiene todas las partículas
Require: `indexes_subtensor` ▷ Lista de índices que identifican las partículas del subtensor
Ensure: `subtensor` ▷ Subtensor con las partículas seleccionadas

```
function devolverSubtensor(tensor, indexes_subtensor)
    subtensor ← nuevo EspacioTiempo      ▷ Inicializa un nuevo subtensor
    for i ← 0 to indexes_subtensor.size()-1 do
        subtensor.T.push_back(tensor.T[indexes_subtensor[i]])   ▷ Agrega la partícula seleccionada al subtensor
    end for
     return subtensor
end function
```

Finalmente, la mejor partícula del tensor se obtiene con la siguiente función.

Algorithm 9 Obtener Mejor Partícula

Require: `tensor` ▷ EspacioTiempo que contiene las partículas
Ensure: `(mejor, index_mejor)` ▷ Par con la mejor partícula y su índice

function OBTENERMEJORPARTICULA(`tensor`)
    `mejor` ← `tensor.T[0]` ▷ Inicializa la mejor partícula con la primera partícula
    `index_mejor` ← 0 ▷ Inicializa el índice de la mejor partícula
    for $i \leftarrow 1$ **to** `tensor.T.size()`-1 **do**
        if `mejor.fitness` < `tensor.T[i].fitness` **then**
            `mejor` ← `tensor.T[i]` ▷ Actualiza la mejor partícula
            `index_mejor` ← i ▷ Actualiza el índice de la mejor partícula
        end if
    end for
     return `(mejor, index_mejor)`
end function

Y la función discreta que obtiene el peso correspondiente para la iteración t es la siguiente:

Algorithm 10 Recta Decreciente

Require: `iteracion` ▷ Número de la iteración actual
Ensure: `resultado` ▷ Valor de la función recta decreciente

function RECTADECRECIENTE(`iteracion`)
    `m` ← $\frac{0{,}9-0{,}1}{1-\texttt{T\_MAX}}$ ▷ Calcula la pendiente
    `n` ← $0{,}9 - \texttt{m}$ ▷ Calcula el término constante
    `resultado` ← $(\texttt{m} \times \texttt{iteracion}) + \texttt{n}$ ▷ Calcula el valor de la función **return** `resultado`
end function

Descripción del algoritmo implementado. Mejoras e hibridaciones

Habiendo descrito todas las estructuras de datos empleadas y las funciones auxiliares con las que se ha buscado modularizar el código para una mejor interpretación, se está en condiciones de exponer en pseudocódigo los principales métodos del algoritmo GRSA. Primero se expondrá el algoritmo diseñado por los autores y después se expondrá el pseudocódigo tanto de la mejora propuesta por el alumno como de la hibridación, concluyendo en la mezcla de la mejora y la hibridación con el objetivo de estudiar su comportamiento conjunto.

3.1. Algoritmo GRSA básico

Se va a suponer que se tiene un tensor de partículas dispuestas a lo largo del espacio-tiempo de dimensión tan alta como número de características tengan las muestras. El primer paso para resolver el algoritmo es el calculo de los radios de energía cinética, determinantes para mover las partículas a zonas de mayor potencial (esto es, mayor fitness).

Algorithm 11 Obtener Radios de Energía Cinética

Require: `subtensor` ▷ Lista de índices de partículas en el subtensor
Require: `index_de_subtensor` ▷ Índice del subtensor actual
Ensure: `radios_e_cinetica` ▷ Lista de radios de energía cinética

```
function obtenerRadiosEnergiaCinetica(subtensor, index_de_subtensor)
    radios_e_cinetica ← lista vacía ▷ Inicializa la lista de radios de energía cinética
    distr ← distribución uniforme de enteros entre 1 y h
    for i ← 0 to h-1 do
        I_rand ← h ×(index_de_subtensor − 1) + Random::get(distr)
        radio ← (1,0×(I_rand−TAM_POBLACION))/(TAM_POBLACION−1) × GM_1 + (1,0×(1−I_rand))/(TAM_POBLACION−1) × GM_2
        radios_e_cinetica.push_back(radio)
    end for
    return radios_e_cinetica
end function
```

A continuación es necesario obtener el vector de transformada de Lorentz para cada partícula, con el que se podrá calcular la velocidad que tomará la misma en la iteración t.

Algorithm 12 Obtener Vectores de Transformada de Lorentz

Require: `radios_e_cinetica` ▷ Lista de radios de energía cinética
Ensure: `vectoresTrLorentz` ▷ Lista de vectores transformados por Lorentz

1: **function** OBTENERVECTORESTRANSFORMADALORENTZ(`radios_e_cinetica`)
2: `vectoresTrLorentz` ← `lista vacía` ▷ Inicializa la lista de vectores transformados por Lorentz
3: `distr` ← `distribución uniforme de reales entre 0.0 y 1.0`
4: `K_g` ← `lista vacía` ▷ Inicializa el vector aleatorio K_g
▷ Inicializa el vector K_g con valores aleatorios
5: **for** $i \leftarrow 0$ **to** `DIMENSION`-1 **do**
6: `K_g.push_back(Random::get(distr))`
7: **end for**
8: **for** $i \leftarrow 0$ **to** `h`-1 **do**
9: `escalarLorentz` ← 1,0 + `radios_e_cinetica`$[i]$ ▷ Calcula el escalar Lorentz
10: `vectorParticula` ← `lista vacía` ▷ Inicializa el vector de la partícula
11: **for** $j \leftarrow 0$ **to** `DIMENSION`-1 **do**
12: `valor` ← `escalarLorentz` + (1,0 − `escalarLorentz`) × `K_g`$[j]$
13: `vectorParticula.push_back(valor)`
14: **end for**
15: `vectoresTrLorentz.push_back(vectorParticula)`
16: **end for**
return `vectoresTrLorentz`
17: **end function**

Finalmente, antes de obtener la velocidad por partícula, se implementa una función para obtener el vector por el que se ha podido transformar la ecuación de la transformada por una más simple y eficiente. Formalmente, si T es el tensor, se obtendría como

$$K_{V,i}(t) = |T^{mejor,s}(t) - T^{rand,s}(t)|$$

donde $T^{mejor,s}(t)$ es la mejor posición hasta el momento de la partícula i en el subtensor $s \in \{1, ..., S\}$ y $T^{rand,s}$ es la posición de una partícula aleatoria en el subtensor s.

Algorithm 13 Obtener K_V Dada una Partícula

Require: `mejorParticula` ▷ Partícula de referencia con mejor posición
Require: `particulaAleatoria` ▷ Otra partícula para comparación
Ensure: `K_V_i` ▷ Lista de diferencias absolutas entre las posiciones de las dos partículas

```
function obtenerK_V_dadaUnaParticula(mejorParticula, particulaAleatoria)
    assert (mejorParticula.posicion.valores.size() = particulaAleatoria.posicion.valores.size())  ▷ Asegura que ambas partículas tienen el mismo tamaño de posición
    K_V_i ← lista vacía  ▷ Inicializa la lista K_Vi
    for i ← 0 to mejorParticula.posicion.valores.size()-1 do
        diferencia ← fabs(mejorParticula.posicion.valores[i] - particulaAleatoria.posicion.valores[i])
        K_V_i.push_back(diferencia)
    end for
     return K_V_i
end function
```

Y la velocidad se calcula como sigue.

Algorithm 14 obtenerVelocidades

```
function obtenerVelocidades(tensor, subtensor, transformadasLorentz)
    velocidades ← lista vacía  ▷ Inicializa la lista de velocidades
    distribucion_subtensor ← distribución uniforme de reales entre 0 y subtensor.size()-1
                          ▷ Inicializa subtensorP con las partículas del subtensor
    subtensorP ← nuevo EspacioTiempo
    for i ← 0 to subtensor.size() - 1 do
        subtensorP.T.push_back(tensor.T[subtensor[i]])
    end for
    for i ← 0 to subtensorP.T.size() - 1 do
        V_i ← lista vacía  ▷ Inicializa el vector de velocidades para la partícula i
        transformada_i ← transformadasLorentz[i]
        mejorParticula ← obtenerMejorParticula(subtensorP).first
        particulaAleatoria ← subtensorP.T[Random::get(distribucion_subtensor)]
        K_V_i ← obtenerK_V_dadaUnaParticula(mejorParticula, particulaAleatoria)
        assert (transformada_i.size() = K_V_i.size())  ▷ Asegura que los tamaños coinciden
        for j ← 0 to transformada_i.size() - 1 do
            V_i.push_back(K_V_i[j] × sqrt((1 / (transformada_i[j] × transformada_i[j])) - 1))
        end for
        velocidades.push_back(V_i)
    end for
    return velocidades
end function
```

Y la distancia a recorrer por la partícula i se obtiene de la siguiente manera.

Algorithm 15 obtenerDistanciasARecorrer

```
1: function obtenerDistanciasARecorrer(velocidades, iteracion)
2:     peso ← rectaDecreciente(iteracion)  ▷ Calcula el peso en función de la iteración
3:     lambdas ← lista vacía  ▷ Inicializa la lista de distancias a recorrer
4:     for i ← 0 to velocidades.size() - 1 do
5:         lambdas_particle_i ← lista vacía  ▷ Inicializa la lista de distancias para la partícula i
6:         for j ← 0 to velocidades[0].size() - 1 do
7:             lambdas_particle_i.push_back(peso × velocidades[i][j])
8:         end for
9:         lambdas.push_back(lambdas_particle_i)
10:    end for
11:    return lambdas
12: end function
```

Por otro lado, las geodésicas mencionadas en la introducción y resumen, que determinan las trayectorias en las que se moverá una partícula, se obtiene implementando el siguiente pseudocódigo.

Algorithm 16 obtenerDeltasGeodesicas

```
1: function obtenerDeltasGeodesicas(tensor, tensor_anterior)
2:     assert (tensor.T.size() = tensor_anterior.T.size())  ▷ Asegura que ambos tensores tengan el mismo tamaño
3:     mejor_global ← obtenerMejorParticula(tensor).first
4:     deltas_particulas ← lista vacía  ▷ Inicializa la lista de deltas para todas las partículas
5:     distrib_0_1 ← distribución uniforme de enteros entre 0 y 1
6:     for i ← 0 to tensor.T.size() - 1 do
7:         T_i ← tensor.T[i]
8:         T_i_anterior ← tensor_anterior.T[i]
9:         mejor_posicion_hasta_ahora_i ← T_i.mejor_posicion
10:        delta_i ← lista vacía ▷ Inicializa la lista de delta para la partícula i
11:        for j ← 0 to T_i.posicion.valores.size() - 1 do
12:            K_f_j ← Random::get(distrib_0_1)  ▷ Valor aleatorio en {0, 1}
13:            aux_time_geo ← sgn(T_i.posicion.valores[j] − T_i_anterior.posicion.valores[j])
14:            aux_space_geo ← sgn(T_i.posicion.valores[j] − mejor_global.posicion.valores[j])
15:            aux_null_geo ← sgn(T_i.posicion.valores[j] − mejor_posicion_hasta_ahora_i.valores[j])
16:            delta_i.push_back( − sgn(aux_time_geo + K_f_j × aux_space_geo + (1 - K_f_j) × aux_null_geo))
17:        end for
18:        deltas_particulas.push_back(delta_i)
19:    end for
20:    return deltas_particulas
21: end function
```

Habiendo conseguido los principales elementos para mover a las partículas, se obtiene la siguiente población aplicando la siguiente expresión por partícula, i.

$T_i(t+1) = T_i(t) + \lambda_i(t)\delta_i(t)$

que ha sido implementado a continuación.

Algorithm 17 obtenerSiguientePoblacion

```
1: function OBTENERSIGUIENTEPOBLACION(tensor, distancias_recorrer, deltas_geodesicas, n_evals)
2:     nuevo_tensor ← tensor
3:     for i ← 0 to tensor.T.size() - 1 and n_evals < MAX_EVALUACIONES do
4:         for j ← 0 to DIMENSION - 1 do
5:             nuevo_tensor.T[i].posicion.valores[j] ← tensor.T[i].posicion.valores[j] + distancias_recorrer[i][j] × deltas_geodesicas[i][j]
6:             if nuevo_tensor.T[i].posicion.valores[j] < 0.0 then
7:                 nuevo_tensor.T[i].posicion.valores[j] ← 0.0
8:             end if
9:             if nuevo_tensor.T[i].posicion.valores[j] > 1.0 then
10:                nuevo_tensor.T[i].posicion.valores[j] ← 1.0
11:            end if
12:        end for
13:        nuevo_tensor.T[i].fitness ← cec17_fitness(&nuevo_tensor.T[i].posicion.
14:        valores[0])
15:        n_evals ← n_evals + 1
16:        if nuevo_tensor.T[i].fitness > tensor.T[i].fitness then
17:            nuevo_tensor.T[i].mejor_posicion ← nuevo_tensor.T[i].posicion
18:        end if
19:    end for
20:    return nuevo_tensor
21: end function
```

Por último, aunque no habría sido necesario, se añade la función mediante la cual los autores proponen una mutación a las peores partículas (fitness más bajo) por subespacio, intentando que avancen más hacia puntos donde el potencial es mayor.

Algorithm 18 mutarPeorParticulaCadaSubespacio

```
1: procedure MUTARPEORPARTICULACADASUBESPACIO(tensor, subespacios, n_evals)
2:     distr ← distribución uniforme de enteros entre 0 y 1
3:     alphas_1 ← lista vacía
4:     alphas_2 ← lista vacía
                                                          ▷ Inicializar alpha
5:     for i ← 0 to DIMENSION - 1 do
6:         alphas_1.push_back(Random::get(distr))
7:         alphas_2.push_back(1 - alphas_1[i])
8:     end for
                                                  ▷ Estructuras de datos auxiliares
9:     R ← par de listas vacías (vector de Particula, vector de enteros)
10:    g ← obtenerMejorParticulaPorSubespacio(tensor, subespacios)
11:    R.second ← buscarParticulasPeorFitnessTodoEspacio(tensor)
12:    for i ← 0 to R.second.size() - 1 do
13:        R.first.push_back(tensor.T[R.second[i]])
14:    end for
15:    for i ← 0 to S - 1 and n_evals < MAX_EVALUACIONES do
16:        for j ← 0 to DIMENSION - 1 do
17:            tensor.T[R.second[i]].posicion.valores[j] ← alphas_1[j] × R.first[i].posicion.valores[j] + alphas_2[j] × g[i].posicion.valores[j]
18:        end for
19:        nuevo_fitness ← cec17_fitness(&tensor.T[R.second[i]].posicion.valores[0])
20:        n_evals ← n_evals + 1
21:        if nuevo_fitness > tensor.T[R.second[i]].fitness then
22:            tensor.T[R.second[i]].mejor_posicion ← tensor.T[R.second[i]].posicion
23:        end if
24:        tensor.T[R.second[i]].fitness ← nuevo_fitness
25:    end for
26: end procedure
```

Se concluye entonces el apartado mostrando el pseudocódigo que implementa GRSA usando las funciones anteriores, habiendo definido un número máximo de iteraciones.

Algorithm 19 GRSA

```
1: procedure GRSA(dim, num_part)
2:     T_anterior, tensor ← nuevos EspacioTiempo
3:     t ← 1
4:     subespacios ← lista vacía
5:     n_evals ← 0
6:     inicializarParametros(dim)
7:     inicializacionDistinta(T_anterior, n_evals)
8:     inicializarTensor(tensor, n_evals)
9:     subespacios ← particionarSubespacios()
10:    while t ≤ T_MAX and n_evals ≤ MAX_EVALUACIONES do
11:        velocs_iteracion ← lista vacía
12:        for i ← 0 to S - 1 do
13:            radios_energia_cinetica_sub_s ← obtenerRadiosEnergiaCinetica(subespacios[i], i + 1)
14:            transformadas_Lorentz ← obtenerVectoresTransformadaLorentz(radios_
15:            energia_cinetica_sub_s)
16:            velocidades_sub_s ← obtenerVelocidades(tensor, subespacios[i], transformadas_Lorentz)
17:            for each veloc in velocidades_sub_s do
18:                velocs_iteracion.push_back(veloc)
19:            end for
20:        end for
21:        step_lengths ← obtenerDistanciasARecorrer(velocs_iteracion, t)
22:        deltas ← obtenerDeltasGeodesicas(tensor, T_anterior)
23:        T_anterior ← tensor
24:        if n_evals < MAX_EVALUACIONES then
25:            tensor ← obtenerSiguientePoblacion(tensor, step_lengths, deltas, n_evals)
26:            if n_evals < MAX_EVALUACIONES then
27:                mutarPeorParticulaCadaSubespacio(tensor, subespacios, n_evals)
28:            end if
29:        end if
30:        escribirFitnessParaConvergencia(num_part, obtenerMejorParticula(tensor).first.fitness, t, "GRSA", çonvergencia/GRSA/")
31:        t ← t + 1
32:    end while
33:    mejorParticula ← obtenerMejorParticula(tensor).first
34:    return make_pair(mejorParticula.posicion, mejorParticula.fitness)
35: end procedure
```

3.2. Algoritmo GRSA mejorado

La idea principal detrás de la mejora propuesta es obtener más diversidad de soluciones. La hipótesis principal es "si tengo una solución con cierto fitness, dada la naturaleza de GRSA de primar explotación de soluciones, va a converger al mismo óptimo que otras soluciones con menor fitness cerca de él".

Considérese la partícula k en el espacio-tiempo con un fitness f_k y posición x_k. Sea $B(x_k, \epsilon) = \{x \in \mathbb{DIMENSION} : \|x_k x - x\| < \epsilon\}$ la bola abierta de centro x_k y radio ϵ. A grandes rasgos, se propone que todas las partículas con menor fitness que k dentro de $B(x_k, \epsilon)$ sean reubicadas en el espacio de soluciones. El objetivo es movernos más en el espacio de soluciones y aumentar la probabilidad de caer en otros óptimos locales, mejores que el óptimo local en el que iba a caer, en el cual ya iba a converger la partícula k.

Hay dos elementos importantes a elegir: la norma y ϵ. Para la práctica se opta por $\epsilon = \text{máx}(0{,}05, f_k - \frac{T_i}{2})$ de tal manera que se controle un radio no excesivamente grande (por ello se resta el valor máximo posible para las componentes) ni excesivamente pequeño, o al menos no menor a 0,05 (fitness es un valor real, por lo que debe ser normalizado a $[0, 1]$ para esta aplicar esta mejora). La norma utilizada, por eficiencia, es la norma l_1 (por lo que la bola es, en realidad, un hipercubo abierto).

Tras elegir las partículas a reubicar, se asignan las nuevas posiciones aleatoriamente.

Algorithm 20 mejoraPropuesta

```
1: procedure MEJORAPROPUESTA(tensor, n_evals)
2:     max_radio ← 0.0
                                                    ▷ Calcular el radio máximo
3:     for each particula in tensor.T do
4:         if particula.fitness > max_radio then
5:             max_radio ← particula.fitness
6:         end if
7:     end for
                                    ▷ Lista para los índices de partículas a eliminar
8:     indices_a_eliminar ← lista vacía
9:     for i ← 0 to tensor.T.size() - 1 do
10:        particula_referencia ← tensor.T[i]
11:        centro_bola ← particula_referencia.posicion.valores
12:        radio_bola ← máx(0,05, particula_referencia.fitness/max_radio-
    T_maxs[i]/2,0)
13:        for j ← 0 to tensor.T.size() - 1 do
14:            if i ≠ j and particula_referencia.fitness > tensor.T[j].fitness
    then
15:                pos_j ← tensor.T[j].posicion.valores
16:                dentro_de_bola ← true
17:                for k ← 0 to DIMENSION - 1 and dentro_de_bola do
18:                    distancia ← (centro_bola[k] − pos_j[k])
19:                    if distancia > radio_bola then
20:                        dentro_de_bola ← false
21:                    end if
22:                end for
23:                if dentro_de_bola then
24:                    indices_a_eliminar.push_back(j)
25:                end if
26:            end if
27:        end for
28:    end for
                                    ▷ Rellenar las partículas faltantes
29:    for each idx in indices_a_eliminar do
30:        if n_evals < MAX_EVALUACIONES then
31:            nueva_particula ← nueva Particula
32:            for p ← 0 to DIMENSION - 1 do
33:                distribution ← distribución uniforme de reales entre
    T_mins[p] y T_maxs[p]
34:                nueva_particula.posicion.valores.push_back(Random::get(distribution))
35:                nueva_particula.mejor_posicion.
36:                valores.push_back(nueva_particula.posicion.valores[p])
37:            end for
38:            nueva_particula.fitness ← cec17_fitness(&nueva_particula.posicion.
39:            valores[0])
40:            n_evals ← n_evals + 1
41:            tensor.T[idx] ← nueva_particula
42:        end if
43:    end for
44: end procedure
```

Y el esquema de GRSA que implementa la mejora propuesta es el siguiente.

Algorithm 21 GRSA_Mej

```
1: procedure GRSA_Mej(dim, num_part)
2:     T_anterior, tensor ← nuevos EspacioTiempo
3:     t ← 1
4:     subespacios ← lista vacía
5:     n_evals ← 0
6:     inicializarParametros(dim)
7:     inicializacionDistinta(T_anterior, n_evals)
8:     inicializarTensor(tensor, n_evals)
9:     subespacios ← particionarSubespacios()
10:    while t ≤ T_MAX and n_evals ≤ MAX_EVALUACIONES do
11:        velocs_iteracion ← lista vacía
12:        for i ← 0 to S - 1 do
13:            radios_energia_cinetica_sub_s ← obtenerRadiosEnergiaCinetica(subespacios[i], i + 1)
14:            transformadas_Lorentz ← obtenerVectoresTransformadaLorentz(radios_
15:            energia_cinetica_sub_s)
16:            velocidades_sub_s ← obtenerVelocidades(tensor, subespacios[i], transformadas_Lorentz)
17:            for each veloc in velocidades_sub_s do
18:                velocs_iteracion.push_back(veloc)
19:            end for
20:        end for
21:        step_lengths ← obtenerDistanciasARecorrer(velocs_iteracion, t)
22:        deltas ← obtenerDeltasGeodesicas(tensor, T_anterior)
23:        T_anterior ← tensor
24:        if n_evals < MAX_EVALUACIONES then
25:            tensor ← obtenerSiguientePoblacion(tensor, step_lengths, deltas, n_evals)
                                                    ▷ Mejora propuesta para la diversidad
26:            if n_evals < MAX_EVALUACIONES then
27:                mejoraPropuesta(tensor, n_evals)
28:            end if
29:            if n_evals < MAX_EVALUACIONES then
30:                mutarPeorParticulaCadaSubespacio(tensor, subespacios, n_evals)
31:            end if
32:        end if
33:        escribirFitnessParaConvergencia(num_part, obtenerMejorParticula(tensor).first.fitness, t, "GRSA-Mej", çonvergencia/GRSA_Mej/")
34:        t ← t + 1
35:    end while
36:    mejorParticula ← obtenerMejorParticula(tensor).first
37:    return make_pair(mejorParticula.posicion, mejorParticula.fitness)
38: end procedure
```

3.3. Algoritmo híbrido. Búsqueda local

Se decide hibridar GRSA con el algoritmo de búsqueda local. Recuérdese que GRSA primaba la explotación de soluciones, intentando llevar las partículas a centros donde el potencial es mayor de acuerdo a los campos gravitatorios generados por cada una. Por ello, dado que búsqueda local permite explorar un vecindario de soluciones, se propone una hibridación que aplica búsqueda local cada 10 iteraciones, buscando con ello que se prime también la exploración en el espacio de soluciones.

Algorithm 22 GRSA_BL

```
1: procedure GRSA_BL(dim, num_part)
2:    T_anterior, tensor ← nuevos EspacioTiempo
3:    t ← 1, n_evals ← 0, distance ← T_MAX / 10
4:    inicializarParametros(dim), inicializacionDistinta(T_anterior,
   n_evals), inicializarTensor(tensor, n_evals)
5:    subespacios ← particionarSubespacios()
6:    while t ≤ T_MAX and n_evals ≤ MAX_EVALUACIONES do
7:       velocs_iteracion ← lista vacía
8:       for i ← 0 to S - 1 do
9:          velocs_iteracion.insertar(velocs_iteracion.end(),
   obtenerVelocidades(tensor, subespacios[i],
   obtenerVectoresTransformadaLorentz(obtenerRadiosEnergiaCinetica(subespacios[i],
   i + 1))).begin(), obtenerVelocidades(tensor, subespacios[i],
   obtenerVectoresTransformadaLorentz(obtenerRadiosEnergiaCinetica(subespacios[i],
   i + 1))).end())
10:      end for
11:      step_lengths ← obtenerDistanciasARecorrer(velocs_iteracion,
   t), deltas ← obtenerDeltasGeodesicas(tensor, T_anterior)
12:      T_anterior ← tensor
13:      if n_evals < MAX_EVALUACIONES then
14:         tensor ← obtenerSiguientePoblacion(tensor, step_lengths,
   deltas, n_evals)
15:         if n_evals < MAX_EVALUACIONES then
16:            mutarPeorParticulaCadaSubespacio(tensor, subespacios,
   n_evals)
17:         end if
18:      end if
19:      if t % distance = 0 and n_evals < MAX_EVALUACIONES then
20:         for each idx in obtenerPosicionesMejorParticulaPorSubespacion(tensor,
   subespacios) do
21:            tras_BL ← BL_dada_solucion_inicial(tensor.T[idx],
   n_evals)
22:            tensor.T[idx].posicion ← tras_BL.first
23:            if tras_BL.second > tensor.T[idx].fitness then
24:               tensor.T[idx].mejor_posicion ← tras_BL.first
25:            end if
26:            tensor.T[idx].fitness ← tras_BL.second
27:         end for
28:      end if
29:      escribirFitnessParaConvergencia(num_part,
   obtenerMejorParticula(tensor).first.fitness, t, "GRSA-BL",
   çonvergencia/GRSA_BL/")
30:      t ← t + 1
31:   end while
32:   return make_pair(obtenerMejorParticula(tensor).first.posicion,
   obtenerMejorParticula(tensor).first.fitness)
33: end procedure
```

3.4. Unificación de mejoras

Para comprobar si la mejora propuesta supone también una mejora para la hibridación con búsqueda local, se implementa el siguiente pseudocódigo.

Algorithm 23 GRSA_Todo

```
1: procedure GRSA_Todo(dim, num_part)
2:     T_anterior, tensor ← nuevos EspacioTiempo
3:     t ← 1, subespacios ← lista vacía, n_evals ← 0
4:     distance ← T_MAX / 10
5:     inicializarParametros(dim)
6:     inicializacionDistinta(T_anterior, n_evals)
7:     inicializarTensor(tensor, n_evals)
8:     subespacios ← particionarSubespacios()
9:     while t ≤ T_MAX and n_evals < MAX_EVALUACIONES do
10:        velocs_iteracion ← lista vacía
11:        for i ← 0 to S - 1 do
12:            velocidades_sub_s ← obtenerVelocidades(tensor,
    subespacios[i], obtenerVectoresTransformadaLorentz(obtenerRadiosEnergiaCinetica(su
    i + 1)))
13:            velocs_iteracion.insertar(velocs_iteracion.end(),
    velocidades_sub_s.begin(), velocidades_sub_s.end())
14:        end for
15:        step_lengths ← obtenerDistanciasARecorrer(velocs_iteracion,
    t)
16:        deltas ← obtenerDeltasGeodesicas(tensor, T_anterior)
17:        T_anterior ← tensor
18:        if n_evals < MAX_EVALUACIONES then
19:            tensor ← obtenerSiguientePoblacion(tensor, step_lengths,
    deltas, n_evals)
20:            if n_evals < MAX_EVALUACIONES then
21:                mejoraPropuesta(tensor, n_evals)
22:                mutarPeorParticulaCadaSubespacio(tensor, subespacios,
    n_evals)
23:            end if
24:        end if
25:        if t % distance = 0 and n_evals < MAX_EVALUACIONES then
26:            for each idx in obtenerPosicionesMejorParticulaPorSubespacion(tensor,
    subespacios) do
27:                tras_BL ← BL_dada_solucion_inicial(tensor.T[idx],
    n_evals)
28:                tensor.T[idx].posicion ← tras_BL.first
29:                if tras_BL.second > tensor.T[idx].fitness then
30:                    tensor.T[idx].mejor_posicion ← tras_BL.first
31:                end if
32:                tensor.T[idx].fitness ← tras_BL.second
33:            end for
34:        end if
35:        escribirFitnessParaConvergencia(num_part,
    obtenerMejorParticula(tensor).first.fitness, t, "GRSA-Todo",
    çonvergencia/GRSA_Todo/")
36:        t ← t + 1
37:    end while
38:    return make_pair(obtenerMejorParticula(tensor).first.posicion,
    obtenerMejorParticula(tensor).first.fitness)
39: end procedure
```

Procedimiento para el desarrollo de la práctica

En primer lugar, la implementación de la práctica se ha realizado en C++, con el apoyo de librerías como *vector*,*utility*,*cmath*,..., además de usar *random.hpp*, sin hacer uso de ningún framework.

Por otro lado, se ha hecho uso de las siguientes carpetas y archivos (más adelante se mencionan y explican los archivos desarrollados para implementar la práctica):

- *BIN*: Se almacena el ejecutable y la carpet *BIN/DATA*, en la que se almacenan los archivos con los datos usados en la práctica.
- *FUENTES*: Todo el código desarrollado se almacena en esta carpeta:
  - *FUENTES/INCLUDE*: Se almacenan los archivos .h y .hpp con la declaración de las funciones y constantes a usar. Además, el archivo random.hpp también aparece aquí.
  - *FUENTES/SRC*: Aquí aparecen los archivos donde se implementan las funciones declaradas en los archivos .h y .hpp.
  - *FUENTES/CMakeLists.txt*: Archivo con las órdenes correspondientes para la creación del archivo Makefile, con el que se genera el programa. Para reducir el tiempo de ejecución, se usa el parámetro -O3 para optimizar el código compilado.

Además, los archivos realizados por el alumno para la práctica son:

- *aux.h/aux.cpp*: En estos archivos se declaran e implementan funciones para el preprocesamiento de datos (lectura y normalización), el cálculo de distancias, la declaración de constantes que se consideran generales, la declaración de las estructuras de datos y algunas funciones de depuración.
- *practica1.hpp/practica1.cpp*: Declaración e implementación de los algoritmos de la práctica. Aquí está la implementación del clasificador 1-NN, los algoritmos de búsqueda lineal y Greedy RELIEF y la función para mostrar los resultados.
- *practica2.hpp/practica2.cpp*: Declaración e implementación de los algoritmos de la práctica. Está la implementación de los algoritmos geneéticos generacionales, estaccionarios y meméticos, además de funciones auxiliares y una búsqueda local revisada para los algoritmos meméticos.
- *practica3.hpp/practica3.cpp*: Declaración e implementación de los algoritmos de la práctica 3: BMB,ILS,ES e ILS-ES.

- *practicaAlternativa.hpp/practicaAlternativa.cpp*: Declaración e implementación de los algoritmos necesarios para GRSA.
- *main.cpp*: Establece la semilla para trabajar con números aleatorios y se llama a la función para las soluciones indicando el algoritmo a ejecutar.

Los pasos a seguir para la creación y ejecución del programa de la **práctica alternativa** son los siguientes:

- **Paso 1**: Abrir la terminal, o ir, a la carpeta *software/FUENTES*,
- **Paso 2**: Ejecutar el comando çmake .".
- **Paso 3**: Ejecutar el comando "make". Esto creará el ejecutable en la carpeta *software/BIN*.
- **Paso 4**: Ir a la carpeta *software/BIN*.
- **Paso 5**: Ejecutar el comando "./practicaAlternativa <semilla>".

Experimentación y análisis de resultados

Los resultados de los algoritmos expuestos en la memoria se han comparado con todos los algoritmos expuestos en CEC2017. Las dimensiones usadas en los problemas son 10, 30 y 50, habiendo evaluado 30 funciones, de menor a mayor complejidad.

La métrica empleada para las comparaciones en Tacolab es comparación por ranking, dada su fácil interpretabilidad.

5.1. Comparación entre algoritmos

- Dimensión 10.
  - GRSA.

	1	2	3	5	10	20	30	40	50	60	70	80	90	100
AEO	4.033333	4.966667	6.200000	8.233333	10.400000	11.266667	11.666667	11.900000	12.266667	12.333333	12.333333	12.433333	12.433333	12.466667
DE	13.333333	13.333333	13.233333	12.100000	8.366667	7.933333	7.916667	7.766667	7.566667	7.633333	7.483333	7.383333	7.316667	7.116667
DYYPO	4.133333	4.400000	5.133333	6.266667	7.633333	7.766667	7.600000	7.533333	7.483333	7.500000	7.566667	7.533333	7.633333	7.700000
EBOwithCMAR	9.733333	9.833333	9.100000	7.800000	5.633333	4.400000	3.783333	3.450000	3.200000	3.333333	3.233333	3.100000	3.300000	3.450000
ELSHADE_SPACMA	11.500000	11.366667	10.766667	8.966667	7.000000	5.333333	4.100000	3.700000	3.983333	4.233333	4.316667	4.450000	4.550000	4.683333
GRSA	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000
GSKA	6.433333	5.566667	5.166667	5.133333	5.533333	6.700000	7.333333	7.633333	7.633333	7.833333	7.683333	7.483333	7.533333	7.433333
MM_OED	7.066667	5.900000	5.433333	3.800000	3.900000	3.416667	3.566667	3.450000	3.450000	3.116667	3.283333	3.516667	3.516667	3.616667
MOS	2.033333	3.300000	4.433333	5.433333	5.900000	6.350000	7.366667	7.816667	8.250000	8.366667	8.500000	8.616667	8.633333	8.750000
PPSO	9.800000	7.133333	6.166667	5.433333	6.000000	6.500000	7.133333	7.366667	7.300000	7.400000	7.500000	7.533333	7.450000	7.283333
PSO	11.533333	12.033333	12.233333	12.500000	12.633333	12.566667	12.300000	12.100000	12.066667	11.766667	11.600000	11.600000	11.566667	11.500000
RB-IPOP-CMA-ES	3.533333	3.833333	4.300000	5.266667	6.100000	5.866667	5.633333	5.966667	6.000000	5.750000	6.000000	6.116667	6.100000	6.083333
SSA	4.833333	7.300000	8.400000	9.766667	10.966667	11.500000	11.633333	11.833333	11.966667	11.900000	12.000000	11.966667	12.000000	12.066667
TLBO-FL	6.566667	6.500000	5.966667	6.300000	7.500000	8.100000	8.533333	8.766667	8.850000	8.966667	9.066667	9.066667	8.966667	8.833333
jSO	10.466667	9.533333	8.466667	8.000000	7.433333	7.300000	6.433333	5.716667	4.983333	4.866667	4.433333	4.200000	4.000000	4.016667

Cuadro 5.1: Dimensión 10 en GRSA

  - GRSA-BL.

	1	2	3	5	10	20	30	40	50	60	70	80	90	100
AEO	4.033333	4.966667	6.200000	8.233333	10.400000	11.266667	11.666667	11.900000	12.266667	12.333333	12.333333	12.433333	12.433333	12.466667
DE	13.333333	13.333333	13.233333	12.100000	8.366667	7.933333	7.916667	7.766667	7.566667	7.633333	7.483333	7.383333	7.316667	7.116667
DYYPO	4.133333	4.400000	5.133333	6.266667	7.633333	7.766667	7.600000	7.533333	7.483333	7.500000	7.566667	7.533333	7.633333	7.700000
EBOwithCMAR	9.733333	9.833333	9.100000	7.800000	5.633333	4.400000	3.783333	3.450000	3.200000	3.333333	3.233333	3.100000	3.300000	3.450000
ELSHADE_SPACMA	11.500000	11.366667	10.766667	8.966667	7.000000	5.333333	4.100000	3.700000	3.983333	4.233333	4.316667	4.450000	4.550000	4.683333
GRSA-BL	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000
GSKA	6.433333	5.566667	5.166667	5.133333	5.533333	6.700000	7.333333	7.633333	7.633333	7.833333	7.683333	7.483333	7.533333	7.433333
MM_OED	7.066667	5.900000	5.433333	3.800000	3.900000	3.416667	3.566667	3.450000	3.450000	3.116667	3.283333	3.516667	3.516667	3.616667
MOS	2.033333	3.300000	4.433333	5.433333	5.900000	6.350000	7.366667	7.816667	8.250000	8.366667	8.500000	8.616667	8.633333	8.750000
PPSO	9.800000	7.133333	6.166667	5.433333	6.000000	6.500000	7.133333	7.366667	7.300000	7.400000	7.500000	7.533333	7.450000	7.283333
PSO	11.533333	12.033333	12.233333	12.500000	12.633333	12.566667	12.300000	12.100000	12.066667	11.766667	11.600000	11.600000	11.566667	11.500000
RB-IPOP-CMA-ES	3.533333	3.833333	4.300000	5.266667	6.100000	5.866667	5.633333	5.966667	6.000000	5.750000	6.000000	6.116667	6.100000	6.083333
SSA	4.833333	7.300000	8.400000	9.766667	10.966667	11.500000	11.633333	11.833333	11.966667	11.900000	12.000000	11.966667	12.000000	12.066667
TLBO-FL	6.566667	6.500000	5.966667	6.300000	7.500000	8.100000	8.533333	8.766667	8.850000	8.966667	9.066667	9.066667	8.966667	8.833333
jSO	10.466667	9.533333	8.466667	8.000000	7.433333	7.300000	6.433333	5.716667	4.983333	4.866667	4.433333	4.200000	4.000000	4.016667

Cuadro 5.2: Dimensión 10 en GRSA-BL

  - GRSA-Mej.

	1	2	3	5	10	20	30	40	50	60	70	80	90	100
AEO	4.033333	4.966667	6.200000	8.233333	10.400000	11.266667	11.666667	11.900000	12.266667	12.333333	12.333333	12.433333	12.433333	12.466667
DE	13.333333	13.333333	13.233333	12.100000	8.366667	7.933333	7.916667	7.766667	7.566667	7.633333	7.483333	7.383333	7.316667	7.116667
DYYPO	4.133333	4.400000	5.133333	6.266667	7.633333	7.766667	7.600000	7.533333	7.483333	7.500000	7.566667	7.533333	7.633333	7.700000
EBOwithCMAR	9.733333	9.833333	9.100000	7.800000	5.633333	4.400000	3.783333	3.450000	3.200000	3.333333	3.233333	3.100000	3.300000	3.450000
ELSHADE_SPACMA	11.500000	11.366667	10.766667	8.966667	7.000000	5.333333	4.100000	3.700000	3.983333	4.233333	4.316667	4.450000	4.550000	4.683333
GRSA-Mej	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000
GSKA	6.433333	5.566667	5.166667	5.133333	5.533333	6.700000	7.333333	7.633333	7.633333	7.833333	7.683333	7.483333	7.533333	7.433333
MM_OED	7.066667	5.900000	5.433333	3.800000	3.900000	3.416667	3.566667	3.450000	3.450000	3.116667	3.283333	3.516667	3.516667	3.616667
MOS	2.033333	3.300000	4.433333	5.433333	5.900000	6.350000	7.366667	7.816667	8.250000	8.366667	8.500000	8.616667	8.633333	8.750000
PPSO	9.800000	7.133333	6.166667	5.433333	6.000000	6.500000	7.133333	7.366667	7.300000	7.400000	7.500000	7.533333	7.450000	7.283333
PSO	11.533333	12.033333	12.233333	12.500000	12.633333	12.566667	12.300000	12.100000	12.066667	11.766667	11.600000	11.600000	11.566667	11.500000
RB-IPOP-CMA-ES	3.533333	3.833333	4.300000	5.266667	6.100000	5.866667	5.633333	5.966667	6.000000	5.750000	6.000000	6.116667	6.100000	6.083333
SSA	4.833333	7.300000	8.400000	9.766667	10.966667	11.500000	11.633333	11.833333	11.966667	11.900000	12.000000	11.966667	12.000000	12.066667
TLBO-FL	6.566667	6.500000	5.966667	6.300000	7.500000	8.100000	8.533333	8.766667	8.850000	8.966667	9.066667	9.066667	8.966667	8.833333
jSO	10.466667	9.533333	8.466667	8.000000	7.433333	7.300000	6.433333	5.716667	4.983333	4.866667	4.433333	4.200000	4.000000	4.016667

Cuadro 5.3: Dimensión 10 en GRSA-Mej

- GRSA-Todo.

	1	2	3	5	10	20	30	40	50	60	70	80	90	100
AEO	4.033333	4.966667	6.200000	8.233333	10.400000	11.266667	11.666667	11.900000	12.266667	12.333333	12.333333	12.433333	12.433333	12.466667
DE	13.333333	13.333333	13.233333	12.100000	8.366667	7.933333	7.916667	7.766667	7.566667	7.633333	7.483333	7.383333	7.316667	7.116667
DYYPO	4.133333	4.400000	5.133333	6.266667	7.633333	7.766667	7.600000	7.533333	7.483333	7.500000	7.566667	7.533333	7.633333	7.700000
EBOwithCMAR	9.733333	9.833333	9.100000	7.800000	5.633333	4.400000	3.783333	3.450000	3.200000	3.333333	3.233333	3.100000	3.300000	3.450000
ELSHADE_SPACMA	11.500000	11.366667	10.766667	8.966667	7.000000	5.333333	4.100000	3.700000	3.983333	4.233333	4.316667	4.450000	4.550000	4.683333
GRSA-todo	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000
GSKA	6.433333	5.566667	5.166667	5.133333	5.533333	6.700000	7.333333	7.633333	7.633333	7.833333	7.683333	7.483333	7.533333	7.433333
MM_OED	7.066667	5.900000	5.433333	3.800000	3.900000	3.416667	3.566667	3.450000	3.450000	3.116667	3.283333	3.516667	3.516667	3.616667
MOS	2.033333	3.300000	4.433333	5.433333	5.900000	6.350000	7.366667	7.816667	8.250000	8.366667	8.500000	8.616667	8.633333	8.750000
PPSO	9.800000	7.133333	6.166667	5.433333	6.000000	6.500000	7.133333	7.366667	7.300000	7.400000	7.500000	7.533333	7.450000	7.283333
PSO	11.533333	12.033333	12.233333	12.500000	12.633333	12.566667	12.300000	12.100000	12.066667	11.766667	11.600000	11.600000	11.566667	11.500000
RB-IPOP-CMA-ES	3.533333	3.833333	4.300000	5.266667	6.100000	5.866667	5.633333	5.966667	6.000000	5.750000	6.000000	6.116667	6.100000	6.083333
SSA	4.833333	7.300000	8.400000	9.766667	10.966667	11.500000	11.633333	11.833333	11.966667	11.900000	12.000000	11.966667	12.000000	12.066667
TLBO-FL	6.566667	6.500000	5.966667	6.300000	7.500000	8.100000	8.533333	8.766667	8.850000	8.966667	9.066667	9.066667	8.966667	8.833333
jSO	10.466667	9.533333	8.466667	8.000000	7.433333	7.300000	6.433333	5.716667	4.983333	4.866667	4.433333	4.200000	4.000000	4.016667

Cuadro 5.4: Dimensión 10 en GRSA-Todo

- Dimensión 30.

  - GRSA.

	1	2	3	5	10	20	30	40	50	60	70	80	90	100
AEO	5.733333	8.133333	9.400000	10.633333	11.800000	12.600000	12.766667	12.966667	13.033333	13.066667	13.066667	13.000000	12.966667	12.933333
DE	12.966667	13.166667	13.200000	11.400000	10.066667	9.833333	9.500000	9.366667	9.166667	9.066667	9.033333	8.900000	8.700000	8.700000
DYYPO	7.233333	8.133333	8.500000	9.300000	9.066667	8.100000	7.633333	7.633333	8.066667	8.300000	8.366667	8.466667	8.533333	8.600000
EBOwithCMAR	11.033333	10.300000	9.300000	8.566667	7.600000	5.466667	4.900000	4.316667	4.316667	4.216667	3.816667	3.016667	2.900000	2.633333
ELSHADE_SPACMA	12.366667	11.466667	11.033333	9.566667	7.800000	5.600000	4.533333	3.333333	2.533333	2.433333	2.300000	2.533333	2.583333	2.883333
GRSA	14.866667	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000
GSKA	4.200000	4.066667	4.066667	4.733333	5.700000	7.133333	7.866667	8.100000	8.166667	8.233333	8.200000	8.266667	8.300000	8.366667
MM_OED	9.100000	7.233333	6.633333	5.700000	4.266667	3.000000	2.733333	2.550000	2.583333	2.333333	2.583333	2.916667	3.066667	3.666667
MOS	2.233333	2.733333	3.100000	2.833333	3.466667	5.333333	6.500000	7.333333	7.866667	7.966667	8.133333	8.200000	8.250000	8.316667
PPSO	6.066667	5.300000	4.900000	4.700000	5.000000	6.066667	6.666667	7.433333	7.700000	7.866667	8.000000	8.066667	8.166667	8.066667
PSO	10.933333	11.800000	12.233333	12.966667	12.866667	12.866667	12.800000	12.833333	12.766667	12.733333	12.700000	12.666667	12.666667	12.600000
RB-IPOP-CMA-ES	2.800000	2.766667	2.800000	2.900000	3.066667	3.666667	4.000000	4.066667	4.283333	4.316667	4.516667	4.550000	4.633333	4.666667
SSA	5.700000	6.633333	7.200000	8.233333	10.033333	10.700000	11.266667	11.566667	11.700000	11.766667	11.800000	11.966667	12.000000	12.033333
TLBO-FL	4.400000	4.066667	4.100000	4.666667	5.433333	6.733333	7.533333	7.966667	8.133333	8.333333	8.466667	8.600000	8.700000	8.600000
jSO	10.366667	9.200000	8.533333	8.800000	8.833333	7.900000	6.300000	5.533333	4.683333	4.366667	4.016667	3.850000	3.533333	2.933333

Cuadro 5.5: Dimensión 30 en GRSA

  - GRSA-BL.

	1	2	3	5	10	20	30	40	50	60	70	80	90	100
AEO	5.733333	8.133333	9.400000	10.633333	11.800000	12.600000	12.766667	12.966667	13.033333	13.066667	13.066667	13.000000	12.966667	12.933333
DE	12.966667	13.166667	13.200000	11.400000	10.066667	9.833333	9.500000	9.366667	9.166667	9.066667	9.033333	8.900000	8.700000	8.700000
DYYPO	7.233333	8.133333	8.500000	9.300000	9.066667	8.100000	7.633333	7.633333	8.066667	8.300000	8.366667	8.466667	8.533333	8.600000
EBOwithCMAR	11.033333	10.300000	9.300000	8.566667	7.600000	5.466667	4.900000	4.316667	4.316667	4.216667	3.816667	3.016667	2.900000	2.633333
ELSHADE_SPACMA	12.366667	11.466667	11.033333	9.566667	7.800000	5.600000	4.533333	3.333333	2.533333	2.433333	2.300000	2.533333	2.583333	2.883333
GRSA-bl	14.866667	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000
GSKA	4.200000	4.066667	4.066667	4.733333	5.700000	7.133333	7.866667	8.100000	8.166667	8.233333	8.200000	8.266667	8.300000	8.366667
MM_OED	9.100000	7.233333	6.633333	5.700000	4.266667	3.000000	2.733333	2.550000	2.583333	2.333333	2.583333	2.916667	3.066667	3.666667
MOS	2.233333	2.733333	3.100000	2.833333	3.466667	5.333333	6.500000	7.333333	7.866667	7.966667	8.133333	8.200000	8.250000	8.316667
PPSO	6.066667	5.300000	4.900000	4.700000	5.000000	6.066667	6.666667	7.433333	7.700000	7.866667	8.000000	8.066667	8.166667	8.066667
PSO	10.933333	11.800000	12.233333	12.966667	12.866667	12.866667	12.800000	12.833333	12.766667	12.733333	12.700000	12.666667	12.666667	12.600000
RB-IPOP-CMA-ES	2.800000	2.766667	2.800000	2.900000	3.066667	3.666667	4.000000	4.066667	4.283333	4.316667	4.516667	4.550000	4.633333	4.666667
SSA	5.700000	6.633333	7.200000	8.233333	10.033333	10.700000	11.266667	11.566667	11.700000	11.766667	11.800000	11.966667	12.000000	12.033333
TLBO-FL	4.400000	4.066667	4.100000	4.666667	5.433333	6.733333	7.533333	7.966667	8.133333	8.333333	8.466667	8.600000	8.700000	8.600000
jSO	10.366667	9.200000	8.533333	8.800000	8.833333	7.900000	6.300000	5.533333	4.683333	4.366667	4.016667	3.850000	3.533333	2.933333

Cuadro 5.6: Dimensión 30 en GRSA-BL

  - GRSA-Mej.

	1	2	3	5	10	20	30	40	50	60	70	80	90	100
AEO	5.733333	8.133333	9.400000	10.633333	11.800000	12.600000	12.766667	12.966667	13.033333	13.066667	13.066667	13.000000	12.966667	12.933333
DE	12.966667	13.166667	13.200000	11.400000	10.066667	9.833333	9.500000	9.366667	9.166667	9.066667	9.033333	8.900000	8.700000	8.700000
DYYPO	7.233333	8.133333	8.500000	9.300000	9.066667	8.100000	7.633333	7.633333	8.066667	8.300000	8.366667	8.466667	8.533333	8.600000
EBOwithCMAR	11.033333	10.300000	9.300000	8.566667	7.600000	5.466667	4.900000	4.316667	4.316667	4.216667	3.816667	3.016667	2.900000	2.633333
ELSHADE_SPACMA	12.366667	11.466667	11.033333	9.566667	7.800000	5.600000	4.533333	3.333333	2.533333	2.433333	2.300000	2.533333	2.583333	2.883333
GRSA-MEJ	14.866667	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000
GSKA	4.200000	4.066667	4.066667	4.733333	5.700000	7.133333	7.866667	8.100000	8.166667	8.233333	8.200000	8.266667	8.300000	8.366667
MM_OED	9.100000	7.233333	6.633333	5.700000	4.266667	3.000000	2.733333	2.550000	2.583333	2.333333	2.583333	2.916667	3.066667	3.666667
MOS	2.233333	2.733333	3.100000	2.833333	3.466667	5.333333	6.500000	7.333333	7.866667	7.966667	8.133333	8.200000	8.250000	8.316667
PPSO	6.066667	5.300000	4.900000	4.700000	5.000000	6.066667	6.666667	7.433333	7.700000	7.866667	8.000000	8.066667	8.166667	8.066667
PSO	10.933333	11.800000	12.233333	12.966667	12.866667	12.866667	12.800000	12.833333	12.766667	12.733333	12.700000	12.666667	12.666667	12.600000
RB-IPOP-CMA-ES	2.800000	2.766667	2.800000	2.900000	3.066667	3.666667	4.000000	4.066667	4.283333	4.316667	4.516667	4.550000	4.633333	4.666667
SSA	5.700000	6.633333	7.200000	8.233333	10.033333	10.700000	11.266667	11.566667	11.700000	11.766667	11.800000	11.966667	12.000000	12.033333
TLBO-FL	4.400000	4.066667	4.100000	4.666667	5.433333	6.733333	7.533333	7.966667	8.133333	8.333333	8.466667	8.600000	8.700000	8.600000
jSO	10.366667	9.200000	8.533333	8.800000	8.833333	7.900000	6.300000	5.533333	4.683333	4.366667	4.016667	3.850000	3.533333	2.933333

Cuadro 5.7: Dimensión 30 en GRSA-Mej

- GRSA-Todo.

	1	2	3	5	10	20	30	40	50	60	70	80	90	100
AEO	5.733333	8.133333	9.400000	10.633333	11.800000	12.600000	12.766667	12.966667	13.033333	13.066667	13.066667	13.000000	12.966667	12.933333
DE	12.966667	13.166667	13.200000	11.400000	10.066667	9.833333	9.500000	9.366667	9.166667	9.066667	9.033333	8.900000	8.700000	8.700000
DYYPO	7.233333	8.133333	8.500000	9.300000	9.066667	8.100000	7.633333	7.633333	8.066667	8.300000	8.366667	8.466667	8.533333	8.600000
EBOwithCMAR	11.033333	10.300000	9.300000	8.566667	7.600000	5.466667	4.900000	4.316667	4.316667	4.216667	3.816667	3.016667	2.900000	2.633333
ELSHADE_SPACMA	12.366667	11.466667	11.033333	9.566667	7.800000	5.600000	4.533333	3.333333	2.533333	2.433333	2.300000	2.533333	2.583333	2.883333
GRSA-Todo	14.866667	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000
GSKA	4.200000	4.066667	4.066667	4.733333	5.700000	7.133333	7.866667	8.100000	8.166667	8.233333	8.200000	8.266667	8.300000	8.366667
MM_OED	9.100000	7.233333	6.633333	5.700000	4.266667	3.000000	2.733333	2.550000	2.583333	2.333333	2.583333	2.916667	3.066667	3.666667
MOS	2.233333	2.733333	3.100000	2.833333	3.466667	5.333333	6.500000	7.333333	7.866667	7.966667	8.133333	8.200000	8.250000	8.316667
PPSO	6.066667	5.300000	4.900000	4.700000	5.000000	6.066667	6.666667	7.433333	7.700000	7.866667	8.000000	8.066667	8.166667	8.066667
PSO	10.933333	11.800000	12.233333	12.966667	12.866667	12.866667	12.800000	12.833333	12.766667	12.733333	12.700000	12.666667	12.666667	12.600000
RB-IPOP-CMA-ES	2.800000	2.766667	2.800000	2.900000	3.066667	3.666667	4.000000	4.066667	4.283333	4.316667	4.516667	4.550000	4.633333	4.666667
SSA	5.700000	6.633333	7.200000	8.233333	10.033333	10.700000	11.266667	11.566667	11.700000	11.766667	11.800000	11.966667	12.000000	12.033333
TLBO-FL	4.400000	4.066667	4.100000	4.666667	5.433333	6.733333	7.533333	7.966667	8.133333	8.333333	8.466667	8.600000	8.700000	8.600000
jSO	10.366667	9.200000	8.533333	8.800000	8.833333	7.900000	6.300000	5.533333	4.683333	4.366667	4.016667	3.850000	3.533333	2.933333

Cuadro 5.8: Dimensión 30 en GRSA-Todo

- Dimensión 50.

  - GRSA.

	1	2	3	5	10	20	30	40	50	60	70	80	90	100
AEO	6.766667	8.900000	9.966667	11.200000	12.133333	12.666667	12.733333	12.766667	12.800000	12.800000	12.866667	12.933333	12.933333	12.900000
DE	12.966667	13.066667	12.633333	11.600000	10.766667	10.200000	10.266667	10.200000	10.033333	9.933333	9.833333	9.700000	9.500000	9.366667
DYYPO	7.900000	9.633333	10.066667	10.466667	9.866667	8.533333	7.433333	7.100000	7.000000	7.433333	7.633333	7.833333	7.866667	8.000000
EBOwithCMAR	11.533333	9.900000	9.266667	8.566667	7.266667	5.866667	5.700000	5.266667	4.716667	4.616667	4.150000	3.216667	3.116667	2.800000
ELSHADE_SPACMA	12.433333	11.700000	10.866667	9.400000	7.633333	5.566667	4.800000	3.700000	2.383333	1.850000	1.816667	1.833333	1.916667	2.266667
GRSA	14.833333	14.900000	14.966667	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000
GSKA	4.366667	4.600000	4.700000	5.200000	6.366667	8.000000	8.533333	8.633333	8.700000	8.733333	8.700000	8.700000	8.866667	8.833333
MM_OED	9.233333	7.300000	6.933333	5.700000	4.733333	3.533333	2.933333	3.366667	3.500000	2.866667	3.016667	3.450000	3.450000	4.016667
MOS	1.833333	2.366667	2.600000	2.433333	3.100000	4.966667	6.100000	6.866667	7.766667	8.066667	8.033333	8.066667	8.200000	8.200000
PPSO	6.366667	5.633333	5.333333	5.233333	5.366667	5.966667	6.833333	7.200000	7.566667	7.733333	8.100000	8.233333	8.300000	8.333333
PSO	10.066667	11.466667	12.000000	12.733333	13.066667	12.966667	12.900000	12.800000	12.766667	12.733333	12.733333	12.733333	12.733333	12.733333
RB-IPOP-CMA-ES	2.800000	2.600000	2.533333	2.633333	2.333333	3.100000	3.233333	3.333333	3.650000	3.983333	4.166667	4.333333	4.300000	4.450000
SSA	4.866667	5.433333	5.833333	6.733333	8.466667	9.866667	10.433333	10.833333	11.033333	11.100000	11.266667	11.433333	11.533333	11.500000
TLBO-FL	3.900000	3.900000	4.166667	4.500000	5.400000	6.666667	7.233333	7.600000	7.866667	8.133333	8.366667	8.533333	8.600000	8.666667
jSO	10.133333	8.600000	8.133333	8.600000	8.500000	7.100000	5.866667	5.333333	5.216667	5.016667	4.316667	4.000000	3.683333	2.933333

Cuadro 5.9: Dimensión 50 en GRSA

  - GRSA-BL.

	1	2	3	5	10	20	30	40	50	60	70	80	90	100
AEO	6.766667	8.900000	9.966667	11.200000	12.133333	12.666667	12.733333	12.766667	12.800000	12.800000	12.866667	12.933333	12.933333	12.900000
DE	12.966667	13.066667	12.633333	11.600000	10.766667	10.200000	10.266667	10.200000	10.033333	9.933333	9.833333	9.700000	9.500000	9.366667
DYYPO	7.900000	9.633333	10.066667	10.466667	9.866667	8.533333	7.433333	7.100000	7.000000	7.433333	7.633333	7.833333	7.866667	8.000000
EBOwithCMAR	11.533333	9.900000	9.266667	8.566667	7.266667	5.866667	5.700000	5.266667	4.716667	4.616667	4.150000	3.216667	3.116667	2.800000
ELSHADE_SPACMA	12.433333	11.700000	10.866667	9.400000	7.633333	5.566667	4.800000	3.700000	2.383333	1.850000	1.816667	1.833333	1.916667	2.266667
GRSA-BL	14.833333	14.900000	14.966667	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000
GSKA	4.366667	4.600000	4.700000	5.200000	6.366667	8.000000	8.533333	8.633333	8.700000	8.733333	8.700000	8.700000	8.866667	8.833333
MM_OED	9.233333	7.300000	6.933333	5.700000	4.733333	3.533333	2.933333	3.366667	3.500000	2.866667	3.016667	3.450000	3.450000	4.016667
MOS	1.833333	2.366667	2.600000	2.433333	3.100000	4.966667	6.100000	6.866667	7.766667	8.066667	8.033333	8.066667	8.200000	8.200000
PPSO	6.366667	5.633333	5.333333	5.233333	5.366667	5.966667	6.833333	7.200000	7.566667	7.733333	8.100000	8.233333	8.300000	8.333333
PSO	10.066667	11.466667	12.000000	12.733333	13.066667	12.966667	12.900000	12.800000	12.766667	12.733333	12.733333	12.733333	12.733333	12.733333
RB-IPOP-CMA-ES	2.800000	2.600000	2.533333	2.633333	2.333333	3.100000	3.233333	3.333333	3.650000	3.983333	4.166667	4.333333	4.300000	4.450000
SSA	4.866667	5.433333	5.833333	6.733333	8.466667	9.866667	10.433333	10.833333	11.033333	11.100000	11.266667	11.433333	11.533333	11.500000
TLBO-FL	3.900000	3.900000	4.166667	4.500000	5.400000	6.666667	7.233333	7.600000	7.866667	8.133333	8.366667	8.533333	8.600000	8.666667
jSO	10.133333	8.600000	8.133333	8.600000	8.500000	7.100000	5.866667	5.333333	5.216667	5.016667	4.316667	4.000000	3.683333	2.933333

Cuadro 5.10: Dimensión 50 en GRSA-BL

  - GRSA-Mej.

	1	2	3	5	10	20	30	40	50	60	70	80	90	100
AEO	6.766667	8.900000	9.966667	11.200000	12.133333	12.666667	12.733333	12.766667	12.800000	12.800000	12.866667	12.933333	12.933333	12.900000
DE	12.966667	13.066667	12.633333	11.600000	10.766667	10.200000	10.266667	10.200000	10.033333	9.933333	9.833333	9.700000	9.500000	9.366667
DYYPO	7.900000	9.633333	10.066667	10.466667	9.866667	8.533333	7.433333	7.100000	7.000000	7.433333	7.633333	7.833333	7.866667	8.000000
EBOwithCMAR	11.533333	9.900000	9.266667	8.566667	7.266667	5.866667	5.700000	5.266667	4.716667	4.616667	4.150000	3.216667	3.116667	2.800000
ELSHADE_SPACMA	12.433333	11.700000	10.866667	9.400000	7.633333	5.566667	4.800000	3.700000	2.383333	1.850000	1.816667	1.833333	1.916667	2.266667
GRSA-Mej	14.833333	14.900000	14.966667	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000
GSKA	4.366667	4.600000	4.700000	5.200000	6.366667	8.000000	8.533333	8.633333	8.700000	8.733333	8.700000	8.700000	8.866667	8.833333
MM_OED	9.233333	7.300000	6.933333	5.700000	4.733333	3.533333	2.933333	3.366667	3.500000	2.866667	3.016667	3.450000	3.450000	4.016667
MOS	1.833333	2.366667	2.600000	2.433333	3.100000	4.966667	6.100000	6.866667	7.766667	8.066667	8.033333	8.066667	8.200000	8.200000
PPSO	6.366667	5.633333	5.333333	5.233333	5.366667	5.966667	6.833333	7.200000	7.566667	7.733333	8.100000	8.233333	8.300000	8.333333
PSO	10.066667	11.466667	12.000000	12.733333	13.066667	12.966667	12.900000	12.800000	12.766667	12.733333	12.733333	12.733333	12.733333	12.733333
RB-IPOP-CMA-ES	2.800000	2.600000	2.533333	2.633333	2.333333	3.100000	3.233333	3.333333	3.650000	3.983333	4.166667	4.333333	4.300000	4.450000
SSA	4.866667	5.433333	5.833333	6.733333	8.466667	9.866667	10.433333	10.833333	11.033333	11.100000	11.266667	11.433333	11.533333	11.500000
TLBO-FL	3.900000	3.900000	4.166667	4.500000	5.400000	6.666667	7.233333	7.600000	7.866667	8.133333	8.366667	8.533333	8.600000	8.666667
jSO	10.133333	8.600000	8.133333	8.600000	8.500000	7.100000	5.866667	5.333333	5.216667	5.016667	4.316667	4.000000	3.683333	2.933333

Cuadro 5.11: Dimensión 50 en GRSA-Mej

- GRSA-Todo.

	1	2	3	5	10	20	30	40	50	60	70	80	90	100
AEO	6.766667	8.900000	9.966667	11.200000	12.133333	12.666667	12.733333	12.766667	12.800000	12.800000	12.866667	12.933333	12.933333	12.900000
DE	12.966667	13.066667	12.633333	11.600000	10.766667	10.200000	10.266667	10.200000	10.033333	9.933333	9.833333	9.700000	9.500000	9.366667
DYYPO	7.900000	9.633333	10.066667	10.466667	9.866667	8.533333	7.433333	7.100000	7.000000	7.433333	7.633333	7.833333	7.866667	8.000000
EBOwithCMAR	11.533333	9.900000	9.266667	8.566667	7.266667	5.866667	5.700000	5.266667	4.716667	4.616667	4.150000	3.216667	3.116667	2.800000
ELSHADE_SPACMA	12.433333	11.700000	10.866667	9.400000	7.633333	5.566667	4.800000	3.700000	2.383333	1.850000	1.816667	1.833333	1.916667	2.266667
GRSA-Todo	14.833333	14.900000	14.966667	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000	15.000000
GSKA	4.366667	4.600000	4.700000	5.200000	6.366667	8.000000	8.533333	8.633333	8.700000	8.733333	8.700000	8.700000	8.866667	8.833333
MM_OED	9.233333	7.300000	6.933333	5.700000	4.733333	3.533333	2.933333	3.366667	3.500000	2.866667	3.016667	3.450000	3.450000	4.016667
MOS	1.833333	2.366667	2.600000	2.433333	3.100000	4.966667	6.100000	6.866667	7.766667	8.066667	8.033333	8.066667	8.200000	8.200000
PPSO	6.366667	5.633333	5.333333	5.233333	5.366667	5.966667	6.833333	7.200000	7.566667	7.733333	8.100000	8.233333	8.300000	8.333333
PSO	10.066667	11.466667	12.000000	12.733333	13.066667	12.966667	12.900000	12.800000	12.766667	12.733333	12.733333	12.733333	12.733333	12.733333
RB-IPOP-CMA-ES	2.800000	2.600000	2.533333	2.633333	2.333333	3.100000	3.233333	3.333333	3.650000	3.983333	4.166667	4.333333	4.300000	4.450000
SSA	4.866667	5.433333	5.833333	6.733333	8.466667	9.866667	10.433333	10.833333	11.033333	11.100000	11.266667	11.433333	11.533333	11.500000
TLBO-FL	3.900000	3.900000	4.166667	4.500000	5.400000	6.666667	7.233333	7.600000	7.866667	8.133333	8.366667	8.533333	8.600000	8.666667
jSO	10.133333	8.600000	8.133333	8.600000	8.500000	7.100000	5.866667	5.333333	5.216667	5.016667	4.316667	4.000000	3.683333	2.933333

Cuadro 5.12: Dimensión 50 en GRSA-Todo

A grandes rasgos se puede observar que el algoritmo GRSA es malo con respecto al resto de algoritmos con los que se compara en *Tacolab*. A mayor dimensión sí es cierto que las primeras funciones mejoran ligeramente.

Dados los resultados que aparecen en el artículo, se intuye que el algoritmo GRSA y sus variantes tienen una convergencia bastante lenta, y no acaba convergiendo tras $10000 \cdot dimension$ evaluaciones. Además, esto se puede sustentar dado a que a mayor dimensión, los resultados empiezan a mejorar ligeramente.

En conclusión, si se tuviese que usar una metaheurística para resolver un problema, se optaría por usar otros algoritmos como genéticos, antes de usar algunos como GRSA. Sin embargo, para comprobar si los cambios propuestos son buenos con respecto al original, se hará un pequeño análisis comparativo.

5.2. Análisis de resultados de GRSA

Se va a realizar el análisis del fitness medio, error medio y tiempo de ejecución medio para comparar algoritmos. Dada la gran diferencia de valores entre dimensiones, se aplicará una escala logarítmica para poder analizar mejor las gráfica.

- Dimensión 10.

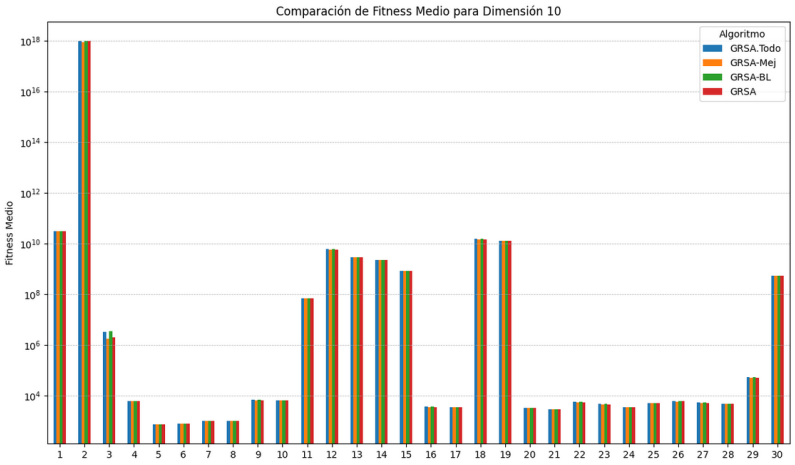

Figura 5.1: Fitness medio

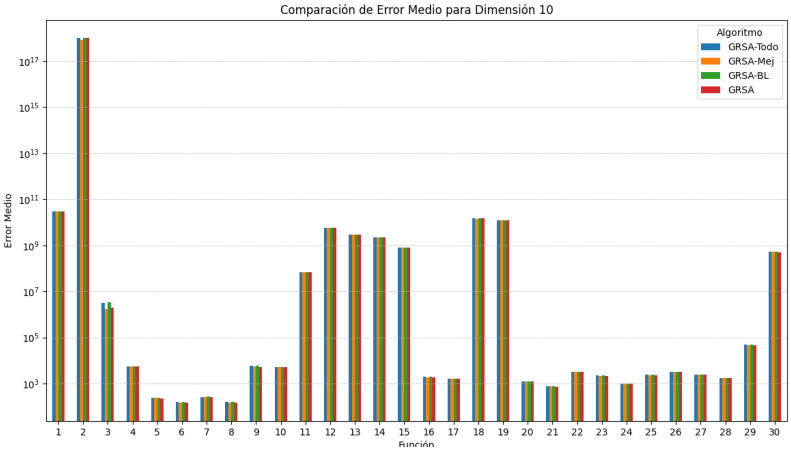

Figura 5.2: Error medio

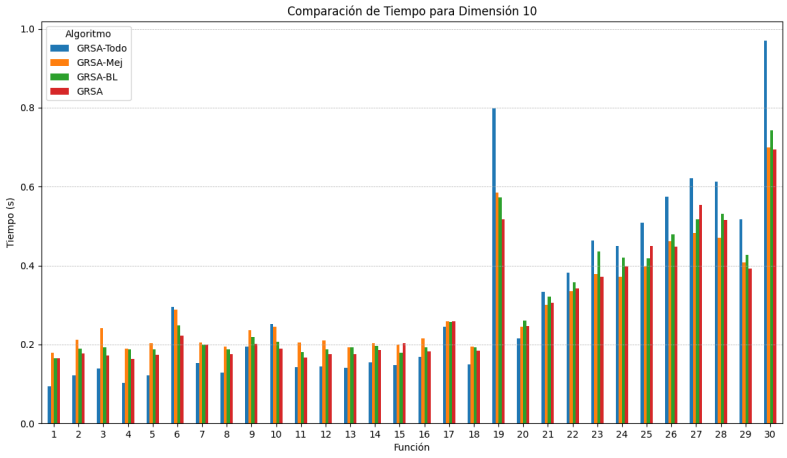

Figura 5.3: Tiempo medio

- Dimensión 30.

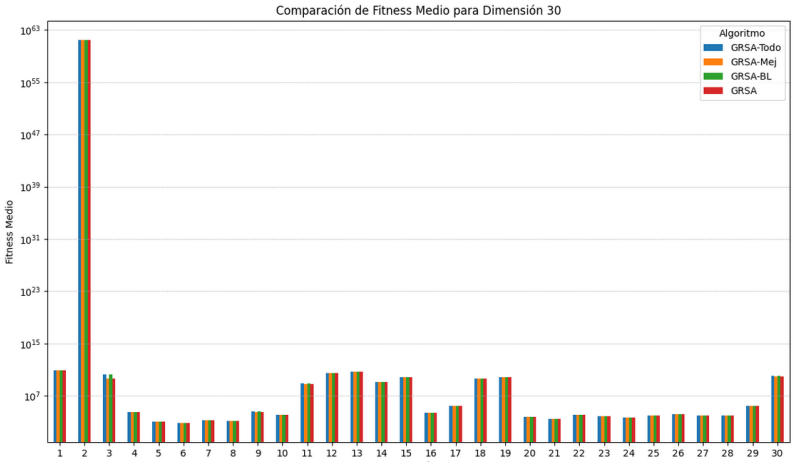

Figura 5.4: Fitness medio

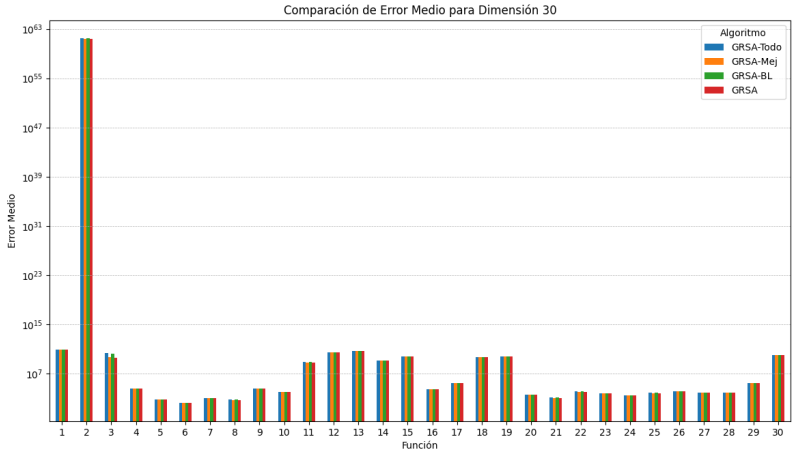

Figura 5.5: Error medio

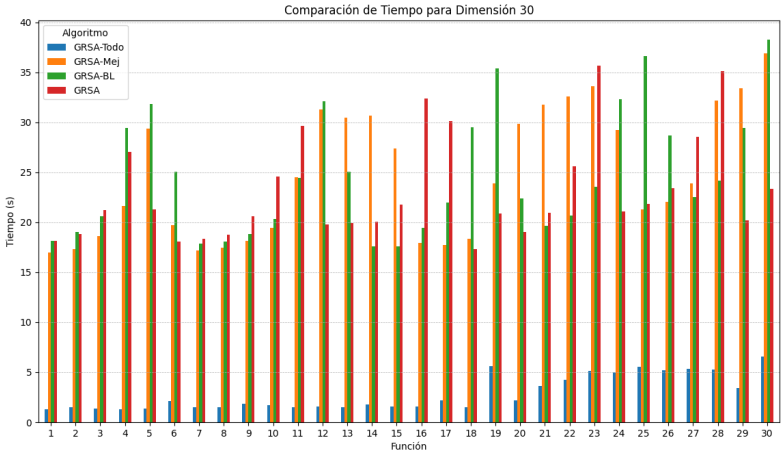

Figura 5.6: Tiempo medio

- Dimensión 50.

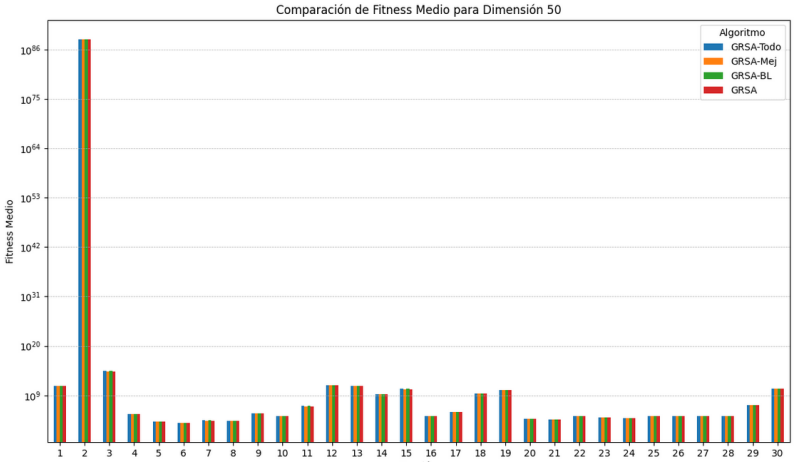

Figura 5.7: Fitness medio

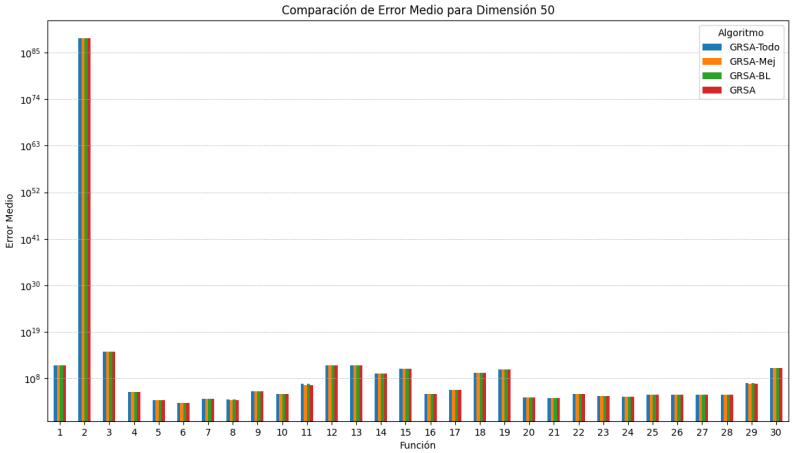

Figura 5.8: Error medio

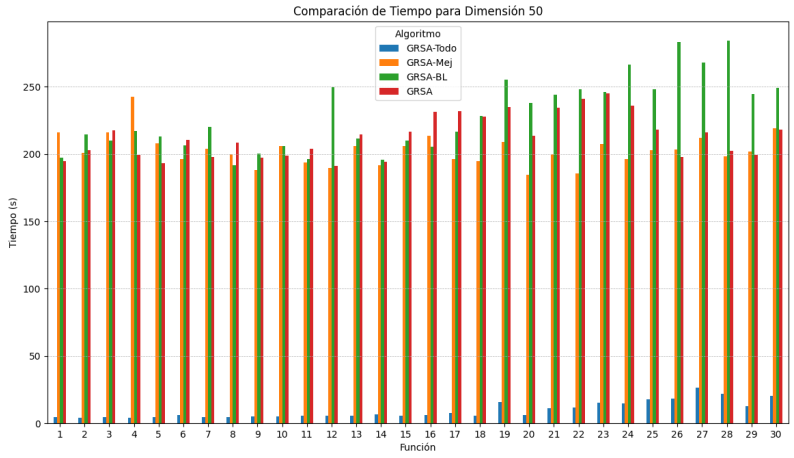

Figura 5.9: Tiempo medio

Se han medido tres métricas para evaluar los cuatro algoritmos: **fitness**, **error** y **tiempo de ejecución** (en segundos).

En primer lugar, el algoritmo GRSA básico ofrece unos valores bastante elevados, especialmente en dimensiones superiores (intuyendo que a mayor dimensión, mejores resultados dará). Se puede comentar lo mismo en el caso del error, lo cual era esperable por la forma de calcularlo. Además, en general el tiempo aumenta con

la dimensión del problema, siendo relativamente bajo para dimensión 10 y alto para dimensión 50.

El algoritmo GRSA con la mejora propuesta presenta un fitness ligeramente mayor que GRSA básico en muchas funciones. El error medio sigue siendo parecido, guardando cierta relación con fitness y el tiempo es algo mayor que en GRSA básico, sobre todo para dimensiones altas, aunque sigue siendo un tiempo manejable.

En el caso de la hibridación con búsqueda local, fitness da mejoras en algunas casos específicos respecto a GRSA básico, y en términos de error presenta magnitudes parecidas al algoritmo GRSA con la mejora propuesta.

Por último, el GRSA que incluye tanto la mejora como la hibridación con búsqueda local desprende un fitness bastante mejor que los tres algoritmos anteriores, además de un error menor por función, reflejando un rendimiento superior en el problema de optimización. Por el contrario, se puede observar que el tiempo de ejecución es bastante menor. Ello se puede deber a que se llega al número de evaluaciones de la función objetivo máximo (que se evalua una gran cantidad de veces entre la mejora propuesta y las evaluaciones en búsqueda local). Sin embargo, los resultados en fitness son bastante buenos para un tiempo de ejecución bajo, por lo que se intuye que el algoritmo converge bastante rápido a óptimos.

Generalmente se podría escoger el algoritmo **GRSA-Todo**, que incluye tanto la hibridación con búsqueda local como la mejora propuesta, independientemente de que el tiempo de ejecución sea o no importante. Ofrece los mejores resultados en cuanto a fitness y error medio para un tiempo de ejecución menor al que da el algoritmo GRSA básico (aunque se deba a la llegada al máximo de evaluaciones de la función objetivo, fitness y error da un buen valor tanto en alta dimensión como en baja dimensión).

5.2.1. Conclusiones de la práctica alternativa

Para exponer las conclusiones obtenidas tras el desarrollo de la práctica, se deben tener en cuenta tres tópicos:

- **Rendimiento general**: El algoritmo **GRSA-Todo** muestra le mejor capacidad de ejecución y en este caso a menor tiempo de ejecución. Si bien es cierto este menor tiempo de ejecución podría deberse a la llegada prematura al máximo de evaluaciones, da a entender que converge bastante rápido y si se escogiese un número de evaluaciones elevado, el algoritmo se estancaría bastante pronto.

- **Compromiso entre tiempo y precisión**: El algoritmo **GRSA-Todo** es el que mejores métricas arroja en todos los sentidos, tanto en el tiempo de ejecución como en el fitness y error medios.

- **Dimensión del problema**: En problemas de baja dimensión (en este caso, dimensión 10), la diferencia en el tiempo de ejecución entre algoritmos es menos pronunciada, y se pueden preferir versiones mejoradas si la calidad del resultado es importante. En problemas de mayor dimensionalidad (dimensión 50 en concreto), el algoritmo **GRSA-Todo** es el que ofrece mejores resultados en relación a su prematura parada (aunque no haya convergido, puede que lso resultados mejoren si se elimina la cantidad máximo de evaluaciones de la función objetivo, o se quede oscilando en torno a cierto valor porque la función haya convergido).

Es importante remarcar que, aunque **GRSA-Todo** sea el mejor algoritmo de los cuatro expuestos, generalmente es peor que otras metaheurísticas, tal y como se ha podido observar en la comparación en Tacolab.