Problema de optimización de compras por Internet

Integrantes:
Juan Andrés Romero (202013449) y Juan Sebastián Alegría (202011282)
Entrega 3: Propuesta de un algoritmo para comparar su resultado con el Modelo Matemático
Modelado, Simulación y Optimización

Departamento de Ingeniería de Sistemas y Computación
Universidad de Los Andes
Bogotá, Colombia

1 Descripción del Problema

1.1 Contexto del problema:

Dada la amplia cantidad de tiendas online que existen actualmente, surge la dificultad de los compradores de revisar manualmente todas las ofertas disponibles y seleccionar las mejores elecciones de compra manteniendo un costo bajo.
Para obtener una solución parcial al problema, existen los llamados comparadores de precio que generan un ranking de precios de dichas ofertas, sin embargo, estos están limitados a trabajar con la comparación de ofertas de un único producto al tiempo.

Ahora bien, la solución más óptima, que es la que se desea implementar, requiere comprobar todas las combinaciones de costos existentes para de este modo, obtener el menor costo de compra posible dado por la combinación entre costos de envío y de productos. Este problema es de naturaleza NP-hard.

1.2 Descripción formal del problema:

Dado un comprador que busca un conjunto de productos N para comprar en las tiendas de L, existe un conjunto de productos disponibles P, un costo C_{jl} de cada producto y un costo D_l de envío que están asociados a cada tienda.

1.3 Limitaciones del problema

Como restricciones se tiene que:

- Si el producto no está disponible, no se puede comprar (costo infinito) y por ende se descarta de N.
- De los productos disponibles que quiere el comprador, es necesario tener solamente uno de ellos seleccionado en alguna de las tiendas.

- El costo de envío de los productos de una tienda, es independiente de la cantidad de productos comprados.

Objetivo que se quiere lograr: Minimizar la sumatoria de costos de envío de tiendas seleccionadas sumado a la sumatoria del conjunto de productos seleccionados en una tienda. En resumen, se quiere minimizar el costo total de los productos seleccionados que a su vez tengan los mínimos costos de envío.

1.4 Ejemplo

A continuación se presenta una muestra de ejemplo, en la cual se evalúan 6 tiendas y se desea comprar 5 libros. Luego, en la segunda tabla se evidencia la solución limitada que usualmente ofrecen los comparadores de precio. Por otro lado, la tercera tabla muestra la solución óptima de menor costo.

Table 1. Prices of books and delivery costs offered by six internet shops.

Cost	Book a	Book b	Book c	Book d	Book e	Delivery	Total
Shop 1	18	39	29	48	59	10	203
Shop 2	24	45	23	54	44	15	205
Shop 3	22	45	23	53	53	15	211
Shop 4	28	47	17	57	47	10	206
Shop 5	24	42	24	47	59	10	206
Shop 6	27	48	20	55	53	15	218

Table 2. Price comparator solution—the result of the selection process.

	Book a	Book b	Book c	Book d	Book e	Delivery	Total
Price	18	39	17	47	44	45	210
Shop	Shop 1	Shop 1	Shop 4	Shop 5	Shop 2		

Table 3. Optimal purchase cost in selected shops.

	Book a	Book b	Book c	Book d	Book e	Delivery	Total
Cost	18	39	17	48	47	20	189
Shop	Shop 1	Shop 1	Shop 4	Shop 1	Shop 4		

Como se evidencia en el anterior ejemplo, dado un set de costos productos por cada tienda y un costo de entrega para esa tienda, un comparador de precios intenta elegir el libro de su tipo que tenga el menor costo y luego suma los costos de envío correspondientes, pero esto no garantiza obtener el menor costo total de compra.

De este modo, si probamos todas las posibles combinaciones de costos se elige

el libro d y e de una tienda con un costo más caro para estos, pero al final obtenemos un costo total menor. Por esto, este problema es NP-Hard.

2 Conjuntos, Parámetros y Variables de decisión

A continuación se presenta la simbología para los conjuntos, parámetros y variables a utilizar en el problema

Table 1. Conjuntos y parámetros.

Sets and Parameters	**Description**
P	Conjunto de Productos.
L	Conjunto de Tiendas.
N	Conjunto de Productos disponibles que quiere el comprador
C_{jl}	Costo de cada producto j en la tienda l
D_l	Costo de envío desde la tienda l

Table 2. Variables de decisión

Variables	**Description**
X_{jl}	Determina si un producto j de una tienda l es escogido o no para ser comprado (Variable Binaria)
Y_l	Determina si se realiza una compra en la tienda l. (Variable Binaria)

3 Función Objetivo y Restricciones

3.1 Función Objetivo

$$min\left(\sum_{l\in L}\left(\sum_{j\in P}(C_{jl}X_{jl})+D_lY_l\right)\right) \tag{1}$$

Explicación: La función objetivo busca minimizar la sumatoria de los costos de los productos elegidos para todas las tiendas. Si una tienda tiene al menos un producto elegido, la variable binaria Y_l nos permitirá incluir a la sumatoria final el costo de envío correspondiente a dicha tienda.

3.2 Restricciones:

Si se compra en l, se activa Y_l

$$\sum_{j \in P} X_{jl} - card(P) Y_l \leq 0 \qquad \forall l \in L \tag{2}$$

Explicación: Esta restricción indica que si se elige al menos un producto de una tienda l, se tiene que activar la variable Y, pues esta significa que se está realizando una compra en la tienda l.

Nota: Card(P) significa la cardinalidad del conjunto P, se eligió esta como constante para multiplicar la Y en la ecuación, debido a que por otras restricciones, la suma de las X_{ji} nunca será igual a ella.

Se deben de comprar los productos que quiere el comprador

$$\sum_{l \in L} X_{jl} = 1 \qquad \forall j \in N \tag{3}$$

Explicación: En esta restricción se indica que por cada producto disponible que quiere el comprador se tiene que comprar exactamente 1 de ellos. Es decir, no es posible comprar más de un producto específico por tienda.

Nota: Para descartar los productos no disponibles de N, se debe de realizar un preprocesamiento del problema a partir de los costos de cada producto. Esto se puede ver como un ciclo antes del cálculo del modelo que va a agregando a N los productos que están disponibles para comprar.

4 Implementación y resultados del Modelo Matemático

Para la comprobación de la implementación del modelo matemático se decidió implementar dos escenarios diferentes, uno sencillo y uno más complejo. Para el primero, se realizará un escenario simple con 3 tiendas y 3 productos, de los cuales, el cliente solo quiere 2 de ellos. De igual manera, se colocarán valores de precios similares de manera que no haya una solución evidente para este problema.

Por otro lado, en el segundo escenario se usarán los mismos datos enseñados en la prueba de ejemplo de la sección 1.4 de este documento y se intentará revisar si los resultados del modelo implementado en Pyomo coinciden.
En este caso hay un problema con 6 tiendas y 5 productos diferentes, donde el cliente quiere obtener por lo menos uno de cada uno. Esta selección de elementos, hace que el segundo escenario sea más complejo porque la cantidad de posibilidades a elegir aumenta con la cantidad de productos que quiere el cliente.

4.1 Escenario 1

Para el primer escenario, se utilizarán únicamente 3 tiendas y 3 productos, haciendo también que el cliente quiera solo dos productos. Para este caso, dichos productos serán: Producto A y Producto C.

La distribución de precios por tienda se puede evidenciar en la siguiente tabla

Table 3. Escenario 1. Precio de productos y costo de envío ofrecidos por 3 tiendas online

Costo	Producto A	Producto B	Producto C	Envío
Tienda 1	7	6	15	12
Tienda 2	13	4	10	10
Tienda 3	10	6	9	15

4.2 Resultados Escenario 1

En la siguiente imagen se puede ver que tras la ejecución del algoritmo de solución con Pyomo, se eligió únicamente la tienda 2 para obtener el menor costo de la compra de A y C.
Igualmente, en esta selección se llegó a un costo total de 33 unidades. Esto demuestra que la solución más factible no fue la tienda que tenía los precios más bajos sino otra que era más cara, pero su costo de envío era más bajo.

```
project on  main via 🐍 v3.10.8
➜ /opt/homebrew/bin/python3 /Users/zejiran/hack/uniandes/MSO/project/scenario1.py
Model unknown

  Variables:
    x : Size=9, Index=x_index
        Key    : Lower : Value : Upper : Fixed : Stale : Domain
        (1, 1) :     0 :   0.0 :     1 : False : False : Binary
        (1, 2) :     0 :   1.0 :     1 : False : False : Binary
        (1, 3) :     0 :   0.0 :     1 : False : False : Binary
        (2, 1) :     0 :   0.0 :     1 : False : False : Binary
        (2, 2) :     0 :   0.0 :     1 : False : False : Binary
        (2, 3) :     0 :   0.0 :     1 : False : False : Binary
        (3, 1) :     0 :   0.0 :     1 : False : False : Binary
        (3, 2) :     0 :   1.0 :     1 : False : False : Binary
        (3, 3) :     0 :   0.0 :     1 : False : False : Binary
    y : Size=3, Index=L
        Key : Lower : Value : Upper : Fixed : Stale : Domain
          1 :     0 :   0.0 :     1 : False : False : Binary
          2 :     0 :   1.0 :     1 : False : False : Binary
          3 :     0 :   0.0 :     1 : False : False : Binary

  Objectives:
    targetFunc : Size=1, Index=None, Active=True
        Key  : Active : Value
        None :   True :  33.0
```

4.3 Escenario 2

En este segundo escenario se implementará el ejemplo mostrado en la sección 1.4, el cual estaba conformado por 6 tiendas, 5 diferentes productos y el cliente quería obtener todos los anteriores productos, es decir del A al E. En este caso, los valores de los pares son los mismos de dicho ejemplo.

Table 4. Escenario 2. Precio de productos y costo de envío ofrecidos por 6 tiendas online

Costo	Producto A	Producto B	Producto C	Producto D	Producto E	Envío	Total
Tienda 1	18	39	29	48	59	10	203
Tienda 2	24	45	23	54	44	15	205
Tienda 3	22	45	23	53	53	15	211
Tienda 4	28	47	17	57	47	10	206
Tienda 5	24	42	24	47	59	10	206
Tienda 6	27	48	20	55	53	15	218

4.4 Resultados Escenario 2

Tras ejecutar el modelo, se identificó que los resultados coincidían satisfactoriamente con lo expuesto en el ejemplo explicado anteriormente.

Se puede ver que los pares elegidos fueron: Tienda 1 para los productos 1, 2 y 4, y tienda 4 para los productos 3 y 5. Resultando con un costo de compra total de 189, el cual es mucho menor que si se hubiera comprado en la tienda con el costo total más barato.

Ejecución del segundo escenario

```
project on  main via 🐍 v3.10.8
➜ /opt/homebrew/bin/python3 /Users/zejiran/hack/uniandes/MSO/project/scenario2.py
Model unknown

  Variables:
    x : Size=30, Index=x_index
        Key    : Lower : Value : Upper : Fixed : Stale : Domain
        (1, 1) :     0 :   1.0 :     1 : False : False : Binary
        (1, 2) :     0 :   0.0 :     1 : False : False : Binary
        (1, 3) :     0 :   0.0 :     1 : False : False : Binary
        (1, 4) :     0 :   0.0 :     1 : False : False : Binary
        (1, 5) :     0 :   0.0 :     1 : False : False : Binary
        (1, 6) :     0 :   0.0 :     1 : False : False : Binary
        (2, 1) :     0 :   1.0 :     1 : False : False : Binary
        (2, 2) :     0 :   0.0 :     1 : False : False : Binary
        (2, 3) :     0 :   0.0 :     1 : False : False : Binary
        (2, 4) :     0 :   0.0 :     1 : False : False : Binary
        (2, 5) :     0 :   0.0 :     1 : False : False : Binary
        (2, 6) :     0 :   0.0 :     1 : False : False : Binary
        (3, 1) :     0 :   0.0 :     1 : False : False : Binary
        (3, 2) :     0 :   0.0 :     1 : False : False : Binary
        (3, 3) :     0 :   0.0 :     1 : False : False : Binary
        (3, 4) :     0 :   1.0 :     1 : False : False : Binary
        (3, 5) :     0 :   0.0 :     1 : False : False : Binary
        (3, 6) :     0 :   0.0 :     1 : False : False : Binary
        (4, 1) :     0 :   1.0 :     1 : False : False : Binary
        (4, 2) :     0 :   0.0 :     1 : False : False : Binary
        (4, 3) :     0 :   0.0 :     1 : False : False : Binary
        (4, 4) :     0 :   0.0 :     1 : False : False : Binary
        (4, 5) :     0 :   0.0 :     1 : False : False : Binary
        (4, 6) :     0 :   0.0 :     1 : False : False : Binary
        (5, 1) :     0 :   0.0 :     1 : False : False : Binary
        (5, 2) :     0 :   0.0 :     1 : False : False : Binary
        (5, 3) :     0 :   0.0 :     1 : False : False : Binary
        (5, 4) :     0 :   1.0 :     1 : False : False : Binary
        (5, 5) :     0 :   0.0 :     1 : False : False : Binary
        (5, 6) :     0 :   0.0 :     1 : False : False : Binary
    y : Size=6, Index=L
        Key : Lower : Value : Upper : Fixed : Stale : Domain
          1 :     0 :   1.0 :     1 : False : False : Binary
          2 :     0 :   0.0 :     1 : False : False : Binary
          3 :     0 :   0.0 :     1 : False : False : Binary
          4 :     0 :   1.0 :     1 : False : False : Binary
          5 :     0 :   0.0 :     1 : False : False : Binary
          6 :     0 :   0.0 :     1 : False : False : Binary

  Objectives:
    targetFunc : Size=1, Index=None, Active=True
        Key  : Active : Value
        None :   True : 189.0
```

5 Algoritmo propuesto

Para esta etapa se propusieron dos algoritmos. En principio una solución de fuerza bruta que encuentra todas las permutaciones posibles de la solución y devuelve la de menor costo.

Luego, se implementó una segunda meta-heurística basada en un algoritmo evolutivo genético que crea individuos en distintas generaciones tratando de encontrar la solución a partir de una función de fitness (puntaje o ranking), la cual se intenta maximizar para obtener los menores costos de compra. Para este algoritmo evolutivo, se utilizó la librería de PyGAD que nos permite abstraer todo el proceso de creación de un algoritmo genético, definir un tipo de mutación aleatorio, los individuos a variar por generación, el número de genes a alterar en cada cromosoma, el rango de valores para generar mutaciones, y el número de generaciones a utilizar.

En esencia, en estos algoritmos se trabajó con un arreglo sencillo con los costos de envío de cada tienda, trabajando el número de la tienda como los índices de dicho arreglo. Es decir, la posición 0 pertenece a la tienda 1, la posición 2 a la tienda 2 y así hasta acabar con el arreglo. Una matriz de costos de los productos por cada tienda y un arreglo sencillo con los productos que quiere el usuario. Por lo tanto, los parámetros que usó el algoritmo fueron los siguientes:

```
# Costos de cada tienda
store_delivery_costs = [12, 10, 15]

# Productos seleccionados
selected_products = [1, 3]

# Matriz de costos de producto/tienda
product_costs = Matrix(3, 3)
```

Cabe resaltar que creamos una clase *Matrix* que contiene la matriz de costos, pero que en esencia funciona de la misma manera que una matriz normal. En cuanto a la solución devuelta por los algoritmos, esta se interpreta de la siguiente manera:

```
# Fuerza Bruta
Minimum cost: 189
Time: 0.04607200622558594
Permutation: [1, 1, 4, 1, 4]

# Algoritmo Evolutivo
Best fitness value reached after 454 generations.
Minimum cost found: 189
Time taken: 0.12108993530273438
Solution: [1 1 4 1 4]
```

Minimum Cost contiene el costo total mínimo encontrado por el algoritmo, time es el tiempo que se demoró encontrándolo y Permutation o Solution es el arreglo que contiene la solución de la mejor combinación de producto/tienda encontrado que resulta en el costo mencionado anteriormente. Este arreglo de solución se interpreta de la siguiente manera: La posición representa el producto, es decir, posición 0 = producto 1, posición 1 = producto 2, etc. Y adicionalmente, el valor de cada posición representa la tienda de la cual se seleccionó el producto.

Ejemplo:

```
Permutation: [1, 1, 4, 1, 4]
```

- El producto 1 se compra en la tienda 1
- El producto 2 se compra en la tienda 1
- El producto 3 se compra en la tienda 4
- El producto 4 se compra en la tienda 1
- El producto 5 se compra en la tienda 4

5.1 Pseudocódigo del algoritmo

Algorithm 1 Brute Force Algorithm Pseudocode.

```
1:  Initialize Store Delivery Costs = []
2:  Initialize Selected Products = []
3:  Initialize Product Costs = [[], [], ...[]]
4:  Initialize Cost List = []
5:
6:  Find all possible permutations from the input parameters
7:  for every permutation do
8:      if product not in selected_products then
9:          Skip permutation
10:     end if
11:     Initialize total_cost = 0
12:     for store/product pair in permutation do
13:         total_cost = total_cost + product_cost from said store
14:     end for
15:     for Store selected in permutation do
16:         total_cost = total_cost + delivery cost from said store
17:     end for
18:     cost_list.insert(total_cost)
19: end for
20:
21: Find minimum cost in Cost List
22: Return cost found
```

Para el pseudocódigo del algoritmo de fuerza bruta, se implementó una secuencia sencilla de instrucciones que permiten hallar con certeza el costo mínimo del problema según sus parámetros de entrada.
En esencia, este pseudocódigo se basa en crear las estructuras de datos necesarias en las líneas 1 a 4, luego hallar todas las permutaciones posibles de los parámetros de entrada en la línea 6, y finalmente, de la 7 a la 19 por cada permutación revisar

si ésta tiene algún producto que no haya sido seleccionado por el cliente, cosa que en dado caso, se salta y se procede a la siguiente permutación.
Luego, si se cumple con los productos incluidos en la permutación, por cada par de tienda/producto seleccionado, se calcula el costo de dicha combinación en el total de la orden y luego se le agrega el costo de envío de las tiendas involucradas en dicha permutación.
Finalmente, después de calcularlo, se agrega el costo total de la permutación y se inserta en la lista de costos. Una vez se completa el procesamiento de todas las permutaciones, se procede a encontrar el valor mínimo de los costos y su respectiva permutación.
De esta manera, se encuentra el costo mínimo que es pedido por el problema.

Algorithm 2 Evolutionary Algorithm Pseudocode.

```
1:  Initialize Store Delivery Costs = []
2:  Initialize Selected Products = []
3:  Initialize Product Costs = [[], [], ...[]]
4:
5:  Start of fitness function definition
6:  if product not in selected_products then
7:      Selected Store = 0
8:  end if
9:  Initialize Total Cost of Individual = 0
10: for index in individual do
11:     Selected Store = stores[index]
12:     if Selected Store not equals 0 and Product is selected then
13:         Add current cost to total cost of individual
14:     end if
15: end for
16: for store in Selected Stores do
17:     Sum delivery cost to total
18: end for
19: Return negative total cost
20: End of fitness function definition
21:
22: Initialize PyGAD Instance = [Genetic Execution Parameters]
23:
24: Execute PyGAD Instance
25: Print Fitness Value, Generations and Solution
26: Plot Generation VS Fitness
```

Ahora en el algoritmo evolutivo, inicializamos los parámetros del escenario de la línea 1 a 3. Luego de esto, se creó una función de fitness de la línea 5 a la 20, la cual es la encargada de revisar cada individuo para otorgarle un puntaje de acuerdo al costo de compra total que se calcula a partir de los costos de los productos seleccionados y los costos de envío. Este costo de compra se retorna en la función de fitness de forma negativa para que la instancia de PyGAD lo intente maximizar durante la creación de individuos.
En la línea 22, se crea la instancia de PyGAD con todos los parámetros géneticos de ejecución.
Finalmente, de la línea 24 a la 26, se ejecuta la instancia de PyGAD y se muestran los resultados.

6 Resultados del Algoritmo vs Modelo Matemático

Notas Adicionales:
Variando algunos de los atributos de entrada del algoritmo genético, obtuvimos tiempos de ejecución mucho más rápidos pero que se alejaban un poco de la respuesta de los modelos matemáticos, por lo que decidimos aumentar el número de generaciones con el fin de crear un algoritmo exacto.
De igual manera, debido al bajo número de permutaciones en el algoritmo de fuerza bruta, éste terminó siendo mucho más rápido en ambos escenarios que los algoritmos genéticos.

Sin embargo, la complejidad del algoritmo de fuerza bruta es O(n!), lo cual hace que sea extremadamente ineficiente si los parámetros de entrada tienen dimensiones lo suficientemente grandes.
Mientras que la complejidad del algoritmo genético, es calculada en base a las generaciones, creación de individuos, y formación de cromosomas. En escenarios con grandes cantidades de datos podría llegar a encontrar aproximaciones de solución de forma más eficiente que el algoritmo de fuerza bruta, siendo un algoritmo no exacto con el fin de ejecutarse bastante rápido.

6.1 Escenario 1

Parámetros a tener en cuenta del escenario 1:
Productos Seleccionados: 1 y 3

Table 5. Escenario 1. Precio de productos y costo de envío ofrecidos por 3 tiendas online

Costo	Producto A	Producto B	Producto C	Envío
Tienda 1	7	6	15	12
Tienda 2	13	4	10	10
Tienda 3	10	6	9	15

Resultados Escenario 1 En este escenario obtuvimos los mismos resultados del modelo matemático tanto en el algoritmo de fuerza bruta como el algoritmo genético, encontramos que el costo mínimo fue de 33 y la solución de este fue la siguiente:

- Producto 1 comprado en tienda 2
- Producto 3 comprado en tienda 2

Estos resultados se pueden apreciar en los siguientes pantallazos.

Resultados Escenario 1 Algoritmo Fuerza Bruta

```
project on  main via  v3.10.8
 3% ➜ /opt/homebrew/bin/python3 /Users/zejiran/hack/uniandes/MSO/project/brute-force-algorithm/scenario1.py
Minimum cost: 33
Time: 4.315376281738281e-05
Permutation: [2, 0, 2]
```

Resultados Escenario 1 Algoritmo Genético/Evolutivo

```
project on  main via  v3.10.8
 3% ➜ /opt/homebrew/bin/python3 /Users/zejiran/hack/uniandes/MSO/project/evolutionary-algorithm/scenario1.py
Best fitness value reached after 2 generations.

Minimum cost found: 33
Time taken: 0.01537013053894043
Solution: [2 0 2]
```

Estos resultados son exactamente los mismos que los del modelo matemático debido a que en primer lugar, el algoritmo de fuerza bruta siempre analiza todas las posibilidades, por lo que logra encontrar el costo menor. Y por otro lado, en el algoritmo evolutivo, generación tras generación, sobreviven los resultados que más se acercan al costo objetivo. En principio, decidimos por establecer una cantidad de generaciones elevado para que se acerque mucho más al resultado matemático, esto hace que el tiempo de ejecución pueda ser más lento, pero casi que nos garantiza un valor mucho más preciso.

6.2 Escenario 2

Parámetros a tener en cuenta para el escenario 2:
Productos Seleccionados: 1, 2, 3, 4 y 5

Table 6. Escenario 2. Precio de productos y costo de envío ofrecidos por 6 tiendas online

Costo	Producto A	Producto B	Producto C	Producto D	Producto E	Envío	Total
Tienda 1	18	39	29	48	59	10	203
Tienda 2	24	45	23	54	44	15	205
Tienda 3	22	45	23	53	53	15	211
Tienda 4	28	47	17	57	47	10	206
Tienda 5	24	42	24	47	59	10	206
Tienda 6	27	48	20	55	53	15	218

Resultados Escenario 2 En este escenario, los resultados obtenidos fueron de naturaleza similar a los encontrados en el escenario 1. Es decir, se encontró que se obtuvieron los mismos resultados que los encontrados en el modelo matemático donde el costo menor hallado fue de 189 y la solución de las combinaciones de producto/tienda fueron las siguientes:

- El producto 1 se compra en la tienda 1
- El producto 2 se compra en la tienda 1
- El producto 3 se compra en la tienda 4
- El producto 4 se compra en la tienda 1
- El producto 5 se compra en la tienda 4

Lo cual se puede observar en los siguientes pantallazos:

Resultados Escenario 2 Algoritmo de Fuerza Bruta

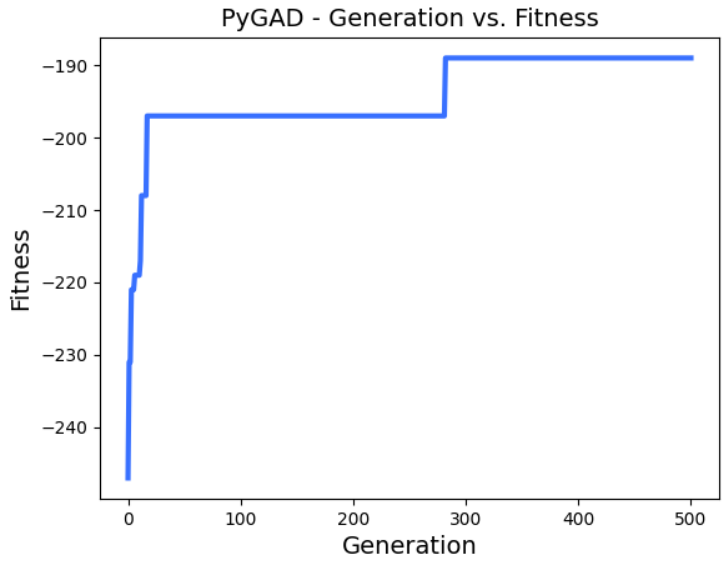
```
project on  main via 🐍 v3.10.8
 3% ➜ /opt/homebrew/bin/python3 /Users/zejiran/hack/uniandes/MSO/project/brute-force-algorithm/scenario2.py
Minimum cost: 189
Time: 0.0039522647857666016
Permutation: [1, 1, 4, 1, 4]
```

Resultados Escenario 2 Algoritmo Genético/Evolutivo

```
project on  main via 🐍 v3.10.8 took 2s
 3% ➜ /opt/homebrew/bin/python3 /Users/zejiran/hack/uniandes/MSO/project/evolutionary-algorithm/scenario2.py
Best fitness value reached after 142 generations.

Minimum cost found: 189
Time taken: 0.04370880126953125
Solution: [1 1 4 1 4]
```

Gráfica del avance de los valores del fitness a medida que avanzan las generaciones



7 Conclusiones

De estos resultados, se puede concluir que los dos algoritmos implementados trabajan de una buena manera y logran encontrar la misma solución que la que es entregada por el modelo matemático. Para este caso, en ambos escenarios se utiliza un número mayor de generaciones y por tanto el algoritmo evolutivo siempre toma mucho más para completar que el algoritmo de fuerza bruta, sin embargo, se estima que a partir de parámetros mayores a 8 en tanto productos como tiendas, ya hará que este algoritmo tome demasiado tiempo para completar.