TRABAJO DE PRÁCTICAS

Técnicas de Inteligencia Artificial

Autora
Aitana Menárguez Box
Octubre 2023
MUIARFID UPV

Índice

1 Introducción

El objetivo de esta memoria es reflejar el trabajo realizado en las prácticas de laboratorio de la asignatura de Técnicas de Inteligencia Artificial. A lo largo de este documento, se describen, implementan y evalúan diferentes técnicas mateheurísticas para abordar un problema de optimización concreto planteado.

2 Descripción del problema

El problema a resolver es el siguiente: se tiene un terreno de medidas $x \times y$ en el cual se quieren colocar una serie de placas solares de medidas $m \times n$ cada una. De éstas se dispone, en principio, un número ilimitado. Para simplificar, se ha dividido el terreno en sectores del tamaño de las placas, de forma que el terreno quedará distribuido en una serie de superficies (posiciones) en las cuales podrán ir situadas las placas. De este modo, el número posiciones en el terreno será $N = \lfloor \frac{x}{m} \times \frac{y}{n} \rfloor$. En la tabla 1 se tiene un ejemplo gráfico de una posible división del terreno.
Por otro lado, las placas tienen un precio y una potencia asociados. Asimismo, cada una de las posiciones en el terreno mencionadas tiene un coste de colocación de una placa en él y una cantidad de energía (potencia) que generaría la placa que se colocara dentro.
El objetivo, por tanto, es escoger la distribución de las placas, es decir la posición de las mismas en el terreno, que maximice la potencia que se genera en toda su extensión a partir de un presupuesto. Esto significa, elegir en qué posiciones (p_i) del terreno se colocará o no una placa, respetando siempre el presupuesto o coste máximo C, que bajo ningún concepto puede superarse, a la vez que se busca generar la mayor cantidad de energía (potencia). La representación de una solución posible para el ejemplo planteado antes puede verse en la figura 1.

Tras realizar diversas pruebas con los ajustes de los parámetros iniciales del problema, se ha decidido que, por limitaciones computacionales y temporales, los valores para el problema serán los siguientes:

- Ancho de las placas (m): $1.7m$
- Alto de las placas (n): $1m$
- Precio de cada placa (*precio-p*): 500 €
- Potencia de cada placa (*potencia-p*): $300W$
- Ancho del terreno (x): $17m$
- Alto del terreno (y): $10m$
- Presupuesto (C): 20000 €
- Cantidad de posiciones (N): $\lfloor \frac{17}{1.7} \times \frac{10}{1} \rfloor = 100$

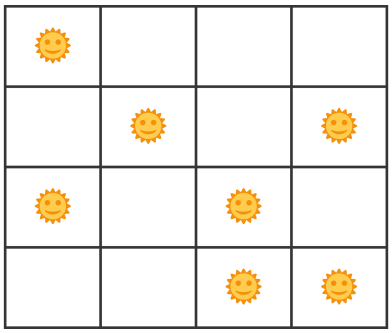

Figura 1: Representación gráfica de una posible solución para el problema cuando $N = 16$. En las posiciones en las cuales hay un sol es donde iría una placa.

p_1	p_2	p_3	p_4
p_5	p_6	p_7	p_8
p_9	p_{10}	p_{11}	p_{12}
p_{13}	p_{14}	p_{15}	p_{16}

Tabla 1: Ejemplo de la representación de un terreno $x \times y$ dividido en posiciones p_i de tamaño $m \times n$ para $1 \leq i \leq N$ siendo $N = 16$.

3 Representación de los datos

En esta sección se explica cómo se han representado los datos matemáticamente sobre el código con tal de poder aplicar los algoritmos mencionados en la introducción. Para ello se plantea la codificación de los individuos y algunas funciones necesarias para operar sobre los datos del problema.

3.1 Codificación y decodificación de los individuos

Cada individuo representa una posible solución, eso es una posible distribución de las placas en el terreno. Se ha escogido la representación en forma de *array*, de modo que éste tendrá tantos elementos como N. Así, cada elemento representará cada una de las posibles posiciones y en cada una de estas posiciones habrá un 1 o un 0 en función de si en dicha posición se coloca una placa o no, respectivamente. En definitiva, cada individuo vendrá representado como un *array* con 100 elementos $p_i \in \{0, 1\}$.
Para decodificar una solución bastará con colocar en el terreno una placa solar en las posiciones correspondientes a las del *array* en las que haya un 1. Por ejemplo, para el terreno de la figura 1 la codificación sería $[1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1]$. Para decodificar la solución, la equivalencia es prácticamente directa.

3.2 Cálculo de la potencia y el coste de cada posición

Como se ha comentado anteriormente, cada posición del terreno tiene un coste de colocación de la placa y una potencia de generación de energía determinada. Estos valores son independientes del precio y la potencia individual de cada placa. Las funciones en el código 1 y el código 2 indican cómo se calcula para una posición del terreno su potencia y su coste asociado. Estos cálculos vienen influenciados por la potencia y el precio individual de cada

placa así como por el coste de la mano de obra en esa zona del terreno. La influencia de estos valores en el cálculo se determina de forma aleatoria pero siempre con la misma semilla (relativa a la posición de entrada) para poder reproducir las mismas condiciones de experimentación cada vez.

```
def calculo_potencia(n, potencia_p):
    random.seed(n+1)
    return (random.randint(1,5) * 0.25 * potencia_p)
```

Código 1: Función de cálculo de la potencia en una posición del terreno

```
def calculo_coste(n, precio_p):
    random.seed(n-1)
    return precio_p + mano_obra * (0.2 * random.randint(1,5))
```

Código 2: Función de cálculo del coste en una posición del terreno

3.3 Evaluación del *fitness*

Con tal de medir cómo de buena es una solución particular dada, se ha determinado una función de *fitness(individuo)*[1]. Ésta, dado un individuo en concreto, devuelve la potencia obtenida en todo el terreno que representa (a partir de las posiciones en las que sí se ha colocado una placa). Esta función a su vez, penaliza mucho aquellas soluciones que superan el presupuesto máximo, ya que es una restricción que debe cumplirse *siempre*. El código 3 indica cómo se ha calculado este valor de evaluación del individuo. Además de devolver el *fitness*, esta función también devuelve la potencia y el coste total del individuo (del terreno con una determinada distribución de placas).

```
def fitness(ind):
    potencia = 0
    costo = 0

    for i in range(len(ind)):
        if ind[i] == 1:
            potencia += calculo_potencia(i, potencia_p)
            costo += calculo_coste(i, precio_p)
    if costo > presupuesto:
        penalizacion = costo *100000
    else: penalizacion = 0

    return potencia, costo, (potencia - penalizacion)
```

Código 3: Función de fitness

[1]Esta nomenclatura se corresponde con la del algoritmo genético. En el caso del enfriamiento simulado se trata de la misma función pero se ha llamado simplemente *f(individuo)*

4 Implementación

La implementación, tanto del algoritmo genético como el algoritmo de enfriamiento simulado han sido implementados *from scratch*, esto significa sin ningún tipo de librería relativa a dichos algoritmos, a mano. Ambos se han programado en lenguaje *Python*. Cabe destacar que sí se han utilizado otras librerías como*numpy*, *math*, *random* o *time* con la finalidad de realizar cálculos más complejos que proporciones medidas de evaluación y condiciones experimentales determinadas. En las dos secciones siguientes, se describen el diseño e implementación de cada uno de los algoritmos, así como las pruebas realizadas en cada uno de ellso y los resultados obtenidos en las experimentaciones.

5 Solución con el AG

5.1 Construcción del algoritmo

Dentro de este algoritmo, se han parametrizado algunas variables y métodos para poder realizar la experimentación. A continuación se expresa cómo se han construido cada uno de los pasos del algoritmo:

5.1.1 Número de generaciones e individuos por generación

Estos valores se han dejado libres para poder variarlos y comprobar con qué cifras funciona mejor el algoritmo

5.1.2 Tipo de generación

Se han implementado dos tipos de generación de los individuos diferentes para realizar pruebas. Éstos son:

- **Generación aleatoria**: se generan *arrays* 100 elementos que contienen en cada posición un 1 o un 0 de forma totalmente aleatoria. Las soluciones con este tipo de generación pueden llegar a ser no factibles.
- **Generación aleatoria con coste**: es igual que la anterior pero cada vez que se determina si en una posición del *array* va un 1 o un 0 se calcula el coste acumulado de la solución. Si se supera, no se añade una placa en esa posición determinada. De esta forma se inicializan soluciones factibles que cumplen las restricciones.

5.1.3 Gestión de las restricciones

En cada iteración del algoritmo se evalúa la aptitud de cada individuo en la población. Para ello se usa la función *fitness*. Esta sección determina cómo se gestionan los individuos que no cumplen la restricción de coste. Se han planteado dos tipos de gestión diferentes:

- **Reparación**: se trata de, dada una solución, ir eliminando las placas que supongan un mayor coste total (estos elementos, por el diseño de las soluciones, se encuentran en las posiciones más altas del *array*) hasta que éste sea menor que el presupuesto dado o hasta que se supere el número de iteraciones (también parametrizado para realizar

pruebas) de reparación. Esto permite explotar en cierto modo la vecindad de cada individuo, realizando pequeñas variaciones en él.

- **Descarte**: si una solución determinada no cumple con las restricciones, se elimina directamente y se genera una solución aleatoria en su lugar. Esto permite explorar nuevas soluciones.

5.1.4 Tipo de selección

La selección de los padres en cada iteración se ha implementado también de dos formas diferentes:

- **Selección elitista**: de esta forma, se ordena la población según el **fitness** de cada individuo y se elige la mitad de la población que mejor resultado obtenga para que sean los padres.
- **Selección por ruleta**: en este caso, no se elige la mejor mitad de la población, sino que se eligen los individuos según una probabilidad asociada a su *fitness*. De esta forma, se da alguna oportunidad de selección a las soluciones que no son tan buenas, aunque con menor probabilidad que a las mejores.

5.1.5 Tipo de cruce

En este caso, se ha optado por dejar fija la forma en que se cruzan los individuos. Ésta se ha implementado a partir del método de los dos puntos, de forma que por cada pareja de padres, se generan dos hijos diferentes fruto de la combinación de las mitades de la codificación genética de los padres.

5.1.6 Mutación

Se muta cada posición de la solución representada por un hijo con una probabilidad aleatoria.

5.1.7 Reemplazo

Aquí también se ha dejado fijo el método de reemplazo, de forma que se eligen los mejores individuos (teniendo en cuenta tanto a padres como a hijos) según el número de individuos por generación indicado. Se trata de un remplazo elitista por generación.

5.2 Evaluación y resultados

Las pruebas han sido lanzadas a partir de un fichero en el cual se indicaban los valores de los parámetros que se querían probar. El código principal iteraba sobre este fichero para realizar las ejecuciones necesarias del algoritmo y probar todos los valores deseados.
Se han probado los diferentes tipos de funciones mencionados anteriormente junto con variedad de parámetros generacionales.
En la tabla a continuación, se muestra un resumen de los resultados obtenidos más destacables:

p [a]	c [b]	g [c]	*n-gen* [d]	*n-ind* [e]	*g-type* [f]	*s-type* [g]	*ev-type* [h]	it [i]	t [j]
11400	**19845**	68	100	100	coste	ruleta	reparación	30	40.03
11025	19857	80	100	50	coste	ruleta	reparación	30	**21.58**
10875	19827	26	50	100	coste	elitista	reparación	30	**15.66**
10875	19836	93	100	100	aleatoria	elitista	reparación	30	34.71

[a] potencia total (en W)
[b] coste total (en €)
[c] generación donde se ha encontrado la solución
[d] número de generaciones
[e] número de individuos
[f] tipo de generación
[g] tipo de selección
[h] tipo de evaluación
[i] número de iteraciones si el tipo de evaluación es *reparación*
[j] tiempo de ejecución (en segundos)

Aquí se puede observar que el mejor resultado obtenido ha sido el de potencia igual a $11400W$. Además, a medida que se optimiza la potencia, también lo hace el coste. Dentro de las soluciones de la tabla, cabe destacar la segunda mejor obtenida, ya que no presenta una diferencia tan significativa respecto del mejor resultado y sin embargo, su tiempo de ejecución es la mitad que el mejor. En la figura 2 se puede ver la evolución de estas dos configuraciones en conjunto. Durante la mayoría del tiempo evolucionan a la par y dado un momento, la configuración que mayor potencia obtiene crece más que la segunda opción. Esto puede deberse al número de individuos en la generación, que en el segundo caso es menor y por tanto tiene menos opciones de exploración.

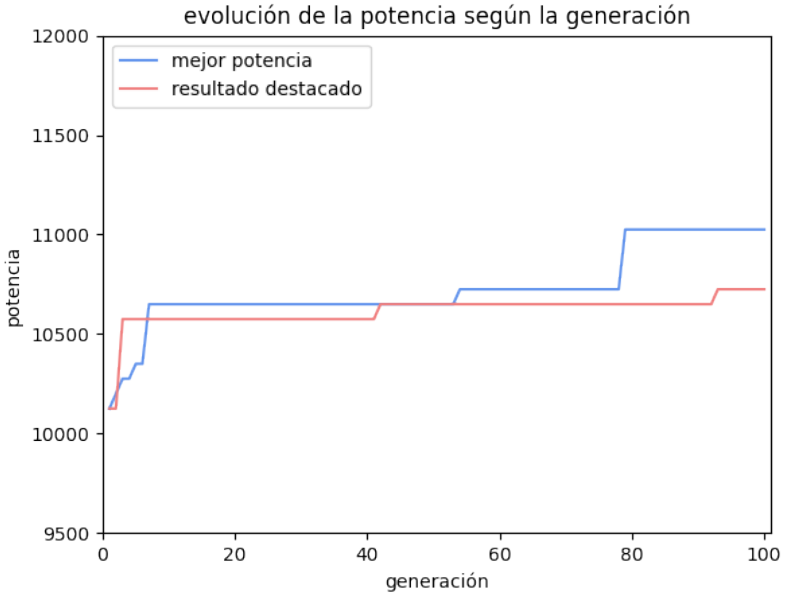

Figura 2: Comparativa de la evolución de la potencia con el paso de las generaciones para las dos mejores soluciones obtenidas.

Por último, si se tiene en cuenta un caso de ejemplo con 10 generaciones y 10 individuos por generación (en la figura 3), se ve cómo evoluciona la potencia según el número de

iteraciones de reparación en la evaluación. Esto indica una tendencia del algoritmo a bajar y estabilizarse la calidad de las soluciones cuando se supera un número determinado de iteraciones. Posiblemente por esta razón el número de iteraciones en las mejores soluciones expuestas anteriormente no varía.

Figura 3: Variación de la potencia según el número de iteraciones de reparación para 10 generaciones y 10 individuos por generación del algorimto genético.

6 Solución con el ES

6.1 Construcción del algoritmo

Para este algoritmo también se han dejado variables algunos de los parámetros y se han aplicado las funciones requeridas de una forma en específico. A continuación se indican los detalles.

6.1.1 Solución inicial

Se ha escogido como solución inicial una de las mejores soluciones para el algoritmo genético (de potencia igual a $10275W$). No se ha elegido la mejor para comprobar cómo de rápido encontraba una mejor solución el algoritmo de enfriamiento simulado.

6.1.2 Generación de vecindad

Se ha escogido que esta generación sea lo más aleatoria posible. La cantidad de cambios a realizar dentro del individuo es aleatoria, al igual que las posiciones concretas que se cambian, así como el cambio a realizar.

6.1.3 Variables libres

Las variables que se han dejado parametrizables para la experimentación han sido las siguientes:

- **Temperatura inicial (T_0)**: siendo un porcentaje de la *f(individuo)* inicial.
- Factor de **decremento de la temperatura (k)**
- **Iteraciones en cada temperatura (L)**

6.1.4 Función de decremento de la temperatura

Se ha optado por un decremento geométrico que aplica la variable k anterior y sigue la forma del código 4.

```
def omega(T,k):
    return T*k
```

Código 4: Función de decremento de la temperatura

6.2 Evaluación y resultados

Como en el caso del algoritmo genético, se han lanzado diferentes ejecuciones de forma automática probando distintos valores. Los mejores resultados se muestran en la tabla siguiente:

T_0 [a]	k [b]	L [c]	t [d]	p [e]	c [f]	T^* [g]
1027.5	0.88	70	16.3	**12675**	19827	0.4219
5137.5	0.9	70	21.07	12525	19824	0.0653
9247.5	0.9	70	22.11	12525	19833	0.3369
5137.5	0.9	70	17.92	12450	19842	0.0127
1027.5	0.88	20	**4.7**	11925	19860	0.4219

[a] temperatura inicial
[b] factor de decremento de T
[c] iteraciones en cada temperatura
[d] tiempo de ejecución (en segundos)
[e] potencia total (en W)
[f] coste total (en €)
[g] temperatura de la mejor solución

Aquí se puede observar que el mejor resultado obtenido es el de $T_0 = 1027.5$ (el 10% de la f inicial) con una potencia total de $12675W$. Cabe destacar el último resultado obtenido, que con menos iteraciones en cada temperatura (por tanto, menor tiempo de ejecución) obtiene un resultado también aceptable. En la figura 4 se observa cómo evolucionan estas dos opciones mencionadas a lo largo de la ejecución del algoritmo; la opción destacada llega a superar a la mejor opción aunque más adelante no consigue llegar a una potencia tan elevada.

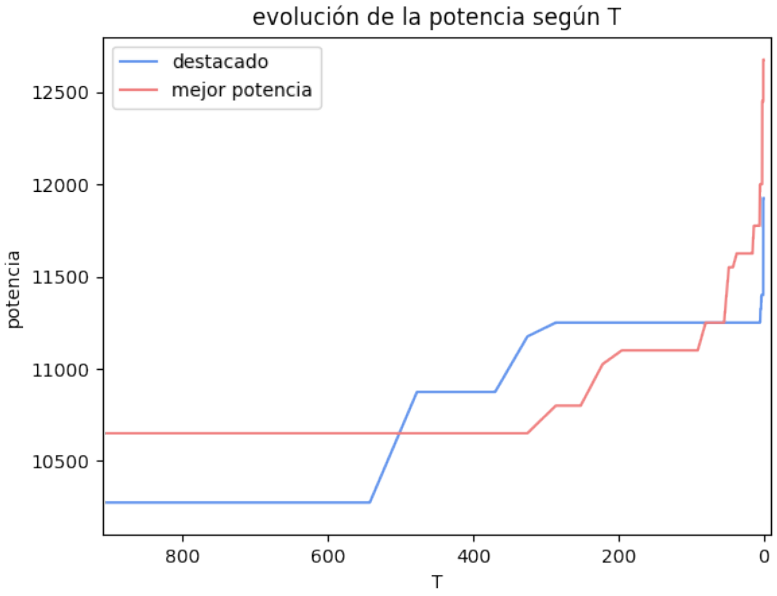

Figura 4: Comparativa de la variación de la potencia según el decremento de la temperatura (T) para la mejor combinación de parámetros y la combinación destacada.

Por otro lado, en la figura 5 se puede observar cómo influye la cantidad de tiempo que se está iterando en cada temperatura en la calidad de los resultados. A más tiempo explorando las soluciones en cada temperatura, mejores resultados. De todas formas, debido al factor de aleatoriedad del algoritmo, esta gráfica no presenta unas evoluciones totalmente estables, de forma que se da el caso en el cual alguna de las opciones que se mantiene menos tiempo en cada temperatura obtenga resultados mejores que la configuración que se mantiene más tiempo iterando.

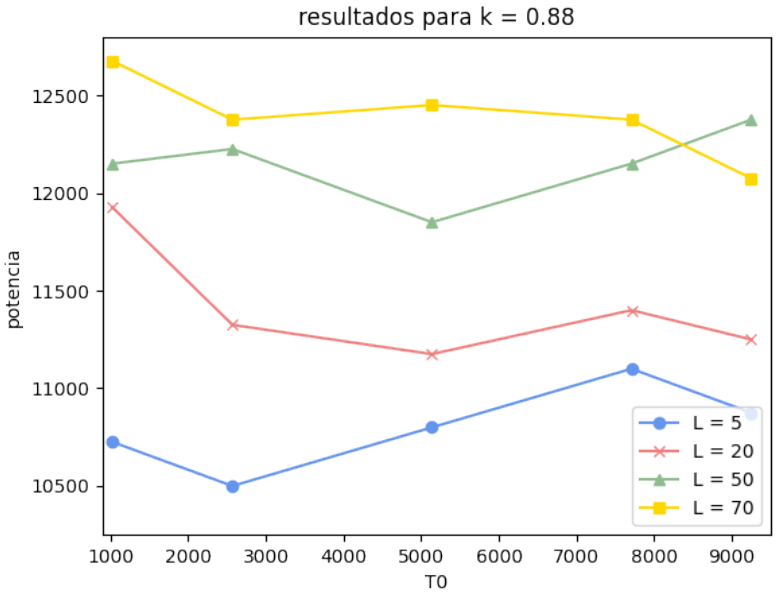

Figura 5: Comparativa para diferentes valores de L de la variación de la mejor potencia obtenida según la temperatura inicial (T_0) utilizada para el factor de decremento de la temperatura (k) utilizado.

7 Conclusiones

En cuanto al algoritmo genético, un factor a tener en cuenta, sobre todo en lo relativo al tiempo de ejecución, son el número de generaciones y la cantidad de individuos por generación. Si bien es verdad que a mayor cantidad de datos, mejor puede llegar a ser la solución, esta cantidad de datos también influencia en el coste computacional y temporal. A la vista de los resultados, está claro que puede encontrarse un equilibrio entre la cantidad de datos con la cual se trabaja y la calidad de la solución. Si hay un compromiso entre ambos se podrán obtener soluciones que, aunque sean un poco peores que la mejor obtenida, tardarán menos en ser calculadas.
Por otro lado, la elección de la forma de generación inicial influye en la calidad de las soluciones iniciales. La generación aleatoria puede dar lugar a soluciones no factibles que superen el presupuesto. En contraste, la generación aleatoria con coste asegura que las soluciones iniciales sean factibles. Esto explica por qué las configuraciones que utilizan la generación aleatoria con coste tienden a obtener mejores resultados, ya que las soluciones iniciales cumplen con las restricciones de presupuesto. Además, la selección elitista suele conducir a soluciones de mejor calidad, especialmente cuando se inicia con estas soluciones factibles. También cabe considerar la reparación, que permite que las soluciones no factibles evolucionen y mejoren con el tiempo, explorando la vecindad.

En cuando al algoritmo de enfriamiento simulado, se puede decir que la temperatura inicial y el factor de decremento son cruciales. Una temperatura inicial más alta permite una exploración amplia del espacio de soluciones en las primeras etapas, mientras que un factor de decremento más bajo da paso a un enfriamiento gradual. La combinación adecuada de

ambos parámetros lleva a encontrar soluciones de alta calidad.
Otro factor es el número de iteraciones en cada temperatura, que también desempeña un papel importante. Un tiempo mayor en cada temperatura nos deja explorar soluciones de forma más exhaustiva, pero también puede aumentar el tiempo de ejecución. En general, como en el algoritmo genético, es necesario encontrar equilibrio adecuado entre el número de iteraciones y la temperatura inicial.

Ambos algoritmos han demostrado ser capaces de resolver el problema de optimización planteado de forma adecuada, aunque el enfriamiento simulado a resultado en soluciones de mejor calidad en un menor tiempo de ejecución.
El mejor resultado ha sido el de $p = 12675W$ con una $T_0 = 1027.5$, una $k = 0.88$ y $L = 70$[2].
En cuanto a las limitaciones del trabajo y la perspectiva de futuro, hay margen para mejorar la eficiencia y la calidad de las soluciones. Se podría explorar con estrategias más avanzadas de inicialización (a partir de algún algoritmo de optimización previo), por ejemplo, además de con tamaños de problemas más realistas y complejos para evaluar su escalabilidad.

[2]No se muestra la codificación del mejor individuo a pesar de haberse calculado, ya que el *array* de cien elementos ocuparía mucho espacio visual en esta memoria.