



**Escuela Técnica Superior de Ingeniería
Universidad de Huelva**

GRADO EN INGENIERÍA INFORMÁTICA

OPTIMIZACIÓN DE UNA PLANTA SOLAR

Algoritmos basados en Entornos y Trayectorias

Autora:
Alba Márquez-Rodríguez

Profesor:
Miguel Ángel Rodríguez Román

Huelva, Marzo 2023

Índice

1. Enunciado	2
1.1. Objetivos	2
1.2. Diseño, justificación y desarrollo de un Algoritmo de optimización de una planta solar	2
2. Base	3
2.1. Esquema de representación	3
2.2. Función de evaluación	3
2.3. Generación de la solución inicial	5
2.4. Operador de movimiento	6
3. Algoritmos	7
3.1. Algoritmos de Heurística Constructiva	7
3.1.1. Greedy	7
3.2. Algoritmos de Búsquedas Locales	8
3.2.1. Búsqueda Aleatoria	8
3.2.2. El Mejor	8
3.2.3. El Primero El Mejor	9
3.2.4. Búsqueda Local VND	9
3.2.5. Simulated Annealing	9
3.2.6. Búsqueda Tabú	10
4. Experimentación	11
4.1. Algoritmos de Heurística Constructiva	11
4.1.1. Greedy	11
4.2. Algoritmos de Búsquedas Locales	12
4.2.1. Búsqueda Aleatoria	12
4.2.2. El Mejor	14
4.2.3. El Primero El Mejor	16
4.2.4. Búsqueda Local VND	19
4.2.5. Simulated Annealing	20
4.2.6. Búsqueda Tabú	21
5. Análisis	23
5.1. Algoritmos de Heurística Constructiva	23
5.1.1. Greedy	23
5.2. Algoritmos de Búsquedas Locales	24
5.2.1. Búsqueda Aleatoria	24
5.2.2. El Mejor	25
5.2.3. El Primero El Mejor	26
5.2.4. Búsqueda Local VND	27
5.2.5. Simulated Annealing	27
5.2.6. Búsqueda Tabú	28
6. Resultados	29
6.1. Problema 1	29
6.2. Problema 2	29
7. Conclusiones	30

1. Enunciado

1.1. Objetivos

El objetivo de esta práctica es estudiar el funcionamiento de los Algoritmos de Búsqueda Aleatoria, El Mejor, Primero El Mejor, Enfriamiento Simulado y Búsqueda Tabú. Para ello, se requerirá implementar estos algoritmos, para resolver un problema de optimización una planta solar. El comportamiento de los algoritmos de OCH implementados deberá compararse con un Algoritmo Greedy.

1.2. Diseño, justificación y desarrollo de un Algoritmo de optimización de una planta solar

Este algoritmo debe maximizar los ingresos diarios para una instalación de planta solar que tiene una capacidad total de 1000 m² con un rendimiento del 20 % de la energía recibida. La planta posee una batería que puede almacenar energía antes de venderse de 300 kWh de capacidad.

Recibirás el día anterior una previsión meteorológica con la radiación por hora en un vector R (24 enteros con w por m²) y el precio previsto de la energía cada hora en P (24 valores reales en euros por cada kWh).

El algoritmo debe decidir que decidir por cada hora cuánta energía se venderá / comprará, la energía que se produce cada hora (o compra) y no se vende se almacena en la batería (que tiene un límite de 300kwh y comienza vacía todos los días). Toda la energía que no almacena ni vende se desperdicia.

Precio por día por kWh: a continuación, se muestran los precios reales para el día de hoy en la península, genera una lista de valores a mano deduciéndolos de la gráfica en una variable o en un fichero con los valores en céntimos de euro por horas (24 valores) . Asume estos valores como precios de venta de los kWh producidos en planta.

Problema 1(datos reales) pc y pv (precio por hora en cts por kW) y r (radiación por hora en W * m²)

```
1 pc = [26, 26, 25, 24, 23, 24, 25, 27, 30, 29, 34, 32, 31, 31, 25, 24, 25, 26, 34, 36, 39, 40, 38, 29]
2 pv = [24, 23, 22, 23, 22, 22, 20, 20, 20, 19, 19, 20, 19, 20, 22, 23, 22, 23, 26, 28, 34, 35, 34, 24]
3 r = [0, 0, 0, 0, 0, 0, 0, 100, 313, 500, 661, 786, 419, 865, 230, 239, 715, 634, 468, 285, 96, 0, 0]
```

Problema 2 (aleatorizado)

```
1 Pc= [7,7,50,25,11,26,48,45,10,14,42,14,42,22,40,34,21,31,29,34,11,37,8,50]
2 Pv= [1,3,21,1,10,7,44,35,4,1,23,12,30,7,30,4,9,10,6,9,8,27,7,10]
3 r =
[274,345,605,810,252,56,964,98,77,816,68,261,841,897,75,489,833,96,117,956,970,255,74,926]
```

Una posible solución del experto podría ser: 00- Vender 30 kwh 01- Comprar 100 kwh ... 23- Vender 0 Kwh (también es posible no hacer nada)

El resultado esperado es la cantidad en euros conseguida en el día, calculada como el precio por kWh vendidos en esa Hora. Dado que lo que podamos vender dependerá de la carga de la batería en dicho momento no se puede saber de antemano el resultado, por lo que el resultado será calculado mediante una simulación usando los precios y radiación. La simulación interpretará las instrucciones en la solución actual a evaluar calculando cada hora la carga de la batería y la cantidad en euros, que puede ser negativa.

2. Base

El siguiente código es total o parcialmente reutilizable. Codifica con la suficiente generalización para su uso en los algoritmos de esta y siguientes prácticas con la denominación, parámetros y retorno que estimes oportuno. Ten en cuenta el uso de semillas preestablecidas para el generador de números aleatorios.

Las soluciones deben ser siempre válidas (dar un valor en euros en la simulación sin errores).

2.1. Esquema de representación

El esquema de representación será un vector:

- **solución:** representa un vector de 24 elementos en el que cada elemento representa cada hora del día desde las 00h a las 23h. En cada elemento se almacenará un número desde -100 a 100. Las acciones a seguir según este número serán:

- + (Positivo): Compra el porcentaje de batería introducido respecto a la fracción de almacenaje disponible, después de introducir la energía recibida en esa hora. Al hacer la compra sobre el porcentaje de energía almacenada tras añadir la nueva energía, solo se superará la capacidad de la batería con la nueva energía a almacenar, en este caso, esa energía sobrante se venderá. Es decir, si a las 07h tenemos +20 y la capacidad de la batería está al 0. Entonces el operario comprará un 20 % de la batería total. Por otro lado, si a las 09h tenemos +30 y la batería se encuentra al 30 % de su capacidad. Entonces comprará el 30 % del (100-30 %) = 30 % del 70 % de la capacidad de la batería total que es igual al 21Si a las 11h tenemos +50 y la batería se encuentra al 90 % pero tras introducir la energía queda al 110 %, el 10 % restante se venderá y habrá que comprar el 50 % del (100-100 %) = 30 % del 0 % = 0.
- 0: Guarda la energía recibida. En el caso de guardar energía y que haya más energía que la que se puede almacenar, se venderá la energía sobrante. Si a las 18h tenemos la capacidad de la batería al 30 % y llega para incrementar al 20 %, se almacena y lo tendremos al 50 %. Por otro lado, si estamos al 80 % y llega un 30 %, se almacenará hasta completar el 100 % y se venderá el 10 % sobrante.
- - (Negativo): Vende el porcentaje de energía almacenada introducido. Es decir, si estamos a las 10h y tenemos -40, esto quiere decir que de la energía que tenemos almacenada a las 10, supongamos un 20 % del total, venderemos un 40 %. Es decir el 40 % del 20 % = 8 % del total. A las 19h si tenemos la batería al 0 % y tenemos que vender un 10 %, el 10 % del 0 % es 0 por lo que no habrá problemas.

Ejemplo de la tabla del esquema de representación:

1	[00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23]
2	[0 0 0 0 0 0 0 20 10 30 -40 50 -20 6 -30 0 -50 10 0 10 0 0 0 0]

En nuestro programa no será necesario tener las horas en la tabla, por lo que será un único vector, para el entendimiento del esquema de representación se ha decidido colocar las horas en el ejemplo superior, pero será como el siguiente:

1	[0 0 0 0 0 0 0 20 10 30 -40 50 -20 6 -30 0 -50 10 0 10 0 0 0 0]
---	---

2.2. Función de evaluación

La función evaluación recibe como argumentos:

- **Solucion:** vector de 24 elementos que indica la acción a realizar a determinada hora.
- **Radiación:** vector que indica por hora la radiación
- **Precios:** vector prevision por hora precios.

Primero se calcula la cantidad de energía producida en cada hora a partir de la radiación y la capacidad de la planta solar. Luego se itera sobre las 24 horas para determinar cuánta energía se venderá, comprará o almacenará en cada hora de acuerdo con la solución propuesta. Por último, se calcula el ingreso obtenido en el día a partir de los precios de la energía y la cantidad de energía vendida y comprada en cada hora.

La función devuelve el ingreso obtenido en el día como resultado de evaluar la solución propuesta.

```

49         # 3. Vende energía
50
51     bateria_vender_porcentaje = (abs(solucion[hora]) * bateria_capacidad_porcentaje)
52     / 100
53
54     bateria_capacidad_porcentaje -= bateria_vender_porcentaje
55
56     if bateria_capacidad_porcentaje < 0: # pasa con números decimales muy pequeños
57         por el redondeo
58         bateria_capacidad_porcentaje = round(bateria_capacidad_porcentaje)
59
60     assert 0 <= bateria_capacidad_porcentaje <= 100, "Batería fuera de los posibles
61     límites, al vender: " + str(bateria_capacidad_porcentaje)
62
63     dinero_total += cents_to_euros((porcentaje_a_energía(bateria_vender_porcentaje))
64     * precio_venta_actual)
65
66     # si es la última hora vender todo:
67     if hora == (len(horas) - 1):
68         bateria_vender_porcentaje = 100 * bateria_capacidad_porcentaje / 100
69
70         bateria_capacidad_porcentaje -= bateria_vender_porcentaje
71
72         dinero_total += cents_to_euros((porcentaje_a_energía(bateria_vender_porcentaje))
73     * precio_venta_actual)
74
75         assert int(bateria_capacidad_porcentaje) == 0, "La batería no se ha vaciado
76     completamente al terminar el día: " + str(bateria_capacidad_porcentaje)
77
78     bateria_hora[hora] = bateria_capacidad_porcentaje
79     dinero_hora[hora] = dinero_total
80
81     if representar:
82         representar_gráfica(dinero_hora, bateria_hora, precios_venta, precios_compra,
83         img_name)
84
85     return dinero_total

```

Listing 1: Función de Evaluación

2.3. Generación de la solución inicial

En este apartado se verá la elaboración de la función 'solucion_inicial()' Para evitar soluciones novedosas, el algoritmo comienza generando una solución inicial de 24 elementos que contendrán números enteros entre -100 y 100.

Esta función genera un vector de 24 elementos, estos elementos, como se ha indicado, son números enteros entre -100 y 100 que indicará la acción a realizar a cada hora.

```

1 def solucion_inicial():
2     # implementación de la generación de solución inicial
3     solucion = [0] * 24 # Inicializar el vector con 24 ceros
4
5     # Generar una solución valida
6     for hora in range(0,24):
7
8         numero_aleatorio = random.randint(-100,100) # Generar un número aleatorio entre el -100
9         y 100
10
11         assert -100 <= numero_aleatorio <= 100, "Un número no está en los límites"
12
13         solucion[hora] = numero_aleatorio
14
15     return solucion

```

Listing 2: Función Solución Inicial

2.4. Operador de movimiento

El operador de movimiento genera una lista de vecinos a partir de una solución dada. Recibe como parámetro:

- **solucion_actual**: solución a partir de la cual se van a generar nuevos vecinos
- **n_vecinos**: número de vecinos que se quiere generar
- **granularidad**: granularidad definida para la creación de vecinos

Los vecinos se generan cogiendo el vector *solucion_actual*, *elemento* a *elemento* hacia *down* incremento y decrecimiento de la *granularidad*. Si cuando se genere un vecino tras hacer el decremento o incremento de la granularidad indicada se supera el valor límite 100 o -100 entonces este vecino no se considerará para la lista de vecinos generados.

```
1 # se genera un numero de vecinos, por ej 40 -> elegir una posicion y hacer + / -
2 # granularidad de esa posicion, yendo de 1 en 1 o de 10 en 10
3 # Función para generar vecino
4 def generador_vecino(solucion_actual, n_vecinos = 1, granularidad = 10):
5     # implementación de la generación de vecino
6     vecinos = []
7
8     for _ in [n_vecinos]:
9
10        vecino = solucion_actual.copy()
11
12        # incremento
13        for idx, elemento in enumerate(range(len(vecino))):
14            nuevo_vecino = vecino.copy()
15            nuevo_valor = vecino[idx] + granularidad
16            nuevo_vecino[idx] = nuevo_valor
17
18            # Se comprueba que el número resultante est dentro del rango [-100, 100]
19            # vecino[idx] = max(min(vecino[idx], 100), -100)
20
21            #Para evitar que pueda haber valores por encima de 100 (porque es suma), si es
22            #+100 -> no appendiza el valor
23            if nuevo_vecino[idx] <= 100:
24                vecinos.append(nuevo_vecino)
25                assert -100 <= nuevo_vecino[idx] <= 100, "Vecino fuera de los límites: " +
26                str(nuevo_vecino[idx])
27
28        # decremento
29        for idx, elemento in enumerate(range(len(vecino))):
30            nuevo_vecino = vecino.copy()
31            nuevo_valor = vecino[idx] - granularidad
32            nuevo_vecino[idx] = nuevo_valor
33
34            # Se comprueba que el número resultante est dentro del rango [-100, 100]
35            # vecino[idx] = max(min(vecino[idx], 100), -100)
36            # vecinos.append(vecino)
37
38            #Para evitar que pueda haber valores por debajo de -100 (porque es resta), si es
39            #-100 -> no appendiza el valor
40            if nuevo_vecino[idx] >= -100:
41                vecinos.append(nuevo_vecino)
42                assert -100 <= nuevo_vecino[idx] <= 100, "Vecino fuera de los límites: " +
43                str(nuevo_vecino[idx])
44
45        # vecinos.append(vecino)
46
47    return vecinos
```

Listing 3: Función del Operador de Movimiento

3. Algoritmos

3.1. Algoritmos de Heurística Constructiva

En este apartado se aborda un algoritmo de heurística constructiva, una técnica de búsqueda que se utiliza para resolver problemas de optimización combinatoria. Esta técnica consiste en construir una solución paso a paso, utilizando criterios heurísticos para tomar decisiones locales óptimas en cada etapa, con el objetivo de obtener una solución global óptima [1] [2].

Las heurísticas constructivas son más rápidas pero dan soluciones de peor calidad que las de Búsqueda Local (BL). Ambos son procesos de búsqueda efectuados sobre un espacio de soluciones al problema. En los métodos constructivos, el espacio es de soluciones parciales, mientras que en la BL es de soluciones completas (candidatas) [1].

Uno de los algoritmos de heurísticas constructivas más conocidos y utilizados es el algoritmo *Greedy*.

3.1.1. Greedy

El algoritmo *Greedy* se basa en la idea de que, en cada paso, se elige la opción que parece ser la mejor en ese momento, sin considerar necesariamente el impacto de esa elección en etapas posteriores. A pesar de su simplicidad, el algoritmo *Greedy* puede proporcionar soluciones rápidas y efectivas para muchos problemas de optimización [3].

Para efectuar la comparativa de resultados entre los distintos algoritmos de búsqueda, se ha implementado como algoritmo básico, un *Greedy*, siguiendo la heurística de guardar desde el principio hasta que se llene y luego vender en el pico de precio del día todo. A partir de ese momento vender todo.

3.2. Algoritmos de Búsquedas Locales

Los algoritmos de Búsquedas Locales son una técnica de optimización para problemas de optimización combinatoria. La idea detrás de estos algoritmos es la de iniciar la búsqueda desde una solución inicial e iterativamente intentar mejorarlala mediante la exploración del vecindario de la solución actual (espacio de búsqueda). La búsqueda se detiene cuando no se pueden encontrar soluciones mejores dentro del vecindario actual. A diferencia de los algoritmos de Búsqueda Completa, los algoritmos de Búsquedas Locales no exploran todo el espacio de búsqueda, lo que los hace más eficientes para problemas de gran tamaño [3].

La búsqueda Local clásica presenta mucha intensificación (explotación del espacio actual) pero poca diversificación (exploración del espacio distante).

3.2.1. Búsqueda Aleatoria

Los algoritmos de búsqueda aleatoria son una técnica de búsqueda heurística que utiliza una exploración aleatoria del espacio de búsqueda en lugar de seguir una estrategia determinística. Aunque la búsqueda aleatoria no garantiza la obtención de la solución óptima, puede ser muy eficaz para encontrar soluciones aceptables en problemas de optimización combinatoria [4].

Para el problema propuesto el algoritmo de Búsqueda Aleatoria (BA) consistirá en generar aleatoriamente una solución en cada iteración debiéndose ejecutar 100 iteraciones con cada semilla devolviendo la mejor de las iteraciones.

3.2.2. El Mejor

El Mejor es un algoritmo de búsqueda local. Este algoritmo comienza con una solución inicial y examina todas las soluciones vecinas de esa solución, seleccionando la mejor solución vecina en cada iteración. Es decir, busca el mejor vecino del entorno, generando todo el entorno de la solución actual y seleccionando la mejor solución vecina. A continuación, comprueba si esa mejor vecina de la solución actual es mejor que la propia solución actual, y:

- En caso afirmativo la sustituye y vuelve a iterar
- En caso de que no, el algoritmo finaliza

El Mejor puede ser muy efectivo para resolver problemas de optimización combinatoria, especialmente cuando se utiliza en combinación con otros algoritmos de búsqueda local. Sin embargo, como con cualquier técnica de búsqueda local, existe el riesgo de que el algoritmo se atasque en un óptimo local y no pueda encontrar la solución global óptima.

Para el problema propuesto, se partirá de una solución inicial aleatoria. Los algoritmos de búsqueda local tienen su propia condición de parada, pero adicionalmente, en prevención de tiempos excesivos en algún caso, se añadirá una condición de parada alternativa (OR) basada en el número de evaluaciones que esté realizando la búsqueda, es decir, el número de veces se llame al cálculo de la función de coste. Este valor para la Búsqueda Local será de 3000 llamadas a la función de coste.

3.2.3. El Primero El Mejor

El algoritmo El Primero El Mejor es un método heurístico para la solución de problemas de optimización combinatoria. En este algoritmo, se evalúa la mejor solución posible en cada paso, considerando solo los movimientos que llevan a soluciones factibles, y se selecciona el primer movimiento que mejore la solución actual.

El algoritmo El Primero El Mejor es simple y eficiente, pero no garantiza la obtención de la solución óptima global.

Para el problema propuesto, para el número de vecinos de *El Primero el Mejor* se pasará a la siguiente solución cuando se encuentre un vecino mejor mediante una operación de movimiento o se haya alcanzado el número máximo de intentos sin mejora.

3.2.4. Búsqueda Local VND

El algoritmo *VND* (Vecindario Variable en español) es una técnica de búsqueda local que utiliza una estrategia de búsqueda en múltiples vecindarios para encontrar soluciones óptimas o subóptimas en un espacio de búsqueda [5]. El VND comienza con una solución inicial y explora su vecindario para encontrar una solución localmente óptima. Luego, se cambia al siguiente vecindario y se repite el proceso hasta que no se puedan encontrar mejoras en ningún vecindario. Es decir, se basa en el algoritmo de *El Mejor*, permitiendo el operador de movimiento cambie de entorno (ampliéndolo) cuando el mejor vecino generado no mejora a la solución actual [6].

En el VND, la selección del vecindario se puede hacer de manera aleatoria o en un orden específico. Los vecindarios pueden variar en tamaño y forma, dependiendo del problema en cuestión. Este paso es el considerado exploración del espacio, es decir, diversifica.

Para el problema propuesto se implementarán 5 niveles de velocidad variable lo que permitirá modificar la estructura de entornos. La búsqueda VND modifica la estructura de entornos como mecanismo para evitar caer en óptimos locales.

3.2.5. Simulated Annealing

El algoritmo Simulated Annealing es una técnica de búsqueda heurística inspirada en la física estadística, que utiliza la simulación de un proceso de enfriamiento gradual para escapar de óptimos locales y encontrar soluciones globales óptimas. El algoritmo se basa en un procedimiento que simula el proceso de enfriamiento de un material, donde la energía del sistema disminuye gradualmente hasta alcanzar un estado de equilibrio estable [7].

La idea detrás del algoritmo Simulated Annealing es aceptar movimientos que empeoren la función objetivo con cierta probabilidad, lo que permite escapar de óptimos locales y explorar el espacio de búsqueda de manera más amplia [8]. A medida que el proceso de enfriamiento avanza, la probabilidad de aceptar movimientos desfavorables disminuye, lo que lleva a una convergencia gradual hacia una solución óptima.

El algoritmo de Enfriamiento Simulado es un método de búsqueda por entornos, caracterizado por un criterio de aceptación de soluciones vecinas que se adapta a lo largo de su ejecución. Hace uso de una variable llamada Temperatura, T, cuyo valor determina en qué medida pueden ser aceptadas soluciones vecinas peores que la actual. La variable Temperatura se inicializa a un valor alto, denominado Temperatura inicial, T₀, y se va reduciendo cada iteración mediante un mecanismo de enfriamiento de la temperatura hasta alcanzar una Temperatura final, T_f.

En otros términos, este algoritmo busca primero diversificar para luego intensificar.

3.2.6. Búsqueda Tabú

La Búsqueda Tabú es un algoritmo de búsqueda local que evita el estancamiento en óptimos locales mediante la utilización de una lista tabú que mantiene un registro de las soluciones ya visitadas y restringe la exploración en su vecindario durante un número determinado de iteraciones. Esta técnica permite al algoritmo escapar de óptimos locales subóptimos y continuar la exploración de soluciones [9]. Es decir, permite el empeoramiento. Además, hace uso de una memoria de largo plazo, lo que permite la diversificación e intensificación.

Para el problema propuesto se plantean 3 estrategias de reinicialización:

- **Solución inicial aleatoria:** esta técnica en concreto permite la diversificación (explotación, búsqueda en regiones distantes en el espacio).
- **Memoria a largo plazo:** esta estrategia permite la diversificación al evitar soluciones recientemente estudiadas.
- **Reinicialización desde la mejor solución:** esta técnica permite la intensificación (exploración, búsqueda en la región actual en el espacio).

Durante la ejecución se realizarán 4 reinicializaciones repartidas a lo largo del total de iteraciones a realizar durante la ejecución del algoritmo. Además, en cada reinicialización la lista tabú se aumentará o decrementará de tamaño en un 50 % según una decisión aleatoria uniforme.

4. Experimentación

4.1. Algoritmos de Heurística Constructiva

4.1.1. Greedy

El algoritmo *Greedy* varía según el valor de radiación y la previsión de precios, pero la solución es siempre la misma siempre y cuando esto sea no varíe.

Los resultados obtenidos son los siguientes:

Greedy	
Evaluaciones Medias	1
Evaluación Mejor	1
Evaluacion Desviación	0.00
Mejor €	331.91
Media €	331.91
Desviación €	0.00
Evaluación Mejor Dinero	1
Evaluación Mejor Dinero	1

Cuadro 1: Resultados Greedy

La solución se ha graficado en la siguiente imagen, pudiéndose ver la evolución de la batería y del dinero acumulado durante el día. Además, como el algoritmo Greedy vende toda la energía que recibe a partir del pico máximo de la hora de venta se ha marcado esta hora.

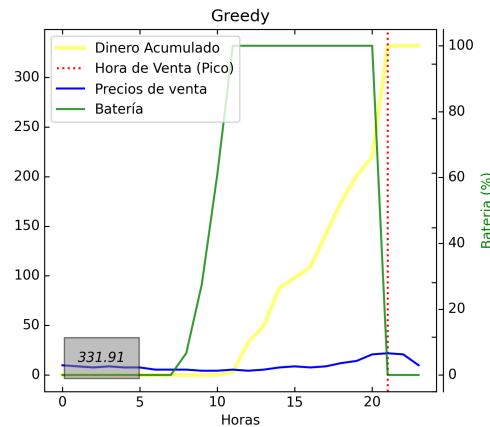


Figura 1: Resultado Greedy

4.2. Algoritmos de Búsquedas Locales

4.2.1. Búsqueda Aleatoria

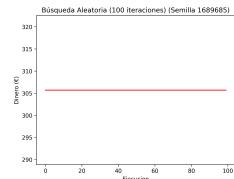
Para Búsqueda Aleatoria tenemos un parámetro que variar durante las experimentaciones, el número de iteraciones.

4.2.1.1. 100 iteraciones Primero hemos hecho los experimentos con 100 iteraciones, los resultados han sido los siguientes para las diferentes semillas:

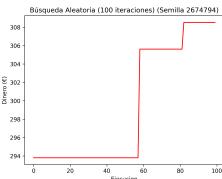
Búsqueda Aleatoria	
Evaluaciones Medias	101
Evaluación Mejor	101
Evaluacion Desviación	0.0
Mejor €	312.86
Media €	305.7
Desviación €	4.84
Evaluación Mejor Dinero	5: 70
Evaluación Media Mejor	50

Cuadro 2: Resultados Búsqueda Aleatoria (100 iteraciones)

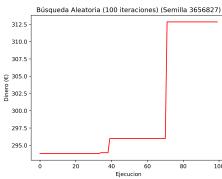
Se ha graficado la evolución del total dinero en las siguientes imágenes:



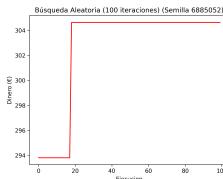
Evolución de la Búsqueda Aleatoria (Semilla 1689685)



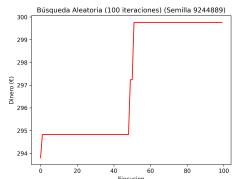
Evolución de la Búsqueda Aleatoria (Semilla 2674794)



Evolución de la Búsqueda Aleatoria (Semilla 3656827)

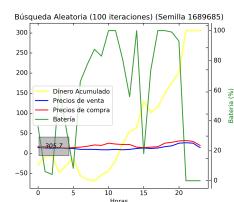


Evolución de la Búsqueda Aleatoria (Semilla 6885052)

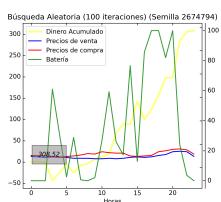


Evolución de la Búsqueda Aleatoria (Semilla 9244889)

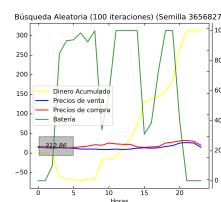
Además, se han graficado los resultados de las mejores soluciones para cada semilla en las siguientes imágenes:



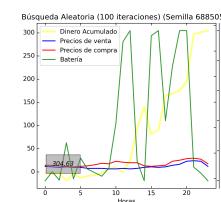
Mejor Resultado Búsqueda Aleatoria (Semilla 1689685)



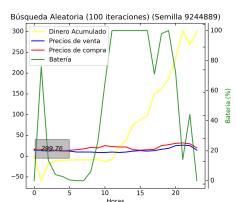
Mejor Resultado Búsqueda Aleatoria (Semilla 2674794)



Mejor Resultado Búsqueda Aleatoria (Semilla 3656827)



Mejor Resultado Búsqueda Aleatoria (Semilla 6885052)



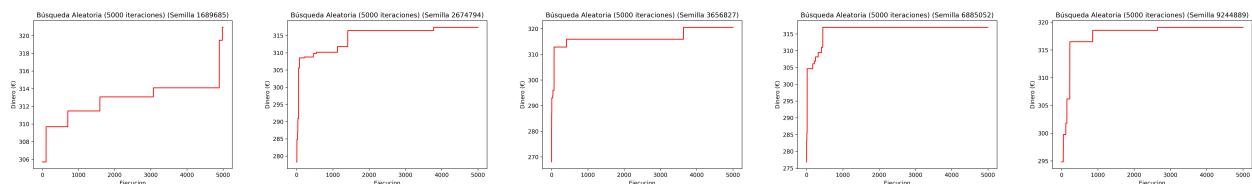
Mejor Resultado Búsqueda Aleatoria (Semilla 9244889)

4.2.1.2. 5000 iteraciones Al ser generadas las soluciones candidatas de forma aleatoria, a más soluciones se generen mayor probabilidad de encontrar una solución mejor. Por ello se ha hecho un experimento incrementando el número de iteraciones a 5000. Esperamos que se mejore la mejor solución al coste de incrementar el número de iteraciones. Aunque no podemos asegurar que esto pueda pasar.

Búsqueda Aleatoria	
Evaluaciones Medias	5001
Evaluación Mejor	5001
Evaluacion Desviación	0.0
Mejor €	320.94
Media €	319.08
Desviación €	1.79
Evaluación Mejor Dinero	3: 4980
Evaluación Media Mejor	3634

Cuadro 3: Resultados Búsqueda Aleatoria (5000 iteraciones)

Se ha graficado la evolución del total dinero en las siguientes imágenes:



Evolución de la
Busqueda Aleatoria
(Semilla 1689685)

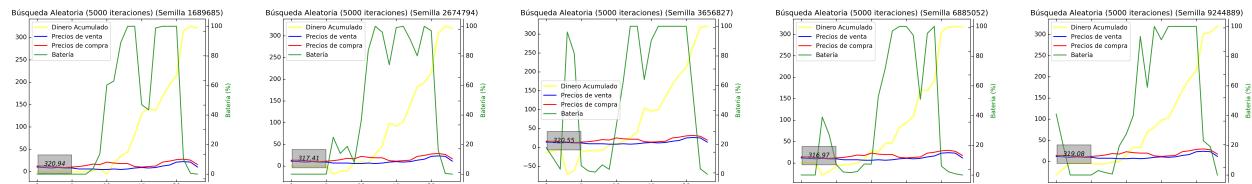
Evolución de la
Busqueda Aleatoria
(Semilla 2674794)

Evolución de la
Busqueda Aleatoria
(Semilla 3656827)

Evolución de la
Busqueda Aleatoria
(Semilla 6885052)

Evolución de la
Busqueda Aleatoria
(Semilla 9244889)

Además, se han graficado los resultados de las mejores soluciones para cada semilla en las siguientes imágenes:



Mejor Resultado
Busqueda Aleatoria
(Semilla 1689685)

Mejor Resultado
Busqueda Aleatoria
(Semilla 2674794)

Mejor Resultado
Busqueda Aleatoria
(Semilla 3656827)

Mejor Resultado
Busqueda Aleatoria
(Semilla 6885052)

Mejor Resultado
Busqueda Aleatoria
(Semilla 9244889)

4.2.2. El Mejor

En El Mejor se genera un conjunto de vecinos a partir de la solución actual para quedarnos con el mejor y generar el nuevo conjunto de vecinos a partir de este mejor vecino. Esto se hace mientras que el valor del mejor vecino no mejore el de la solución actual o no se haya hecho el máximo de evaluaciones establecido.

Para este algoritmo variaremos el parámetro *max_evaluaciones* para la experimentación.

4.2.2.1. Máximas Evaluaciones: 3000 Los resultados obtenidos son los siguientes:

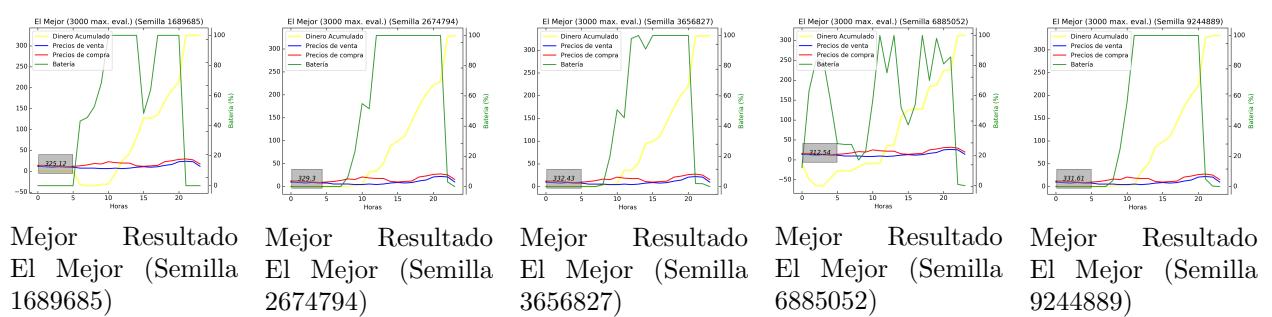
El Mejor	
Evaluaciones Medias	3038
Evaluación Mejor	3009
Evaluacion Desviación	15.03
Mejor €	332.43
Media €	329.3
Desviación €	8.15
Evaluación Mejor Dinero	5: 64
Evaluación Media Mejor	64

Cuadro 4: Resultados El mejor (3000 Max. Eval.)

Y se han graficado de la siguiente manera:



Además, se han graficado los resultados de las mejores soluciones para cada semilla en las siguientes imágenes:

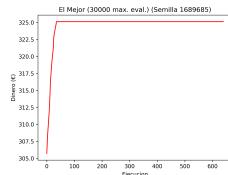


4.2.2.2. Máximas Evaluaciones: 30000 Los resultados obtenidos son los siguientes:

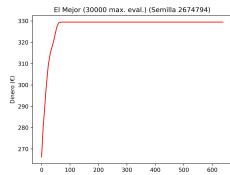
El Mejor	
Evaluaciones Medias	30034
Evaluación Mejor	30012
Evaluacion Desviación	13.27
Mejor €	333.25
Media €	331.61
Desviación €	3.17
Evaluación Mejor Dinero	5: 76
Evaluación Media Mejor	75

Cuadro 5: Resultados El mejor (30000 Max. Eval.)

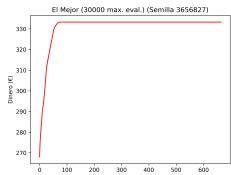
Y se han graficado de la siguiente manera:



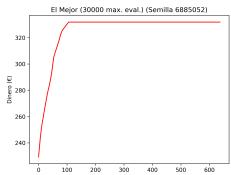
Evolución del Mejor
(Semilla 1689685)



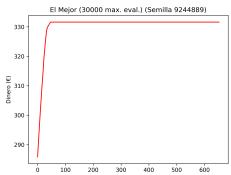
Evolución del Mejor
(Semilla 2674794)



Evolución del Mejor
(Semilla 3656827)

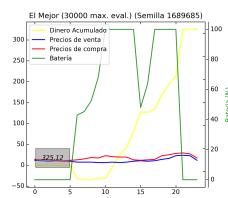


Evolución del Mejor
(Semilla 6885052)

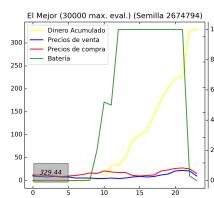


Evolución del Mejor
(Semilla 9244889)

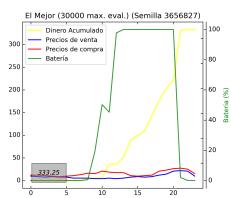
Además, se han graficado los resultados de las mejores soluciones para cada semilla en las siguientes imágenes:



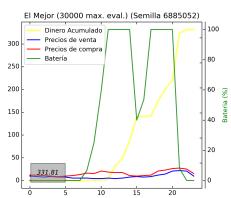
Mejor Resultado
El Mejor (Semilla
1689685)



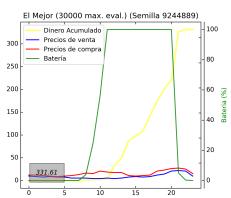
Mejor Resultado
El Mejor (Semilla
2674794)



Mejor Resultado
El Mejor (Semilla
3656827)



Mejor Resultado
El Mejor (Semilla
6885052)



Mejor Resultado
El Mejor (Semilla
9244889)

4.2.3. El Primero El Mejor

En *El Primero El Mejor* se generan vecinos y se comparan con la solución actual, el primer vecino encontrado que mejore la solución actual es con el que se ejecuta el resto del algoritmo.

Tras los experimentos de *El Mejor* se realizaran los experimentos de *El Primero El Mejor* con los parámetros de *máximas evaluaciones*=5000 y *máximas iteraciones sin mejora*=100. El parámetro a variar será la granularidad con la que se generarán los vecinos. Las granularidades para experimentar elegidas son:

- **Granularidad = 5**
- **Granularidad = 10**
- **Granularidad = 20**

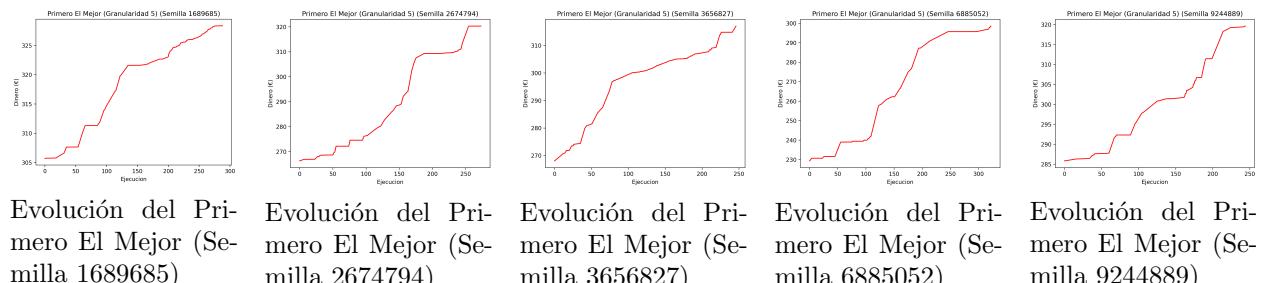
4.2.3.1. Máximas Evaluaciones: 5000 y Máximas iteraciones sin mejora: 100, Granularidad = 5

Los resultados obtenidos son los siguientes:

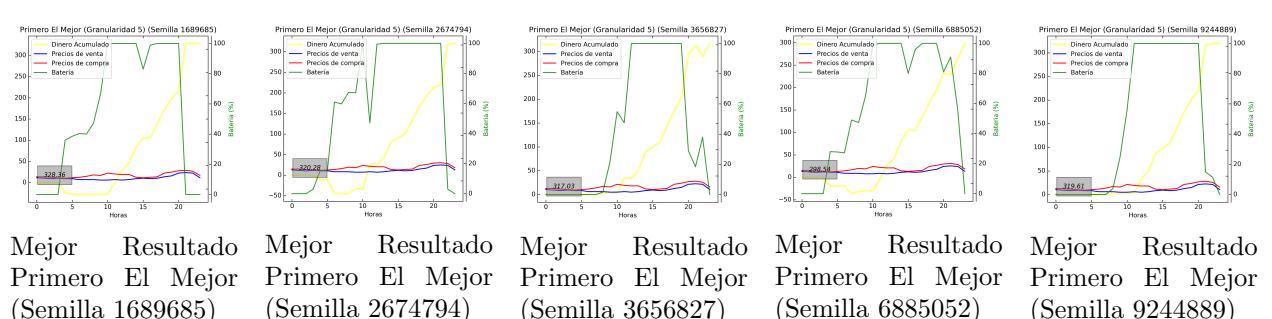
Primero El Mejor	
Evaluaciones Medias	5016
Evaluación Mejor	5002
Evaluacion Desviación	10.28
Mejor €	328.36
Media €	319.61
Desviación €	11.03
Evaluación Mejor Dinero	3: 287
Evaluación Media Mejor	255

Cuadro 6: Resultados El Primero El Mejor (Granularidad 30)

Y se han graficado de la siguiente manera:



Además, se han graficado los resultados de las mejores soluciones para cada semilla en las siguientes imágenes:

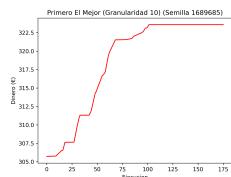


4.2.3.2. Máximas Evaluaciones: 5000 y Máximas iteraciones sin mejora: 1000, Granularidad = 10 Los resultados obtenidos son los siguientes:

Primero El Mejor	
Evaluaciones Medias	5008
Evaluación Mejor	5006
Evaluacion Desviación	13.9
Mejor €	333.25
Media €	323.58
Desviación €	5.63
Evaluación Mejor Dinero	5: 141
Evaluación Media Mejor	141

Cuadro 7: Resultados El Primero El Mejor (Granularidad 10)

Y se han graficado de la siguiente manera:



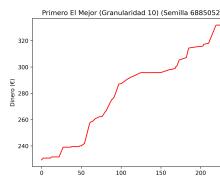
Evolución del Primero El Mejor (Semilla 1689685)



Evolución del Primero El Mejor (Semilla 2674794)



Evolución del Primero El Mejor (Semilla 3656827)

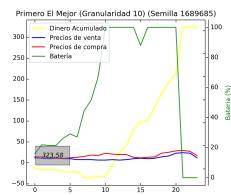


Evolución del Primero El Mejor (Semilla 6885052)

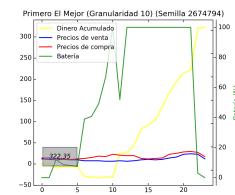


Evolución del Primero El Mejor (Semilla 9244889)

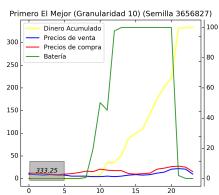
Además, se han graficado los resultados de las mejores soluciones para cada semilla en las siguientes imágenes:



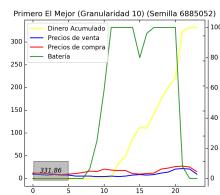
Mejor Resultado Primero El Mejor (Semilla 1689685)



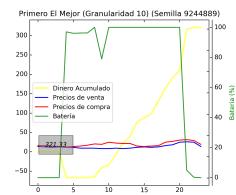
Mejor Resultado Primero El Mejor (Semilla 2674794)



Mejor Resultado Primero El Mejor (Semilla 3656827)



Mejor Resultado Primero El Mejor (Semilla 6885052)



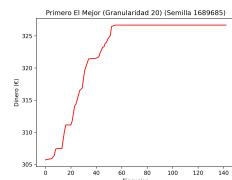
Mejor Resultado Primero El Mejor (Semilla 9244889)

4.2.3.3. Máximas Evaluaciones: 5000 y Máximas iteraciones sin mejora: 1000, Granularidad = 30 Los resultados obtenidos son los siguientes:

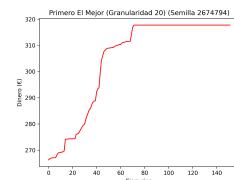
	<i>Primero El Mejor</i>
Evaluaciones Medias	5025
Evaluación Mejor	5009
Evaluacion Desviación	15.25
Mejor €	333.4
Media €	332.65
Desviación €	6.71
Evaluación Mejor Dinero	2: 124
Evaluación Media Mejor	73

Cuadro 8: Resultados El Primero El Mejor (Granularidad 20)

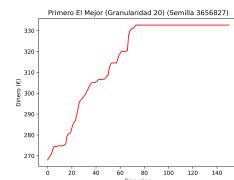
Y se han graficado de la siguiente manera:



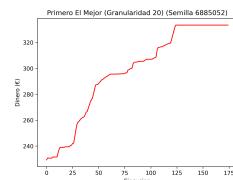
Evolución del Primero El Mejor (Semilla 1689685)



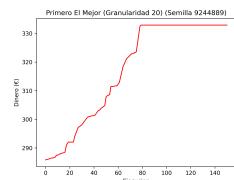
Evolución del Primero El Mejor (Semilla 2674794)



Evolución del Primero El Mejor (Semilla 3656827)

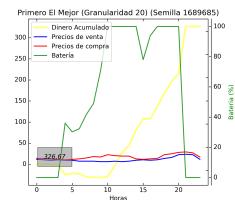


Evolución del Primero El Mejor (Semilla 6885052)

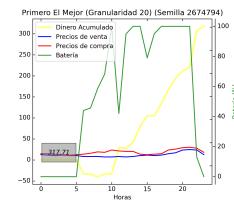


Evolución del Primero El Mejor (Semilla 9244889)

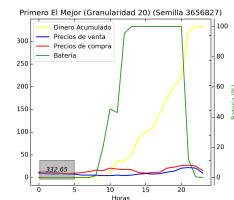
Además, se han graficado los resultados de las mejores soluciones para cada semilla en las siguientes imágenes:



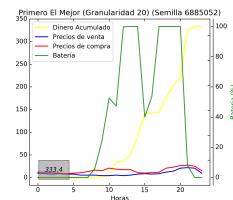
Mejor Resultado Primero El Mejor (Semilla 1689685)



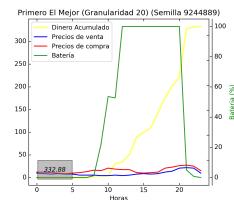
Mejor Resultado Primero El Mejor (Semilla 2674794)



Mejor Resultado Primero El Mejor (Semilla 3656827)



Mejor Resultado Primero El Mejor (Semilla 6885052)



Mejor Resultado Primero El Mejor (Semilla 9244889)

4.2.4. Búsqueda Local VND

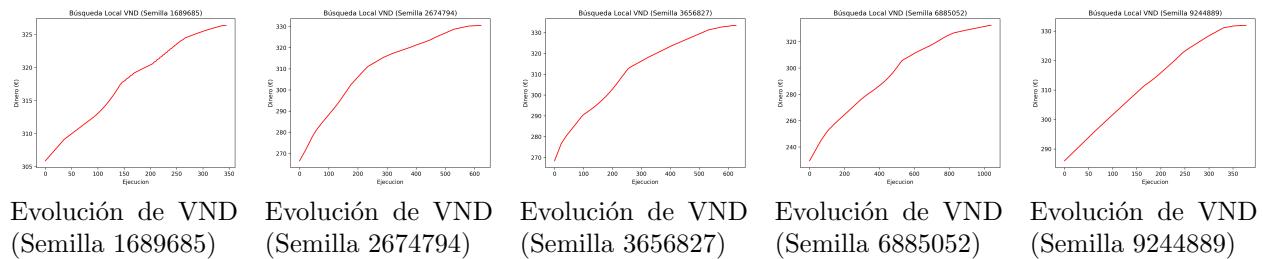
En búsqueda local no se realizarán multiples experimentaciones ya que no hay parámetros que variar.

Los resultados obtenidos son los siguientes:

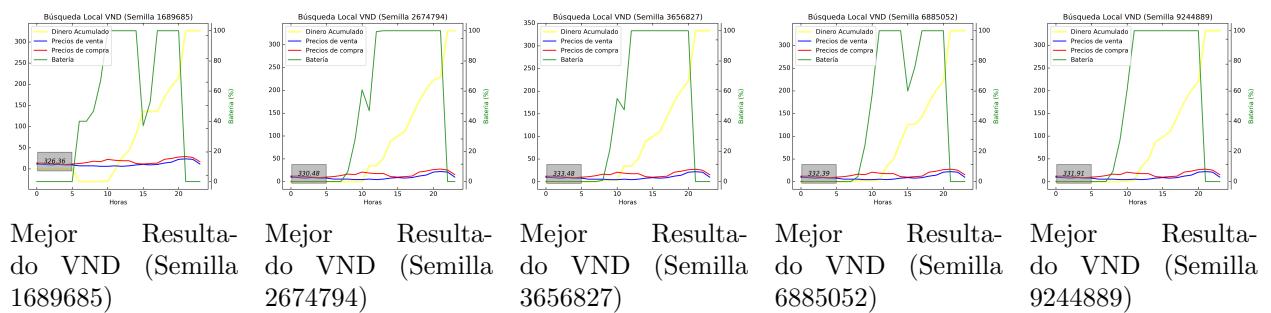
Búsqueda Local VND	
Evaluaciones Medias	29340
Evaluación Mejor	15871
Evaluacion Desviación	13097.5
Mejor €	333.48
Media €	331.91
Desviación €	2.77
Evaluación Mejor Dinero	5: 622
Evaluación Media Mejor	617

Cuadro 9: Resultados Búsqueda Local VND

Y se han graficado de la siguiente manera:



Además, se han graficado los resultados de las mejores soluciones para cada semilla en las siguientes imágenes:



4.2.5. Simulated Annealing

Para el algoritmo de *Simulated Annealing* la experimentación realizada ha sido para encontrar unos buenos valores para ϕ y μ .

Se han considerado valores entre 0.1 y 0.3 para ambos parámetros mediante una pequeña experimentación que determina que el número de soluciones iniciales no aceptadas sea de un 20% para T_0 , tomando $C(s)$ como el coste de la solución Greedy.

La experimentación intentará que la proporción sea lo más cercana posible al 20%, por lo que cuando la proporción sea más cercana a 1, mejor será el valor para la condición propuesta. La proporción comienza siendo infinita y se intentará reducir por lo que cada vez que se mejore, se mostrarán los nuevos mejores.

Se han realizado varias ejecuciones con diferentes iteraciones, los resultados son los siguientes:

```

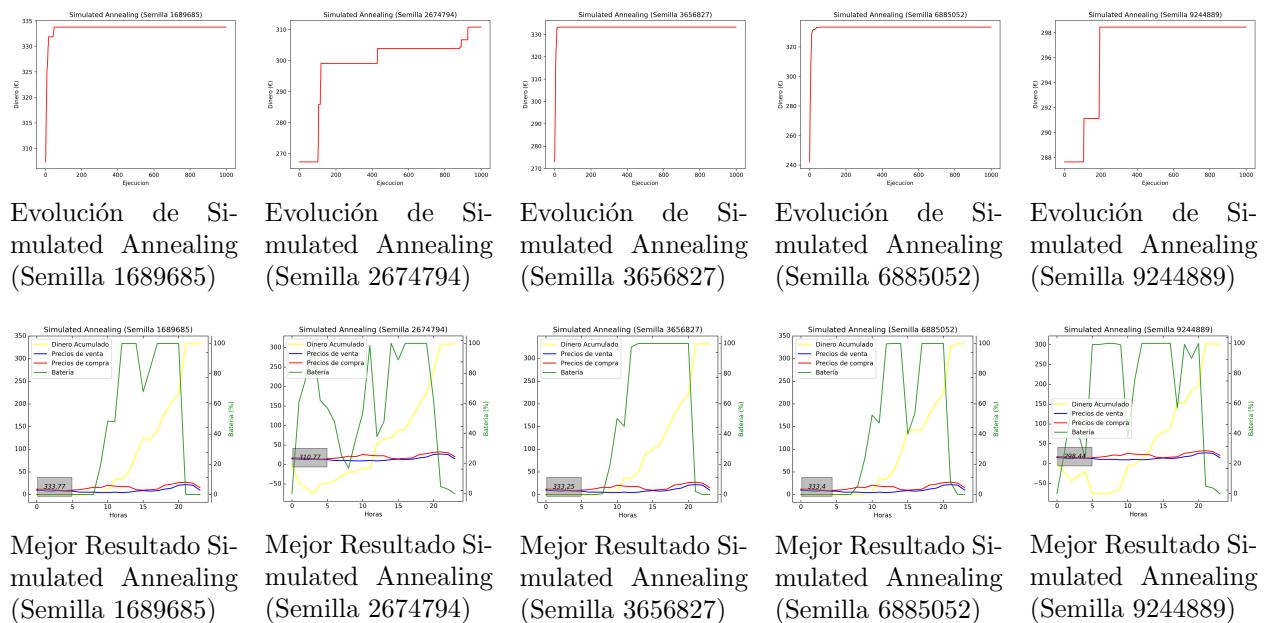
1 ===== Nuevos mejores =====
2 mu:          0.1
3 phi:          0.1
4 proporcion aceptada: 2.0
5 ===== Nuevos mejores =====
6 mu:          0.1
7 phi:          0.2
8 proporcion aceptada: 1.0
9
10 Mejores valores mu y phi: (0.1, 0.2)

```

Los resultados con estos valores son los siguientes:

Simulated Annealing	
Evaluaciones Medias	40000
Evaluación Mejor	40000
Evaluacion Desviación	0
Mejor €	333.77
Media €	333.25
Desviación €	16.04
Evaluación Mejor Dinero	3: 51
Evaluación Media Mejor	63

Cuadro 10: Resultados Búsqueda Local VND



4.2.6. Búsqueda Tabú

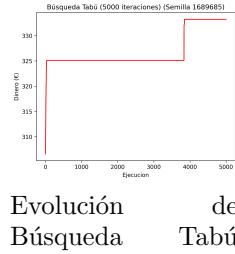
La *Búsqueda Tabú* tiene un parámetro por el que se pueden realizar las experimentaciones, este es *num_iteraciones*.

Los resultados son los siguientes:

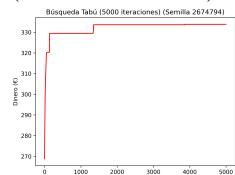
Búsqueda Tabú	
Evaluaciones Medias	188112
Evaluación Mejor	183408
Evaluacion Desviación	2916.12
Mejor €	333.81
Media €	333.65
Desviación €	0.27
Evaluación Mejor Dinero	4: 3867
Evaluación Media Mejor	2620

Cuadro 11: Resultados Búsqueda Tabú

4.2.6.1. 5000 iteraciones

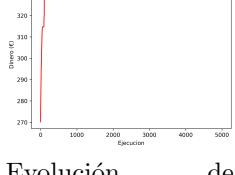


(Semilla 1689685)



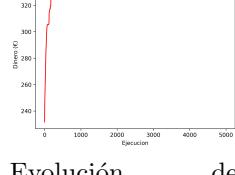
Evolución de Búsqueda Tabú (Semilla 2674794)

Búsqueda Tabú (5000 iteraciones) (Semilla 3656827)



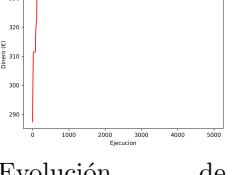
Evolución de Búsqueda Tabú (Semilla 3656827)

Búsqueda Tabú (5000 iteraciones) (Semilla 6885052)



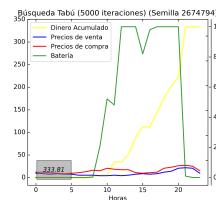
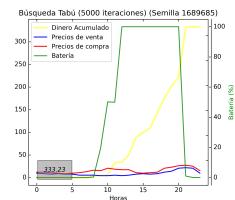
Evolución de Búsqueda Tabú (Semilla 6885052)

Búsqueda Tabú (5000 iteraciones) (Semilla 9244889)

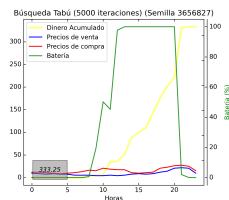


Evolución de Búsqueda Tabú (Semilla 9244889)

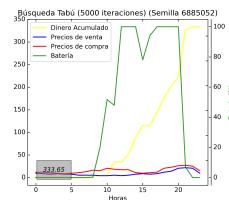
Evolución de Búsqueda Tabú (Semilla 2674794)



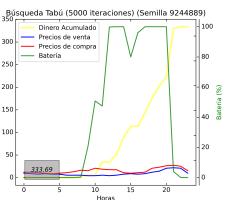
Mejor Resultado Búsqueda Tabú (Semilla 2674794)



Mejor Resultado Búsqueda Tabú (Semilla 3656827)



Mejor Resultado Búsqueda Tabú (Semilla 6885052)

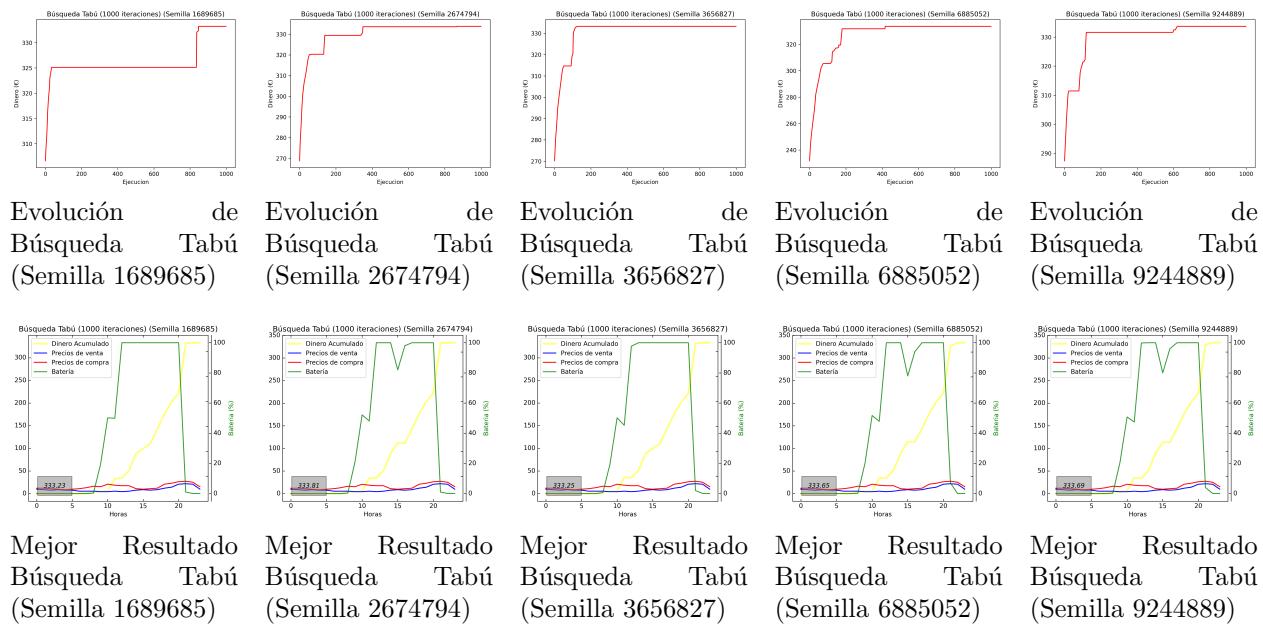


Mejor Resultado Búsqueda Tabú (Semilla 9244889)

4.2.6.2. 1000 iteraciones

Búsqueda Tabú	
Evaluaciones Medias	38556
Evaluación Mejor	37592
Evaluacion Desviación	617.33
Mejor €	333.81
Media €	333.65
Desviación €	0.27
Evaluación Mejor Dinero	4: 867
Evaluación Media Mejor	620

Cuadro 12: Resultados Búsqueda Tabú



5. Análisis

5.1. Algoritmos de Heurística Constructiva

5.1.1. Greedy

Dado que los cálculos de las soluciones se han hecho siempre con los mismos valores de radiación y de precios, el valor de *Greedy* se mantiene constante.

En la Tabla 1 podemos verificar esto. Al ser una única solución, los resultados no nos indican mucho acerca del algoritmo ya que el número de evaluaciones es único y sólo hay un único valor de total dinero, el cual es 331.91.

Por eso en los próximos algoritmos se usará para comparar los resultados.

En la Figura 1 observamos que, como es de esperar, a partir de la hora de venta máxima la batería permanece vacía. Sería de esperar que el dinero acumulado siguiese incrementando. Si observamos el valor de la radiación vemos que a partir de la hora pico no hay más radiación, esa es la razón por la que no aumenta el dinero acumulado.

5.2. Algoritmos de Búsquedas Locales

5.2.1. Búsqueda Aleatoria

5.2.1.1. 100 iteraciones Analizando los parámetros de resultados de la Tabla 2. Observamos que el número de evaluaciones se mantiene constante, es de esperar, ya que en Búsqueda Aleatoria se generan tantas nuevas soluciones (y por lo tanto se tienen que evaluar) como iteraciones se hagan. En este caso hay 1 más porque la solución inicial no está dentro de las iteraciones. Por esta misma razón la *Desviación de las evaluaciones* es 0.

Los resultados de *total_dinero* no son malos, pero no llegan a superar al Greedy. Esto es de esperar ya que la mejora o empeoramiento es totalmente arbitrario. A pesar de la arbitrariedad se ha conseguido una desviación para el dinero de 4.84, lo que demuestra que es robusta.

Las mediciones extras que se han incluído nos da información para saber que la evaluación en la que mejor resultado se ha obtenido es en la 5^o semilla en la iteración 70. Mientras que la media para encontrar la mejor solución en 100 iteraciones es en la iteración 50.

A continuación analizaremos los resultados graficados.

En los resultados de la búsqueda aleatoria con 100 iteraciones podemos ver que en la mayoría hay al menos una mejora excepto en la Figura 4.2.1.1 que el *total_dinero* se mantiene constante, esto quiere decir que no hay ninguna mejora en el resultado en 100 iteraciones. Para el resto de semillas sí hay mejoras aunque en la mayoría de casos en esas 100 iteraciones hay 1 o 2 mejoras.

En las gráficas de los mejores resultados para cada semilla podemos observar que la batería aprender a llenarse y vaciarse. En las figuras Figura 4.2.1.1 y Figura 4.2.1.1 se ve más claramente. Se podría haber pensado que el algoritmo no iba a vaciar la batería y por lo tanto no explorar todo el espacio de búsqueda, en este caso, al ser aleatorio, podría haber o no haber pasado. Pero las mejores soluciones que se han generado y han sido las mejores vacían y llenan las baterías.

5.2.1.2. 5000 iteraciones Analizando los parámetros de resultados de la Tabla 3. Tal y como en el experimento anterior, observamos que el número de evaluaciones se mantiene constante.

Los resultados de *total_dinero* mejoran los resultados logrados con el experimento 4.2.1.1, pero siguen sin superar al Greedy. Esta vez la desviación del dinero ha disminuido a 1.79, mostrando mayor robustez. Aunque, al ser arbitrario, este indicador no debe ser del todo concluyente.

Las mediciones extras que se han incluído nos indica que la mejor solución se ha encontrado en la iteración 4980 de la semilla 3, es decir, finalizando el total de las iteraciones introducidas para este experimento. Por lo que podríamos seguir aumentando el número de iteraciones y seguir consiguiendo mejoras. Por otro lado, la media de evaluaciones en la que se ha encontrado la mejor solución es la 3634 que es aproximadamente el 75 % de las 5000 iteraciones. Reafirmando que podríamos seguir aumentando el número de iteraciones y podría mejorarse la solución.

Analizando las gráficas de forma general podemos observar que la mayor mejora se obtiene al inicio, a las 500 iteraciones aproximadamente. Excepto en la figura 4.2.1.2 en donde observamos que hay otro gran incremento en la evaluación de la solución casi al finalizar el total de iteraciones.

Respecto a las imágenes que grafican la mejor solución, casi todas tienden a mantener la batería vacía al inicio para pasar a intentar mantenerla llena hasta el final. Esto nos podría indicar o bien que a pesar de hacer una búsqueda aleatoria no está explotando bien todo el espacio de posibles soluciones o que las soluciones óptimas tendrían esta forma.

5.2.2. El Mejor

5.2.2.1. Máximas Evaluaciones: 3000 Analizando los parámetros de resultados de la tabla 4, podemos observar que la evolución de *dinero_total* tiene una forma exponencial. Esto quiere decir que al inicio las mejoras son muy grandes pero a medida que pasa el tiempo las mejoras son muy pequeñas. Es decir, si incrementamos el número máximo de evaluaciones, sería de esperar no mejorar mucho los resultados.

Esta vez el mejor resultado de dinero supera, quedando muy cercano, a la evaluación del *Greedy*.

Algo llamativo es que las evaluaciones superan el número máximo de evaluaciones, esto puede darse por dos razones, debido a las condiciones que hemos puesto al algoritmo *El Mejor*:

- Se ha completado el total de evaluaciones máximas (5000) y se han encontrado consecutivamente soluciones que han mejorado a la actual.
- Se ha completado el total de evaluaciones máximas (5000) y como se incrementa el *contador_evaluaciones* con el número de vecinos generados, el último incremento superó directamente el número de evaluaciones máximas.

Además, el medidor *Evaluación Media Mejor* coge el índice de *totales_dinero*, en este caso como se incrementa de número de vecinos generados en número de vecinos generados por iteración, no da exacto. Para calcularlo con mayor exactitud tendríamos que multiplicarlo por la media de vecinos generados y evaluados por cada iteración. Pongamos que son 45, ya que de máximo se generan 48 pero si algún valor es inferior a -100 o superior a +100 no se incluye, por ello este valor sería 2880, casi al final de las iteraciones.

Analizando ahora las gráficas de las mejores soluciones podemos ver como casi en todas la solución se acerca mucho a la gráfica del *Greedy*, volvemos a poder intuir que puede que la solución óptima que nuestra función evaluación es capaz de producir es cercana a lo que consigue el *Greedy*.

Sin embargo, podríamos pensar que estamos cayendo en un óptimo local, pero la figura 4.2.2.1 nos muestra una representación diferente de la batería por lo que se está explorando un espacio de soluciones más extenso del que podríamos pensar.

5.2.2.2. Máximas Evaluaciones: 30000 Para el experimento hemos incrementado notablemente el número de iteraciones. En la tabla 5 podemos ver como, tal y como se había previsto, los resultados apenas mejoran. Se consigue un valor de *Mejor €* muy cercano al del *Greedy* y que vuelve a mejorarlo. La mejora es superior a la que se consigue con 100 iteraciones pero es mínima. Además de una desviación entre las evaluaciones de las mejores soluciones muy pequeña.

Por otro lado los medidores extras añadidos nos indican que la mejor solución se alcanzo en, como en el caso anterior $76*45 = 3420$. Por lo que no nos interesa incrementar tanto el número de evaluaciones máximas. Dejandolo tal y como estaba en 3000 se alcanzan casi los mismos resultados con mucho menos tiempo y menos evaluaciones.

En las gráficas confirmamos esto ya que a partir del valor 100 que equivale a la evaluación 4500 nos encontramos con una línea recta, es decir, la solución no mejora.

Esta vez en las gráficas de la mejor solución de cada semilla nos encontramos que tienen formas muy similares llegando a ser casi idénticas esta vez al *Greedy*. Es el caso de la figura 4.2.2.2 que, de hecho, corresponde con la mejor solución como se indica en la tabla en el medidor extra *Evaluación Mejor Dinero* que es en la 5º semilla la evaluación 3420 ($76*25$).

5.2.3. El Primero El Mejor

Para los experimentos de *El Primero El Mejor* se pueden variar 3 parámetros:

- **Máximo Evaluaciones**
- **Intentos sin mejora**
- **Granularidad**

Pero tras los experimentos de *El Mejor* se ha decidido que el valor óptimo para sacar la mejor solución son para *Máximo de Evaluaciones*: 5000. Ya que a partir de esta cifra no mejora mucho más la solución. Y para limitar el tiempo, *Intentos sin mejora* se ha establecido en 100.

Por ello los experimentos se harán variando la granularidad de generación de nuevos vecinos. Los valores que se van a estudiar son los siguientes:

- **5**
- **10**
- **20**

5.2.3.1. Granularidad 5 Para granularidad 5 los resultados que se obtienen y pueden visualizar en la tabla 6 no son malos. Se llegan a ejecutar las 5000 evaluaciones lo que quiere decir que al menos cada 100 iteraciones hay una mejora. Por otro lado las evaluaciones no son malas pero no consiguen superar la evaluación del *Greedy*.

En las gráficas podemos ver como hay una mejora constante de la evaluación.

Por otro lado, las gráficas de las mejores soluciones vuelven a ser muy similares a la gráfica 1 del *Greedy*.

A pesar de tener mejoras continuamente en algunas semillas llega un momento en el que el resultado no mejora. Esto podría ser porque la granularidad no es lo suficientemente grande como para expandirse y salir de un posible óptimo local.

5.2.3.2. Granularidad 10 Para el segundo experimento se ha incrementado la granularidad a 10. Este es el valor que toma la granularidad del generador de vecinos de forma predeterminada para cualquier algoritmo.

En la tabla 7 podemos ver como esta vez mejora la evaluación del *Greedy* y la media de evaluaciones es muy similar a la mejor conseguida. Hasta el momento es la mejor solución que se ha encontrado.

En las gráficas podemos volver a observar como hay mejoras constantes y que en la mayoría de semillas llega un punto que es próximo al 75 % de las evaluaciones en el que no se consigue mejora. A pesar de tener el parámetro de *intentos sin mejora* se podría reducir el número de *máximas evaluaciones*. Solo hay una semilla en la que esto no pasa y se consigue mejora hasta el final, es en la figura 4.2.3.2. Aún así la mejor solución se obtiene en la 5^a semilla por lo que se podría reducir sin problema.

En las gráficas de las mejores soluciones podemos volver a ver como sigue siendo muy similar a la gráfica 1 del *Greedy*. Sin embargo, la mejor solución que es la gráfica 4.2.3.2 mantiene la batería llena hasta el final, que coincide cuando la radiación disminuye a 0 y es la hora de venta máxima.

5.2.3.3. Granularidad 20 Para terminar con la experimentación, como se han hecho experimentos disminuyendo la granularidad predeterminada, también se ha aumentado para estudiar el impacto.

Lo esperado podría ser que no haya muy buenas mejoras ya que pueda alcanzar rápido el límite de la batería tanto superior como inferior. Sin embargo también podría encontrar buenas soluciones de forma más rápida.

En la solución que se puede estudiar en la tabla 8 volvemos a conseguir una mejora a la evaluación del *Greedy*. Volviendo a conseguir la mejor solución hasta el momento aunque por muy poco respecto al experimento anterior. La media de dinero también es muy buena. Y se vuelven a gastar las evaluaciones máximas introducidas por lo que se intuye que hay mejoras constantes.

En las gráficas de evolución podemos confirmar la teoría de que la solución óptima podría ser encontrada antes ya que los resultados se atascan al 50% de las evaluaciones aproximadamente. Por lo que podríamos reducir el número de intentos sin mejora y de máximas evoluciones.

Por otro lado en las gráficas de mejores soluciones se vuelven a parecer mucho a las de *Greedy*. Sin embargo esta vez la mejor solución corresponde a la semilla 2, figura 4.2.3.2. Que no es tan parecida al *Greedy* ya que intenta vaciar un poco la batería.

5.2.4. Búsqueda Local VND

En este algoritmo no se han realizado múltiples experimentaciones como para el resto.

La mejor evaluación obtenida con este algoritmo mejora la mejor conseguida hasta ahora por unos decimales. Además la media es bastante buena coincidiendo con la evaluación del *Greedy*. Sin embargo la desviación de evaluaciones es muy alta, siendo casi la mitad de las evaluaciones medias. Es decir, no es un algoritmo robusto ya que a veces requiere muchas evaluaciones y otras no.

Analizando las gráficas observamos que todas las semillas tienen un comportamiento muy similar consiguiendo mejoras exponencialmente.

Por otro lado, en las gráficas de las mejores soluciones tienen comportamientos también muy similares al *Greedy*. La semilla que mejor resultado obtiene es la 5^a, que corresponde a la figura 4.2.4 que es el más parecido al *Greedy*.

5.2.5. Simulated Annealing

Tras las experimentaciones del algoritmo *Simulated Annealing* para conseguir los mejores valores de mu y phi, los resultados se muestran en la tabla 10.

La mejor evaluación sigue sin mejorar mucho pero vuelve a mejorar la última mejor obtenida. La media de evaluaciones sigue siendo muy alta. Sin embargo la evaluación de mejor dinero se encuentra en las primeras iteraciones por lo que podríamos reducir el valor de iteraciones para *Simulated Annealing*.

Esto lo podemos verificar con las gráficas de evolución de la solución. En todas excepto en la semilla 2, se deja de mejorar antes de llegar al 50% de iteraciones. Aún así la mejor solución se obtiene de la semilla 3, por lo que podríamos reducir el valor de iteraciones y seguir obteniendo muy buenos resultados.

En las gráficas de las mejores soluciones vuelven a asemejarse a la gráfica 1 del *Greedy*, sobre todo la de la 3^a semilla que es la mejor solución. Aún así el resto de evaluaciones también tienen valores similares y son un poco más diferentes.

5.2.6. Búsqueda Tabú

Para la *Búsqueda Tabú* se han realizado dos experimentos variendo el número de iteraciones.

5.2.6.1. 5000 iteraciones

Para la *Búsqueda Tabú* con 5000 iteraciones tenemos un número de evaluaciones muy alto, comparado con las evaluaciones medias la desviación no es tan alta como aparenta. Por otro lado, el mejor dinero supera a los algoritmos antes estudiados al igual que la media. Además la desviación es muy pequeña por lo que este algoritmo a pesar de requerir muchas evaluaciones consigue los mejores resultados con cualquier semilla.

Pero por otra parte en las gráficas de evolución de la solución podemos observar que tantas iteraciones no son necesarias pues a partir de cierto punto la mejora es mínima e incluso no se consigue ninguna mejora.

En las gráficas de las mejores soluciones vuelve a repetirse el patrón del algoritmo *Greedy*.

5.2.6.2. 1000 iteraciones

Tras observar que tantas iteraciones no eran necesarias se han reducido y se han obtenido los mismos mejores resultados y medias. Es decir, no eran necesarias tantas iteraciones, con muchas menos se han conseguido los mismos resultados.

6. Resultados

6.1. Problema 1

	Greedy	Búsq. Aleatoria	El Mejor	Primerº El Mejor	VND	Simulated Annealing	Búsq. Tabú
Evaluaciones Medias	1	101	3038	3025	29340	40000	38556
Evaluación Mejor	1	101	3009	3000	15871	40000	37592
Evaluacion Desviación	0.00	0.00	15.03	14.92	13097.5	0.00	617.33
Mejor €	331.91	312.86	332.43	333.4	333.48	333.77	333.81
Media €	331.91	305.7	329.3	332.65	331.91	333.25	333.65
Desviación €	0.00	4.84	8.15	6.71	2.77	16.4	0.27
Evaluación Mejor Dinero	1	5: 70	5: 64	2: 124	5: 622	3: 51	4: 867
Evaluación Medias Dinero	1	50	64	73	617	63	620

Cuadro 13: Problema 1

6.2. Problema 2

	Greedy	Búsq. Aleatoria	El Mejor	Primerº El Mejor	VND	Simulated Annealing	Búsq. Tabú
Evaluaciones Medias	1	101	3017	3027	47978	40000	36338
Evaluación Mejor	1	101	3009	3015	34183	40000	34911
Evaluacion Desviación	0.00	0.00	9.91	11.33	10448.65	0.00	727.86
Mejor €	398.026	469.07	553.95	596.07	608.97	675.14	674.55
Media €	398.026	424.21	541.29	570.59	579.58	664.72	637.02
Desviación €	0.00	23.06	46.51	60.36	61.43	123.87	28.34
Evaluación Mejor Dinero	1	3: 81	5: 65	1: 60	5: 1353	3: 81	5: 255
Evaluación Medias Dinero	1	61	65	54	1142	99	480

Cuadro 14: Problema 2

7. Conclusiones

Las experimentaciones solo se han realizado sobre el Problema 1, por lo que los parámetros se han modificado en función de este problema. Sin embargo los mejores resultados se han obtenido en el Problema 2.

Estudiando las evaluaciones de las soluciones obtenidas en el primer problema, al graficar los resultados se observaba que todos tendían a seguir la solución *Greedy*. Esto lleva a pensar que el problema podría ser que los algoritmos no aprendían a vaciar y llenar la batería durante el día. Sin embargo, al ejecutar los algoritmos sobre el Problema 2, al estudiar todas las gráficas se observa que los algoritmos si saben llenar y vaciar la batería durante el día. Por lo que el problema podría ser los valores dados para la radiación y precios para ese preciso día, pero en otro día podrían mejorar.

Analizando los medidores de ambos problemas para todos los algoritmos podemos observar como en el segundo problema en el *Greedy* se obtienen mejores resultados.

El algoritmo de Búsqueda Aleatoria es el único que en ambos problemas empeora la solución del *Greedy*. Por otro lado, los mejores algoritmos han resultado ser *Simulated Annealing* y *Búsqueda Tabú*. Que son ambos los únicos algoritmos que permiten el empeoramiento frente al resto que tiene otras técnicas.

Con *Búsqueda Aleatoria* podríamos obtener buenas o malas resultados, es un algoritmo impredecible ya que la obtención de una buena solución es arbitrario. Se ha demostrado con los experimentos que con las iteraciones dadas no ha dado buenas soluciones. Incrementandolo en los experimentos. De hecho para el Problema 2 se han incrementado las iteraciones de 100 a 10000 y el problema 2 ha obtenido como mejor evaluación 526.31. La desviación va en función de esta arbitrariedad por lo que no tiene porque ser un medidor concluyente para este algoritmo.

Entre *El Mejor* y *Primero El Mejor* son algoritmos muy similares. Mientras que *El Mejor* genera un conjunto de vecinos y selecciona el mejor, *Primero El Mejor* deja de comparar vecinos en cuanto uno supera la solución inicial. Sería de esperar que *El Mejor* superase a *El Primero El Mejor* a pesar de tener un costo de tiempo superior y hacer más evaluaciones. Sin embargo, los resultados en ambos problemas han dado como resultados mejores soluciones con *Primero El Mejor* aunque no hay una diferencia muy grande. El número de evaluaciones también es muy parecido y se acerca al número introducido como máximo de iteraciones. Ambos algoritmos tienen un rendimiento medio, equilibrado, entre el número de evaluaciones que realizan y el valor de las soluciones obtenidas.

El algoritmo *Búsqueda Local VND* explora localmente el espacio de soluciones con diferentes velocidades. Puede ser muy lento y no garantiza la solución óptima. Por esta razón es de los algoritmos que más evaluaciones necesita. En el primer problema no tiene demasiadas pero en el Problema 2 es el algoritmo que más evaluaciones ha necesitado. Además, es el algoritmo menos robusto pues la desviación entre el número de evaluaciones de las diferentes semillas es muy alto siendo entre un cuarto y la mitad de las evaluaciones medias. Los resultados respecto a la evaluación de la solución no son los mejores ni los peores. Pero teniendo en cuenta la cantidad de evaluaciones necesarias no es el mejor algoritmo para este problema.

Otro algoritmo lento y que requiere un mayor número de evaluaciones es el *Simulated Annealing*. La forma de explorar el espacio de soluciones es similar al de *Búsqueda Aleatoria* pero con una probabilidad de aceptar soluciones peores. Al aceptar soluciones peores, es decir, permitir el empeoramiento, puede escapar de óptimos locales y encontrar soluciones de alta calidad, pero requiere ajustar los parámetros y es menos predecible. Por ello hemos necesitado ejecutar experimentos para encontrar los valores de mu y phi. En ambos problemas optiene muy buenas soluciones con un número de evaluaciones razonable. De hecho si ejecutamos en algoritmo de *Búsqueda Aleatoria* con el mismo número de iteraciones no conseguimos resultado tan buenos. Por lo que el empeoramiento permite la mejora de los resultados al escapar de posibles óptimos locales. Lo más negativo es la robustez de las mejores soluciones pues en ambos problemas la desviación para las evaluaciones de las mejores soluciones es muy alta, la peor.

Búsqueda Tabú, como *Simulated Annealing*, permite el empeoramiento, y también ha permitido encontrar

mejores resultados, por lo que podríamos suponer que el resto de algoritmos estaban cayendo en óptimos locales. Es el algoritmo más robusto pues la desviación de las evaluaciones es mínima. Además, las evaluaciones obtenidas son casi óptimas en ambos problemas. Sin embargo, el número de evaluaciones es muy alto.

En general, los algoritmos más rápidos (*Greedy, Búsqueda Aleatoria*) pueden encontrar soluciones buenas pero no óptimas al poder caer en óptimos locales sin conseguir escapar de ellos. Mientras que los algoritmos más lentos (*El Mejor, Búsqueda Tabú*) pueden encontrar soluciones óptimas pero son más costosos computacionalmente, pues necesitan ejecutar la función de evaluación muchas más veces que el resto de algoritmos. Los otros algoritmos (*Primero el mejor, Búsqueda Local VND, Simulated Annealing*) buscan un equilibrio entre velocidad y calidad de la solución, aunque en el caso de este problema VND no ha conseguido encontrar este equilibrio pues tiene un número de evaluaciones muy alto para la solución encontrada.

Por ello habría que valorar si tenemos suficiente capacidad computacional para ejecutar los algoritmos más costosos para este problema para encontrar mejores soluciones.

Referencias

- [1] C. Blum and A. Roli, “Metaheuristics in combinatorial optimization: Overview and conceptual comparison,” *ACM computing surveys (CSUR)*, vol. 35, no. 3, pp. 268–308, 2003.
- [2] M. Gendreau, J.-Y. Potvin, *et al.*, *Handbook of metaheuristics*, vol. 2. Springer, 2010.
- [3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2022.
- [4] Z. B. Zabinsky *et al.*, “Random search algorithms,” *Department of Industrial and Systems Engineering, University of Washington, USA*, 2009.
- [5] P. Hansen and N. Mladenović, “Variable neighborhood search: Principles and applications,” *European journal of operational research*, vol. 130, no. 3, pp. 449–467, 2001.
- [6] P. Hansen and N. Mladenović, *An introduction to variable neighborhood search*. Springer, 1999.
- [7] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi, “Optimization by simulated annealing,” *science*, vol. 220, no. 4598, pp. 671–680, 1983.
- [8] E. Aarts and J. Korst, *Simulated annealing and Boltzmann machines: a stochastic approach to combinatorial optimization and neural computing*. John Wiley & Sons, Inc., 1989.
- [9] F. Glover, “Tabu search—part i,” *ORSA Journal on computing*, vol. 1, no. 3, pp. 190–206, 1989.