



Universidad
de Huelva



Universidad de Huelva

GRADO EN INGENIERÍA INFORMÁTICA

PRÁCTICA 3. ALGORITMOS HEURÍSTICOS NO CONSTRUCTIVOS POBLACIONALES. ALGORITMOS GENÉTICOS

Autor: Alberto Fernández Merchán

Asignatura: Modelos Bioinspirados y Heurísticas de Búsquedas

Índice

1. Introducción	3
2. Algoritmo Genético Básico	4
2.1. Código	4
2.1.1. Función de Cruce	4
2.1.2. Función de Mutación	4
2.1.3. Algoritmo Genético	5
2.2. Parámetros	6
2.3. Análisis del algoritmo	6
2.4. Resultados	7
2.4.1. Problema Real	7
2.4.2. Problema Aleatorio	9
2.5. Experimentación	10
2.5.1. Número de Iteraciones	10
2.5.2. Porcentaje de Mutación	10
2.5.3. Población Inicial	11
3. CHC	11
3.1. Código	11
3.1.1. Operador de Cruce	11
3.1.2. Operador de Selección	12
3.1.3. Algoritmo CHC	13
3.2. Parámetros	14
3.3. Análisis del algoritmo	14
3.4. Resultados	15
3.4.1. Problema Real	15
3.4.2. Problema Aleatorio	17
3.5. Experimentación	18
3.5.1. Parámetro α del operador de cruce	18
3.5.2. Número de Iteraciones sin mejora	18
4. Algoritmo Genético Multimodal	19
4.1. Código	19
4.1.1. Función de Clearing	19
4.2. Parámetros	19
4.3. Análisis del algoritmo	20
4.4. Resultados	20
4.4.1. Problema Real	20
4.4.2. Problema Aleatorio	22
4.5. Experimentación	23
4.5.1. Número de generaciones para hacer clearing	23
4.5.2. Radio de Clearing	23
4.5.3. Kappa	24
5. Análisis de Resultados	24
5.1. Problema Real	24
5.2. Problema Aleatorio	25
6. Artículos Relacionados	25
6.1. Genético Básico	25
6.2. CHC	26
6.3. Multimodal	26
7. Conclusiones	26

Índice de cuadros

1.	Semillas	3
2.	Parámetros del algoritmo genético generacional básico.	6
3.	Fitness Genético Básico Problema Real	7
4.	Convergencia Genético Básico Problema Real	7
5.	Simulación Genético Básico Problema Real	8
6.	Fitness Genético Básico Problema Aleatorio	9
7.	Convergencia Genético Básico Problema Aleatorio	9
8.	Simulación Genético Básico Problema Aleatorio	10
9.	Experimentación de iteraciones sin mejora	10
10.	Experimentación de porcentaje de mutación	10
11.	Experimentación de población inicial	11
12.	Parámetros del algoritmo genético CHC.	14
13.	Fitness CHC Problema Real	15
14.	Convergencia CHC Problema Real	15
15.	Simulación CHC Problema Real	16
16.	Fitness CHC Problema Aleatorio	17
17.	Convergencia CHC Problema Aleatorio	17
18.	Simulación CHC Problema Aleatorio	18
19.	Experimentación del parámetro alfa	18
20.	Experimentación de número de iteraciones	18
21.	Parámetros del algoritmo genético multimodal.	19
22.	Fitness Multimodal Problema Real	20
23.	Convergencia Multimodal Problema Real	21
24.	Simulación Multimodal Problema Real	21
25.	Fitness Multimodal Problema Aleatorio	22
26.	Convergencia Multimodal Problema Aleatorio	22
27.	Simulación Multimodal Problema Aleatorio	23
28.	Experimentación de generaciones	23
29.	Experimentación de Radio de Clearing	23
30.	Experimentación de Kappa	24
31.	Comparación de métricas para los datos del problema real.	24
32.	Comparación de métricas para los datos del problema aleatorio.	25

1. Introducción

En esta práctica, estudiaremos el funcionamiento de algoritmos heurísticos no constructivos para resolver el problema de administración de energía de una central solar, que abordamos en la práctica 1. Estos algoritmos mejoran soluciones candidatas a través de un conjunto de operaciones, a partir de un conjunto de soluciones iniciales.

Analizaremos tres tipos de algoritmos: el algoritmo genético básico, CHC y el algoritmo genético multimodal. Además, compararemos sus resultados con el algoritmo de búsqueda local primero el mejor. Realizaremos la ejecución de los algoritmos con tres semillas diferentes (1) y aplicaremos los algoritmos tanto para datos reales como para datos aleatorios.

Luego, analizaremos los resultados obtenidos en función de la convergencia, el grado de adaptación (fitness), y simularemos el problema con el mejor individuo de cada ejecución. Asimismo, compararemos los resultados utilizando una tabla, en la que mediremos el número de veces que se llama a la función de evaluación y el dinero que se ha conseguido.

Por último, mostraremos algunos artículos en los que se resuelven problemas reales mediante el uso de algoritmos genéticos.

Las semillas que se han utilizado han sido las siguientes:

Semillas		
123456	654321	456789

Cuadro 1: Semillas que se han utilizado para estudiar el funcionamiento de los algoritmos.

2. Algoritmo Genético Básico

En primer lugar, analizaremos un algoritmo genético básico generacional con una élite de 5 individuos y una población de 17 inicializada de forma aleatoria.

2.1. Código

En esta sección veremos los códigos utilizados para realizar el cruce entre individuos, la mutación de cada uno y, por último, el código del algoritmo genético propiamente dicho.

2.1.1. Función de Cruce

Para el operador de cruce he utilizado el cruce en dos puntos. Este operador consiste en seleccionar dos puntos aleatorios de un cromosoma y generar los hijos concatenando cada parte del cromosoma de cada padre de la siguiente manera:

```
1 def cruce(individuo1, individuo2):
2
3     pos1 = np.random.randint(0, len(individuo1) - 1)
4     pos2 = np.random.randint(0, len(individuo1) - 1)
5     min_pos = pos2 if pos2 < pos1 else pos1
6     max_pos = pos2 if pos2 > pos1 else pos1
7     hijo1 = np.append(individuo1[0:min_pos].copy(),
8                       np.append(individuo2[min_pos:max_pos].copy(),
9                               individuo1[max_pos:len(individuo1)].copy()))
10
11     hijo2 = np.append(individuo2[0:min_pos].copy(),
12                      np.append(individuo1[min_pos:max_pos].copy(),
13                              individuo2[max_pos:len(individuo2)].copy()))
14     return hijo1, hijo2
```

2.1.2. Función de Mutación

El operador de mutación consiste en cambiar un gen del cromosoma un 1 % de las veces que se genera un hijo. Este cambio consiste en sumar o restar, aleatoriamente, un valor de 10. Este operador se utiliza para aumentar la diversidad entre los individuos de la población.

```
1 def mutacion(individuo):
2     ind_mutado = individuo.copy()
3     if np.random.random() <= Utils.PORCENTAJE_MUTACION:
4         # Obtiene el valor de la posición asignada
5         posicion = np.random.randint(0, 24)
6         valor = 10
7         # Sumar o Restar
8         if np.random.uniform() < 0.5:
9             ind_mutado[posicion] += valor
10        else:
11            ind_mutado[posicion] -= valor
12
13        # Comprobamos que no supere los límites
14        if ind_mutado[posicion] > 100:
15            ind_mutado[posicion] = 100
16        elif ind_mutado[posicion] < -100:
17            ind_mutado[posicion] = -100
18    return ind_mutado
```

2.1.3. Algoritmo Genético

```
1 def algoritmo_genetico_generacional(semilla, isRandom):
2     np.random.seed(semilla)
3     t = 0
4     # Inicializar P(t)
5     poblacion = Utils.inicializar_poblacion(Utils.POBLACION_INICIAL)
6     # Evaluar P(t)
7     valores_poblacion, dinero_acumulado, bateria_acumulada = Utils.fitnessPoblacion(poblacion, isRandom)
8     indice_maximo = np.argmax(valores_poblacion)
9     # Obtenemos el mejor individuo de la población inicial.
10    mejor_individuo = poblacion[indice_maximo]
11    mejor_valor = valores_poblacion[indice_maximo]
12    while t < Utils.NUM_ITERACIONES:
13        t = t + 1
14        # Seleccionamos la élite de la población:
15        elite = elitismo(poblacion, Utils.ELITE, isRandom)
16        numero_evaluaciones += len(poblacion) # Evaluamos a toda la poblacion para escoger la élite.
17
18        # Seleccionar los índices de los padres (K=3)
19        candidatos = np.array([torneo(valores_poblacion, int(Utils.K * Utils.POBLACION_INICIAL)) for _ in
20                                range(Utils.POBLACION_INICIAL)])
21
22        # Elegimos a los L=2 mejores de cada trío de padres
23        padres = np.zeros(shape=(Utils.POBLACION_INICIAL-Utils.ELITE, 2), dtype=int)
24        # Generamos el numero de la población menos el número de individuos
25        # de la élite parejas.
26        for i in range(Utils.POBLACION_INICIAL-Utils.ELITE):
27            padres[i] = candidatos[i][np.argsort(valores_poblacion[candidatos[i]])[-2:]]
28
29        # Recombinar P(t)
30        hijos = np.zeros(shape=(Utils.POBLACION_INICIAL, 24), dtype=int)
31        # Añado la élite a la siguiente generación:
32        hijos[0:Utils.ELITE, :] = elite
33        for i in range(Utils.ELITE, Utils.POBLACION_INICIAL, 2):
34            h1, h2 = cruce(poblacion[padres[i-Utils.ELITE][0].copy()],
35                           poblacion[padres[Utils.ELITE][1].copy()])
36            hijos[i] = h1
37            hijos[i + 1] = h2
38
39        # Mutar P(t)
40        hijos_mutados = np.apply_along_axis(mutacion, 1, hijos.copy())
41        poblacion = hijos_mutados
42
43        # Evaluar P(t)
44        valores_poblacion, dinero_acumulado, bateria_acumulada = Utils.fitnessPoblacion(poblacion, isRandom)
45        numero_evaluaciones += len(poblacion)
46        indice_maximo = np.argmax(valores_poblacion)
47
48        # Obtenemos el mejor individuo de la población.
49        if mejor_valor < valores_poblacion[indice_maximo]:
50            mejor_individuo = poblacion[indice_maximo].copy()
51            mejor_valor = valores_poblacion[indice_maximo].copy()
52            t = 0 # Cuando mejora, reiniciamos el contador de iteraciones.
53
54    return mejor_valor
```

2.2. Parámetros

Los parámetros que se han utilizado son los siguientes:

Parámetro	Valor
Población Inicial	17
Porcentaje de Mutación	1 %
Número de Iteraciones sin mejora	500
Individuos de Élite	5
Porcentaje de la población para el torneo	20 %

Cuadro 2: Parámetros del algoritmo genético generacional básico.

2.3. Análisis del algoritmo

Los algoritmos genéticos generacionales trabajan mediante la generación iterativa de nuevas poblaciones de soluciones candidatas. Utilizan operadores genéticos como la selección, el cruce y la mutación para explorar el espacio de búsqueda y encontrar soluciones óptimas.

Una de las características clave de estos algoritmos es la capacidad de explotación. Esta se refiere a la capacidad del algoritmo para mejorar gradualmente la calidad de la población al seleccionar y reproducir las mejores soluciones en cada generación. Esta estrategia permite que el algoritmo converja hacia una solución óptima y se acerque a la región del espacio de búsqueda donde se encuentra la mejor solución.

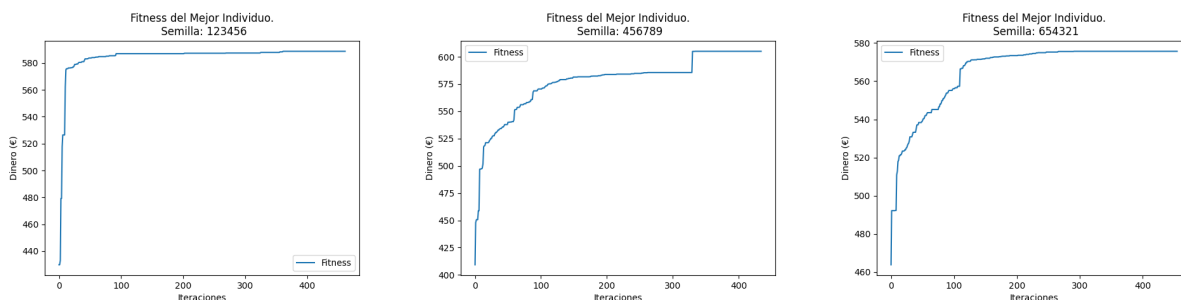
Sin embargo, a la hora de explorar nuevas soluciones, este algoritmo genético básico no consigue obtener una diversidad poblacional alta. Para intentar mejorar este aspecto, se introduce el operador de mutación, que realiza una perturbación en uno de los genes de un individuo para aumentar dicha diversidad. Según vemos en las tablas 4 y 7, la convergencia se realiza en una etapa temprana de la ejecución del algoritmo, lo que nos puede indicar que existe una convergencia prematura y, por tanto, poca diversidad genética.

2.4. Resultados

2.4.1. Problema Real

Fitness del mejor individuo

En las gráficas de la tabla 3 podemos observar el fitness del mejor individuo.

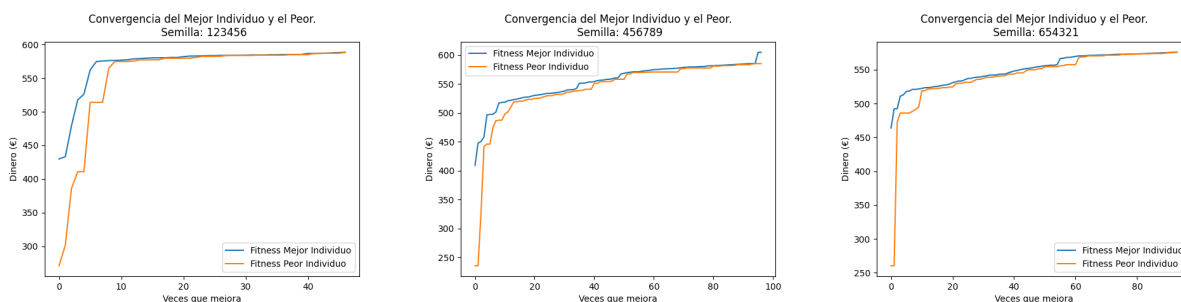


Cuadro 3: *Fitness* del mejor individuo en cada iteración del algoritmo sobre los datos del problema real con el algoritmo genético básico.

En las gráficas podemos ver que la adaptación de los individuos se realiza de forma escalonada y no continua. Además, tanto en el primer ejemplo como en el tercero, vemos que la función objetivo tiene una mejora considerable dejando como marca un codo en el perfil de la gráfica.

Convergencia del mejor y peor individuo

En las siguientes gráficas veremos el *fitness* del mejor y peor individuo en cada iteración que mejora la solución global del problema.

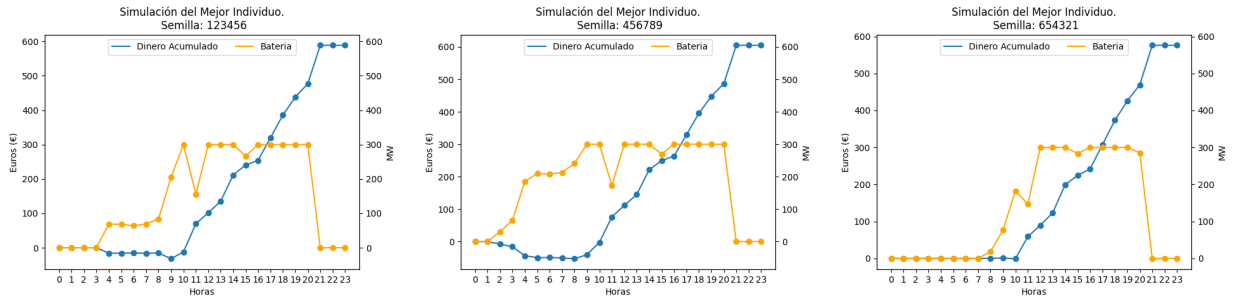


Cuadro 4: Convergencia del *fitness* del mejor y peor individuo en cada iteración del algoritmo sobre los datos del problema real con el algoritmo genético básico.

Podemos observar como, al comienzo de la ejecución del algoritmo, el peor individuo tiene un valor muy inferior al del mejor individuo. Sin embargo, conforme van sucediendo las iteraciones, comprobamos que el *fitness* de ambos individuos comienza a converger. Pasadas unas pocas iteraciones más, el valor del mejor individuo se incrementa y, seguidamente, el del peor individuo también lo hace más tarde.

Simulación del problema

Por último visualizaremos las gráficas de simulación del mejor individuo.



Cuadro 5: Simulación del mejor individuo sobre los datos del problema real con el algoritmo genético básico.

En las gráficas del cuadro 5 podemos ver que la solución obtenida decide comprar energía al comienzo del día para llenar la batería y venderla más adelante junto al sobrante.

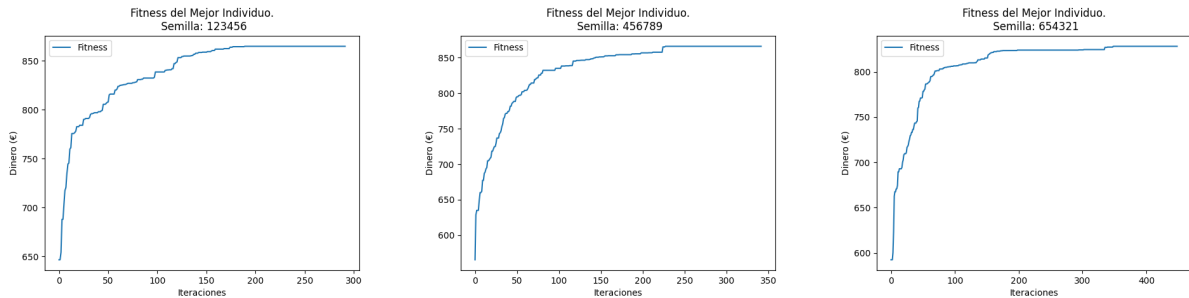
Si visualizamos las gráficas de la práctica 1, se parecen mucho a las que obteníamos con la búsqueda tabú, incluso, en el caso del algoritmo genético, llega a superar el valor máximo de ganancia obtenida.

En este caso, se llega a obtener una cantidad superior a los 500€ en todos los casos, cuando en la búsqueda tabú obteníamos un beneficio medio inferior a esa cifra.

2.4.2. Problema Aleatorio

Fitness del mejor individuo

A continuación se muestran las gráficas del *fitness* del mejor individuo durante cada iteración del algoritmo sobre los datos del problema aleatorio.

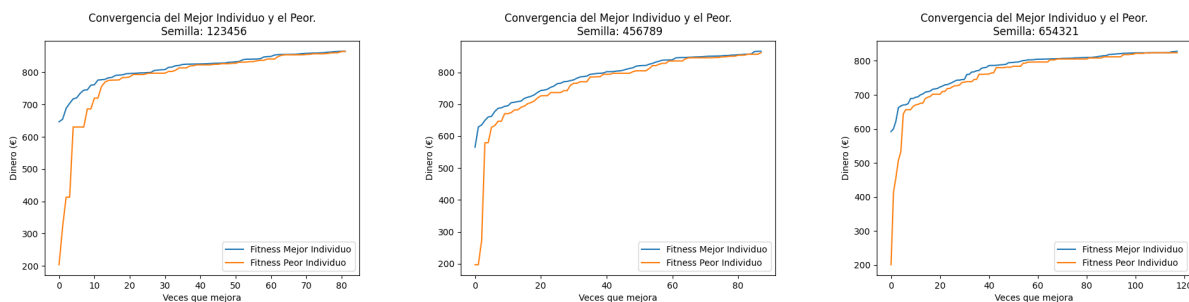


Cuadro 6: *Fitness* del mejor individuo en cada iteración del algoritmo sobre los datos del problema aleatorio con el algoritmo genético básico.

Al igual que en el problema de los datos reales, el *fitness* del mejor individuo se incrementa de forma escalonada hasta llegar a una planicie donde no consigue mejorar más. En este caso, no se marca tanto el codo de la mejora del algoritmo genético en el perfil de la gráfica. Sin embargo, los resultados obtenidos son mucho mejores con estos datos que con los del problema anterior.

Convergencia del mejor y peor individuo

En este párrafo se muestran las gráficas de convergencia del mejor y peor individuo en cada iteración que cambia el mejor individuo.

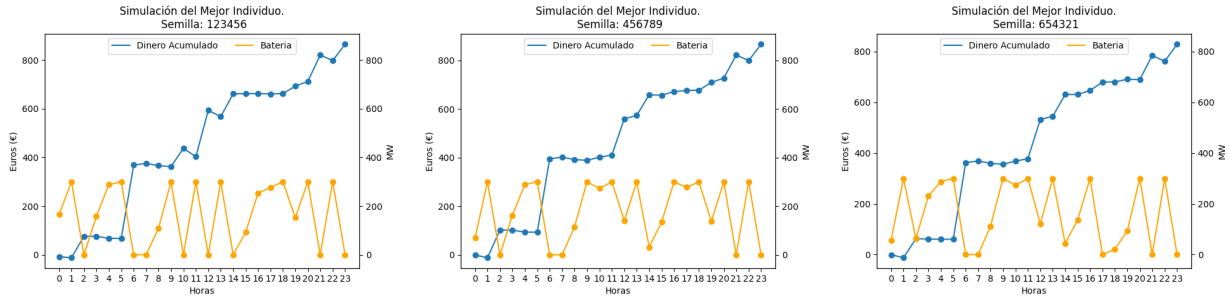


Cuadro 7: Convergencia del *fitness* del mejor y peor individuo en cada iteración del algoritmo sobre los datos del problema aleatorio con el algoritmo genético básico.

En cuanto a la convergencia de la población en este problema, sucede mucho antes de lo esperado. Esto puede ser debido a un operador de mutación con un efecto muy débil sobre los individuos o que el operador de cruce en dos puntos utilizado no genera la suficiente diversidad en los hijos.

Simulación del problema

Por último, mostraremos la simulación de la batería y la ganancia obtenida para el problema con los datos aleatorios.



Cuadro 8: Simulación del mejor individuo sobre los datos del problema aleatorio con el algoritmo genético básico.

Como ocurría con el anterior problema, las gráficas de simulación se parecen mucho a las que obteníamos con el algoritmo de búsqueda tabú. Estas gráficas son erráticas debido a la naturaleza aleatoria de los datos de venta y compra de energía.

2.5. Experimentación

En esta sección he tratado de experimentar con algunos parámetros para conseguir mejorar los resultados obtenidos por el algoritmo genético básico. Los parámetros que he seleccionado para experimentar son los siguientes:

2.5.1. Número de Iteraciones

Respecto al número de iteraciones, he escogido los siguientes valores para ejecutar el algoritmo y tratar de ver cuál es la mejor cantidad para colocar en la condición de parada del algoritmo:

Iteraciones sin Mejora						
100	200	300	500	1000	1500	2000

Cuadro 9: Número de iteraciones que se han utilizado para estudiar el funcionamiento de los algoritmos.

2.5.2. Porcentaje de Mutación

En cuanto al porcentaje de mutación se han elegido los siguientes valores:

Porcentaje de Mutación				
0.01	0.05	0.1	0.2	0.3

Cuadro 10: Porcentaje de mutación que se han utilizado para estudiar el funcionamiento de los algoritmos.

2.5.3. Población Inicial

Por último, he tomado los siguientes valores para experimentar con el número de individuos que tendría la población inicial:

Población Inicial							
15	17	19	21	23	25	27	29

Cuadro 11: Número de individuos iniciales que se han utilizado para estudiar el funcionamiento de los algoritmos.

Finalmente he elegido un número de iteraciones de 100, un porcentaje de mutación de 0,3 y una población inicial de 25 individuos.

3. CHC

El algoritmo CHC fue diseñado para cromosomas binarios, sin embargo, en este caso utilizaremos un operador de cruce conocido como *BLX- α* que nos servirá para generar hijos con codificación real.

3.1. Código

3.1.1. Operador de Cruce

El operador de cruce en este algoritmo consiste en obtener un hijo lo suficientemente diferente de los padres y lo suficientemente parecido para que mantenga la diversidad y pueda alcanzar una buena solución.

```
1 def blx_alpha(parent_1, parent_2, alpha):
2     """
3     Operador de cruce BLX-alpha para dos padres con valores reales.
4
5     :param parent_1: Cromosoma padre 1.
6     :param parent_2: Cromosoma padre 2.
7     :param alpha: Valor de alpha para el operador BLX-alpha.
8     :return: Dos hijos generados a partir de los padres mediante el operador BLX-alpha.
9     """
10    # Creamos dos hijos vacíos con el mismo tamaño que los padres.
11    child_1 = np.zeros_like(parent_1)
12    child_2 = np.zeros_like(parent_2)
13    # Recorremos los genes de los padres y generamos los genes de los hijos.
14    for i in range(len(parent_1)):
15        # Calculamos los límites del intervalo de cruce para el gen i.
16        cmin = min(parent_1[i], parent_2[i])
17        cmax = max(parent_1[i], parent_2[i])
18        I = cmax - cmin
19
20        # Generamos los genes de los hijos utilizando el operador BLX-alpha.
21        delta = alpha * I
22        rand = np.random.uniform(-delta, 1 + delta)
23        child_1[i] = 0.5 * ((1 + rand) * parent_1[i] + (1 - rand) * parent_2[i])
24        child_2[i] = 0.5 * ((1 - rand) * parent_1[i] + (1 + rand) * parent_2[i])
25
26        # Ajustamos los valores de los hijos si superan los límites permitidos.
27        child_1[i] = max(min(child_1[i], 100), -100)
28        child_2[i] = max(min(child_2[i], 100), -100)
29
30    return child_1, child_2
```

3.1.2. Operador de Selección

El operador de selección del algoritmo CHC consiste en reemplazar a los peores individuos de los padres por los mejores individuos de los hijos, aumentando así la calidad de los individuos de la población resultante.

```
1 def select_s(padres, hijos, isRandom):
2     """
3     Reemplaza los peores miembros de padres por los mejores miembros de hijos mientras haya algún miembro en
4     hijos mejor que en padres.
5
6     Args:
7     padres (numpy.ndarray): La matriz de padres.
8     hijos (numpy.ndarray): La matriz de hijos.
9
10    Returns:
11    numpy.ndarray: La matriz de padres actualizada.
12    """
13    valores_padres = Utils.fitnessPoblacion(padres, isRandom)[0]
14    valores_hijos = Utils.fitnessPoblacion(hijos, isRandom)[0]
15
16    while np.any(valores_hijos > valores_padres):
17        indices_padres = np.argsort(valores_padres)
18        indices_hijos = np.argsort(valores_hijos)[::-1]
19        for i in range(len(indices_hijos)):
20            if valores_hijos[indices_hijos[i]] > valores_padres[indices_padres[i]]:
21                padres[indices_padres[i]] = hijos[indices_hijos[i]]
22                valores_padres[indices_padres[i]] = valores_hijos[indices_hijos[i]]
23
24    return padres
```

3.1.3. Algoritmo CHC

```
1 def CHC(semilla, israndom):
2     np.random.seed(semilla)
3     t = 0
4     d = 24 / 4
5     # Inicializar Poblacion
6     poblacion = Utils.inicializar_poblacion(Utils.POBLACION_INICIAL - 1)
7     # Evaluar Poblacion
8     valores_poblacion = Utils.fitnessPoblacion(poblacion, israndom)[0]
9     indice_maximo = np.argmax(valores_poblacion)
10
11     # Mejor Valor Inicial
12     mejor_individuo = poblacion[indice_maximo]
13     mejor_valor = valores_poblacion[indice_maximo]
14
15     # Mientras no se cumpla la condicion
16     while t < Utils.NUM_ITERACIONES_CHC:
17         t += 1
18         # Seleccionar_r
19         parejas = poblacion.copy()
20         np.random.shuffle(parejas)
21
22         # Cruzar
23         hijos = np.empty_like(poblacion)
24         for i in range(0, len(parejas), 2):
25             hijo1, hijo2 = blx_alpha(parejas[i], parejas[i + 1], Utils.ALPHA)
26             hijos[i] = hijo1
27             hijos[i + 1] = hijo2
28
29         # Seleccionar
30         poblacion = select_s(parejas, hijos, israndom)
31         valores_poblacion, _, _ = Utils.fitnessPoblacion(poblacion, israndom)
32         indice_maximo = np.argmax(valores_poblacion)
33         indice_minimo = np.argmin(valores_poblacion)
34
35         # Actualizamos el mejor valor de toda la historia
36         if valores_poblacion[indice_maximo] > mejor_valor:
37             mejor_valor = valores_poblacion[indice_maximo].copy()
38
39         # Obtenemos el mejor individuo de la generaci3n
40         mejorIndividuoGenAnterior = poblacion[indice_maximo].copy()
41
42         # Si  $P(t) == P(t-1)$ 
43         if np.array_equal(poblacion, hijos):
44             d -= 1
45         # Si  $d < 0 \rightarrow$  diverge  $P(t)$  y  $d = 24/4$ 
46         if d < 0:
47             # En el arranque los valores de un cromosoma corresponden al mejor
48             # individuo de la generaci3n anterior y el resto ser3n aleatorios
49             d = 24/4
50             poblacion = Utils.inicializar_poblacion(Utils.POBLACION_INICIAL - 1)
51             poblacion[0] = mejorIndividuoGenAnterior.copy()
52
53     return mejor_valor
```

3.2. Parámetros

Los parámetros que utilizaremos para este algoritmo son los siguientes:

Parámetro	Valor
Población Inicial	24
Alpha (α)	0.05
Número de Iteraciones sin mejora	300

Cuadro 12: Parámetros del algoritmo genético CHC.

3.3. Análisis del algoritmo

Este algoritmo utiliza la técnica de cruce *BLX- α* para introducir diversidad en la población y evitar la convergencia prematura. El algoritmo CHC mantiene un equilibrio entre diversidad y convergencia utilizando varias reinicializaciones de la población.

En cuanto a la capacidad de exploración, el algoritmo CHC consigue potenciar esta característica debido a su mecanismo de reinicialización. Este permite volver a reiniciar la población de individuos cuando convergen todos ellos un número determinado de veces.

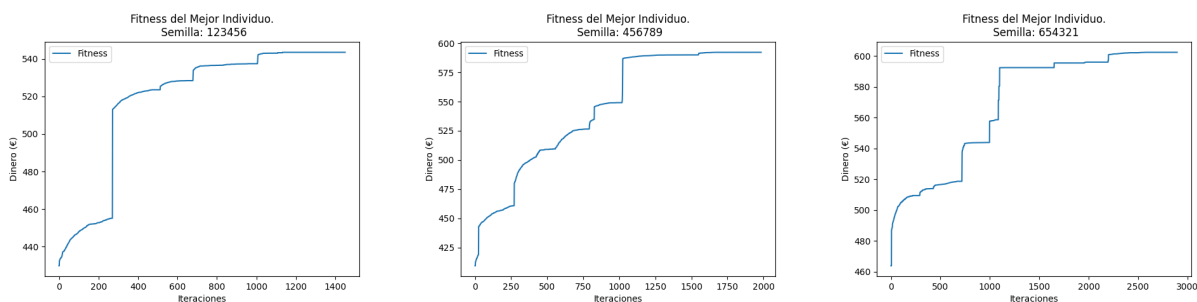
Por otro lado, la capacidad de explotación del algoritmo viene dada por el operador de cruce que permite explorar el entorno de las soluciones ya encontradas generando hijos similares a los padres. Además, para aumentar esta explotación, al reiniciar también se copia el mejor individuo en la población siguiente para conservar una especie de heurística sobre la mejor solución encontrada en la iteración anterior.

3.4. Resultados

3.4.1. Problema Real

Fitness del mejor individuo

A continuación se observa el *fitness* en las diferentes semillas probadas con el algoritmo CHC sobre los datos del problema real.

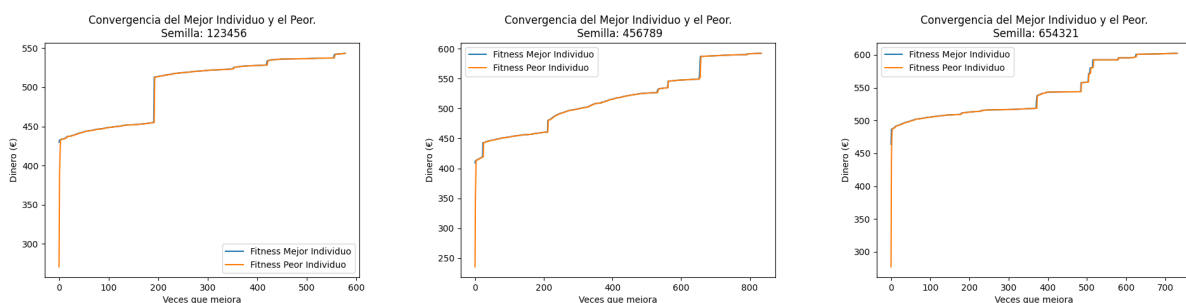


Cuadro 13: *Fitness* del mejor individuo en cada iteración del algoritmo sobre los datos del problema real con el algoritmo genético CHC.

Podemos ver que, como en el anterior algoritmo, la mejora del *fitness* es escalonada. Sin embargo, en este caso podemos ver como existen periodos de una mayor estabilidad tras cada mejora. Esto puede ser debido al contador de reinicios del algoritmo CHC, que reinicia la población solo cuando han pasado un número concreto de poblaciones iguales.

Convergencia del mejor y peor individuo

En este párrafo veremos el *fitness* del mejor y peor individuo de la población en cada iteración en la que el mejor individuo se actualiza con los datos del problema real.

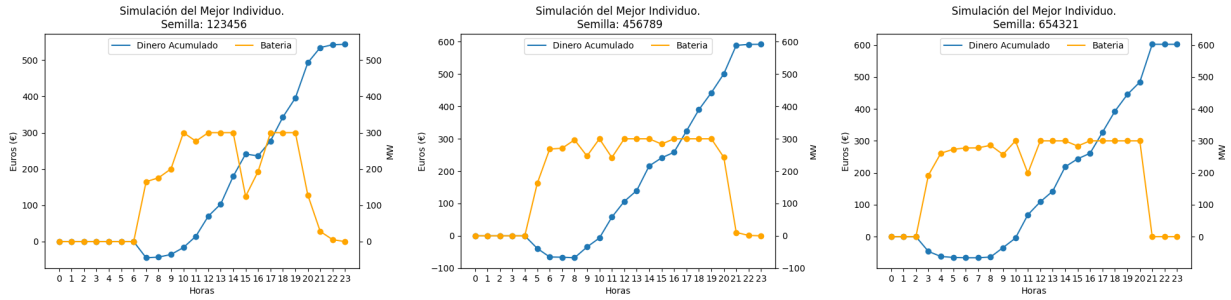


Cuadro 14: Convergencia del *fitness* del mejor y peor individuo en cada iteración del algoritmo sobre los datos del problema real con el algoritmo CHC.

En este caso podemos observar que el algoritmo CHC converge prematuramente en comparación con el algoritmo anterior. En teoría debería de retardar la convergencia debido a su método de reinicialización, pero en las gráficas podemos ver que para este problema, la distancia entre el peor y mejor individuo se reduce mucho muy rápidamente.

Simulación del problema

Por último, veremos como se muestra la capacidad de la batería y el dinero obtenido después de cada hora del día en la simulación de la solución obtenida con los datos del problema real.



Cuadro 15: Simulación del mejor individuo sobre los datos del problema real con el algoritmo genético CHC.

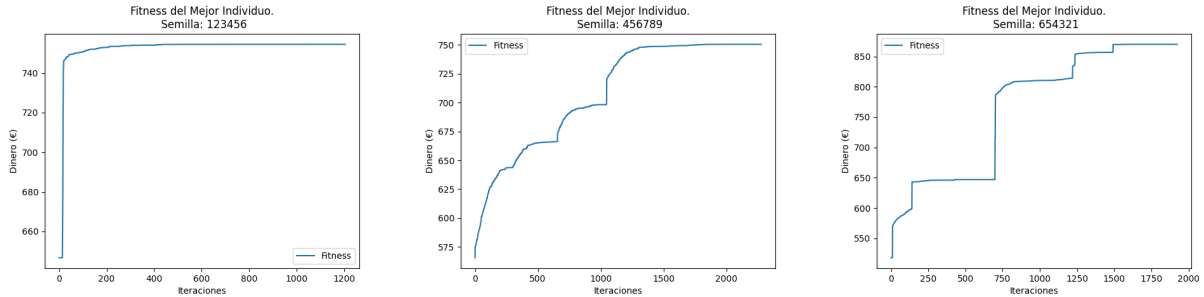
Observamos que, como en el algoritmo anterior, las gráficas son similares a las de los algoritmos de búsqueda tabú y enfriamiento simulado que vimos en la práctica 1.

Podemos notar una diferencia con la gráfica del algoritmo anterior ya que la segunda semilla no comienza comprando energía y llenando la batería. Por lo demás, el algoritmo CHC llega a conseguir un resultado un poco mejor que el genético básico con un número de llamadas a la función de evaluación mucho menor.

3.4.2. Problema Aleatorio

Fitness del mejor individuo

En las siguientes gráficas veremos el *fitness* del mejor individuo para el problema de datos aleatorios. Podemos observar que, en el problema aleatorio, los periodos de estabilidad del algoritmo son mucho más



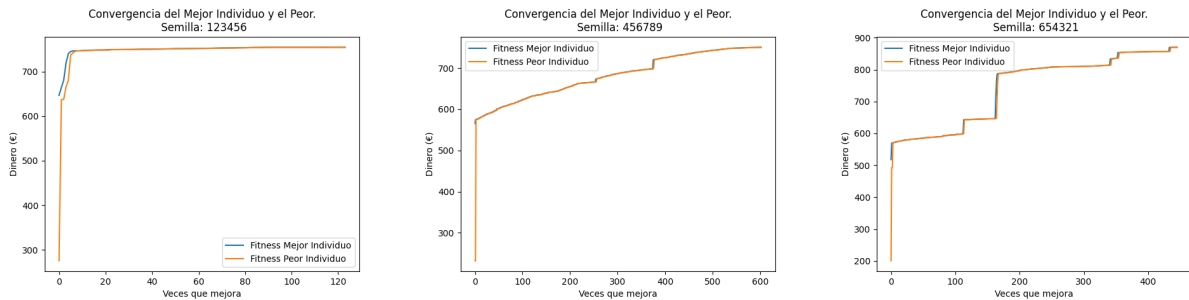
Cuadro 16: *Fitness* del mejor individuo en cada iteración del algoritmo sobre los datos del problema aleatorio con el algoritmo genético CHC.

prolongados que en el de datos reales. Comparándolo con el algoritmo genético, podemos ver que la mejora del mejor individuo se realiza, sobre todo, en unos pocos escalones. Esto puede ser debido a la reinicialización de la población del algoritmo.

Cabe destacar el borde redondeado de los codos de cada gráfica, esta característica puede ser debida a la pequeña mejora que proporciona el cruce $blx-\alpha$. Puede indicarnos que el cruce que hemos elegido está mal escogido o tiene un α demasiado pequeño.

Convergencia del mejor y peor individuo

A continuación se muestran los *fitness* del mejor y peor individuo para el problema de datos aleatorios.

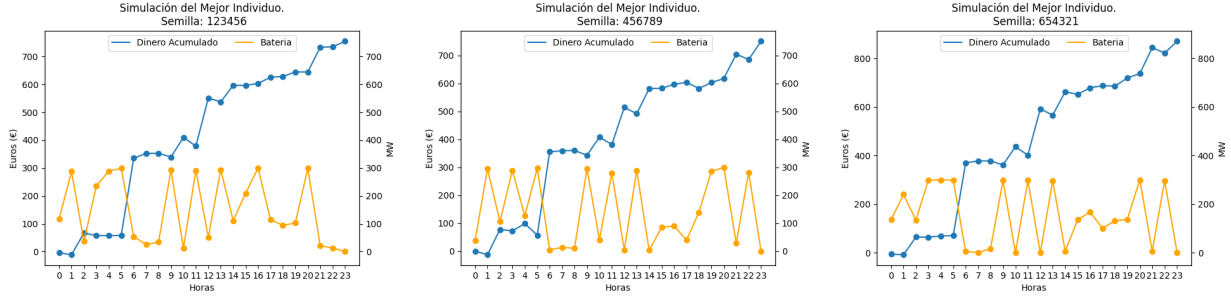


Cuadro 17: Convergencia del *fitness* del mejor y peor individuo en cada iteración del algoritmo sobre los datos del problema aleatorio con el algoritmo genético CHC.

En estas gráficas podemos ver que, como en el problema de datos reales, el peor individuo converge con el mejor individuo muy rápidamente, probablemente por las mismas razones que he detallado anteriormente.

Simulación del problema

Por último, veremos el estado de la batería y la ganancia obtenida cada hora por la solución obtenida para el problema de datos aleatorios.



Cuadro 18: Simulación del mejor individuo sobre los datos del problema aleatorio con el algoritmo genético CHC.

En estas gráficas podemos ver que, como en el algoritmo anterior, la capacidad de la batería no sigue un patrón determinado debido a la naturaleza aleatoria de los datos del problema. Las gráficas se parecen, también, a las obtenidas por el algoritmo de búsqueda tabú de la práctica 1.

3.5. Experimentación

En esta sección anotaremos los valores de los diferentes parámetros escogidos para el algoritmo CHC. Experimentaremos sobre los siguientes parámetros:

3.5.1. Parámetro α del operador de cruce

El operador de cruce que hemos escogido es el BLX- α , este operador tiene un parámetro (α) que permite aumentar o disminuir la diversidad de la población escogida. He probado los siguientes valores para él:

Parámetro Alpha					
0.001	0.01	0.05	0.075	0.1	0.2

Cuadro 19: Parámetros α escogidos para el operador de cruce para estudiar el funcionamiento del algoritmo CHC.

3.5.2. Número de Iteraciones sin mejora

En este apartado experimentaremos con el número de iteraciones sin mejora del algoritmo CHC. Los valores escogidos han sido los siguientes:

Número de Iteraciones sin mejorar						
100	200	300	500	1000	1500	2000

Cuadro 20: Número de iteraciones que se han utilizado para estudiar el funcionamiento del algoritmo CHC.

Finalmente, he elegido $\alpha = 0,05$ y el número de iteraciones sin mejora será de 300 ya que esta ha sido la combinación que mejores resultados ha obtenido (31).

4. Algoritmo Genético Multimodal

4.1. Código

4.1.1. Función de Clearing

```
1 def clearing(poblacion, israndom):
2     # Realizamos la seleccion sobre los individuos dominantes, por tanto, hay que limpiar a los individuos
3     # de cada nicho. Obtenemos los valores del fitness de la poblacion
4     fitness_poblacion = Utils.fitnessPoblacion(poblacion,israndom)[0]
5     # Clasificación de la población de forma descendente
6     indices_ordenados = np.argsort(-fitness_poblacion)
7     # Se escoge al mejor individuo y se compara con el resto de forma descendente.
8     # Aquellos individuos que estén dentro de su radio, quedan eliminados.
9     for i in range(len(fitness_poblacion)):
10        if fitness_poblacion[indices_ordenados[i]] > 0:
11            numGanadores = 1
12            for j in range(i+1,len(fitness_poblacion)):
13                if fitness_poblacion[indices_ordenados[j]] > 0 and \
14                    Utils.distanciaEuclidea(poblacion[indices_ordenados[i]],
15                                            poblacion[indices_ordenados[j]]) < Utils.RADIO_CLEARING:
16                    if numGanadores < Utils.KAPPA:
17                        numGanadores += 1
18                    else:
19                        fitness_poblacion[indices_ordenados[j]] = 0
20    # Devuelve los individuos que no han sido eliminados.
21    return poblacion[fitness_poblacion[indices_ordenados] != 0]
```

4.2. Parámetros

Como este algoritmo es una modificación del algoritmo genético básico, algunos parámetros son los mismos:

Parámetro	Valor
Población Inicial	29
Porcentaje de Mutación	10 %
Número de Iteraciones sin mejora	100
Individuos de Élite	5
Porcentaje de la población para el cruce	80 %
Generaciones antes de hacer clearing	10
Radio de Clearing	100
Kappa	2

Cuadro 21: Parámetros del algoritmo genético multimodal.

4.3. Análisis del algoritmo

Este algoritmo es una modificación del algoritmo genético básico. Consiste en cruzar tan solo los individuos dominantes de la población, es decir, los que sobrevivan a la función de *clearing*. Este cruce se producirá con una probabilidad del 80 %, en caso de que no se produzca, se copian los individuos directamente a la siguiente población. Este mecanismo, al matar a los individuos que son parecidos entre sí, favorece la diversidad de la población. Sin embargo, más adelante veremos que el algoritmo padece de convergencia prematura.

En teoría, al mantener a un subconjunto de individuos en cada solución lo suficientemente distinta entre sí, favorecemos a la diversidad genética y, consecuentemente, aumentaríamos el grado de exploración del algoritmo al cruzar los individuos. Por otro lado, el grado de explotación de las soluciones se mantiene como en el algoritmo genético básico, ya que el operador de cruce es el mismo (cruce en dos puntos).

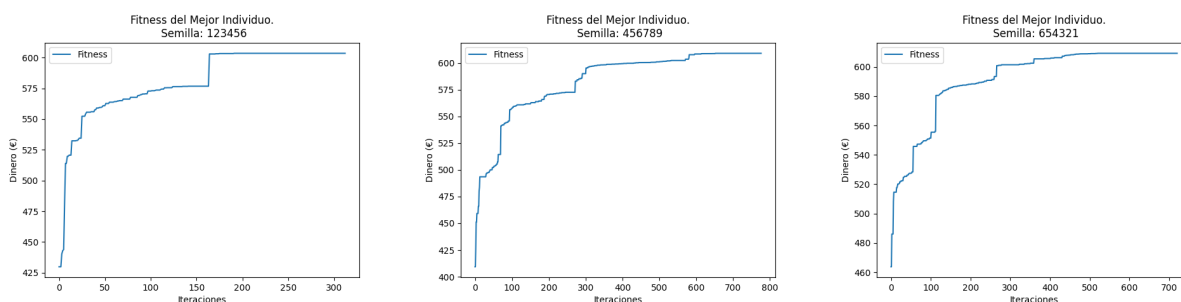
Para evitar la convergencia prematura, se han añadido 2 individuos aleatorios a la población después de hacer el cruce. Esto permite mantener la diversidad genética y obtener mejores resultados.

4.4. Resultados

4.4.1. Problema Real

Fitness del mejor individuo

A continuación observamos las gráficas del *fitness* del mejor individuo sobre las diferentes ejecuciones del problema de datos reales.



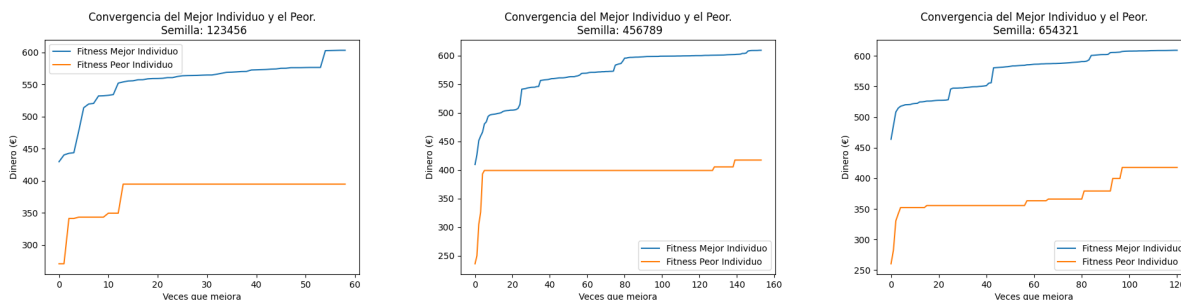
Cuadro 22: *Fitness* del mejor individuo en cada iteración del algoritmo sobre los datos del problema real con el algoritmo genético Multimodal.

En estas gráficas podemos observar que, como en el algoritmo genético básico, el grado de adaptación del mejor individuo mejora considerablemente. Podemos observar que la mejora del individuo se realiza de forma escalonada, al igual que en el algoritmo básico.

El resultado es el esperado, puesto que la base del algoritmo es la misma que el algoritmo genético básico, es normal que la forma de la gráfica sea parecida. Sin embargo, al añadir dos individuos aleatorios cada vez que se generan los hijos, el resultado mejora considerablemente llegando a superar los 600€ de beneficio.

Convergencia del mejor y peor individuo

Ahora mostraré las gráficas relativas al *fitness* del mejor y peor individuo de las diferentes ejecuciones del algoritmo multimodal sobre los datos del problema real.

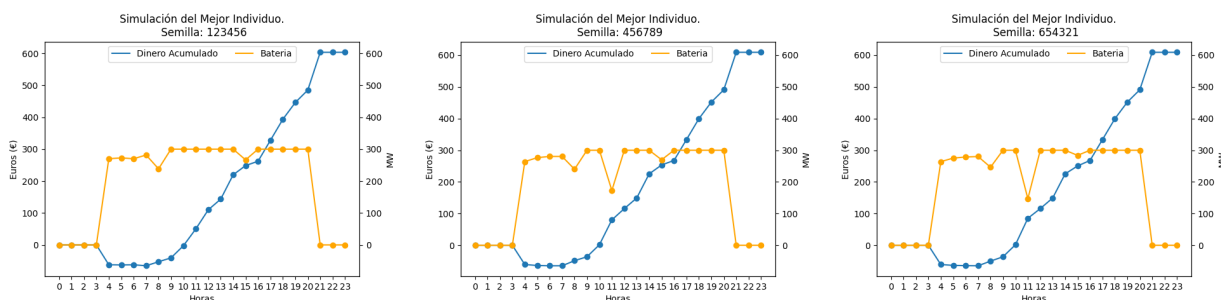


Cuadro 23: Convergencia del *fitness* del mejor y peor individuo en cada iteración del algoritmo sobre los datos del problema real con el algoritmo Multimodal.

En esta variación del algoritmo básico vemos que, al generar diferentes nichos, el peor individuo se mantiene alejado del mejor de la población. Esto es debido al método de *clearing* y por añadir dos individuos aleatorios a la población. Esto permite generar la suficiente diversidad para que la población no converja en un solo individuo.

Simulación del problema

Por último, se muestran las gráficas que realizan la simulación de los beneficios y estado de la batería por cada hora que proporciona la solución dada por el mejor individuo de cada ejecución del algoritmo sobre los datos del problema real.



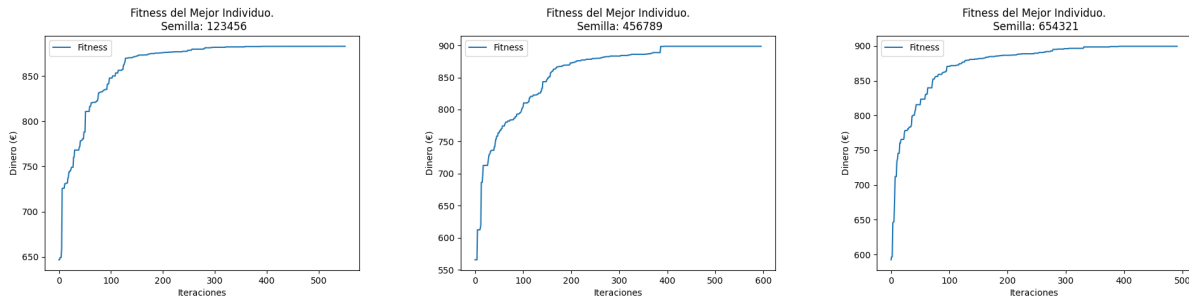
Cuadro 24: Simulación del mejor individuo sobre los datos del problema real con el algoritmo genético Multimodal.

En la simulación de la mejor solución generada por el mejor individuo de cada ejecución podemos ver que se asemeja mucho a las obtenidas por el algoritmo genético básico. Sin embargo, podemos observar que el algoritmo decide llenar la batería mucho antes que en la otra versión del algoritmo. Esto le permite obtener un resultado superior a los otros dos algoritmos que hemos visto en esta práctica.

4.4.2. Problema Aleatorio

Fitness del mejor individuo

En este párrafo se muestra el *fitness* del mejor individuo de las tres ejecuciones realizadas sobre los datos del problema aleatorio.

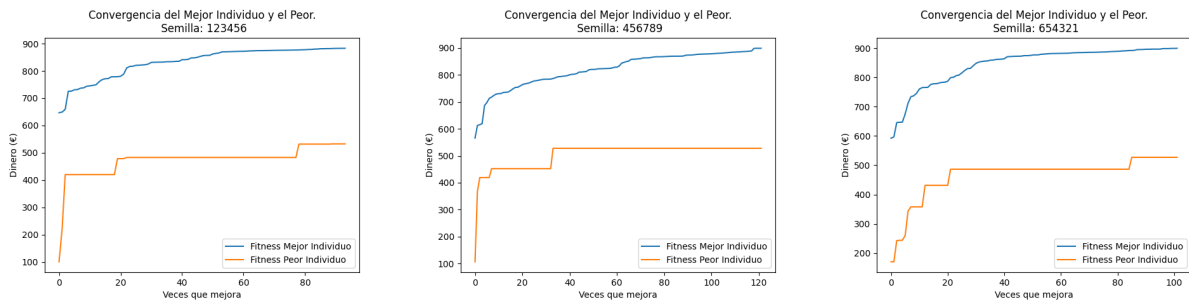


Cuadro 25: *Fitness* del mejor individuo en cada iteración del algoritmo sobre los datos del problema aleatorio con el algoritmo genético Multimodal.

Las gráficas son muy parecidas a las del problema de datos reales. El mejor individuo mejora escalonadamente hasta llegar a converger a un valor cercano a 900e.

Convergencia del mejor y peor individuo

A continuación se muestran las gráficas del *fitness* del peor y mejor individuo de cada ejecución del algoritmo sobre los datos del problema aleatorio.

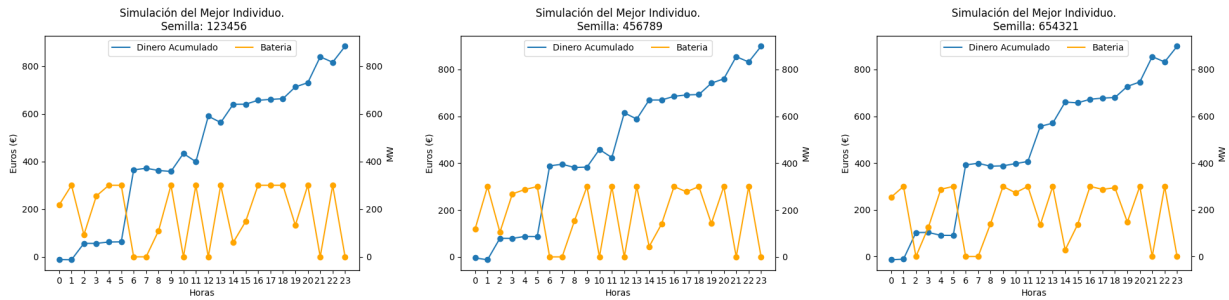


Cuadro 26: Convergencia del *fitness* del mejor y peor individuo en cada iteración del algoritmo sobre los datos del problema aleatorio con el algoritmo genético Multimodal.

Al igual que en el problema anterior, el peor individuo se mantiene alejado del mejor de la población debido a la combinación del método de *clearing* junto a la adición de dos individuos aleatorios a la población tras cruzar a los padres seleccionados.

Simulación del problema

Finalmente, mostraremos la simulación del estado de la batería y de las ganancias obtenidas en cada ejecución con los datos del problema aleatorio.



Cuadro 27: Simulación del mejor individuo sobre los datos del problema aleatorio con el algoritmo genético Multimodal.

Las gráficas de simulación son similares a las de los otros algoritmos que hemos visto en esta práctica al igual que a las de los algoritmos de búsqueda tabú y enfriamiento simulado. Podemos observar que la ganancia de la mejor solución llega a superar los 800 euros, siendo esta la solución más alta obtenida en la práctica.

4.5. Experimentación

Para este algoritmo dejaremos los parámetros que hemos obtenido con la experimentación del algoritmo genético básico y nos centraremos en experimentar con los valores siguientes:

4.5.1. Número de generaciones para hacer clearing

Los valores que hemos escogido para iterar para encontrar el mejor número de generaciones tras el que hacer *clearing* han sido:

Cantidad de generaciones antes de hacer Clearing						
10	20	50	100	150	200	300

Cuadro 28: Cantidad de generaciones que se han utilizado para estudiar el funcionamiento del algoritmo multimodal.

4.5.2. Radio de Clearing

En cuanto al radio para generar los nichos hemos elegido los siguientes valores:

Radio de Clearing								
10	20	30	50	80	100	120	150	200

Cuadro 29: Radios para hacer *clearing* que se han utilizado para estudiar el funcionamiento del algoritmo CHC.

4.5.3. Kappa

Por último, el número de individuos que dominan cada nicho se ha escogido entre los siguientes valores:

Número de Kappas									
2	3	4	5	6	7	8	9	10	

Cuadro 30: Número de iteraciones que se han utilizado para estudiar el funcionamiento del algoritmo CHC.

5. Análisis de Resultados

5.1. Problema Real

Problema Real						
Algoritmo	Ev. Medias	Ev. Mejor	Desv. Evaluaciones	Mejor Fitness	Media Fitness	Desv. Fitness
Búsqueda Local	53.8	52	1.79	361.1	309.1	31.54
G. Básico	22525.0	21725	708.87	605.02	589.79	14.7
G. Multimodal	36971.0	19155.0	15525.34	609.15	607.26	3.26
CHC	33760.0	23200	11699.00	602.4	579.38	31.51

Cuadro 31: Comparación de métricas para los datos del problema real.

En la tabla 31 podemos observar los resultados obtenidos en los algoritmos genéticos implementados en esta práctica aplicados al problema de datos reales. Además, también se compara con los resultados obtenidos por la búsqueda local de primero el mejor que vimos en la práctica 1.

En cuanto a los resultados obtenidos, podemos ver que el algoritmo genético multimodal obtiene la mayor ganancia respecto a todos los algoritmos. Sin embargo, el número de llamadas a la función de evaluación que realiza es superior a los demás.

En cuanto a la robustez de estos algoritmos, podemos observar que, en cuanto a los resultados, generan soluciones muy parecidas entre sí. El algoritmo genético multimodal consigue una mayor robustez en relación con los resultados obtenidos. Este algoritmo tiene una desviación típica de 3,26e que es inferior a la del algoritmo genético básico y a la del CHC (14,7 y 31,51e respectivamente).

Sin embargo, en cuanto a la robustez en relación a las llamadas a la función de evaluación, el algoritmo multimodal tiene una desviación típica demasiado alta siendo poco fiable para ejecutarlo unas veces concretas. Por otra parte, el algoritmo genético básico obtendría una mayor robustez en este aspecto.

5.2. Problema Aleatorio

Problema Aleatorio						
Algoritmo	Ev. Medias	Ev. Mejor	Desv. Evaluaciones	Mejor Fitness	Media Fitness	Desv. Fitness
Búsqueda Local	53.6	53	0.55	523.69	430.15	71.55
G. Básico	18025.0	14575	4010.29	865.95	853.04	21.53
G. Multimodal	33478.66	30132	3193.92	893.68	893.68	9.33
CHC	28826.66	19264	8754.94	870.03	791.7	67.86

Cuadro 32: Comparación de métricas para los datos del problema aleatorio.

Por otro lado, en la tabla 32 podemos observar los resultados obtenidos en los algoritmos genéticos implementados en esta práctica aplicados al problema de datos aleatorios. También se compara con los resultados obtenidos por la búsqueda local de la primera práctica.

En cuanto a los resultados obtenidos, podemos ver que, al igual que en el problema anterior, el algoritmo genético multimodal consigue mejores resultados que sus otros competidores. Además, consigue una desviación típica de 9,33e. Esto indica una alta robustez por parte del algoritmo genético multimodal.

En cuanto a la robustez de los resultados de los otros dos algoritmos genéticos, podemos observar como es más alta que la del multimodal, aunque no tan altas como la de la búsqueda local.

Por otro lado, si observamos el número de llamadas a la función de evaluación, notamos que el multimodal realiza, de media, un número mayor de evaluaciones. En este aspecto, el algoritmo CHC realiza un número muy inferior de evaluaciones respecto a los otros dos algoritmos genéticos. Esto puede ser debido al problema de la convergencia prematura que hemos visto anteriormente en este algoritmo.

La mayor robustez en cuanto al número de llamadas a la función de evaluación la consigue el algoritmo multimodal, ya que tiene una desviación típica de 3193,92 llamadas a la función. Sin embargo, en comparación con la búsqueda local, este número sigue siendo demasiado alto.

6. Artículos Relacionados

6.1. Genético Básico

Este artículo ([Shopova and Vaklieva-Bancheva, 2006]) presenta en detalle un algoritmo genético llamado BASIC, diseñado para aprovechar los esquemas genéticos conocidos y ser capaz de resolver numerosos problemas de optimización. BASIC GA sigue los pasos comunes de los algoritmos genéticos e incluye esquemas de representación real para variables enteras y reales. También se aplican tres esquemas de selección sesgada para la reproducción, cuatro para la recombinación y tres para la mutación. Además, se aborda un nuevo esquema de selección para la sustitución.

BASIC GA puede ajustarse fácilmente a problemas concretos mediante la adaptación de sus parámetros globales y locales, y ofrece la posibilidad de ampliar los operadores genéticos con nuevos esquemas. Se han resuelto una serie de problemas de optimización para probar su capacidad, utilizando funciones de penalización estáticas y dinámicas para manejar todo tipo de restricciones. Las soluciones obtenidas son comparables con otros algoritmos genéticos y técnicas de solución.

Por último, menciona algunos ejemplos de aplicaciones de algoritmos genéticos en la optimización de problemas de ingeniería, como la optimización de la fermentación de cerveza o la programación de trabajos en talleres.

6.2. CHC

En este artículo [Rathee and Ratnoo, 2020] se presenta un algoritmo genético multiobjetivo para la selección de características, llamado MOCHC-FS. El objetivo es seleccionar características no redundantes e informativas para el preprocesamiento de conjuntos de datos antes de aplicar algoritmos de minería de datos.

El algoritmo utiliza una combinación de clasificación no dominada y algoritmos genéticos CHC para obtener un conjunto de soluciones no dominadas que cumplen con ambos objetivos de precisión y tasa de reducción.

Este algoritmo se ha validado en veinte conjuntos de datos disponibles en el repositorio de datos de UCI y se demuestra que puede encontrar soluciones óptimas que cumplen ambos objetivos.

Por último, se extrae un subconjunto de características de las soluciones no dominadas y se registran las tasas de precisión y reducción para varios conjuntos de datos experimentales utilizando el algoritmo de clasificación KNN en las características seleccionadas.

6.3. Multimodal

El artículo [Dilettoso and Salerno, 2006] propone un nuevo algoritmo genético de nicho llamado *Self-adaptive NGA* (SANGA), que utiliza una variable adicional en el problema de optimización para estimar el radio del nicho en lugar de asignarlo a priori, como se hace en otros algoritmos genéticos del mismo estilo.

SANGA se combina con el método de búsqueda de patrones determinista para formar un método híbrido de optimización que funciona bien en la optimización de funciones multimodales y en el diseño de dispositivos electromagnéticos. La capacidad de estos algoritmos para localizar múltiples óptimos puede ser útil para los problemas de optimización del mundo real que a menudo exhiben múltiples óptimos. Este algoritmo consigue identificar correctamente los nichos y estimar sus radios con un bajo número de evaluaciones de la función objetivo, y se puede combinar con el método de búsqueda de patrones determinista.

7. Conclusiones

En esta práctica hemos visto como implementar tres algoritmos genéticos diferentes: un algoritmo genético básico, el algoritmo CHC y, por último, mejorar el algoritmo genético básico añadiéndole una función de *clearing* para obtener mayor diversidad genética.

Atendiendo a la tabla de resultados, podemos ver una clara mejora en relación a la primera práctica donde vimos las búsquedas locales. Los algoritmos genéticos mejoran los resultados obtenidos por la búsqueda local drásticamente.

Entre los algoritmos genéticos, destacan los resultados del algoritmo genético multimodal, ya que obtiene los mejores resultados en ambos problemas. Sin embargo, el número de llamadas a la función de evaluación que realiza es superior al de los otros dos algoritmos poblacionales, sobretodo al de CHC.

El algoritmo CHC no ha conseguido alcanzar los resultados obtenidos por sus rivales. Esto puede ser debido a una mala elección de parámetros para el cruce $BLX-\alpha$ o una mala gestión de la diversidad genética, ya que, en las gráficas, hemos comprobado que sufre de una convergencia prematura.

Finalmente, podemos concluir que, tanto para el problema de datos reales como para el de datos aleatorios, deberíamos utilizar el algoritmo genético multimodal ya que ofrece unos mejores resultados a cambio de unas pocas de evaluaciones más que en genético básico.

Referencias

- [Dilettoso and Salerno, 2006] Dilettoso, E. and Salerno, N. (2006). A self-adaptive niching genetic algorithm for multimodal optimization of electromagnetic devices. *IEEE Transactions on Magnetics*, 42(4):1203–1206.
- [Rathee and Ratnoo, 2020] Rathee, S. and Ratnoo, S. (2020). Feature selection using multi-objective chaotic genetic algorithm. *Procedia Computer Science*, 167:1656–1664.
- [Shopova and Vaklieva-Bancheva, 2006] Shopova, E. G. and Vaklieva-Bancheva, N. G. (2006). Basic—a genetic algorithm for engineering problems solution. *Computers & chemical engineering*, 30(8):1293–1309.