Práctica 3

Sistema de Colonias de Hormigas

Autores:
David Díaz Jiménez 77356084T
Andrés Rojas Ortega 77382127F

Grupo 2

Metaheurísticas

Informe de prácticas

David Díaz Jiménez, Andrés Rojas Ortega

Contents

1 Definición y análisis del problema

Dado un conjunto N de tamaño n, se pide encontrar un subconjunto M de tamaño m, que maximice la función:

$$C(M) = \sum_{s_i, s_j \in M} d_{ij}$$

donde d_{ij} es la diversidad del elemento s_i respecto al elemento s_j

1.1 Representación de la solución

Para representar la solución se ha optado por el uso de un vector de enteros, en el que el elemento contenido en cada posición se corresponde con un integrante de la solución. La solución vendrá dada por las siguientes restricciones:

- La solución no puede contener elementos repetidos.
- Debe tener exactamente m elementos.
- El orden de los elementos es irrelevante.

1.2 Función objetivo

$$C(M) = \sum_{s_i, s_j \in M} d_{ij}$$

1.3 Operadores comunes

El operador de intercambio es el 1-opt, se seleccionara un elemento de la solución actual en base a un criterio y se sustituirá por un elemento que no pertenece a la solución.

2 Clases auxiliares

A continuación se enumeran las diferentes clases auxiliares utilizadas en el programa acompañadas de una breve descripción de las mismas.

nota: Para obtener información detallada se deben consultar los comentarios insertados en el código de cada una de las clases.

2.1 Archivo

Esta clase se encarga de almacenar toda la información que se encuentra dentro de cada archivo que contiene cada uno de los problemas.

2.2 Configurador

Utilizamos esta clase para leer y almacenar los parámetros del programa que se encuentran dentro del archivo de configuración.

2.3 Greedy

Esta clase implementa la funcionalidad del algoritmo Greedy, se utiliza para calcular el coste Greedy de cada problema. Este coste es necesario como parámetro para el Sistema de Colonias de Hormigas.

2.4 Hormiga

Esta clase almacena toda la información necesaria de cada hormiga de la colonia.

2.5 GestorLog

La función principal de esta clase es la administración de los archivos Log del programa y el almacenamiento de información para debug en los mismos.

2.6 Metaheuristicas

Esta clase se utiliza para lanzar la ejecución de los algoritmos para cada problema facilitado como parámetro.

2.7 Pair

Representa un par formado por un candidato y un coste asociado a este.

2.8 RandomP

Clase para generar números aleatorios.

2.9 Timer

Clase para gestionar los tiempos de ejecución del algoritmo.

3 Pseudocódigo

3.1 algoritmo principal

Algorithm 1 Sistema Colonia Hormigas

```
matrizFeromonas ← InicializarFeromona()
while iteracionesRealizadas<= limiteIteraciones ∧ tiempoEjecucion <= limiteTiempoEjecucion do
    colonia ← InicializarColonia()
    for i=1; i<tamñoHormiga; i++ do
        for hormiga ∈ colonia do
            lrc ← GenerarLRC(hormiga)
            AplicarReglaTransicion(lrc, hormiga)
        end for
        ActualizarFeromonaLocal()
    end for
    TareasDemonio()
    ∅ ← colonia
    iteracionesRealizadas ← iteracionesRealizadas + 1
end while
```

Lo primero que realiza el algoritmo es inicializar la matriz de feromonas con el valor inicial. Esta tarea la realiza la función "InicializarFeromona()", la matriz de feromonas se almacena en la variable "matrizFeromonas". El valor inicial de feromona se pasa como parámetro del programa.

A continuación, entramos en un bucle while cuyas condiciones de parada son: las iteraciones realizadas y el tiempo de ejecución del algoritmo. En el caso de las iteraciones realizadas, almacenadas en la variable "iteracionesRealizadas", deben ser inferiores al límite almacenado en la variable "limiteIteraciones". El valor de "limiteIteraciones" se pasa como parámetro del programa. En el caso del tiempo de ejecución, que se almacena en la variable "tiempoEjecucion", debe de ser inferior al valor almacenado en la variable "limiteTiempoEjecucion". El valor de "limiteTiempoEjecucion" se pasa como parámetro del programa.

Cada vez que empieza una nueva iteración del bucle, lo primero que se realiza es inicializar la colonia de hormigas. La colonia se almacena en la variable "colonia" y la función encargada de inicializarla es "InicializarColonia()".

Una vez inicializada la colonia, iniciamos dos bucles for. El primero se ejecutará hasta que el valor de "i" alcance el número de elementos que debe contener cualquier solución (parámetro del programa, almacenado en la variable "tamañoHormiga"). El segundo bucle for recorrerá cada una de las hormigas pertenecientes a la variable "colonia".

Lo primero que realizamos es almacenar en la variable "lrc" la lista de candidatos restringida de la hormiga, calculada por la función "GenerarLRC(hormiga)".

Una vez almacenada la lista restringida, seleccionamos uno de los candidatos que contiene aplicando la regla de la transición y lo añadimos como parte de la solución de la hormiga. La función que realiza dicha tarea es "AplicarReglaTransicion(lrc,hormiga)".

Se finaliza el bucle for interior y la función "ActualizarFeromonaLocal()" se encarga de actualizar la feromona local de todas las hormigas después de haber añadido un nuevo elemento.

Una vez finalice el bucle for tendremos todas las hormigas completas, por lo que pasamos a seleccionar la mejor hormiga de la colonia y realizar la actualización de la feromona global. La función encargada de dichas tareas es "TareasDemonio()".

Para finalizar el bucle while, vaciamos la variable "colonia" y actualizamos el valor de la variable "iteracionesRealizadas".

3.2 Inicialización de la matriz de feronoma

Salida Devuelve la matriz de feromonas inicializada.

Algorithm 2 InicializarFeromona()

```
for i=0; i<tamañoMatriz; i++ do
  for j=0; i<tamañoMatriz; j++ do
    matrizFeromonas[i][j] ← feromonaInicial
  end for
end for
return matrizFeromonas
```

El funcionamiento de esta función es trivial: se inicializan todos los elementos de la matriz de feromonas con el valor de "feromonaInicial" (parámetro del programa).

3.3 Inicialización de la colonia de hormigas

Salida Devuelve la colonia de hormigas inicializada.

Algorithm 3 InicializarColonia()

```
while colonia.tamaño()<tamañoColonia do
  hormiga ← ∅
  primerElemento ← GenerarEnteroAleatorio()
  hormiga ← hormiga ∪ {primerElemento}
  colonia ← colonia ∪ {hormiga}
end while
return colonia
```

El funcionamiento de esta función es trivial: se inicializan todas las hormigas con un elemento generado aleatoriamente con la función "GenerarEnteroAleatorio()". Cuando se tengan "tamañoColonia" (parámetro del programa) hormigas inicializadas, se devuelve la colonia como resultado.

3.4 Generación de la Lista Restringida de Candidatos

Entrada La hormiga para la que se quiere obtener su Lista Restringida de Candidatos.

Salida Devuelve la Lista Restringida de Candidatos "lrc".

Algorithm 4 GenerarLRC(hormiga)

```
lrc ← ∅
noSeleccionados ← ∅
min ← +∞
max ← −∞
for i=0;i<tamañoMatriz;i++ do
  if i ∉ hormiga then
    aporte ← 0
    for elemento ∈ hormiga do
      aporte ← aporte + MatrizCostes[elemento][i]
    end for
    if min == +∞ then
      max ← aporte
      min ← aporte
    end if
    if aporte>max then
      max ← aporte
    else if aporte<min then
      min ← aporte
    end if
    noSeleccionados ← noSeleccionados ∪ {i, aporte}
  end if
end for
valorCorte ← min + delta∗(max - min)
for elemento ∈ noSeleccionados do
  if elemento.aporte>=valorCorte then
    lrc ← lrc ∪ {elemento}
  end if
end for
return lrc
```

Lo primero que se realiza es inicializar los valores de las variables "lrc", "noSeleccionados", "min" y "max"."lrc" almacena la Lista Restringida de Candidatos, "noSeleccionados" almacena todos los elementos del problema que no forman parte de la hormiga, "min" y "max" almacena el mínimo y máximo aporte encontrado hasta el momento.

Se recorren todos los elementos del problema y se comprueba que no se encuentren ya en la hormiga.

En el caso de que no se encuentren en la hormiga, se actualiza el valor de la variable "aporte" a cero y se calcula el coste que aportaría a la hormiga si lo incluyéramos en ella. Este coste se almacena en la variable "aporte".

A continuación, se comprueba que aporte no sea mayor que "max" o menor que "min". En el caso de que se cumpla una de estas dos condiciones, se actualiza el valor de "max" o de "min" dependiendo del caso.

Para finalizar, se añade el elemento junto su aporte a la variable "noSeleccionados".

Se realiza lo anterior hasta que se haya recorrido todos los elementos del problema.

Lo siguiente que se realiza es el cálculo de "valorCorte" aplicando la siguiente fórmula, siendo "delta" un parámetro del programa:

$$valorCorte = min + delta * (max - min)$$

Para cada elemento de "noSeleccionados", se comprueba que su aporte sea mayor que "valorCorte". Si se da este caso, se añade el elemento a la Lista Restringida de Candidatos. Cuando se haya hecho la comprobación de todos los elementos "lrc" habrá sido generada.

Una vez se haya realizado esto último, se devuelve la Lista Restringida de Candidatos generada como resultado de la ejecución de la función.

3.5 Aplicacion de la regla de transición

Entradas La Lista Restringida de Candidatos "lrc", y la hormiga a la que queremos aplicar la regla de la transición.

Algorithm 5 AplicarReglaTransicion(lrc, hormiga)

$probabilidadesTransicion \leftarrow \emptyset$
$sumatoria \leftarrow 0$
$q \leftarrow GenerarFloatAleatorio()$
if q0<=q **then**
  $elemento \leftarrow 0$
  $mayorValor \leftarrow -\infty$
  for $elementoLrc \in lrc$ **do**
    $sumatoria \leftarrow 0$
    for $elementoHormiga \in hormiga$ **do**
      $sumatoria \leftarrow sumatoria + (matrizFeromonas[elementoHormiga][elementoLrc]^{alfa} * matrizCostes[elementoHormiga][elementoLrc]^{beta})$
    end for
    if mayorValor<=sumatoria **then**
      $mayorValor \leftarrow sumatoria$
      $elemento \leftarrow elementoLrc$
    end if
  end for
  $hormiga \leftarrow hormiga \cup \{elemento\}$
else
  for $elementoLrc \in lrc$ **do**
    $valorSuperior \leftarrow 0$
    for $elementoHormiga \in hormiga$ **do**
      $valorSuperior \leftarrow valorSuperior + (matrizFeromonas[elementoHormiga][elementoLrc]^{alfa} * matrizCostes[elementoHormiga][elementoLrc]^{beta})$
    end for
    $sumatoria \leftarrow sumatoria + valorSuperior$
    $probabilidadesTransicion \leftarrow probabilidadesTransicion \cup \{valorSuperior\}$
  end for
  $indice \leftarrow 0$
  for $valor \in probabilidadesTransicion$ **do**
    $probabilidad \leftarrow (valor/sumatoria)$
    $probabilidadesTransicion[indice] \leftarrow probabilidad$
    $indice \leftarrow indice + 1$
  end for
  $uniforme \leftarrow GenerarFloatAleatorio()$
  $indice \leftarrow 0$
  $probabilidadAcumulada \leftarrow 0$
  for $probabilidad \in probabilidadesTransicion$ **do**
    $probabilidadAcumulada \leftarrow probabilidadAcumulada + probabilidad$
    if uniforme<= probabilidadAcumulada **then**
      $hormiga \leftarrow hormiga \cup \{lrc[indice]\}$
    end if
    $indice \leftarrow indice + 1$
  end for
  $lrc \leftarrow \emptyset$
end if

Esta función selecciona uno de los candidatos de la Lista Restringida de Candidatos y la añade a la hormiga aplicando la siguiente regla:

$$p_k(r,s) = \begin{cases} \text{si } q0 \leq q & \arg\, max_{u \in J_k(r)}\{(feromona_{ru})^{alfa} * (heuristica_{ru})^{beta}\} \\ \text{en otro caso} & p'_k(r,s) \end{cases} \tag{1}$$

$$p'_k(r,s) = \begin{cases} \text{si } s \in J_k(r) & \frac{(feromona_{rs})^{alfa} * (heuristica_{rs})^{beta}}{\sum_{u \in J_k(r)} (feromona_{ru})^{alfa} * (heuristica_{ru})^{beta}} \\ \text{en otro caso} & 0 \end{cases} \tag{2}$$

Lo primero que se realiza en la función es inicializar las variables "probabilidadesTransicion", "sumatoria" y "q". "probabilidadesTransicion" almacenará la distribución de probabilidades de todos los candidatos y su valor inicial será el conjunto vacío. "sumatoria" almacenará la suma de las dos ecuaciones anteriores y su valor inicial será 0. "q" almacenará un aleatorio generado por la función "GeneraFloatAleatorio()".

Después de esto se comprueba si "q0" es menor o igual que "q". Si se da este caso se realiza lo siguiente.

Se inicializa "elemento" a 0 y "mayorValor" a menos infinito. "elemento" se encargará de almacenar el elemento de "lrc" seleccionado. "mayorValor" almacenará el mayo resultado calculado hasta el momento.

Para cada elemento perteneciente a la Lista Restringida de Candidatos se calcula la fórmula de la transición respecto a cada elemento almacenado en la hormiga, y se adiciona cada resultado a la variable "sumatoria".

Una vez obtenido el valor de "sumatoria", se comprueba si mejora "mayorValor". Si se da este caso, se actualiza "mayorValoe" con "sumatoria" y se guarda el elemento de "lrc" en la variable "elemento".

Cuando se haya recorrido toda la Lista Restringida de Candidatos se añade a la hormiga el mejor elemento de "lrc", almacenado en "elemento".

En el caso de que "q" sea menor que "q0" se realiza los pasos que se describen a continuación.

Se inicia un bucle for que recorrerá todos los elementos de la Lista Restringida de Candidatos.

Dentro de este bucle se inicializa la variable "valorSuperior" a 0, esta variable almacenará el numerador de la fórmula.

A continuación se recorren todos los elementos de la hormiga y se realiza el cálculo de la fórmula del numerador y se adiciona a "valorSuperior".

Una vez se hayan hecho todos los cálculos con todos los elementos de la hormiga, se adiciona a "sumatoria" la variable "valorSuperior" y se añade a "probabilidadesTransicion" esta última.

Una vez terminado el bucle for, se inicializa la variable "indice" a 0, que hará las funciones de iterador.

Se inicia un nuevo bucle for que recorrerá todos los valores almacenado en la variable "probabilidadesTransición".

Dentro de este bucle, se calcula la probabilidad de cada elemento de "lrc" con los valores almacenados en "probabilidadesTransicion" y "sumatoria", y se almacenan en "probabilidadesTransicion". Una vez realizado esto, "probabilidadesTransicion" habrá pasado de almacenar los numeradores de la formula de transición de cada elemento de "lrc" a almacenar las probabilidades calculadas para cada elemento de "lrc".

Al terminar de recorrer todos los elementos de "probabilidadesTransicion", se genera aleatoriamente un número decimal entre 0 y 1 y se almacena en la variable "uniforme". Se inicializan las variables "indice" y "probabilidadAcumulada" a 0. "indice" sigue desempeñando la misma función y "probabilidadAcumulada" almacena la sumatoria de todos los valores de "probabilidadesTransicion" que se hayan recorrido hasta el momento.

Para terminar se inicializa un último bucle for en el que se comprobará a qué elemento de la Lista Restringida de Candidatos corresponde el valor de "uniforme" generado haciendo uso de la distribución de probabilidades almacenada en "probabilidadesTransicion". Cuando sepamos el elemento, se añade a la hormiga como parte de la solución y se termina la ejecución de la función.

3.6 Actualización de la feromona local

Algorithm 6 ActualizarFeromonaLocal()

```
for hormiga ∈ colonia do
  elementoB ← hormiga.UltimoElmento()
  for i=0;i<=hormiga.tamaño()-1;i++ do
    elementoA ← hormiga[i]
    valorAnterior ← matrizFeromonas[elementoA][elementoB]
    matrizFeromonas[elementoA][elementoB] ← (1 − rho) ∗
    valorAnterior + rho ∗ (costeGreedy)
    matrizFeromonas[elementoB][elementoA] ←
    matrizFeromonas[elementoA][elementoB]
  end for
end for
```

Se recorren todas las hormigas de la colonia y se realizan las operaciones que se describen a continuación.

Se guarda en la variable "elementoB" el último elemento introducido en la hormiga.

A continuación, se inicia un nuevo bucle for que recorre todos los elementos pertenecientes a la hormiga.

Dentro de este bucle, se guarda en la variable "elementoA" el elemento 'i' de la hormiga y el valor de la feromona del arco "elementoA"-"elementoB" en la variable "valorAnterior".

Se actualiza el valor de la feromona del arco "elementoA"-"elementoB", almacenada en "matrizFeromonas" con el resultado de la siguiente función:

$$Feromona(elementoA, elementoB) = (1-\phi) * valorAnterior + \phi * costeGreedy$$

Una vez se haya actualizado la feromona de todos los arcos nuevos de todas las hormigas, se tendrá "matrizFeromonas" actualizada localmente como resultado de la ejecución de la función.

3.7 Tareas demonio

Algorithm 7 TareasDemonio()

```
mejorHormiga ← EvaluarMejorHormiga()
ActualizarFeromonaGlobal(mejorHormiga())
```

El funcionamiento de esta función es trivial: se obtiene la mejor hormiga haciendo uso de la función "EvaluarMejorHormiga()" y se actualiza la feromona global.

3.8 Obtener la mejor hormiga de la colonia

Salida Se devuelve la mejor hormiga de la colonia.

Algorithm 8 EvaluarMejorHormiga()

```
mejorHormiga ← 0
valorMejor ← 0
for hormiga ∈ colonia do
  coste ← CalcularCoste(hormiga)
  if coste>= valorMejor then
    mejorHormiga ← hormiga
    valorMejor ← coste
  end if
end for
return mejorHormiga
```

Se inicializan las variables "mejorHormiga" y "valorMejor" a 0. "mejorHormiga" almacenará la hormiga con mejor coste de la colonia. "valorMejor" almacenará el coste de la mejor hormiga encontrada.

Se recorren todas las hormigas pertenecientes a la colonia, se calcula el coste de cada hormiga con "CalcularCoste(hormiga)" y, si se mejora "valorMejor", se guarda la hormiga y su coste en las variables destinadas a tal efecto.

Para finalizar, cuando se haya ejecutado el bucle for, se devuelve "mejorHormiga" como resultado de la ejecución.

3.9 Actualización de la feromona global

Entrada La mejor hormiga de la colonia.

Algorithm 9 ActualizarFeromonaGlobal(mejorHormiga)

```
for i=0;i<tamañoMatriz;i++ do
  for j=0;j<tamañoMatriz;j++ do
    valorAnterior ← matrizFeromonas[i][j]
    matrizFeromonas[i][j] ← (1 − phi) ∗ valorAnterior
    matrizFeromonas[j][i] ← matrizFeromonas[i][j]
  end for
end for
coste ← CalcularCoste(mejorHormiga)
for i=0;i<mejorHormiga.Tamaño()-1;i++ do
  for j=i+1;j<mejorHormiga.Tamaño();j++ do
    matrizFeromonas[i][j] ← matrizFeromonas[i][j] + phi ∗ coste
    matrizFeromonas[j][i] ← matrizFeromonas[i][j]
  end for
end for
```

En los dos primeros bucles for se recorre toda la matriz de feromonas evaporando el valor que contenía cada celda por el resultante de la siguiente función:

$$EvaporacionFeromona(i, j) = (1 - \rho) * valorAnterior$$

Una vez evaporada toda la matriz de feromonas, se procede a adicionar feromona a todos los arcos pertenecientes a la mejor hormiga de la colonia.

Para este propósito se almacena el coste de la hormiga, haciendo uso de "CalcularCoste(mejorHormiga)", en la variable "coste".

A continuación, haciendo uso de un doble bucle for, recorremos todos los arcos de la hormiga en la matriz de feromonas y añadimos la feromona resultante de la siguiente función:

$$AdicionFeromona = \rho * coste$$

Cuando se hayan ejecutado los dos bucles for, tendremos la matriz de feromonas actualziada.

4 Experimentos y análisis de resultados

4.1 Procedimiento de desarrollo de la práctica

Para realizar la práctica, se ha optado por implementar las heurísticas propuestas en el lenguaje de programación Java. El ejecutable que se entrega junto a este documento ha sido compilado bajo Apache NetBeansIDE 12.0.

4.1.1 Equipo de pruebas

Los resultados de las heurísticas han sido obtenidos en el siguiente equipo:

- Host:
- S.O:
- Kernel:
- CPU:
- GPU:
- GPU:
- Memoria RAM :

4.1.2 Manual de usuario

Para ejecutar el software asegúrese de que el archivo .jar proporcionado se ubica en el mismo directorio que la carpeta *archivos*.

Cuando se muestre la GUI, podrá seleccionar la heurística que desee mediante el botón correspondiente. Una vez empiece la ejecución de una heurística no sera posible seleccionar otra hasta que finalice su ejecución. Los resultados finales se mostrarán en el cuadro de texto, a su vez, se generan los log correspondientes a cada archivo y semilla en la carpeta Log.

Figure 1: GUI

4.2 Parámetros de los algoritmos

4.2.1 Sistema de Colonias de Hormigas

Para regular el comportamiento del Sistema de Colonias de Hormigas, se han definido los siguientes parámetros en el archivo de configuración:

- Iteraciones: Número máximo de iteraciones realizadas por el algoritmo principal, valor por defecto: 10.000.
- Número de hormigas: Número de hormigas pertenecientes a cada colonia generada en cada iteración, valor por defecto: 10.
- Q0: Probabilidad de no elegir directamente el candidato que mayor coste aporta a la hormiga en la regla de la transición, valor por defecto: 0,95.
- Beta: Exponente aplicado al coste de los arcos en las fórmulas de la regla de la transición, valores por defecto: 1 y 2.
- Alfa: Exponente aplicado a la feromona de los arcos en las fórmulas de la regla de la transición, valores por defecto: 1 y 2.
- Rho: Constante aplicada en la fórmula de la evaporación global de feromona, valor por defecto: 0,1.
- Phi: Constante aplicada en la fórmula de la evaporación local de feromona, valor por defecto: 0,1.
- Delta: Porcentaje de restricción aplicado sobre la lista de candidatos total de una hormiga, valor por defecto: 0,75.

4.2.2 Semillas

Para la generación de números pseudoaleatorios se utiliza una semilla previamente definida en el archivo de configuración, en este caso es 77356084. Esta semilla se va rotando en las 5 iteraciones de cada archivo.

77356084 $\rightarrow$ 73560847 $\rightarrow$ 35608477 ...

4.3 Análisis de los resultados

4.3.1 SCH con $\alpha = 1$, $\beta = 1$

De la experimentación realizada, obtenemos las siguientes tablas, correspondientes al S.C.H. con $\alpha = 1$ y $\beta = 1$:

SCH $\alpha = 1$, $\beta = 1$						
	GKD-c_1_n500_m50		GKD-c_2_n500_m50		GKD-c_3_n500_m50	
Ejecución	Coste	Tiempo	Coste	Tiempo	Coste	Tiempo
1	19447,05	86283	19677,01	85025	19525,05	85636
2	19460,39	85767	19674,26	85178	19523,76	85188
3	19462,50	87097	19670,57	85011	19519,77	85287
4	19458,63	86364	19661,08	85106	19523,81	85303
5	19462,67	86277	19674,56	85214	19531,01	85232
Media	-0,14%	86357,60	-0,15%	85106,80	-0,12%	85329,20
Devs. típica	0,03%	476,35	0,03%	90,04	0,02%	177,47

Table 1: Resultados GKD

SCH $\alpha = 1$, $\beta = 1$						
	SOM-b_11_n300_m90		SOM-b_12_n300_m120		SOM-b_13_n400_m40	
Ejecución	Coste	Tiempo	Coste	Tiempo	Coste	Tiempo
1	20662	167862	35800	263791	4630	53726
2	20658	167622	35793	264292	4626	53753
3	20645	167410	35780	263622	4646	53726
4	20623	167560	35828	263551	4631	53832
5	20634	167525	35776	263251	4629	54074
Media	-0,48%	167595,80	-0,24%	263701,40	-0,55%	604,40
Devs. típica	0,08%	167,70	0,06%	383,61	0,17%	7,44

Table 2: Resultados SOM

SCH $\alpha = 1$, $\beta = 1$						
	MDG-a_21_n2000_m200		MDG-a_22_n2000_m200		MDG-a_23_n2000_m200	
Ejecución	Coste	Tiempo	Coste	Tiempo	Coste	Tiempo
1	113383,00	600256,00	113153,00	600090,00	113132,00	600032,00
2	113539,00	600065,00	113180,00	600185,00	113110,00	600298,00
3	113368,00	600107,00	113439,00	600184,00	113088,00	600292,00
4	113483,00	600283,00	113232,00	600161,00	113064,00	600116,00
5	113307,00	600149,00	113258,00	600124,00	113085,00	600223,00
Media	-0,74%	600172,00	-0,94%	600178,80	-0,90%	6000192,20
Devs. típica	0,08%	94,31	0,10%	41,14	0,02%	115,73

Table 3: Resultados MDG

Como puede observarse en los resultados, el algoritmo exhibe un comportamiento robusto, manteniendo las desviación típica de los resultados por debajo

del 0,10% en casi todos los archivos estudiados, a excepción de dos. En todo caso la mayor desviación típica alcanzada no sobrepasa el 0,20%. Si evaluamos la calidad de las soluciones obtenidas respecto a costes, el algoritmo es incapaz de encontrar los óptimos globales con los parámetros anteriormente descritos.

En lo referente a tiempos, el algoritmo consigue ejecutar las iteraciones objetivo antes de los 600 segundos para las instancias pequeñas y medianas. No es el caso de las instancias de datos más grandes, dónde el algoritmo finaliza su ejecución en torno las 2000 iteraciones, sin sobrepasar las 3.000 iteraciones en ningún caso,para los archivos MDG.

4.3.2 SCH con $\alpha = 2$, $\beta = 1$

4.3.3 SCH con $\alpha = 1$, $\beta = 2$

De la experimentación realizada, obtenemos las siguientes tablas, correspondientes al S.C.H. con $\alpha = 1$ y $\beta = 2$:

SCH $\alpha = 1$, $\beta = 1$						
	GKD-c_1_n500_m50		GKD-c_2_n500_m50		GKD-c_3_n500_m50	
Ejecución	Coste	Tiempo	Coste	Tiempo	Coste	Tiempo
1	19458,15	77625,00	19671,65	74997,00	19524,13	75122,00
2	19464,52	75155,00	19674,24	75196,00	19517,79	75309,00
3	19461,56	75467,00	19672,58	75163,00	19512,71	75040,00
4	19453,08	76501,00	19663,09	75141,00	19519,27	75298,00
5	19455,44	75050,00	19671,50	75515,00	19513,44	75086,00
Media	-0,14%	75959,60	-0,16%	75202,40	-0,15%	75171,00
Devs. típica	0,02%	1093,63	0,02%	190,57	0,02%	124,46

Table 4: Resultados GKD

SCH $\alpha = 1$, $\beta = 1$						
	SOM-b_11_n300_m90		SOM-b_12_n300_m120		SOM-b_13_n400_m40	
Ejecución	Coste	Tiempo	Coste	Tiempo	Coste	Tiempo
1	20622,00	139098,00	35778,00	213206,00	4623,00	47794,00
2	20642,00	138657,00	35788,00	213358,00	4625,00	47931,00
3	20626,00	139174,00	35801,00	213563,00	4645,00	47870,00
4	20630,00	139116,00	35798,00	213505,00	4628,00	48003,00
5	20659,00	139407,00	35777,00	213245,00	4630,00	48161,00
Media	-0,52%	139090,40	-0,26%	213375,40	-0,60%	47951,80
Devs. típica	0,07%	271,93	0,03%	156,52	0,19%	140,01

Table 5: Resultados SOM

SCH $\alpha = 1$, $\beta = 1$						
	MDG-a_21_n2000_m200		MDG-a_22_n2000_m200		MDG-a_23_n2000_m200	
Ejecución	Coste	Tiempo	Coste	Tiempo	Coste	Tiempo
1	113382,00	600068,00	113280,00	600167,00	113246,00	600160,00
2	113460,00	600100,00	113189,00	600228,00	113009,00	600159,00
3	113257,00	600161,00	113464,00	600081,00	113136,00	600202,00
4	113281,00	600139,00	113209,00	600100,00	113153,00	600118,00
5	113376,00	600014,00	113202,00	600126,00	113109,00	600223,00
Media	-0,79%	600096,40	-0,93%	600140,40	-0,87%	6000174,40
Devs. típica	0,07%	58,30	0,10%	58,63	0,07%	44,22

Table 6: Resultados MDG

Como puede observarse en los resultados, el algoritmo exhibe un comportamiento robusto, manteniendo las desviación típica de los resultados por debajo del 0,10% en casi todos los archivos estudiados, a excepción de dos. En todo caso la mayor desviación típica alcanzada no sobrepasa el 0,20%. Si evaluamos la calidad de las soluciones obtenidas respecto a costes, el algoritmo es incapaz de encontrar los óptimos globales con los parámetros anteriormente descritos.

En lo referente a tiempos, el algoritmo consigue ejecutar las iteraciones objetivo antes de los 600 segundos para las instancias pequeñas y medianas. No es el caso de las instancias de datos más grandes, dónde el algoritmo finaliza su ejecución en torno las 2.000 iteraciones, sin sobrepasar las 3.000 iteraciones en ningún caso, para los archivos MDG.

4.4 Evolución de los resultados durante la experimentación

En las siguientes gráficas se muestra la evolución del coste de la mejor hormiga encontrada hasta el momento en la ejecución de cada uno de los algoritmos conforme aumenta el número de iteraciones realizadas para cada uno de los valores de α y β del S.C.H. La semilla utilizada para las ejecuciones ha sido 35608477.

Hemos decidido que la unidad de tiempo sea las iteraciones realizadas por el algoritmo en vez del tiempo de ejecución ya que este valor es sobre el que se comprueba la condición de parada y, además, puede arrojar información relevante a la hora de la modificación de dicho parámetro para obtener mejoras de tiempos o en el valor de las soluciones obtenidas.

Otro de los motivos por los que no se ha escogido el tiempo transcurrido como unidad de tiempo para las gráficas de convergencia de los costes es porque en un principio la única condición de parada era el número de iteraciones realizadas. No obstante, el escoger las iteraciones nos permite comparar resultados

obtenidos en otros equipos de experimentación, en los que pueden variar sus tiempos de ejecución debido a su configuración de hardware.

En este apartado, se hará referencia a las diferentes configuraciones de α y β haciendo uso de la siguiente codificación:

- $\alpha = 1$ $\beta = 1$: **A1B1**
- $\alpha = 1$ $\beta = 2$: **A1B2**
- $\alpha = 2$ $\beta = 1$: **A2B1**

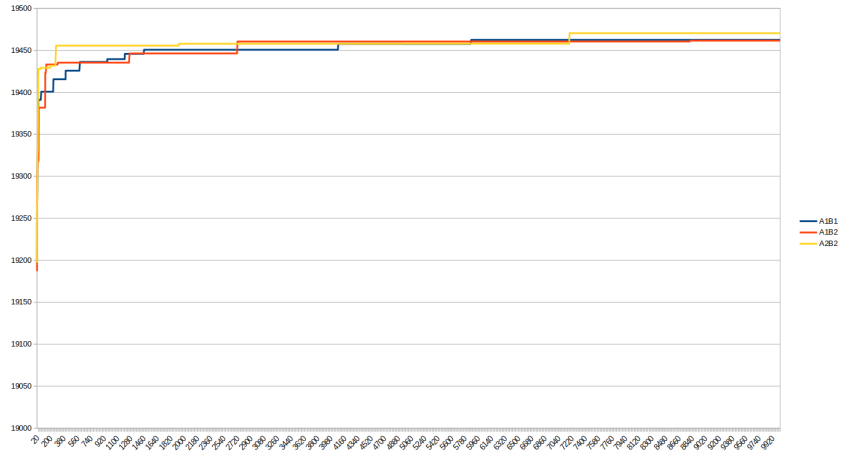

Figure 2: Evolución del mejor coste en la ejecución de todos los algoritmos respecto al número de evaluaciones para el problema GKD-c1, semilla 35608477

Para el problema GKD-c1 todas las configuraciones empiezan a encontrar soluciones aceptables desde la primera iteración realizada. Hemos considerado una solución aceptable aquella solución que difiere como máximo un 10% respecto al óptimo global. En este problema las soluciones aceptables son todas aquellas cuyo coste es superior a 17.536,669.

En la ejecución de la configuración A1B1, el mejor coste obtenido se estabiliza en 19.462,505859375 a partir de las 5.844 iteraciones, permaneciendo inalterable hasta finalizar las 10.000 iteraciones objetivo. Esta configuración es capaz de encontrar costes con una diferencia inferior a un 1% respecto al óptimo global a partir de las 10 iteraciones.

En la ejecución de la configuración A1B2, el mejor coste obtenido se estabiliza en 19.461,5625 a partir de las 8.791 iteraciones, permaneciendo inalterable hasta finalizar las 10.000 iteraciones objetivo. Esta configuración es capaz de encontrar

costes con una diferencia inferior a un 1% respecto al óptimo global a partir de las 3 iteraciones.

En la ejecución de la configuración A2B1, el mejor coste obtenido se estabiliza en 19.470,498046875 a partir de las 7.166 iteraciones, permaneciendo inalterable hasta finalizar las 10.000 iteraciones objetivo. Esta configuración es capaz de encontrar costes con una diferencia inferior a un 1% respecto al óptimo global a partir de las 7 iteraciones.

En la configuración A1B1 obtenemos una diferencia entre el coste de la solución obtenida y el óptimo global de un 0.0011% aproximadamente, siendo este último 19.485,188.

En la configuración A1B2 obtenemos una diferencia entre el coste de la solución obtenida y el óptimo global de un 0.0012% aproximadamente.

En la configuración A2B1 obtenemos una diferencia entre el coste de la solución obtenida y el óptimo global de un 0.0007% aproximadamente.

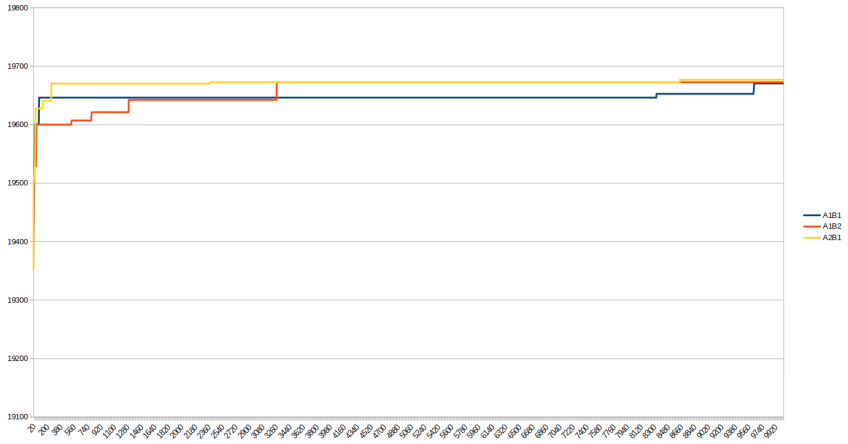

Figure 3: Evolución del mejor coste en la ejecución de todos los algoritmos respecto al número de evaluaciones para el problema GKD-c2, semilla 35608477

Para el problema GKD-c2 todas las configuraciones empiezan a encontrar soluciones aceptables desde la primera iteración realizada. Hemos considerado una solución aceptable aquella solución que difiere como máximo un 10% respecto al óptimo global. En este problema las soluciones aceptables son todas aquellas cuyo coste es superior a 17.731,3833.

En la ejecución de la configuración A1B1, el mejor coste obtenido se estabiliza en 19.670,576171875 a partir de las 9.609 iteraciones, permaneciendo inalterable hasta finalizar las 10.000 iteraciones objetivo. Esta configuración es capaz de encontrar costes con una diferencia inferior a un 1% respecto al óptimo global a partir de las 11 iteraciones.

En la ejecución de la configuración A1B2, el mejor coste obtenido se estabiliza en 19.672,58203125 a partir de las 3.243 iteraciones, permaneciendo inalterable hasta finalizar las 10.000 iteraciones objetivo. Esta configuración es capaz de encontrar costes con una diferencia inferior a un 1% respecto al óptimo global a partir de las 11 iteraciones.

En la ejecución de la configuración A2B1, el mejor coste obtenido se estabiliza en 19.676,869140625 a partir de las 8.619 iteraciones, permaneciendo inalterable hasta finalizar las 10.000 iteraciones objetivo. Esta configuración es capaz de encontrar costes con una diferencia inferior a un 1% respecto al óptimo global a partir de las 8 iteraciones.

En la configuración A1B1 obtenemos una diferencia entre el coste de la solución obtenida y el óptimo global de un 0.0015% aproximadamente, siendo este último 19.701,537.

En la configuración A1B2 obtenemos una diferencia entre el coste de la solución obtenida y el óptimo global de un 0.0014% aproximadamente.

En la configuración A2B1 obtenemos una diferencia entre el coste de la solución obtenida y el óptimo global de un 0.0012% aproximadamente.

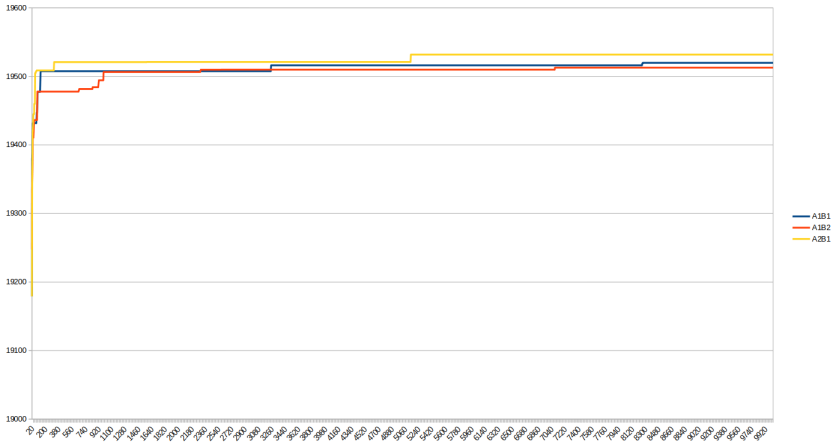

Figure 4: Evolución del mejor coste en la ejecución de todos los algoritmos respecto al número de evaluaciones para el problema GKD-c3, semilla 35608477

Para el problema GKD-c3 todas las configuraciones empiezan a encontrar soluciones aceptables desde la primera iteración realizada. Hemos considerado una solución aceptable aquella solución que difiere como máximo un 10% respecto al óptimo global. En este problema las soluciones aceptables son todas aquellas cuyo coste es superior a 17.592,4863.

En la ejecución de la configuración A1B1, el mejor coste obtenido se estabiliza en 19.519,779296875 a partir de las 8.244 iteraciones, permaneciendo inalterable hasta finalizar las 10.000 iteraciones objetivo. Esta configuración es capaz de encontrar costes con una diferencia inferior a un 1% respecto al óptimo global a partir de las 2 iteraciones.

En la ejecución de la configuración A1B2, el mejor coste obtenido se estabiliza en 19.512,7109375 a partir de las 7.058 iteraciones, permaneciendo inalterable hasta finalizar las 10.000 iteraciones objetivo. Esta configuración es capaz de encontrar costes con una diferencia inferior a un 1% respecto al óptimo global a partir de las 9 iteraciones.

En la ejecución de la configuración A2B1, el mejor coste obtenido se estabiliza en 19.531,783203125 a partir de las 5.114 iteraciones, permaneciendo inalterable hasta finalizar las 10.000 iteraciones objetivo. Esta configuración es capaz de encontrar costes con una diferencia inferior a un 1% respecto al óptimo global a partir de las 5 iteraciones.

En la configuración A1B1 obtenemos una diferencia entre el coste de la solución obtenida y el óptimo global de un 0.0014% aproximadamente, siendo este último 19.547,207.

En la configuración A1B2 obtenemos una diferencia entre el coste de la solución obtenida y el óptimo global de un 0.0017% aproximadamente.

En la configuración A2B1 obtenemos una diferencia entre el coste de la solución obtenida y el óptimo global de un 0.0007% aproximadamente.

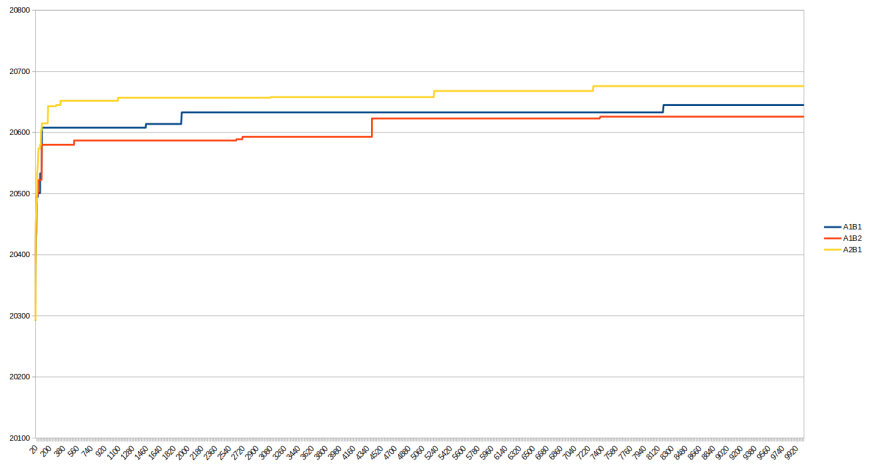

Figure 5: Evolución del mejor coste en la ejecución de todos los algoritmos respecto al número de evaluaciones para el problema SOM-b11, semilla 35608477

Para el problema SOM-b11 todas las configuraciones empiezan a encontrar soluciones aceptables desde la primera iteración realizada. Hemos considerado una solución aceptable aquella solución que difiere como máximo un 10% respecto al óptimo global. En este problema las soluciones aceptables son todas aquellas cuyo coste es superior a 18.668,7.

En la ejecución de la configuración A1B1, el mejor coste obtenido se estabiliza en 20.645 a partir de las 8.174 iteraciones, permaneciendo inalterable hasta finalizar las 10.000 iteraciones objetivo. Esta configuración es capaz de encontrar costes con una diferencia inferior a un 1% respecto al óptimo global a partir de las 84 iteraciones.

En la ejecución de la configuración A1B2, el mejor coste obtenido se estabiliza en 20.626 a partir de las 7.351 iteraciones, permaneciendo inalterable hasta finalizar las 10.000 iteraciones objetivo. Esta configuración es capaz de encontrar costes con una diferencia inferior a un 1% respecto al óptimo global a partir de las 82 iteraciones.

En la ejecución de la configuración A2B1, el mejor coste obtenido se estabiliza en 20.676 a partir de las 7.260 iteraciones, permaneciendo inalterable hasta finalizar las 10.000 iteraciones objetivo. Esta configuración es capaz de encontrar costes con una diferencia inferior a un 1% respecto al óptimo global a partir de las 31 iteraciones.

En la configuración A1B1 obtenemos una diferencia entre el coste de la solución obtenida y el óptimo global de un 0.0047% aproximadamente, siendo

este último 20.743.

En la configuración A1B2 obtenemos una diferencia entre el coste de la solución obtenida y el óptimo global de un 0.0056% aproximadamente.

En la configuración A2B1 obtenemos una diferencia entre el coste de la solución obtenida y el óptimo global de un 0.0032% aproximadamente.

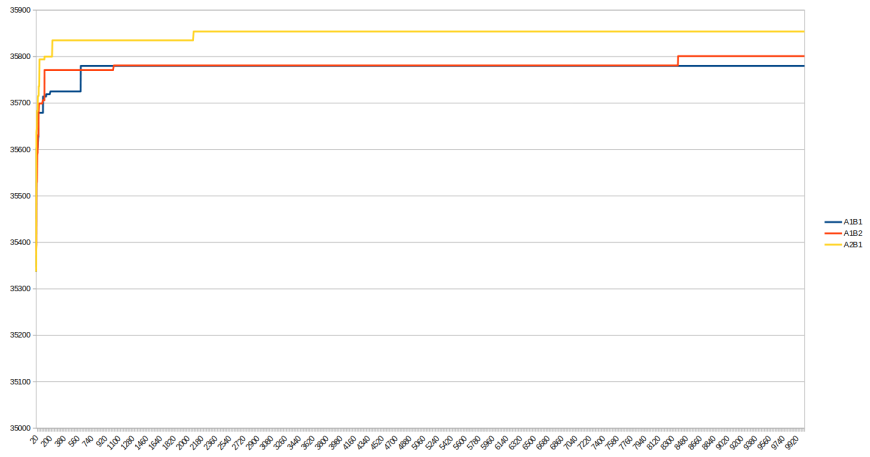

Figure 6: Evolución del mejor coste en la ejecución de todos los algoritmos respecto al número de evaluaciones para el problema SOM-b12, semilla 35608477

Para el problema SOM-b12 todas las configuraciones empiezan a encontrar soluciones aceptables desde la primera iteración realizada. Hemos considerado una solución aceptable aquella solución que difiere como máximo un 10% respecto al óptimo global. En este problema las soluciones aceptables son todas aquellas cuyo coste es superior a 32.292,9.

En la ejecución de la configuración A1B1, el mejor coste obtenido se estabiliza en 35.780 a partir de las 582 iteraciones, permaneciendo inalterable hasta finalizar las 10.000 iteraciones objetivo. Esta configuración es capaz de encontrar costes con una diferencia inferior a un 1% respecto al óptimo global a partir de las 6 iteraciones.

En la ejecución de la configuración A1B2, el mejor coste obtenido se estabiliza en 35.801 a partir de las 8.356 iteraciones, permaneciendo inalterable hasta finalizar las 10.000 iteraciones objetivo. Esta configuración es capaz de encontrar costes con una diferencia inferior a un 1% respecto al óptimo global a partir de las 7 iteraciones.

En la ejecución de la configuración A2B1, el mejor coste obtenido se estabiliza en 35.854 a partir de las 2.051 iteraciones, permaneciendo inalterable hasta finalizar las 10.000 iteraciones objetivo. Esta configuración es capaz de encontrar costes con una diferencia inferior a un 1% respecto al óptimo global a partir de las 4 iteraciones.

En la configuración A1B1 obtenemos una diferencia entre el coste de la solución obtenida y el óptimo global de un 0.0028% aproximadamente, siendo este último 35.881.

En la configuración A1B2 obtenemos una diferencia entre el coste de la solución obtenida y el óptimo global de un 0.0022% aproximadamente.

En la configuración A2B1 obtenemos una diferencia entre el coste de la solución obtenida y el óptimo global de un 0.0007% aproximadamente.

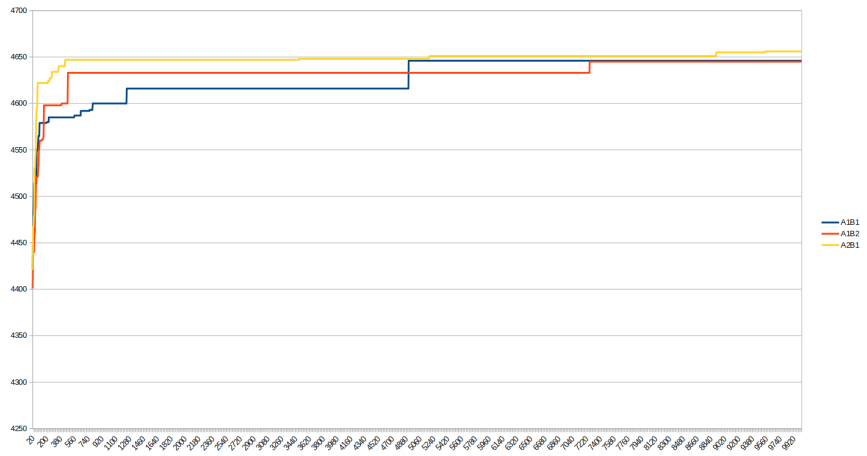

Figure 7: Evolución del mejor coste en la ejecución de todos los algoritmos respecto al número de evaluaciones para el problema SOM-b13, semilla 35608477

Para el problema SOM-b13 todas las configuraciones empiezan a encontrar soluciones aceptables desde la primera iteración realizada. Hemos considerado una solución aceptable aquella solución que difiere como máximo un 10% respecto al óptimo global. En este problema las soluciones aceptables son todas aquellas cuyo coste es superior a 4.192,2.

En la ejecución de la configuración A1B1, el mejor coste obtenido se estabiliza en 4.646 a partir de las 4.891 iteraciones, permaneciendo inalterable hasta finalizar las 10.000 iteraciones objetivo. Esta configuración es capaz de encontrar costes con una diferencia inferior a un 1% respecto al óptimo global a partir de las 1.225 iteraciones.

En la ejecución de la configuración A1B2, el mejor coste obtenido se estabiliza en 4.645 a partir de las 7.245 iteraciones, permaneciendo inalterable hasta finalizar las 10.000 iteraciones objetivo. Esta configuración es capaz de encontrar costes con una diferencia inferior a un 1% respecto al óptimo global a partir de las 460 iteraciones.

En la ejecución de la configuración A2B1, el mejor coste obtenido se estabiliza en 4.656 a partir de las 9.525 iteraciones, permaneciendo inalterable hasta finalizar las 10.000 iteraciones objetivo. Esta configuración es capaz de encontrar costes con una diferencia inferior a un 1% respecto al óptimo global a partir de las 67 iteraciones.

En la configuración A1B1 obtenemos una diferencia entre el coste de la solución obtenida y el óptimo global de un 0.0025% aproximadamente, siendo este último 4.658.

En la configuración A1B2 obtenemos una diferencia entre el coste de la solución obtenida y el óptimo global de un 0.0027% aproximadamente.

En la configuración A2B1 obtenemos una diferencia entre el coste de la solución obtenida y el óptimo global de un 0.0004% aproximadamente.

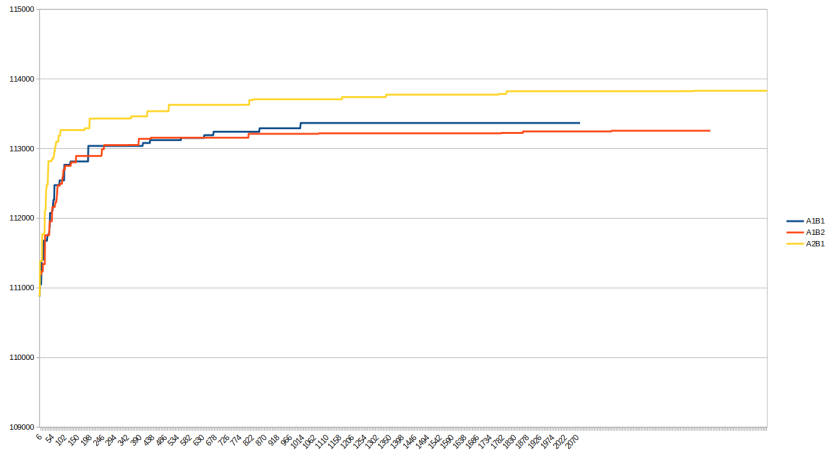

Figure 8: Evolución del mejor coste en la ejecución de todos los algoritmos respecto al número de evaluaciones para el problema MDG-a21, semilla 35608477

Para el problema MDG-a21 todas las configuraciones empiezan a encontrar soluciones aceptables desde la primera iteración realizada. Hemos considerado una solución aceptable aquella solución que difiere como máximo un 10% respecto al óptimo global. En este problema las soluciones aceptables son todas aquellas cuyo coste es superior a 102.833,1.

En la ejecución de la configuración A1B1, el mejor coste obtenido se estabiliza en 113.368 a partir de las 1.003 iteraciones, permaneciendo inalterable hasta realizar 2.076 iteraciones, donde se termina la ejecución al haber alcanzado los 600 segundos límites de tiempo de ejecución. Esta configuración es capaz de encontrar costes con una diferencia inferior a un 1% respecto al óptimo global a partir de las 425 iteraciones.

En la ejecución de la configuración A1B2, el mejor coste obtenido se estabiliza en 113.257 a partir de las 2.196 iteraciones, permaneciendo inalterable hasta realizar 2.575 iteraciones, donde se termina la ejecución al haber alcanzado los 600 segundos límites de tiempo de ejecución. Esta configuración es capaz de encontrar costes con una diferencia inferior a un 1% respecto al óptimo global a partir de las 382 iteraciones.

En la ejecución de la configuración A2B1, el mejor coste obtenido se estabiliza en 113.831 a partir de las 2.511 iteraciones, permaneciendo inalterable hasta realizar 2.793 iteraciones, donde se termina la ejecución al haber alcanzado los 600 segundos límites de tiempo de ejecución. Esta configuración es capaz de encontrar costes con una diferencia inferior a un 1% respecto al óptimo global a partir de las 74 iteraciones.

En la configuración A1B1 obtenemos una diferencia entre el coste de la solución obtenida y el óptimo global de un 0.0077% aproximadamente, siendo este último 114.259.

En la configuración A1B2 obtenemos una diferencia entre el coste de la solución obtenida y el óptimo global de un 0.0087% aproximadamente.

En la configuración A2B1 obtenemos una diferencia entre el coste de la solución obtenida y el óptimo global de un 0.0037% aproximadamente.

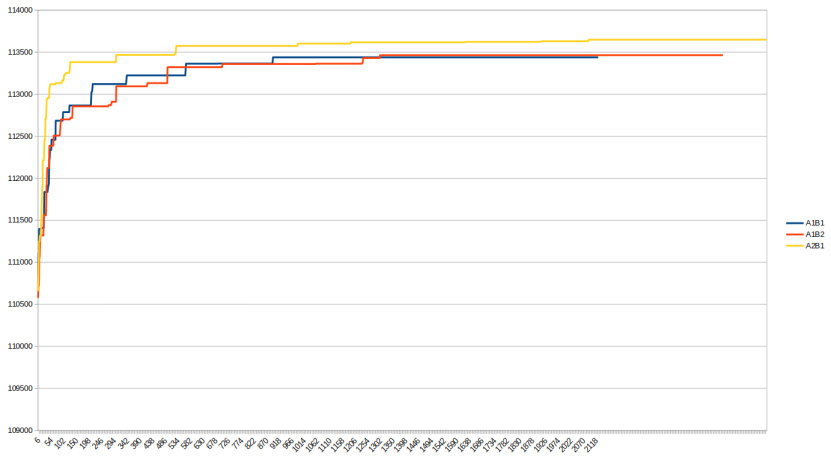

Figure 9: Evolución del mejor coste en la ejecución de todos los algoritmos respecto al número de evaluaciones para el problema MDG-a22, semilla 35608477

Para el problema MDG-a22 todas las configuraciones empiezan a encontrar soluciones aceptables desde la primera iteración realizada. Hemos considerado una solución aceptable aquella solución que difiere como máximo un 10% respecto al óptimo global. En este problema las soluciones aceptables son todas aquellas cuyo coste es superior a 102.894,3.

En la ejecución de la configuración A1B1, el mejor coste obtenido se estabiliza en 113.439 a partir de las 890 iteraciones, permaneciendo inalterable hasta realizar 2.121 iteraciones, donde se termina la ejecución al haber alcanzado los 600 segundos límites de tiempo de ejecución. Esta configuración es capaz de encontrar costes con una diferencia inferior a un 1% respecto al óptimo global a partir de las 337 iteraciones.

En la ejecución de la configuración A1B2, el mejor coste obtenido se estabiliza en 113.464 a partir de las 1.295 iteraciones, permaneciendo inalterable hasta realizar 2.592 iteraciones, donde se termina la ejecución al haber alcanzado los 600 segundos límites de tiempo de ejecución. Esta configuración es capaz de encontrar costes con una diferencia inferior a un 1% respecto al óptimo global a partir de las 491 iteraciones.

En la ejecución de la configuración A2B1, el mejor coste obtenido se estabiliza en 113.649 a partir de las 2.085 iteraciones, permaneciendo inalterable hasta realizar 2.758 iteraciones, donde se termina la ejecución al haber alcanzado los 600 segundos límites de tiempo de ejecución. Esta configuración es capaz de encontrar costes con una diferencia inferior a un 1% respecto al óptimo global a partir de las 100 iteraciones.

En la configuración A1B1 obtenemos una diferencia entre el coste de la solución obtenida y el óptimo global de un 0.0077% aproximadamente, siendo este último 114.327.

En la configuración A1B2 obtenemos una diferencia entre el coste de la solución obtenida y el óptimo global de un 0.0075% aproximadamente.

En la configuración A2B1 obtenemos una diferencia entre el coste de la solución obtenida y el óptimo global de un 0.0059% aproximadamente.

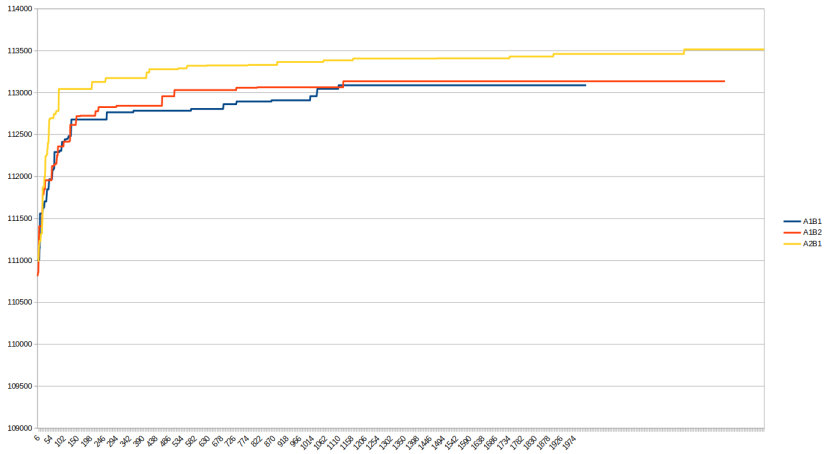

Figure 10: Evolución del mejor coste en la ejecución de todos los algoritmos respecto al número de evaluaciones para el problema MDG-a23, semilla 35608477

Para el problema MDG-a23 todas las configuraciones empiezan a encontrar soluciones aceptables desde la primera iteración realizada. Hemos considerado una solución aceptable aquella solución que difiere como máximo un 10% respecto al óptimo global. En este problema las soluciones aceptables son todas aquellas cuyo coste es superior a 102.710,7.

En la ejecución de la configuración A1B1, el mejor coste obtenido se estabiliza en 113.088 a partir de las 1.105 iteraciones, permaneciendo inalterable hasta realizar 2.012 iteraciones, donde se termina la ejecución al haber alcanzado los 600 segundos límites de tiempo de ejecución. Esta configuración es capaz de encontrar costes con una diferencia inferior a un 1% respecto al óptimo global a partir de las 1.027 iteraciones.

En la ejecución de la configuración A1B2, el mejor coste obtenido se estabiliza en 113.136 a partir de las 1.122 iteraciones, permaneciendo inalterable hasta realizar 2.521 iteraciones, donde se termina la ejecución al haber alcanzado los

600 segundos límites de tiempo de ejecución. Esta configuración es capaz de encontrar costes con una diferencia inferior a un 1% respecto al óptimo global a partir de las 503 iteraciones.

En la ejecución de la configuración A2B1, el mejor coste obtenido se estabiliza en 113.516 a partir de las 2.372 iteraciones, permaneciendo inalterable hasta realizar 2.665 iteraciones, donde se termina la ejecución al haber alcanzado los 600 segundos límites de tiempo de ejecución. Esta configuración es capaz de encontrar costes con una diferencia inferior a un 1% respecto al óptimo global a partir de las 79 iteraciones.

En la configuración A1B1 obtenemos una diferencia entre el coste de la solución obtenida y el óptimo global de un 0.0090% aproximadamente, siendo este último 114.123.

En la configuración A1B2 obtenemos una diferencia entre el coste de la solución obtenida y el óptimo global de un 0.0086% aproximadamente.

En la configuración A2B1 obtenemos una diferencia entre el coste de la solución obtenida y el óptimo global de un 0.0053% aproximadamente.

References

[1] Umbarkar, Dr. Anantkumar & Sheth, P.. (2015). CROSSOVER OPERATORS IN GENETIC ALGORITHMS: A REVIEW. ICTACT Journal on Soft Computing (Volume: 6 , Issue: 1). 6. 10.21917/ijsc.2015.0150.

[2] https://sci2s.ugr.es/graduateCourses/Metaheuristicas

[3] https://sci2s.ugr.es/graduateCourses/Algoritmica