Doble Grado en
Ingeniería Informática y Matemáticas

METAHEURÍSTICAS
(E. Computación y Sistemas Inteligentes)

UNIVERSIDAD DE GRANADA

PROYECTO FINAL

Cuckoo Search
aplicada al problema de Máxima Diversidad.

Alberto Estepa Fernández
Email: albertoestep@correo.ugr.es

24 de junio de 2020

Índice

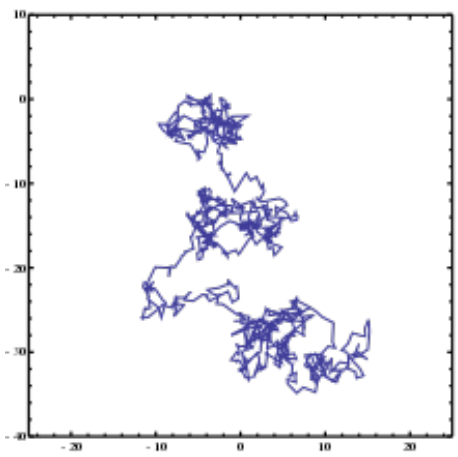
1. Explicación metaheurística

En este trabajo estudiaremos la metaheurística Cuckoo Search (CS) y la adaptaremos al problema de Máxima Diversidad.

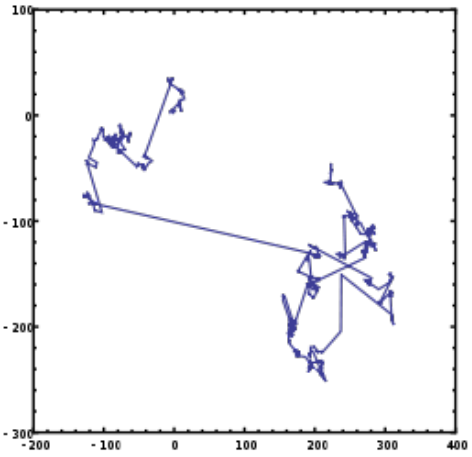
La metaheurística se basa en el comportamiento de reproducción (el parasitismo de cría) de ciertas especies de cucos. Primero presentaremos el comportamiento reproductivo de los cucos y las características de los vuelos de Lévy de algunas aves y moscas de la fruta, y luego formularemos el algoritmo, seguido de su implementación.

1.1. Vuelo de Lévy

Un vuelo de Lévy [2], es un tipo de paseo aleatorio en el cual los incrementos son distribuidos de acuerdo a una distribución de probabilidad de cola pesada.



(a) Representación de un movimiento browniano de 1000 pasos



(b) Representación de un vuelo de Lévy de 1000 pasos

Un ejemplo que ilustra dicho movimiento es el que nos proporcionan los tiburones y otros depredadores del océano cuando no pueden encontrar alimento. Estos abandonan el movimiento browniano (el movimiento al azar visto en moléculas de gas) por el vuelo de Lévy (una mezcla de trayectorias largas y movimientos al azar cortos encontrados en líquidos turbulentos). En la imagen 1b podemos apreciar grandes saltos en la localización con respecto a la imagen 1a.

Por otro lado, varios estudios han demostrado que el comportamiento de vuelo de muchos animales e insectos ha demostrado las características típicas de los vuelos de Lévy. Concretamente algunas moscas de la fruta exploran su paisaje utilizando una serie de rutas de vuelo rectas puntuadas por un giro repentino de 90 grados, lo que lleva a un patrón de búsqueda libre de escala intermitente estilo Lévy. Posteriormente, dicho comportamiento se ha aplicado a la optimización y la búsqueda óptima, y los resultados preliminares muestran su capacidad prometedora.

1.2. Comportamiento de cría de cuco

Los cucos han desarrollado una estrategia de reproducción agresiva. La mayoria de especies de cucos se dedican al parasitismo de cría obligado al poner sus huevos en los nidos de otras aves hospederas (a menudo otras especies).

Algunas aves hospedadoras pueden entablar un conflicto directo con los cucos intrusos. Si un ave huésped descubre que los huevos no son de su propiedad, tirarán estos huevos alienígenas o simplemente abandonarán su nido y construirán un nuevo nido en otro lugar. Algunas especies de cuco, han evolucionado de tal manera que los cucos parásitos hembras a menudo están muy especializadas en la imitación del color y el patrón de los huevos de unas pocas especies de hospedadores elegidas. Esto reduce la probabilidad de que sus huevos sean abandonados y, por lo tanto, aumenta su reproductividad. Además, el momento de la puesta de huevos de algunas especies también es sorprendente. Los cucos parásitos a menudo eligen un nido donde el ave huésped acaba de poner sus propios huevos. En general, los huevos de cuco eclosionan un poco antes que sus huevos de huésped. Una vez que nace el primer pollito cuco, la primera acción instintiva que tomará es desalojar los huevos del hospedador al expulsar ciegamente los huevos del nido, lo que aumenta la porción de alimento del pollito cuco proporcionado por su ave hospedadora. Los estudios también muestran que un pollito cuco también puede imitar la llamada de los pollitos anfitriones para obtener acceso a más oportunidades de alimentación.

1.3. Algotimo Cuckoo Search

Para simplificar la descripción de nuestra metaheurística (Cuckoo Search), usamos las siguientes tres reglas idealizadas:

1. Cada cuco pone un huevo a la vez y lo deja en un nido elegido al azar.
2. Los mejores nidos con huevos de alta calidad se trasladarán a las próximas generaciones.
3. El número de nidos huéspedes es fijo, y el huevo puesto por un cuco es descubierto por el ave huésped con una probabilidad $p_a \in [0, 1]$. En este caso, el ave huésped puede tirar el huevo o abandonar el nido y construir un nido completamente nuevo. Por simplicidad, esta última suposición puede ser aproximada por la fracción p_a de los n nidos reemplazados por nuevos nidos (con nuevas soluciones aleatorias).

Para simplificar, tal y como explica la documentación del autor [1] representamos a cada huevo en un nido como una solución, y un huevo de cuco representa una nueva solución, el objetivo es usar las soluciones nuevas y potencialmente mejores (cucos) para reemplazar una solución peor en los nidos. En base a estas tres reglas, los pasos básicos de Cuckoo Search (CS) pueden resumirse como el pseudocódigo

que se muestra en 1.

Algorithm 0: Pseudocódigo Cuckoo Search

```
begin
    Función objetivo f(x), x = (x_1, x_2, ..., x_d)^T
    Generamos una población inicial de n nidos huéspedes x_i (i = 1, 2, ..., n)
    while (t < MaxGeneration) or (criterio parada) do
        Obtenemos un cuco al azar por vuelos Lévy y evaluamos su calidad
         (F_i)
        Elegimos un nido entre n (digamos, j) al azar.
        if F_i > F_j then
            reemplazar j por la nueva solución
        end
        Una fracción (p_a) de los peores nidos son abandonados y se
         construyen otros nuevos.
        Mantenemos la mejor solución (o los nidos con mejor soluciones)
        Ordenamos las soluciones y encontramos el mejor actual.
    end
    Devolvemos la mejor solución
end
```

Al generar nuevas soluciones $x^{(t+1)}$ para, por ejemplo, un cuco i, se realiza un vuelo Lévy:

$$x_i^{(t+1)} = x_i^{(t)} + \alpha \otimes \text{Lévy}(\lambda)$$

donde $\alpha > 0$ es el tamaño del paso que debe estar relacionado con las escalas del problema de intereses. En la mayoría de los casos, podemos usar $\alpha = 1$. La ecuación anterior es esencialmente la ecuación estocástica para el paseo aleatorio. En general, una paseo aleatorio es una cadena de Markov cuyo siguiente estado/ubicación solo depende de la ubicación actual (el primer término en la ecuación anterior) y la probabilidad de transición (el segundo término). El producto $\otimes$ significa multiplicaciones de entrada. Este producto de entrada es similar a los utilizados en PSO, pero aquí el paseo aleatorio a través del vuelo de Lévy es más eficiente para explorar el espacio de búsqueda, ya que su longitud de paso es mucho más larga a largo plazo.

El vuelo Lévy esencialmente proporciona un paseo aleatorio, mientras que la longitud aleatoria del paso se extrae de una distribución Lévy.

$$\text{Lévy} \sim t^{-\lambda}, \qquad (1 < \lambda \leq 3)$$

que tiene una varianza infinita con una media infinita. Aquí los pasos esencialmente forman un proceso de paseo aleatorio con una distribución de longitud de paso de ley de potencia con una cola pesada. Algunas de las nuevas soluciones deben ser generadas por Lévy, caminando alrededor de la mejor solución obtenida hasta ahora, esto acelerará la búsqueda local. Sin embargo, una fracción sustancial de las nuevas soluciones debe generarse mediante la aleatorización de campo lejano

y cuyas ubicaciones deben estar lo suficientemente lejos de la mejor solución actual, esto asegurará que el sistema no quede atrapado en un óptimo local.

De un vistazo rápido, parece que hay alguna similitud entre CS y escalada en combinación con alguna aleatorización a gran escala. Pero hay algunas diferencias significativas. En primer lugar, CS es un algoritmo basado en la población, de manera similar a GA y PSO, pero utiliza algún tipo de elitismo y / o selección similar a la utilizada en la búsqueda de armonía. En segundo lugar, la aleatorización es más eficiente ya que la longitud del paso es de cola pesada y es posible cualquier paso grande. En tercer lugar, el número de parámetros a ajustar es menor que GA y PSo, y por lo tanto es potencialmente más genérico adaptarse a una clase más amplia de problemas de optimización. Además, cada nido puede representar un conjunto de soluciones, por lo tanto, CS puede extenderse al tipo de algoritmo de metapoblación.

2. Adaptación al problema de MDP

El Problema de la Máxima Diversidad (*Maximun Diversity Problem*, ***MDP***) es un problema de optimización combinatoria consistente en seleccionar un subconjunto M de m elementos ($|M| = m$) de un conjunto inicial S de n elementos (obviamente, $n > m$) de forma que se maximice la diversidad entre los elementos escogidos.

Además de los n elementos (e_i, $i = 1, \ldots, n$) y el número de elementos a seleccionar m, se dispone de una matriz $D = (d_{ij})$ de dimensión $n \times n$ que contiene las distancias entre ellos.

En este caso la diversidad se calcula como la suma de las distancias entre cada par de elementos seleccionados.

Matemáticamente el problema consistiría en maximizar la función:

$$z_{MS}(x) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d_{ij} x_i x_j$$

Sujeto a que $\sum_{i=1}^{n} x_i = m$ con $x_i = \{0, 1\},\ i = 1, \ldots, n$.

Realmente y como hemos comprobado, la metaheurística está formulada para problemas con valores reales y la representación más natural de nuestro problema es la representación binaria (un vector de tamaño n que contiene un 1 en los índices de los puntos que forman la solución y un 0 en los demás). Esta representación es fácilmente adaptable a una representación en números enteros (un vector de tamaño m que contiene los índices (valores no repetidos entre 1 y n) de los puntos que forman la solución).

Tras un primer intento de adaptar la representación del problema en una representación en valores reales comprobamos que los resultados no eran los esperados.

Decidí por tanto adaptar la metaheurística al problema MDP de otra manera: usaremos la representación binaria de nuestro problema e intentaremos adaptar la metaheurística a dicha representación. A continuación explicaremos dicha adaptación e introduciremos los pseudocódigos de las funciones más importantes.

Por lo pronto, el esquema de representación de una solución usado es el siguiente:

```
struct solucion {
    vector<bool> v;
    double coste;
    bool evaluada;
};
```

Que como hemos comentado, el vector `v` contiene un 1 en los índices de los puntos que forman la solución y un 0 en los demás. Los datos de donde hemos sacado las distancias se han guardado en una matriz simétrica de tamaño $n \times n$. `coste` indica la diversidad de la solución y `evaluada` será un *flag* que indicará si la solución se ha evaluado o no.

La clase `poblacion` permite guardar los individuos de la población en un instante dado, además guardamos también el índice en el vector de de soluciones de la mejor solución actualmente en la población (para evitar cálculos posteriormente), así como la diversidad de ésta solución.

```
class poblacion {
    vector<solucion> v;
    double max_coste;
    int mejor_solucion;
    int n_individuos;

    poblacion();
};
```

La función objetivo (`evaluarSolucion`) calcula, dado un conjunto de elementos que forma la solución, la distancia entre los elementos del conjunto, es decir, la diversidad de la solución y actualiza el *flag* `evaluada`. Hacemos uso de la función `distanciaAUnConjunto` que dado un conjunto de elementos y un elemento concreto, calcula la distancia acumulada entre el elemento y el conjunto de elementos. No incluimos el pseudocódigo de todas la funciones para no hacer pesada la lectura. Dicha función se incluye debidamente comentada en cualquiera de los ficheros del código.

Algorithm 1: evaluarSolucion

```
Data: s = solucion, m = matriz de distancias
Result: coste = diversidad de la solucion
begin
    s.coste ← 0
    foreach i ∈ [0, s.v.size) do
        if s.v[i] then
            s.coste ← s.coste + distanciaAUnConjunto (i, s.v, m)
        end
    end
    s.coste ← s.coste / 2
    s.evaluada ← true
end
```

Si ahora analizamos el pseudocódigo de la metaheurística con detenimiento, es fácil darse cuenta que el método de búsqueda se basa en dos partes bien diferenciadas (una vez inicializada la población inicial):

- Por un lado una búsqueda local, en el que se abandonan los peores nidos (se eliminan las peores soluciones) y se construyen nuevos (nuevas soluciones alterando ligeramente las mejores soluciones encontradas).
- Por otro lado una búsqueda global, en el que se obtiene un cuco (solución) al azar mediante un vuelo Lévy (cambio brusco) para permitir la exploración del dominio y intentando evitar mínimos locales.

Así la búsqueda local la hemos mantenido intacta. Para ello hemos creado una función `reemplazarPoblacion`.

Algorithm 2: reemplazarPoblacion

```
Data: p = población a actualizar
p_a = porcentaje de nidos a abandonar
n = número de elementos a seleccionar para la solución
Result: p : población actualizada
begin
    mejores = Escogemos los p_a mejores de p
    while i ∈ [0, size(mejores)] do
        solucion ← p.v[i]
        alterarSolucion(solucion)
        Añadir solucion a p
        i ← i + 1
    end
    Ordenamos p por diversidad
    Escogemos los n mejores
    Devolvemos la población de n mejores
end
```

Como vemos hacemos uso de la función `alterarSolucion`, que produce una mínima alteración en la solución (intercambiando un solo índice por otro). De nuevo dicha función se incluye debidamente comentada en cualquiera de los ficheros del código. Esto provoca la explotación del algoritmo.

Para la búsqueda global si hemos tenido que hacer cambios respecto a la implementación original, ya que tenemos que expresar de alguna manera el cambio brusco que provoca el vuelo Lévy. Para ello hemos implementado la siguiente función:

Algorithm 3: obtenerCuckooPorLevy

```
Data: m = matriz de distancias
n = número de elementos a seleccionar para la solución
Result: s : solución o cuco obtenido
begin
    n_elegidos ← 0
    Inicializamos solución s con todos los índices de s.v a false.
    while n_elegidos < n do
        na = Calculo número aleatorio
        if !s.v[na] then
            s.v[na] ← true
            n_elegidos ← n_elegidos + 1
        end
    end
    Devolvemos la solución s
end
```

Intentando adaptarnos a la idea del vuelo Lévy (recordemos que intenta hacer un cambio de 90 grados en el espacio de búsqueda y da un salto importante en el desplazamiento, es decir, se produce un cambio brusco), hemos decidido y probado elegir como cambio brusco una nueva solución aleatoria, ya que después de varias pruebas nos daba resultados bastante notables y produce de forma similar lo que el vuelo Lévy quiere provocar, un cambio brusco en el espacio de búsqueda, permitiendo la exploración de dicho espacio.

Obviamente después de obtener dicha solución, la evaluamos y la colocamos en un nido (la sustituimos por una solución).

Lo que acabamos de hacer es una adaptación sencilla de la metaheurística a nuestro problema de MDP y como veremos, los resultados obtenidos con ella serán bastante notables. El código se encuentra en el fichero **CS.cpp**. Posteriormente, en los siguientes apartado, se incluirán mejoras del comportamiento del algoritmo.

3. Mejoras de calidad de la metaheurística con un algoritmo híbrido

Tras numerosas pruebas hemos planteado dos posibles inclusiones de la búsqueda local en nuestro algoritmo que realmente mejoran el resultado.

Primero y antes de todo, para aprovechar el código de la práctica 1 y 2 tal y como implementamos la búsqueda local, usaremos la representación de la solución mediante índices enteros, como ya hemos comentado en la sección 2. Esta representación estará dada por:

```
struct solucion_ints{
    vector<int> v;
    double coste;
};
```

Así un vector `v` de tamaño m que contiene los índices (valores no repetidos entre 1 y n) de los puntos que forman la solución). De igual forma con `coste` guardamos la diversidad de la solución. Para establecer una conexión entre ambas representaciones, como existe una relación biyectiva entre ambas, hemos creado dos funciones: `pasarDeBitsAInts` y `pasarDeIntsABits` cuyo código debidamente comentado se incluye tanto en los ficheros `CS_hibrido_1.cpp`, `CS_hibrido_2.cpp` y `CS_mejorado.cpp`.

La primera inclusión viene motivada porque, una vez que el algoritmo empieza a tener soluciones notables en el paso de generaciones, es difícil encontrar buenas soluciones al realizar el salto provocado por el vuelo Lévy, por lo que, si incluimos una explotación de la solución obtenida justo después del cambio brusco provocado por el vuelo Lévy, tendremos mayor posibilidad de explorar el espacio de búsqueda con éxito.

Esto justamente es lo que hemos hecho:

Algorithm 4: Pseudocódigo Cuckoo Search con búsqueda local. 1

```
Data: m = matriz de distancias
n = número de elementos a seleccionar para la solución
MAX_EVALUACIONES = número máximo de evaluaciones
n_nidos = número de individuos de la población
p_a = probabilidad de abandonar el nido (eliminar soluciones)
max_evaluaciones_bl = número máximo de evaluaciones de la búsqueda
local
Result: mejor solución encontrada
begin
    evaluaciones ← 0
    Generamos una población inicial de n_nidos nidos huéspedes x_i
     (i = 1, 2, ..., n_nidos)
    while (evaluaciones < MAX_EVALUACIONES) do
        cuco ← obtenerCuckooPorLevy(m, n)
        solucion_mejorada ← busquedaLocal(m, cuco,
         max_evaluaciones_bl)
        Evaluamos la calidad de solucion_mejorada (F_i)
        Actualizamos evaluaciones.
        Elegimos un nido entre n (digamos, j) al azar.
        if F_i > F_j then
            reemplazar j por la nueva solución
        end
        Una fracción (p_a) de los peores nidos son abandonados y se
         construyen otros nuevos.
        Mantenemos la mejor solución (o los nidos con mejor soluciones)
        Ordenamos las soluciones y encontramos el mejor actual.
    end
    Devolvemos la mejor solución
end
```

Como vemos hemos incluido una llamada a la función `busquedaLocal` justo después de realizar el vuelo Lévy. Hemos usado la `busquedaLocal` implementada en la Práctica 1 y adaptada en la Práctica 2 a los algoritmos genéticos. Puede encontrar el código en el fichero `CS_hibrido_1.cpp`.

Anotamos que, tras numerosas pruebas y apoyándonos en la documentación del creador [1], hemos inicializado los parámetros como $p_a = 0{,}25$, $n\_nidos = 50$, $max\_evaluaciones\_bl = 1000$ y fijamos $MAX\_EVALUACIONES = 100000$ para todos los experimentos.

La segunda versión de algoritmo híbrido (*Cuckoo Search + Local Search*) viene motivada para permitir una mayor explotación de las mejores soluciones obtenidas hasta el momento (*si una solución tiene potencial para ser buena, dedicamos nuestros recursos a explotarla*).

Esto es lo que intentamos conseguir con esta segunda versión del algoritmo que planteamos a continuación:

Algorithm 5: Pseudocódigo Cuckoo Search con búsqueda local. 2

```
Data: m = matriz de distancias
n = número de elementos a seleccionar para la solución
MAX_EVALUACIONES = número máximo de evaluaciones
n_nidos = número de individuos de la población
p_a = probabilidad de abandonar el nido (eliminar soluciones)
max_evaluaciones_bl = número máximo de evaluaciones de la búsqueda
local
Result: mejor solución encontrada
begin
    evaluaciones ← 0
    generaciones ← 0
    Generamos una población inicial de n_nidos nidos huéspedes x_i
     (i = 1, 2, ..., n_nidos)
    while (evaluaciones < MAX_EVALUACIONES) do
        cuco ← obtenerCuckooPorLevy(m, n)
        Evaluamos la calidad de cuco (F_i)
        Actualizamos evaluaciones.
        Elegimos un nido entre n (digamos, j) al azar.
        if F_i > F_j then
            reemplazar j por la nueva solución
        end
        Una fracción (p_a) de los peores nidos son abandonados y se
         construyen otros nuevos.
        Mantenemos la mejor solución (o los nidos con mejor soluciones)
        Ordenamos las soluciones y encontramos el mejor actual.
        if generaciones es múltiplo de 10 then
            busquedaLocal(m, mejor solución obtenida,
             max_evaluaciones_bl)
            Actualizamos evaluaciones y mejor solución.
        end
    end
    Devolvemos la mejor solución
end
```

Vemos que hacemos una búsqueda local de la mejor solución obtenida hasta el momento cada ciertas generaciones. Esto permite una mayor explotación de la solución. Dicha implementación se incluye en el fichero `CS_hibrido_2.cpp`.

De nuevo anotamos que, tras varias pruebas y para poder comparar con las versiones anteriores, hemos inicializado los parámetros como $p_a = 0{,}25$, $n\_nidos = 50$, $max\_evaluaciones\_bl = 2000$ (aumentamos dicho valor puesto que se realiza la bús-

queda local menos veces que en el algoritmo 5), y de nuevo $MAX\_EVALUACIONES = 100000$ para todos los experimentos.

4. Mejoras de calidad de la metaheurística sobre el comportamiento o el diseño

Como hemos comentado, el algoritmo original, tal y como está diseñado e implementado, peca de inclinar la balanza a la explotación en contra de la exploración (ya que el vuelo Lévy, permite explorar el espacio de búsqueda, pero solo será introducida la solución en la población si es relativamente buena, lo que, al evolucionar el algoritmo a fases avanzadas (cuando las soluciones en la población sean notables) dicha exploración no obtiene ningún resultado. Esto lo intentamos paliar en la versión 5 del algoritmo.

Sin embargo, como veremos, aunque mejoramos generalmente los resultados obtenidos para las instancias del problema dado, dichas mejoras no son muy significantes. Podemos apoyarnos en la segunda versión del algoritmo híbrido que implementamos (6) y mezclar dichas mejoras en una sola. Fichero `CS_mejorado.cpp`.

Esta nueva versión aumenta la exploración del espacio de búsqueda del algoritmo original y también la explotación de las mejores soluciones encontradas, reservando evaluaciones a dichos mecanismos, sin desperdiciar gran cantidad de evaluaciones a explotar soluciones no notables.

Algorithm 6: Pseudocódigo Cuckoo Search mejorado

```
Data: m = matriz de distancias
n = número de elementos a seleccionar para la solución
MAX_EVALUACIONES = número máximo de evaluaciones
n_nidos = número de individuos de la población
p_a = probabilidad de abandonar el nido (eliminar soluciones)
max_evaluaciones_bl = número máximo de evaluaciones de la búsqueda
local
Result: mejor solución encontrada
begin
    evaluaciones ← 0
    generaciones ← 0
    Generamos una población inicial de n_nidos nidos huéspedes x_i
     (i = 1, 2, ..., n_nidos)
    while (evaluaciones < MAX_EVALUACIONES) do
        cuco ← obtenerCuckooPorLevy(m, n)
        solucion_mejorada ← busquedaLocal(m, cuco,
         max_evaluaciones_bl)
        Evaluamos la calidad de solucion_mejorada (F_i)
        Actualizamos evaluaciones.
        Elegimos un nido entre n (digamos, j) al azar.
        if F_i > F_j then
            reemplazar j por la nueva solución
        end
        Una fracción (p_a) de los peores nidos son abandonados y se
         construyen otros nuevos.
        Mantenemos la mejor solución (o los nidos con mejor soluciones)
        Ordenamos las soluciones y encontramos el mejor actual.
        if generaciones es múltiplo de 10 then
            busquedaLocal(m, mejor solución obtenida,
             max_evaluaciones_bl)
            Actualizamos evaluaciones y mejor solución.
        end
    end
    Devolvemos la mejor solución
end
```

Los parámetros iniciales de esta versión del algoritmo, tras varias pruebas con la motivación de poder establecer una comparación equitativa entre los 4 algoritmos implementados y los de las anteriores prácticas, los hemos inicializado como $p_a = 0{,}25$, $n\_nidos = 50$, $max\_evaluaciones\_bl = 1200$ y fijamos $MAX\_EVALUACIONES = 100000$ para todos los experimentos.

Como trabajos futuros, este algoritmo, por supuesto, puede extenderse al caso más complicado en el que cada nido tiene múltiples huevos que representan un con-

junto de soluciones.

Además existen numerosas versiones de dicho algoritmo aceptadas por la comunidad científica como las que encontramos en [3], [4] y [5]. Dichas versiones son una motivación para seguir ampliando este proyecto en un futuro.

5. Comparación con otras metaheurísticas

Lo primero que haremos será visualizar los resultados de cada algoritmo:

CS.cpp

CS				
Caso	Coste obtenido	Mejor coste	Desv.	Tiempo (s.)
GKD-c_11_n500_m50	18256,9	19587,12891	6,79	0,73122
GKD-c_12_n500_m50	17783,2	19360,23633	8,15	1,07045
GKD-c_13_n500_m50	17944,1	19366,69922	7,35	1,11225
GKD-c_14_n500_m50	17817,3	19458,56641	8,43	1,1651
GKD-c_15_n500_m50	17839,5	19422,15039	8,15	1,13317
GKD-c_16_n500_m50	18144,4	19680,20898	7,80	1,07754
GKD-c_17_n500_m50	17954,5	19331,38867	7,12	1,0865
GKD-c_18_n500_m50	17862,2	19461,39453	8,22	1,12865
GKD-c_19_n500_m50	17992,8	19477,32813	7,62	1,16104
GKD-c_20_n500_m50	17887,6	19604,84375	8,76	1,2913
MDG-b_1_n500_m50	761623	778030,625	2,11	1,15283
MDG-b_2_n500_m50	767143	779963,6875	1,64	1,08503
MDG-b_3_n500_m50	757504	776768,4375	2,48	1,04985
MDG-b_4_n500_m50	759598	775394,625	2,04	1,03208
MDG-b_5_n500_m50	759084	775611,0625	2,13	1,03881
MDG-b_6_n500_m50	774554	775153,6875	0,08	1,06727
MDG-b_7_n500_m50	764348	777232,875	1,66	1,0812
MDG-b_8_n500_m50	756083	779168,75	2,96	1,06073
MDG-b_9_n500_m50	759730	774802,1875	1,95	1,05984
MDG-b_10_n500_m50	764294	774961,3125	1,38	1,04606
MDG-a_31_n2000_m200	112205	114139	1,69	12,1935
MDG-a_32_n2000_m200	112350	114092	1,53	11,6897
MDG-a_33_n2000_m200	111932	114124	1,92	11,086
MDG-a_34_n2000_m200	111946	114203	1,98	11,6908
MDG-a_35_n2000_m200	112405	114180	1,55	11,0931
MDG-a_36_n2000_m200	112053	114252	1,92	11,3482
MDG-a_37_n2000_m200	112016	114213	1,92	11,7142
MDG-a_38_n2000_m200	112396	114378	1,73	11,2128
MDG-a_39_n2000_m200	111267	114201	2,57	11,103
MDG-a_40_n2000_m200	111645	114191	2,23	11,0679

Media Desv:	3,8621377
Media Tiempo:	4,5276706

CS_hibrido_1.cpp

CS_híbrido_1				
Caso	Coste obtenido	Mejor coste	Desv.	Tiempo (s.)
GKD-c_11_n500_m50	18266,6	19587,12891	6,74	1,37726
GKD-c_12_n500_m50	17760,9	19360,23633	8,26	1,75826
GKD-c_13_n500_m50	17933,2	19366,69922	7,40	1,06473
GKD-c_14_n500_m50	17898,4	19458,56641	8,02	1,90832
GKD-c_15_n500_m50	17841,3	19422,15039	8,14	1,37606
GKD-c_16_n500_m50	18131	19680,20898	7,87	1,58001
GKD-c_17_n500_m50	17960,4	19331,38867	7,09	1,76716
GKD-c_18_n500_m50	17854,8	19461,39453	8,26	1,31259
GKD-c_19_n500_m50	17974,4	19477,32813	7,72	1,34879
GKD-c_20_n500_m50	17873,8	19604,84375	8,83	1,7881
MDG-b_1_n500_m50	764647	778030,625	1,72	1,48243
MDG-b_2_n500_m50	767790	779963,6875	1,56	1,58734
MDG-b_3_n500_m50	768459	776768,4375	1,07	1,85362
MDG-b_4_n500_m50	764735	775394,625	1,37	0,977355
MDG-b_5_n500_m50	761759	775611,0625	1,79	2,01417
MDG-b_6_n500_m50	770131	775153,6875	0,65	1,83322
MDG-b_7_n500_m50	769937	777232,875	0,94	1,35396
MDG-b_8_n500_m50	765102	779168,75	1,81	1,50525
MDG-b_9_n500_m50	768399	774802,1875	0,83	2,07193
MDG-b_10_n500_m50	765972	774961,3125	1,16	2,08581
MDG-a_31_n2000_m200	112636	114139	1,32	85,5038
MDG-a_32_n2000_m200	112405	114092	1,48	78,1355
MDG-a_33_n2000_m200	112443	114124	1,47	89,4255
MDG-a_34_n2000_m200	112640	114203	1,37	88,5322
MDG-a_35_n2000_m200	112515	114180	1,46	82,6233
MDG-a_36_n2000_m200	112626	114252	1,42	81,5998
MDG-a_37_n2000_m200	112613	114213	1,40	88,8859
MDG-a_38_n2000_m200	112522	114378	1,62	89,2865
MDG-a_39_n2000_m200	112371	114201	1,60	86,4253
MDG-a_40_n2000_m200	112740	114191	1,27	86,3084

Media Desv:	3,5210699
Media Tiempo:	29,6257521

CS_hibrido_2.cpp

CS_híbrido_2				
Caso	Coste obtenido	Mejor coste	Desv.	Tiempo (s.)
GKD-c_11_n500_m50	18280	19587,12891	6,67	0,839124
GKD-c_12_n500_m50	17784	19360,23633	8,14	1,28252
GKD-c_13_n500_m50	17924,7	19366,69922	7,45	1,28391
GKD-c_14_n500_m50	17863,4	19458,56641	8,20	1,31129
GKD-c_15_n500_m50	17876,6	19422,15039	7,96	1,28128
GKD-c_16_n500_m50	18161,5	19680,20898	7,72	1,26381
GKD-c_17_n500_m50	18000,7	19331,38867	6,88	1,21691
GKD-c_18_n500_m50	17868,3	19461,39453	8,19	1,197
GKD-c_19_n500_m50	17994	19477,32813	7,62	1,23333
GKD-c_20_n500_m50	17867,4	19604,84375	8,86	1,24279
MDG-b_1_n500_m50	764816	778030,625	1,70	1,25738
MDG-b_2_n500_m50	773707	779963,6875	0,80	1,23776
MDG-b_3_n500_m50	761368	776768,4375	1,98	1,24731
MDG-b_4_n500_m50	764142	775394,625	1,45	1,21763
MDG-b_5_n500_m50	763173	775611,0625	1,60	1,23778
MDG-b_6_n500_m50	768864	775153,6875	0,81	1,22457
MDG-b_7_n500_m50	765905	777232,875	1,46	1,26692
MDG-b_8_n500_m50	766610	779168,75	1,61	1,18344
MDG-b_9_n500_m50	763185	774802,1875	1,50	1,256
MDG-b_10_n500_m50	766138	774961,3125	1,14	1,19224
MDG-a_31_n2000_m200	113165	114139	0,85	14,7048
MDG-a_32_n2000_m200	113034	114092	0,93	13,9989
MDG-a_33_n2000_m200	113053	114124	0,94	13,4519
MDG-a_34_n2000_m200	112852	114203	1,18	13,501
MDG-a_35_n2000_m200	113289	114180	0,78	15,4844
MDG-a_36_n2000_m200	113228	114252	0,90	14,2811
MDG-a_37_n2000_m200	113289	114213	0,81	14,5236
MDG-a_38_n2000_m200	113077	114378	1,14	14,6166
MDG-a_39_n2000_m200	112944	114201	1,10	16,0288
MDG-a_40_n2000_m200	113181	114191	0,88	15,1774

Media Desv:	3,3749243
Media Tiempo:	5,6747164

CS_mejorado.cpp

CS_mejorado				
Caso	Coste obtenido	Mejor coste	Desv.	Tiempo (s.)
GKD-c_11_n500_m50	18280	19587,12891	6,67	1,98094
GKD-c_12_n500_m50	17777,5	19360,23633	8,18	1,95814
GKD-c_13_n500_m50	17944,8	19366,69922	7,34	1,80788
GKD-c_14_n500_m50	17908,8	19458,56641	7,96	1,98339
GKD-c_15_n500_m50	17859,4	19422,15039	8,05	1,83683
GKD-c_16_n500_m50	18172,4	19680,20898	7,66	1,83767
GKD-c_17_n500_m50	17984,4	19331,38867	6,97	1,78728
GKD-c_18_n500_m50	17865,5	19461,39453	8,20	1,88296
GKD-c_19_n500_m50	17977,8	19477,32813	7,70	1,92882
GKD-c_20_n500_m50	17885,6	19604,84375	8,77	1,69991
MDG-b_1_n500_m50	772475	778030,625	0,71	2,09939
MDG-b_2_n500_m50	769918	779963,6875	1,29	2,02014
MDG-b_3_n500_m50	773384	776768,4375	0,44	1,94241
MDG-b_4_n500_m50	773111	775394,625	0,29	1,94494
MDG-b_5_n500_m50	767456	775611,0625	1,05	2,10756
MDG-b_6_n500_m50	774099	775153,6875	0,14	2,08444
MDG-b_7_n500_m50	772198	777232,875	0,65	2,10983
MDG-b_8_n500_m50	767187	779168,75	1,54	1,94822
MDG-b_9_n500_m50	766078	774802,1875	1,13	1,98063
MDG-b_10_n500_m50	772544	774961,3125	0,31	1,93887
MDG-a_31_n2000_m200	112786	114139	1,19	74,424
MDG-a_32_n2000_m200	112994	114092	0,96	75,2537
MDG-a_33_n2000_m200	112803	114124	1,16	73,7526
MDG-a_34_n2000_m200	113000	114203	1,05	75,7477
MDG-a_35_n2000_m200	112835	114180	1,18	80,192
MDG-a_36_n2000_m200	113086	114252	1,02	76,1775
MDG-a_37_n2000_m200	113135	114213	0,94	75,5421
MDG-a_38_n2000_m200	112932	114378	1,26	74,9742
MDG-a_39_n2000_m200	112926	114201	1,12	77,6755
MDG-a_40_n2000_m200	113061	114191	0,99	75,8428

Media Desv:	3,1971277
Media Tiempo:	26,615411

Podemos resumir la información anterior en tres tablas, para hacerlo más visual, una por cada problema:

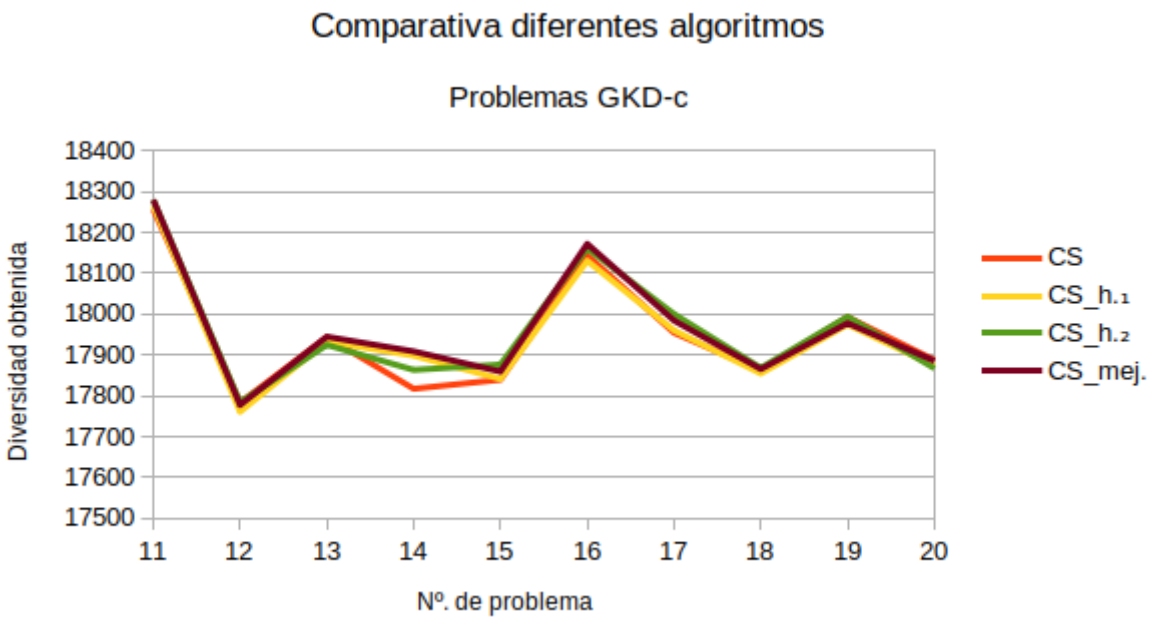


Imagen 2: Comparativa entre los 4 algoritmos para los problemas GFD-c

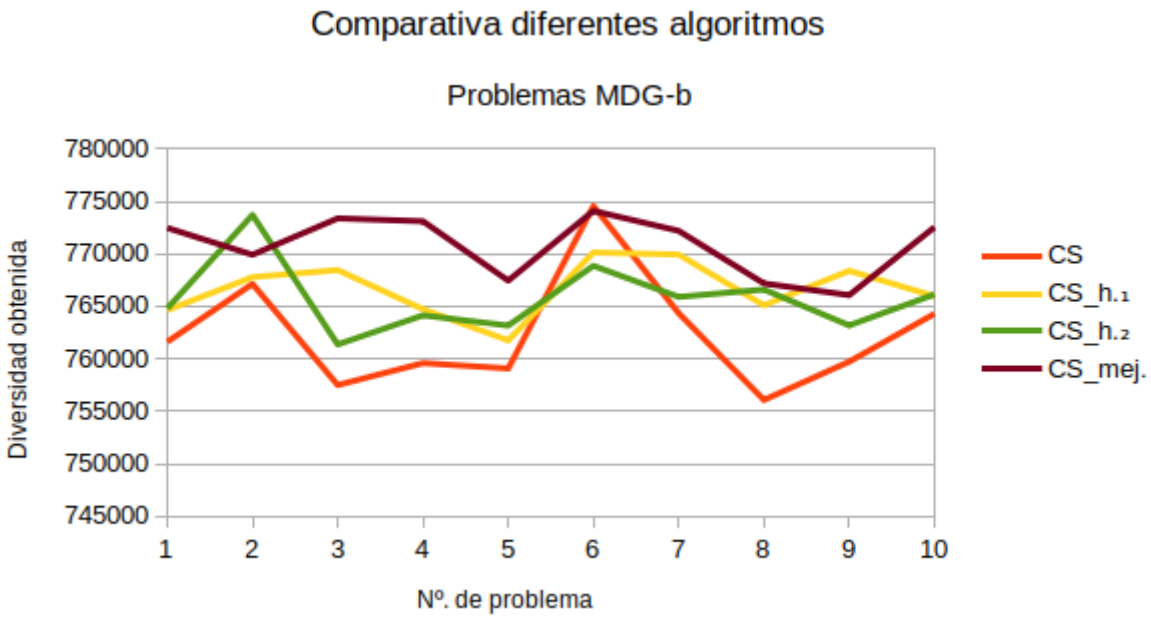


Imagen 3: Comparativa entre los 4 algoritmos para los problemas MDG-b

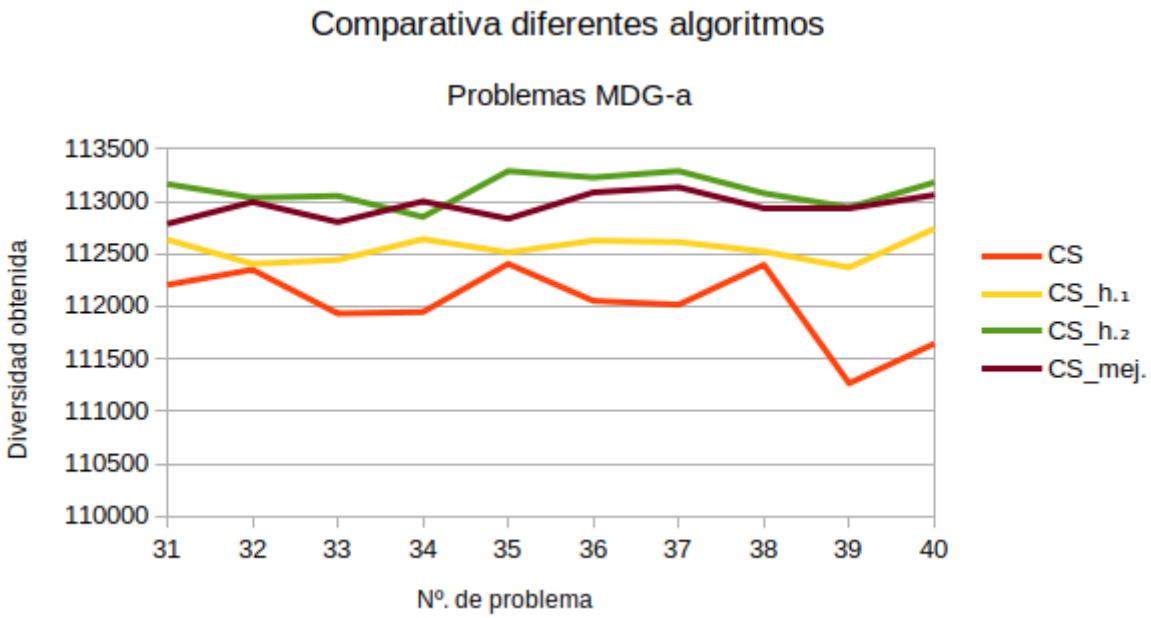


Imagen 4: Comparativa entre los 4 algoritmos para los problemas MDG-a

Por otra parte, resumiendo la información anterior usando la media de los resultados de las treinta instancias de los problemas estudiados y añadiendo la información de las otras prácticas, tenemos que:

Algoritmo	***Desv.***	***Tiempo (s.)***
Greedy	4,44	0,17
BL	3,47	11,39
AGGu	4,6	98,9
AGGp	6,4	3,6
AGEu	3,9	130,6
AGEp	4,2	12,3
AM0	3,8	62,8
AM1	4,6	52,7
AM2	4,2	69,8
ES	4,24	0,13
BMB	3,24	4,35
ILS	3,25	2,10
ILS-ES	5,78	0,32
CS	3,86	4,52
CS_hibrido_1	3,52	29,62
CS_hibrido_2	3,37	5,67
CS_mej.	3,19	26,61

Hay varias cosas importantes que comentar: lo primero es que la adaptación del algoritmo original implementado es bastante notable (los resultados son óptimos ob-

teniendo una desviación media menor a 4, lo que supera a muchos de los algoritmos implementados en las prácticas); lo segundo a comentar que las mejoras incorporadas al algoritmos provocan los resultados esperados, llegando incluso a obtener la menor desviación media en el conjunto de algoritmos implementados en las prácticas (incluso se alcanza casi el óptimo (0.14 de desviación con respecto el óptimo) en alguna instancia del problema).

De esta forma podemos concluir que los pasos de creación de mejoras han sido correctos y la adaptación del algoritmo a nuestro problema ha sido correcta también, obteniendo resultados bastante notables.

Las explicaciones y motivaciones de cada uno de los cuatro tipos se han ido desarrollando a lo largo de la memoria, por lo que no vemos necesario incluir en esta sección, salvo la afirmación de que las suposiciones expuestas han sido confirmadas y aceptadas viendo los resultados. Hemos conseguido una exploración notable del espacio de búsqueda y una explotación bastante buena de las zonas donde los máximos locales tenían una mayor calidad.

Con respecto al tiempo de ejecución de los algoritmos, vemos que las inclusión de la búsqueda local involucra una subida sustancial del tiempo de ejecución del algoritmo. Una posible motivación como trabajo futuro puede ser optimizar dicha búsqueda local. Sin embargo vemos que a pesar de que nuestra metaheurística se encuentre encuadrada dentro de la familia de los algoritmos genéticos, los tiempos de ejecución no son los más altos con respecto al conjuntos de algoritmos genéticos previamente implementados (como AGE_u, AGG_u o los algoritmos meméticos de la práctica 1, por lo que también consideramos positivo dicho aspecto.

Por último y para acabar incluimos las tres gráficas que visualizan los resultados de todas las instancias ejecutadas por todos las metaheurísticas implementadas:

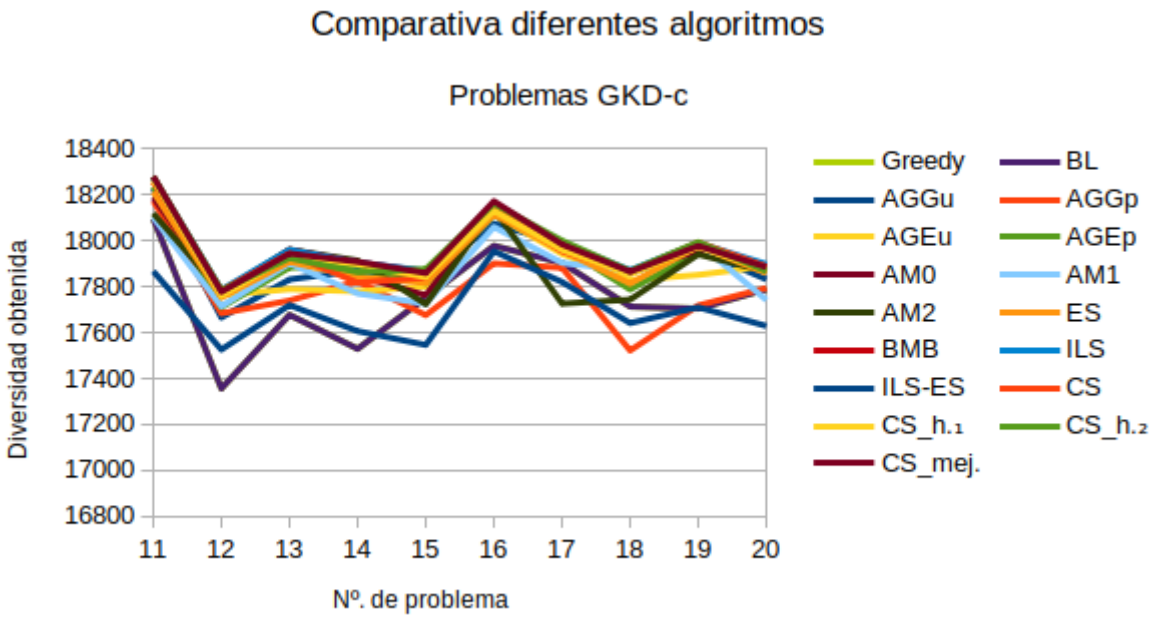


Imagen 5: Comparativa entre todos los algoritmos para los problemas GFD-c

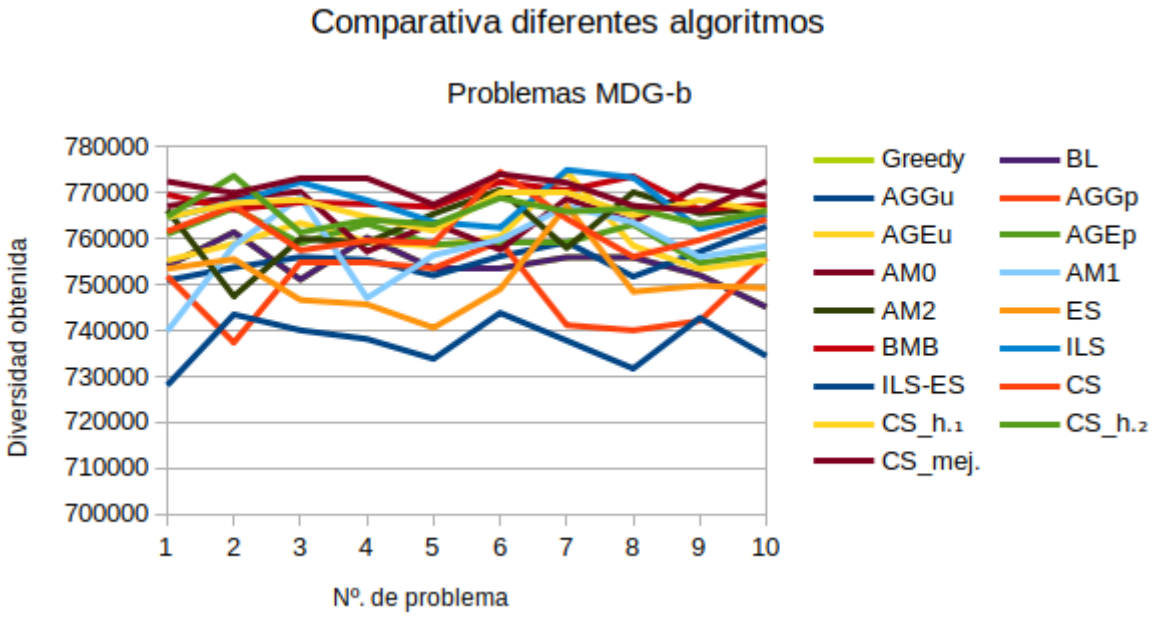


Imagen 6: Comparativa entre todos los algoritmos para los problemas MDG-b

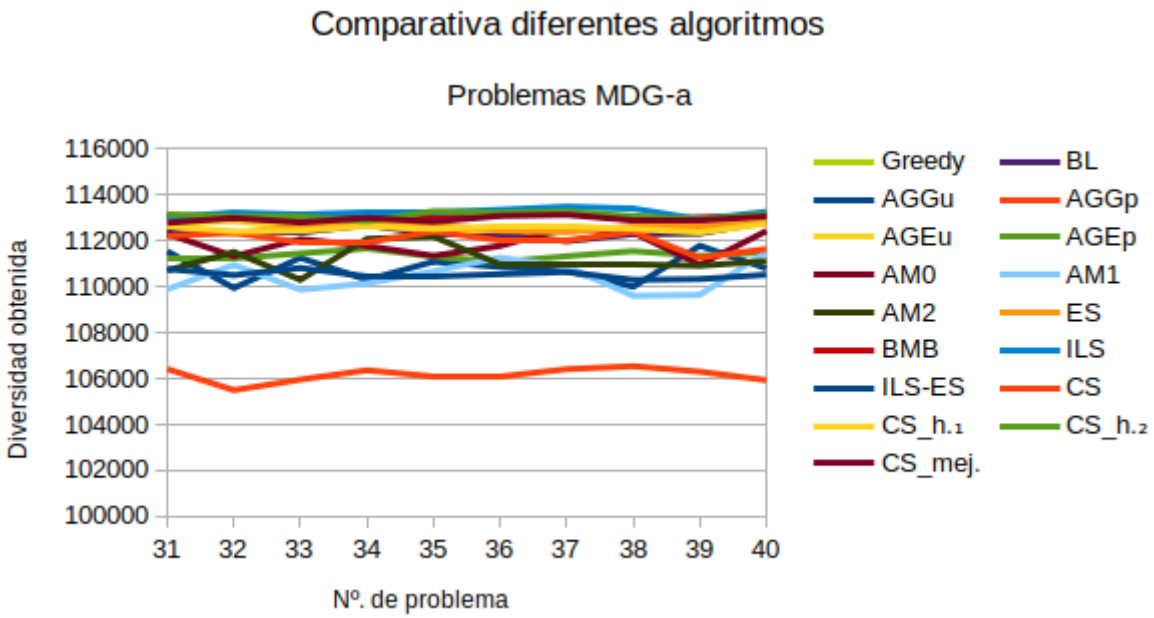


Imagen 7: Comparativa entre todos los algoritmos para los problemas MDG-a

Referencias

[1] Xin-She Yang & Suash Deb, *Cuckoo Search via Lévy Flights* Proc. of World Congress on Nature & Biologically Inspired Computing (NaBIC 2009), December 2009, India. IEEE Publications, USA, **pp. 210-214** (2009)
https://arxiv.org/pdf/1003.1594.pdf

[2] Mandelbrot, Benoit B., *The Fractal Geometry of Nature (Updated and augm. ed.). New York: W. H. Freeman. (2009)*
ISBN 0-7167-1186-9. OCLC 7876824.

[3] Ramin Rajabioun, *Cuckoo Optimization Algorithm* Applied Soft Computing 11 **5508-5518** (2011)
https://doi.org/10.1016/j.asoc.2011.05.008

[4] Hojjat Rakhshani & Amin Rahati, *Snap-drift cuckoo search: A novel cuckoo search optimization algorithm* Applied Soft Computing 52 **771-794** (2017)
https://doi.org/10.1016/j.asoc.2016.09.048

[5] S. Walton, O. Hassan, K. Morgan & M.R. Brown, *Modified cuckoo search: A new gradient free optimisation algorithm* Chaos, Solitons & Fractals Nonlinear Science, and Nonequilibrium and Complex Phenomena 44 **710-718** (2011)
https://doi.org/10.1016/j.chaos.2011.06.004