# Informe TP 3 AE

## Ejercicio 1:

### Punto a:

```
import numpy as np


# función objetivo a maximizar
def f(x):
    return 375 * x[0] + 275 * x[1] + 475 * x[2] + 325 * x[3]  # funcion
objetivo: 3x1 + 5x2


# primera restriccion
def g1(x):
    return 2.5 * x[0] + 1.5 * x[1] + 2.75 * x[2] + 2 * x[3] <= 640  #
restricción de 640 horas fabricacion.
# segunda restriccion
def g2(x):
    return 3.5 * x[0] + 3 * x[1] + 3 * x[2] + 2 * x[3] <= 960  #
restriccion: de 960 horas acabados.


# parametros
n_particles = 20  # numero de particulas en el enjambre
n_dimensions = 4  # dimensiones del espacio de busqueda (x1, x2, x3 y
x4)
max_iterations = 50  # numero máximo de iteraciones para la
optimizacion
c1 = c2 = 1.4944  # coeficientes de aceleracion
w = 0.6  # Factor de inercia
# inicialización de particulas
x = np.zeros((n_particles, n_dimensions))  # matriz para las posiciones
de las particulas
v = np.zeros((n_particles, n_dimensions))  # matriz para las
velocidades de las particulas
```

```
pbest = np.zeros((n_particles, n_dimensions))  # matriz para los
mejores valores personales
pbest_fit = -np.inf * np.ones(n_particles)  # vector para las mejores
aptitudes personales (inicialmente -infinito)
gbest = np.zeros(n_dimensions)  # mejor solución global
gbest_fit = -np.inf  # mejor aptitud global (inicialmente -infinito)


# inicializacion de particulas factibles
for i in range(n_particles):
    while True:  # bucle para asegurar que la particula sea factible
        x[i] = np.random.uniform(0, 10, n_dimensions)  # inicializacion
posicion aleatoria en el rango [0, 10]
        if g1(x[i]) and g2(x[i]):  # se comprueba si la posicion cumple
las restricciones
            break  # Salir del bucle si es factible
    v[i] = np.random.uniform(-1, 1, n_dimensions)  # inicializar
velocidad aleatoria
    pbest[i] = x[i].copy()  # ee establece el mejor valor personal
inicial como la posicion actual
    fit = f(x[i])  # calculo la aptitud de la posicion inicial
    if fit > pbest_fit[i]:  # si la aptitud es mejor que la mejor
conocida
        pbest_fit[i] = fit  # se actualiza el mejor valor personal

# Optimizacion
for _ in range(max_iterations):  # Repetir hasta el número máximo de
iteraciones
    for i in range(n_particles):
        fit = f(x[i])  # Se calcula la aptitud de la posicion actual
        # Se comprueba si la nueva aptitud es mejor y si cumple las
restricciones
        if fit > pbest_fit[i] and g1(x[i]) and g2(x[i]):
            pbest_fit[i] = fit  # Se actualiza la mejor aptitud
personal
            pbest[i] = x[i].copy()  # Se actualizar la mejor posicion
personal
            if fit > gbest_fit:  # Si la nueva aptitud es mejor que la
mejor global
                gbest_fit = fit  # Se actualizar la mejor aptitud
global
                gbest = x[i].copy()  # Se actualizar la mejor posicion
global
```

```
        # actualizacion de la velocidad de la particula
        v[i] = w * v[i] + c1 * np.random.rand() * (pbest[i] - x[i]) +
c2 * np.random.rand() * (gbest - x[i])
        x[i] += v[i]  # Se actualiza la posicion de la particula

        # se asegura de que la nueva posicion esté dentro de las
restricciones
        if not (g1(x[i]) and g2(x[i])):
            # Si la nueva posicion no es válida, revertir a la mejor
posicion personal
            x[i] = pbest[i].copy()


# Se imprime la mejor solucion encontrada y también su valor optimo
print(f"Mejor solucion: [{gbest[0]:.4f}, {gbest[1]:.4f},
{gbest[2]:.4f}, {gbest[3]:.4f}]")
print(f"Valor optimo: {gbest_fit}")
```

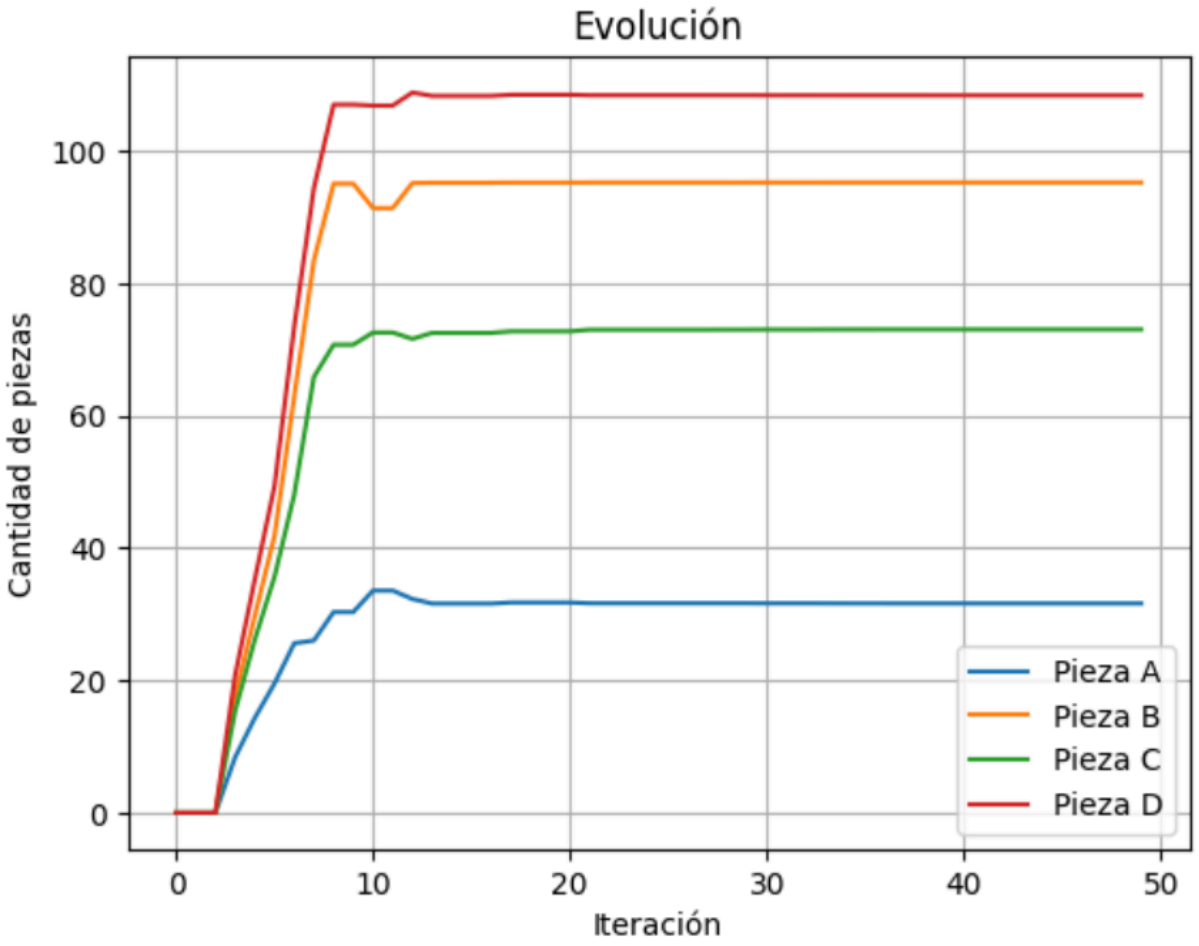
## Punto b:

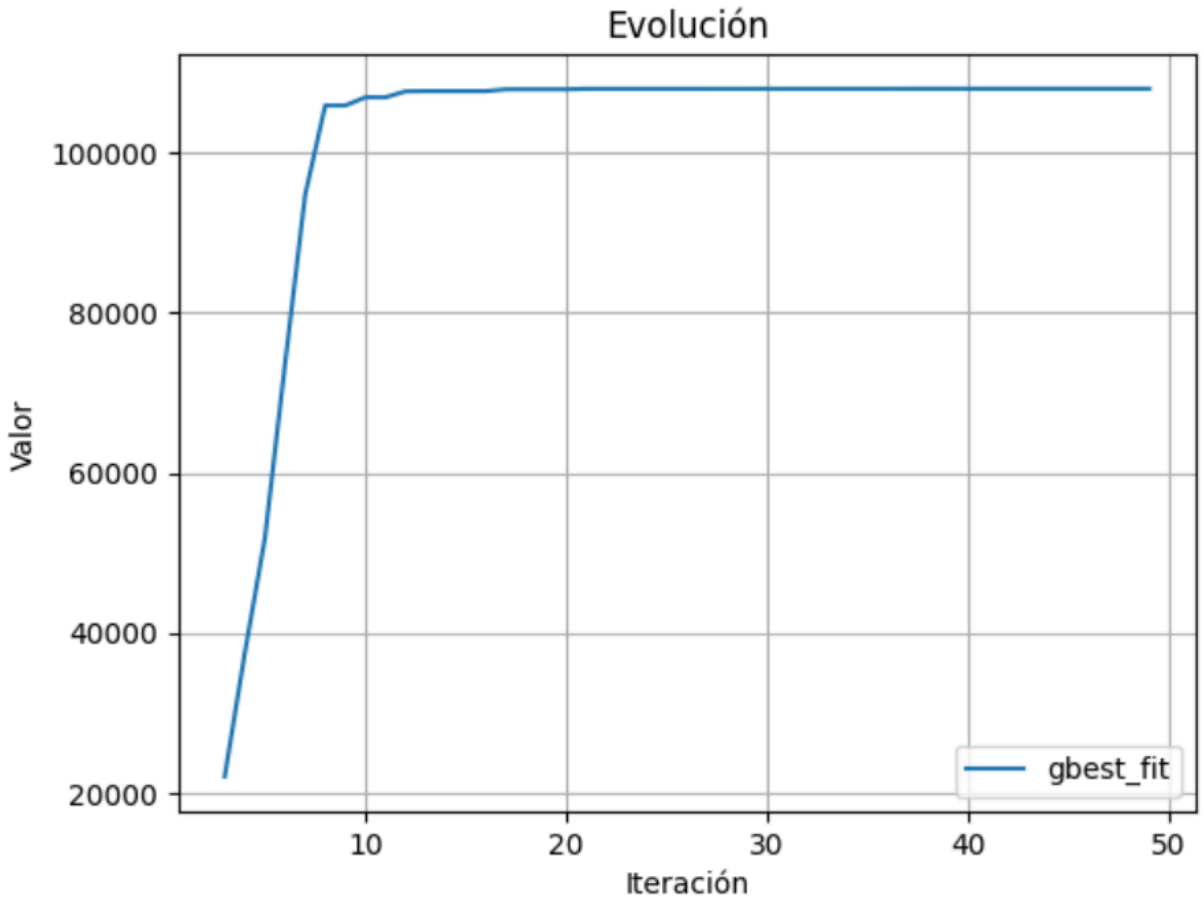
Mejor solución: [31.6545, 95.3047, 73.0835, 108.4635].
Valor óptimo: 108044.5315962174.

## Punto c:

Url al repositorio.

## Punto d:

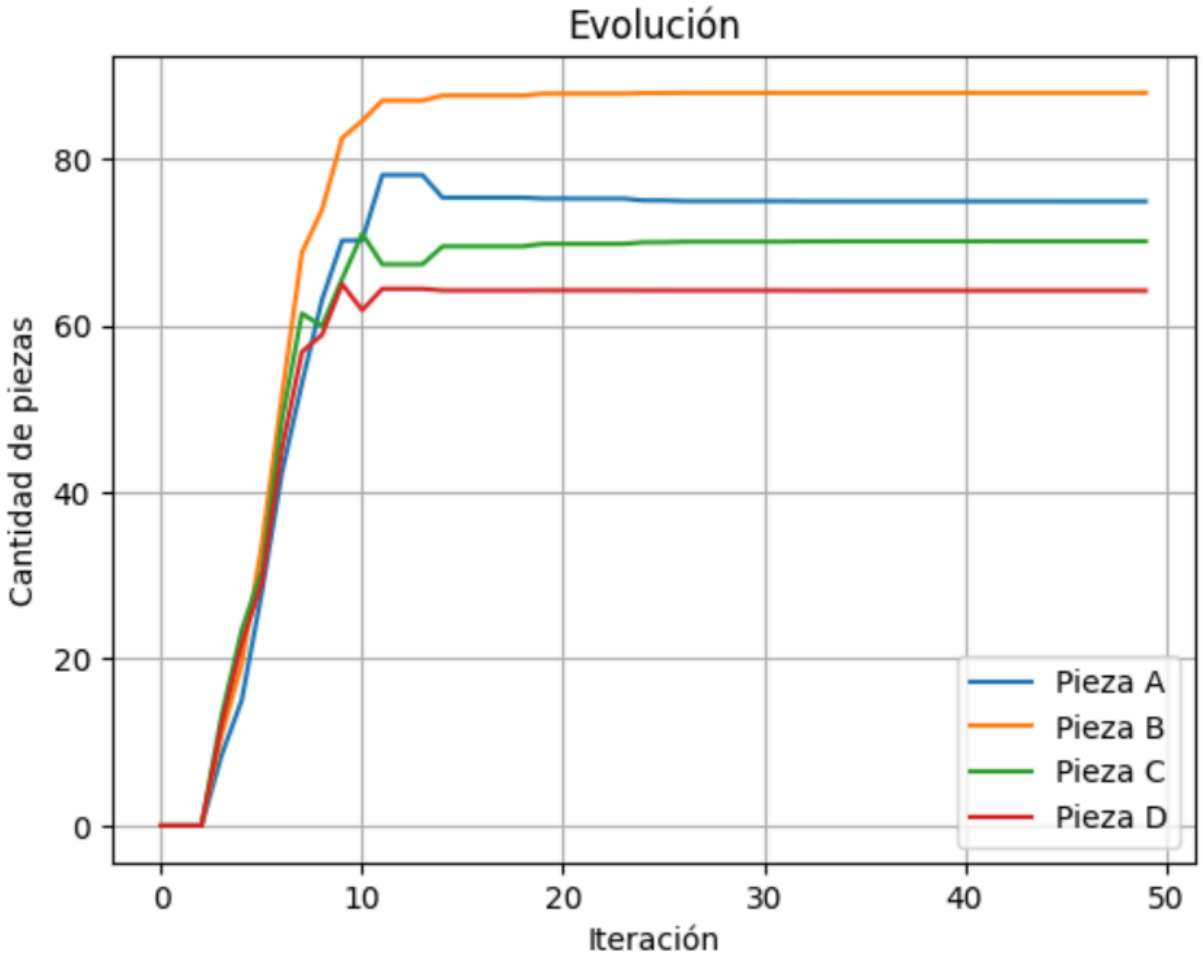

Evolución
Cantidad de piezas
Iteración
0
20
40
60
80
100
0
10
20
30
40
50
Pieza A
Pieza B
Pieza C
Pieza D

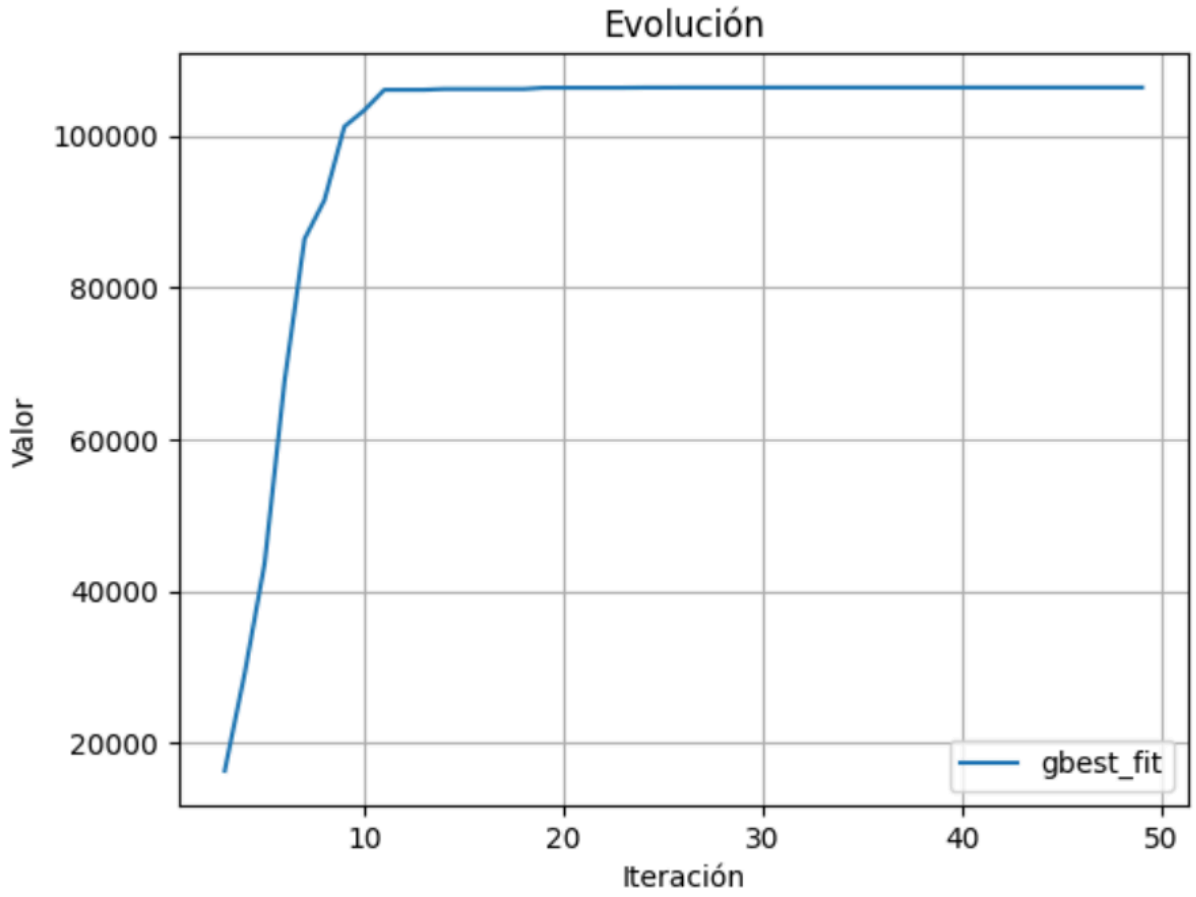



## Punto e:

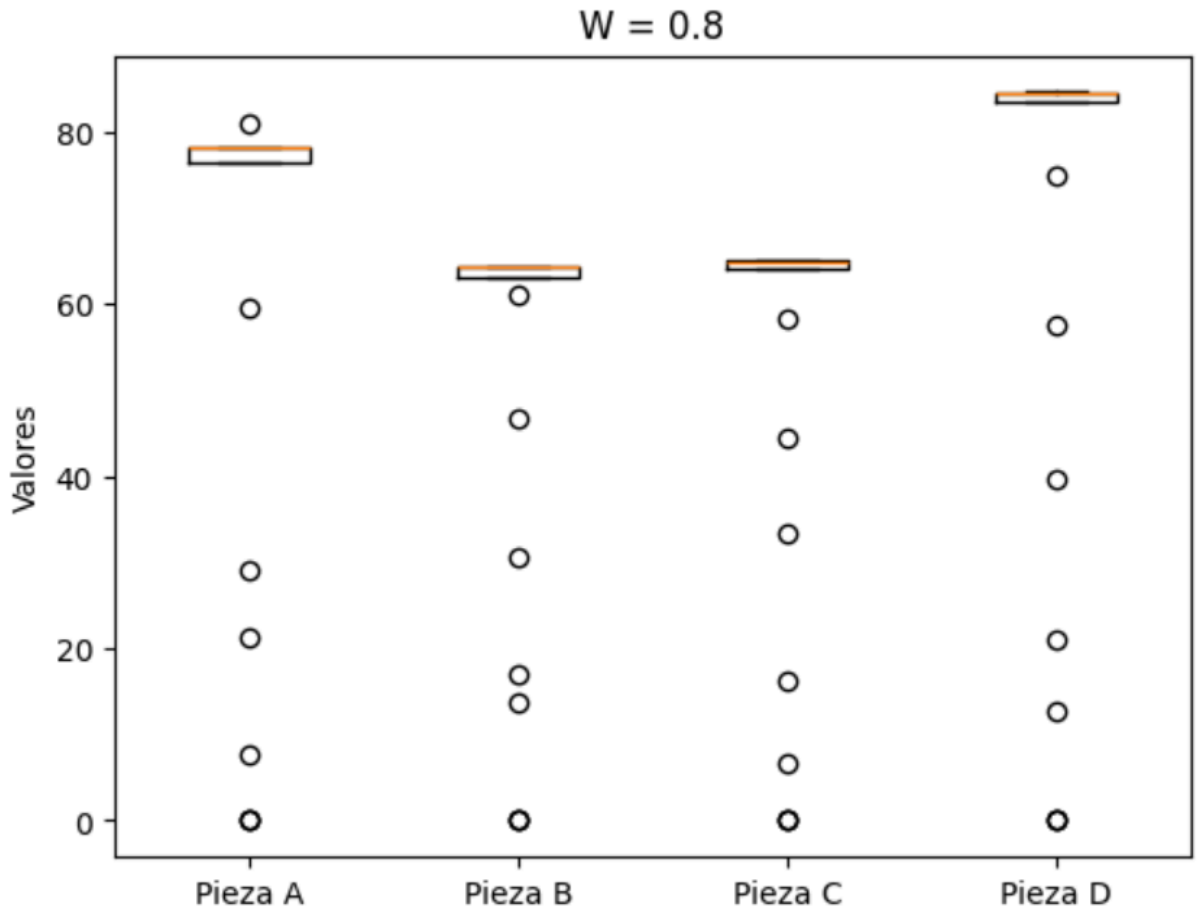
- Si se reduce en una unidad el tiempo de acabado de la pieza B, significa que la producción de piezas B se vuelve más eficiente. Como resultado, el algoritmo de optimización debería priorizar esta pieza, asignándole mayor importancia.

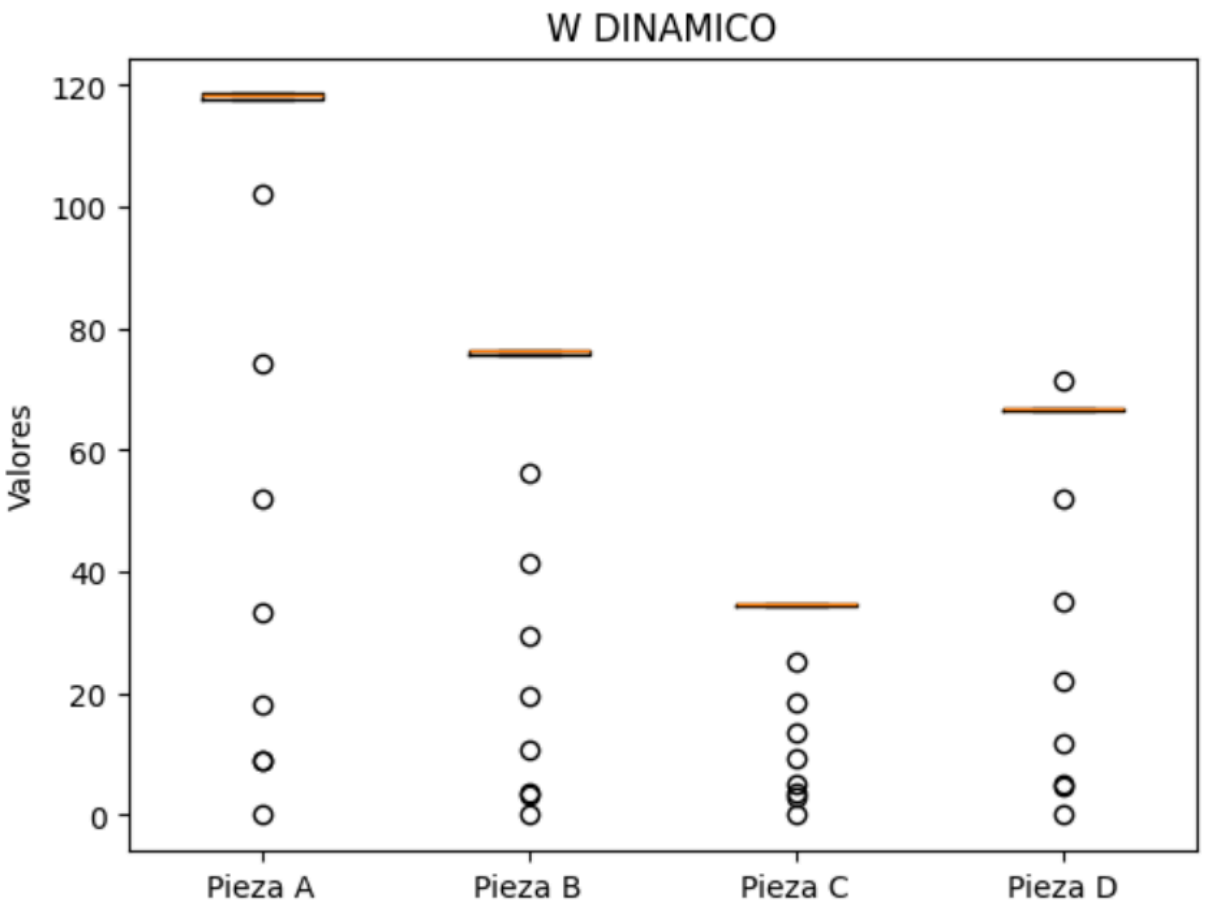
Mejor solución: [74.8678, 87.8772, 70.0685, 64.1633].
Valor óptimo: 106377.21982405565.

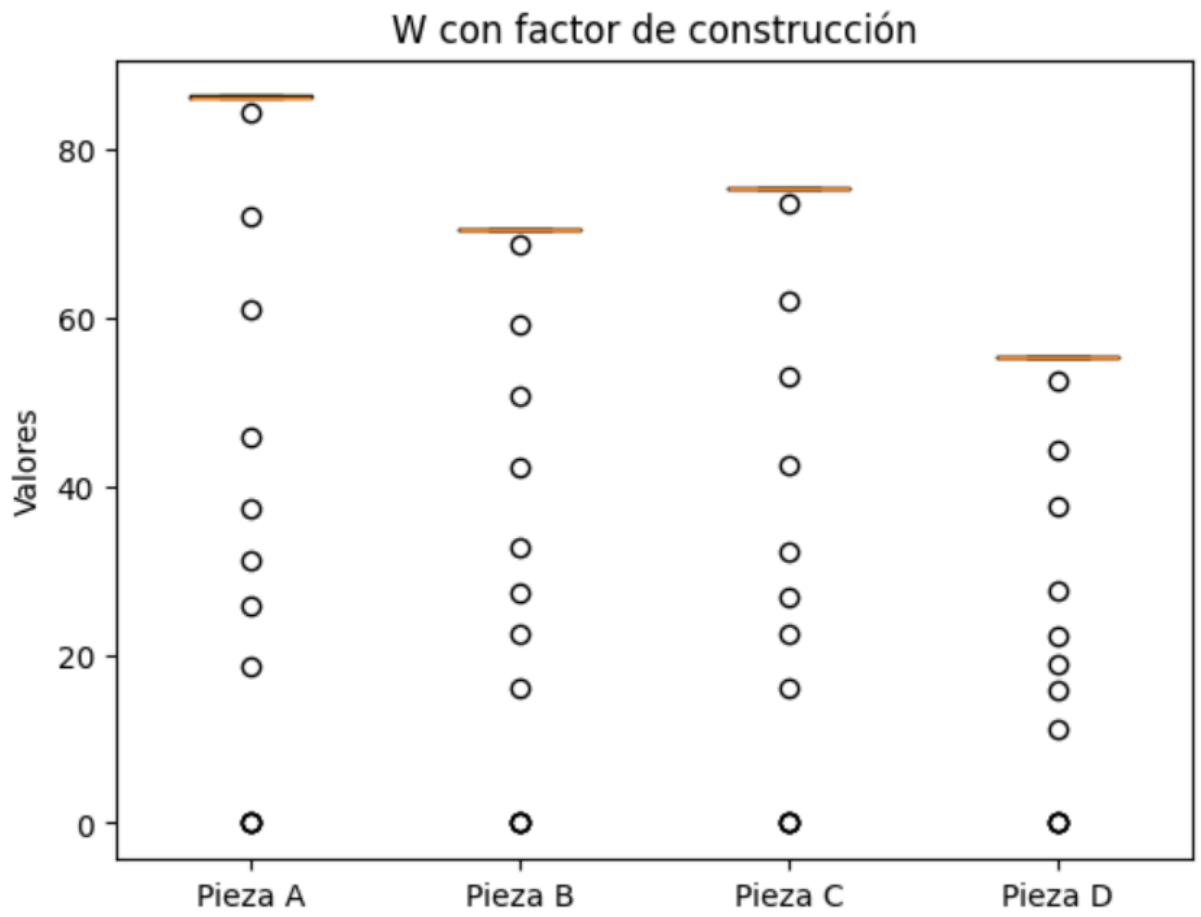

Evolución
Cantidad de piezas
Iteración
0
20
40
60
80
0
10
20
30
40
50
Pieza A
Pieza B
Pieza C
Pieza D

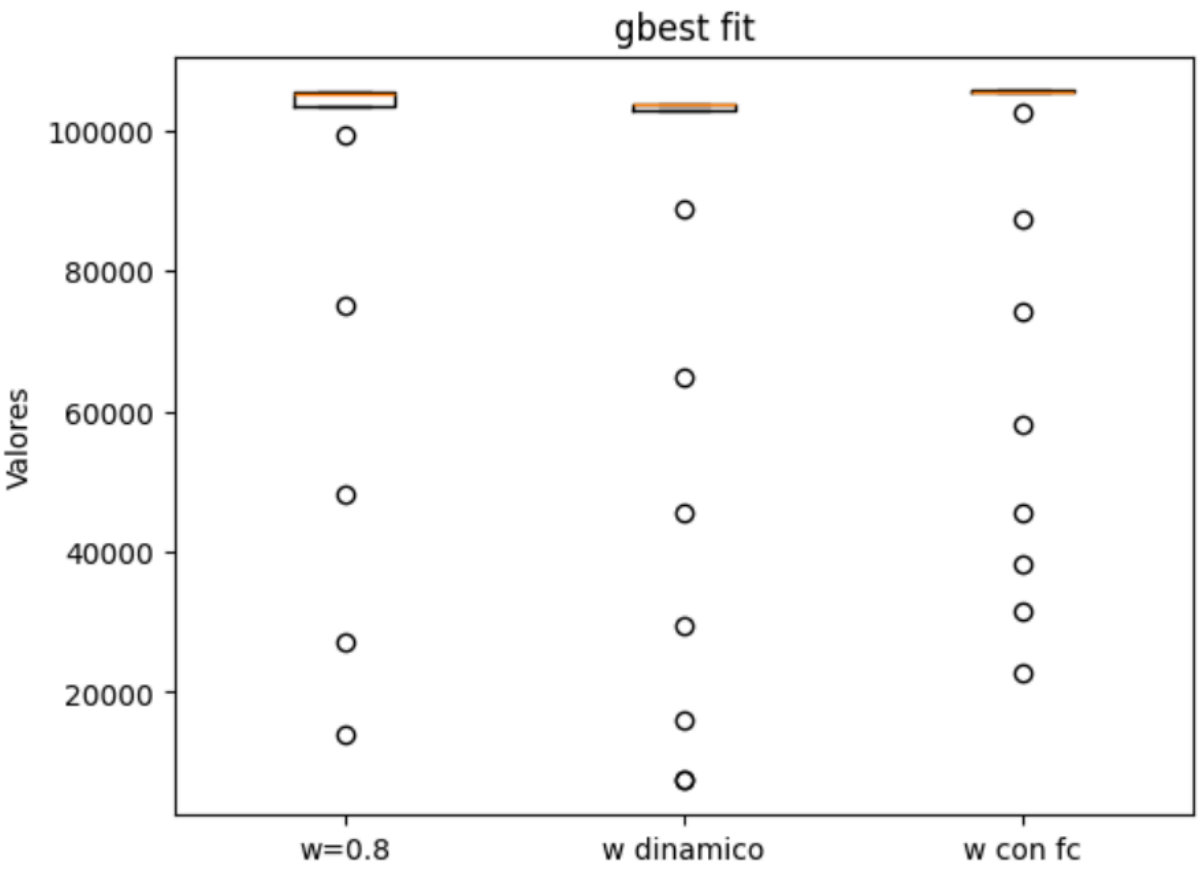

Evolución
100000
80000
60000
40000
20000
Valor
10
20
30
40
50
Iteración
gbest_fit

Punto f:


W = 0.8
Valores
0
20
40
60
80
Pieza A
Pieza B
Pieza C
Pieza D


W DINAMICO
Valores
120
100
80
60
40
20
0
Pieza A
Pieza B
Pieza C
Pieza D


W con factor de construcción
Valores
0
20
40
60
80
Pieza A
Pieza B
Pieza C
Pieza D


gbest fit
Valores
20000
40000
60000
80000
100000
w=0.8
w dinamico
w con fc

# Punto g:

**Configuración con w=0.8 (Inercia constante)**

- La diferencia entre las piezas no es muy marcada, lo que indica que esta configuración tiende a un comportamiento más equilibrado entre todas las piezas.
- Aunque la pieza D tiene una ligera ventaja en términos de las ganancias óptimas, las demás piezas (A, B, C) no están muy lejos, lo que sugiere que con esta inercia constante, el algoritmo optimiza todas las piezas de manera relativamente equitativa.

**Configuración con w Dinámico (Inercia variable)**

- En esta variante, la pieza A obtiene claramente el mejor rendimiento, superando a las demás con una diferencia considerable. Esto sugiere que un factor de inercia dinámico podría estar favoreciendo un comportamiento en el que ciertas soluciones se ven beneficiadas de manera desproporcionada, mientras que otras piezas (B, C, D) quedan más atrás.
- El hecho de que la pieza B sea la segunda mejor sugiere que el algoritmo puede estar enfocándose en mejorar unas pocas soluciones (como A y B) en detrimento de las otras, lo que podría resultar en una convergencia más rápida pero no tan equilibrada como en el caso de la inercia constante.

**Configuración sin w, pero con Factor de Construcción**

- Esta configuración muestra que la pieza A sigue siendo la más favorecida, pero ahora la pieza C mejora significativamente, colocándose en segundo lugar, seguida por B y, finalmente, D.

- El uso del factor de construcción sin inercia parece permitir una mayor flexibilidad en el movimiento de las partículas, lo que lleva a un mayor valor óptimo general. Esto podría indicar que el factor de construcción impulsa a las partículas a explorar más agresivamente el espacio de búsqueda, lo que da lugar a mejores soluciones para algunas piezas, pero también puede resultar en una mayor dispersión.

## Punto h:

- El número de partículas como mínimo tiene que ser igual al número de dimensiones. Aunque se suele usar más que esta cantidad para asegurar una buena exploración del espacio.
- Si hay muy pocas partículas, es posible que colisionan o se agrupen en ciertas áreas, lo que puede llevar a una convergencia prematura y evitar que el algoritmo explore otras áreas.
- Realiza pruebas con diferentes cantidades de partículas: Comenzar con un número bajo (por ejemplo, 5, 10, 20) y aumenta progresivamente. Observar cómo afecta.
- Verificar cuántas iteraciones tarda cada configuración en converger a una solución óptima. Si un número menor de partículas produce soluciones de baja calidad o tarda demasiado en converger, es posible que se necesite aumentar el número.
- Considerar el tiempo de cálculo y los recursos necesarios. Si el número de partículas es demasiado alto, el tiempo de ejecución puede ser muy costoso.

# Ejercicio 2:

## Punto a:

```
import numpy as np
import matplotlib.pyplot as plt

# función objetivo a maximizar
def f(x):
    return 500 * x[0] + 400 * x[1] # funcion objetivo:

# primera restriccion
def g1(x):
    return 300 * x[0] + 400 * x[1] <= 127000   # restricción de $127000
de capital por día.

# segunda restriccion
def g2(x):
    return 20 * x[0] + 10 * x[1] <= 4270   # restriccion: de 4270 horas
de mano de obra por día.

# parametros
n_particles = 10  # numero de particulas en el enjambre
n_dimensions = 2  # dimensiones del espacio de busqueda (x1, x2, x3 y
x4)
max_iterations = 80  # numero máximo de iteraciones para la
optimizacion
c1 = c2 = 2  # coeficientes de aceleracion

w = 0.5 # Factor de inercia
w_dinamico = False # True

factor_contruccion = False #True
phi = 3
chi = 2 / abs(2 - phi - np.sqrt(phi ** 2 - 4 * phi))

# inicialización de particulas
x = np.zeros((n_particles, n_dimensions))  # matriz para las posiciones
de las particulas
```

```
v = np.zeros((n_particles, n_dimensions))  # matriz para las
velocidades de las particulas
pbest = np.zeros((n_particles, n_dimensions))  # matriz para los
mejores valores personales
pbest_fit = -np.inf * np.ones(n_particles)  # vector para las mejores
aptitudes personales (inicialmente -infinito)
gbest = np.zeros(n_dimensions)  # mejor solución global
gbest_fit = -np.inf  # mejor aptitud global (inicialmente -infinito)


# inicializacion de particulas factibles
for i in range(n_particles):
    while True:  # bucle para asegurar que la particula sea factible
        x[i] = np.random.uniform(0, 10, n_dimensions)  # inicializacion
posicion aleatoria en el rango [0, 10]
        if g1(x[i]) and g2(x[i]):  # se comprueba si la posicion cumple
las restricciones
            break  # Salir del bucle si es factible
    v[i] = np.random.uniform(-1, 1, n_dimensions)  # inicializar
velocidad aleatoria
    pbest[i] = x[i].copy()  # ee establece el mejor valor personal
inicial como la posicion actual
    fit = f(x[i])  # calculo la aptitud de la posicion inicial
    if fit > pbest_fit[i]:  # si la aptitud es mejor que la mejor
conocida
        pbest_fit[i] = fit  # se actualiza el mejor valor personal


# Optimizacion

#lista para guardar los gbest
gbest_list = []
gbest_fit_list = []

for i in range(max_iterations):  # Repetir hasta el número máximo de
iteraciones

    # w_dinamico
    if w_dinamico:
        w_max = 0.9
        w_min = 0.4
        t_max = max_iterations
        w = w_max - ((w_max - w_min) / t_max) * (i + 1)
```

```
        #print("w_dinamico:" , w)

    for i in range(n_particles):
        fit = f(x[i])  # Se calcula la aptitud de la posicion actual
        # Se comprueba si la nueva aptitud es mejor y si cumple las
restricciones
        if fit > pbest_fit[i] and g1(x[i]) and g2(x[i]):
            pbest_fit[i] = fit  # Se actualiza la mejor aptitud
personal
            pbest[i] = x[i].copy()  # Se actualizar la mejor posicion
personal
            if fit > gbest_fit:  # Si la nueva aptitud es mejor que la
mejor global
                gbest_fit = fit  # Se actualizar la mejor aptitud
global
                gbest = x[i].copy()  # Se actualizar la mejor posicion
global

        if factor_contruccion:
            # actualizacion de la velocidad de la particula
            v[i] = chi * (v[i] + c1 * np.random.rand() * (pbest[i] -
x[i]) + c2 * np.random.rand() * (gbest - x[i]))
            x[i] += v[i]  # Se actualiza la posicion de la particula
        else:
            # actualizacion de la velocidad de la particula
            v[i] = w * v[i] + c1 * np.random.rand() * (pbest[i] - x[i])
+ c2 * np.random.rand() * (gbest - x[i])
            x[i] += v[i]  # Se actualiza la posicion de la particula

        # se asegura de que la nueva posicion esté dentro de las
restricciones
        if not (g1(x[i]) and g2(x[i])):
            # Si la nueva posicion no es válida, revertir a la mejor
posicion personal
            x[i] = pbest[i].copy()

    gbest_list.append(gbest)
    gbest_fit_list.append(gbest_fit)


# Se imprime la mejor solucion encontrada y también su valor optimo
print(f"Mejor solucion: [{gbest[0]:.4f}, {gbest[1]:.4f}]")
print(f"Valor optimo: {gbest_fit}")
```

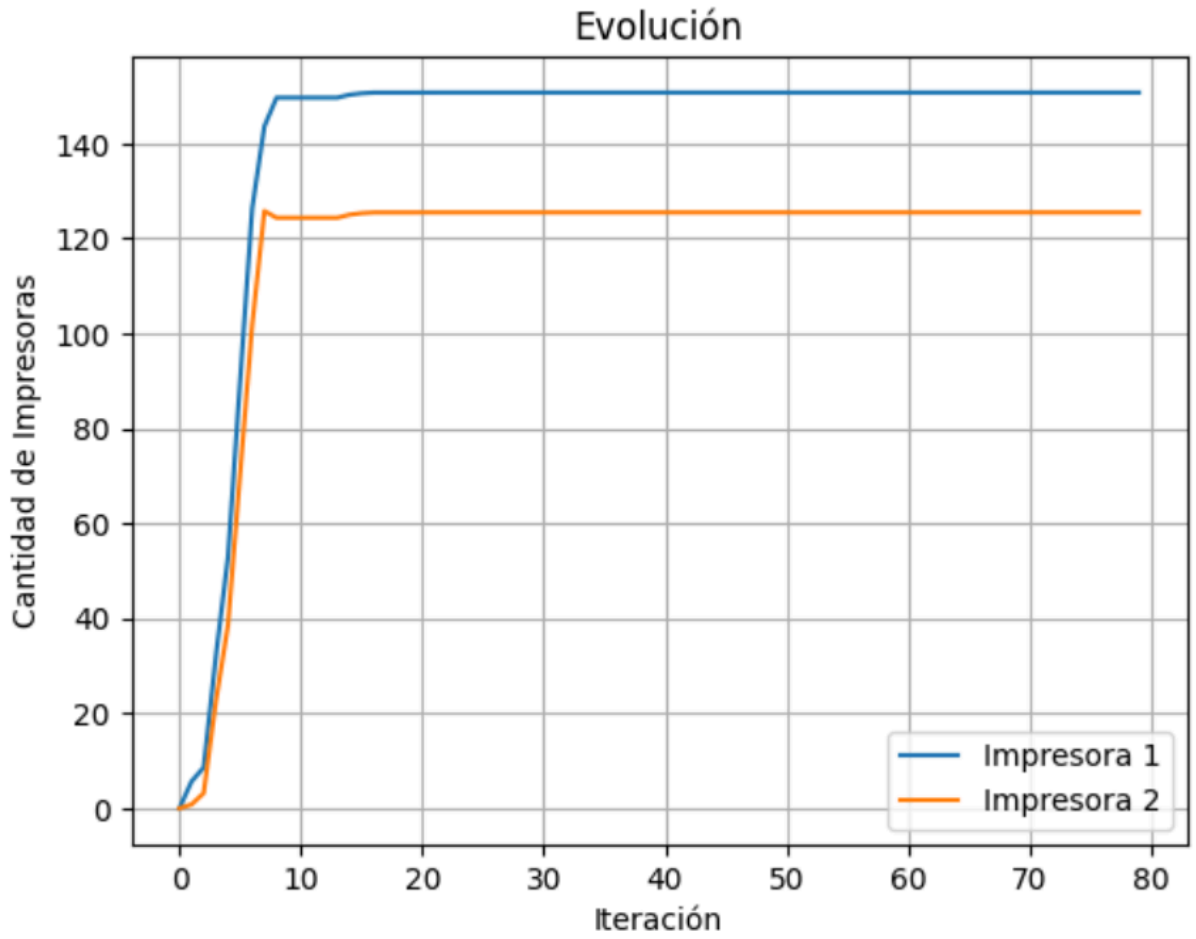
## Punto b:

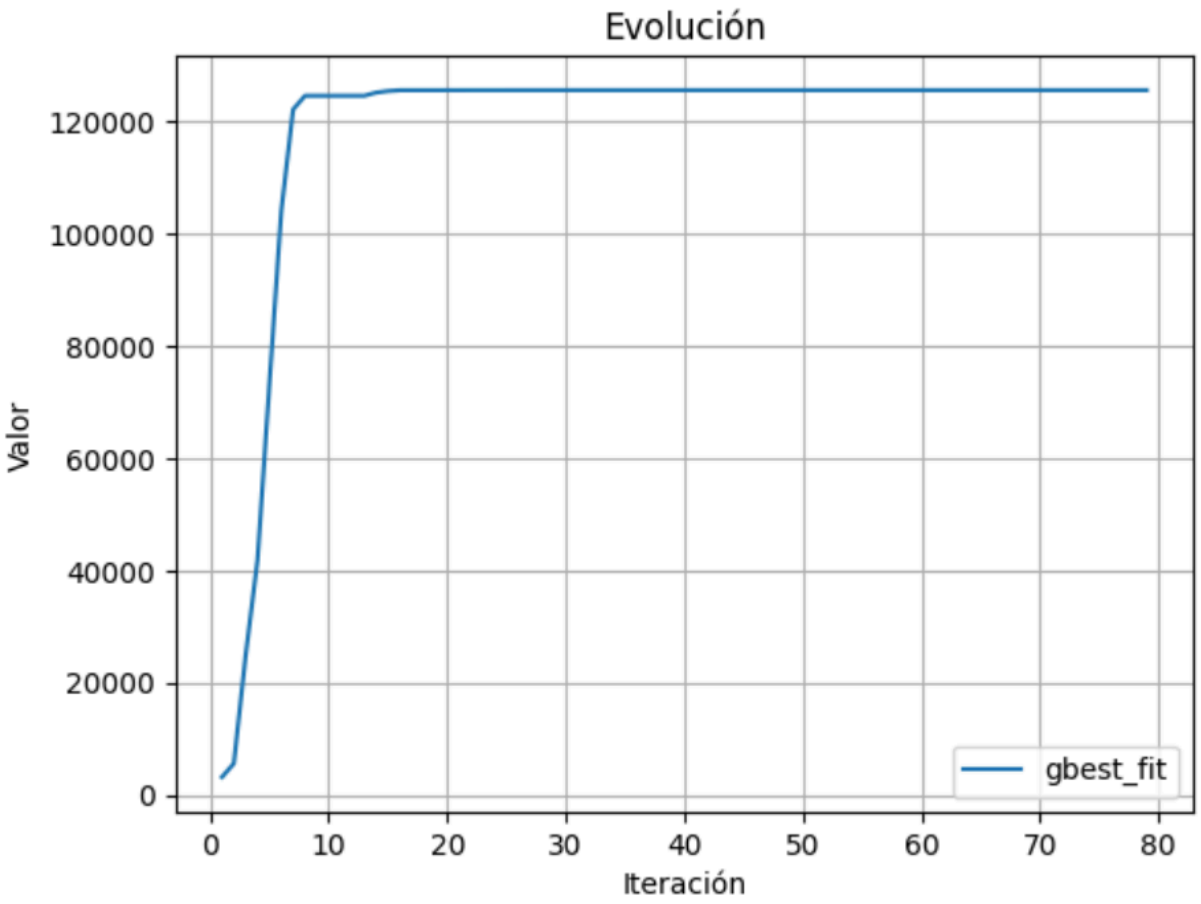
Mejor solución: [150.7470, 125.5060].
Valor óptimo: 125575.89914417824.

## Punto c:

Url al repositorio.

## Punto d:

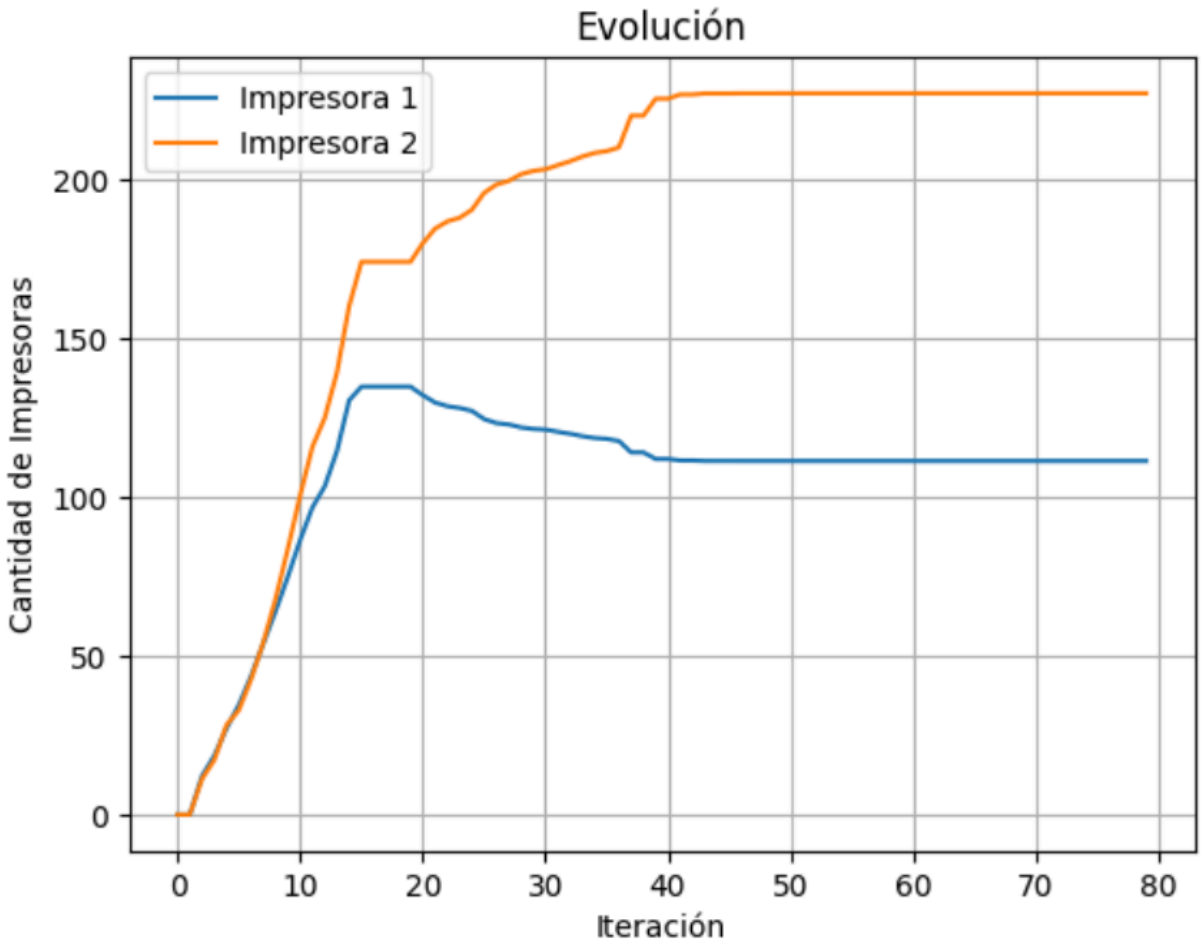


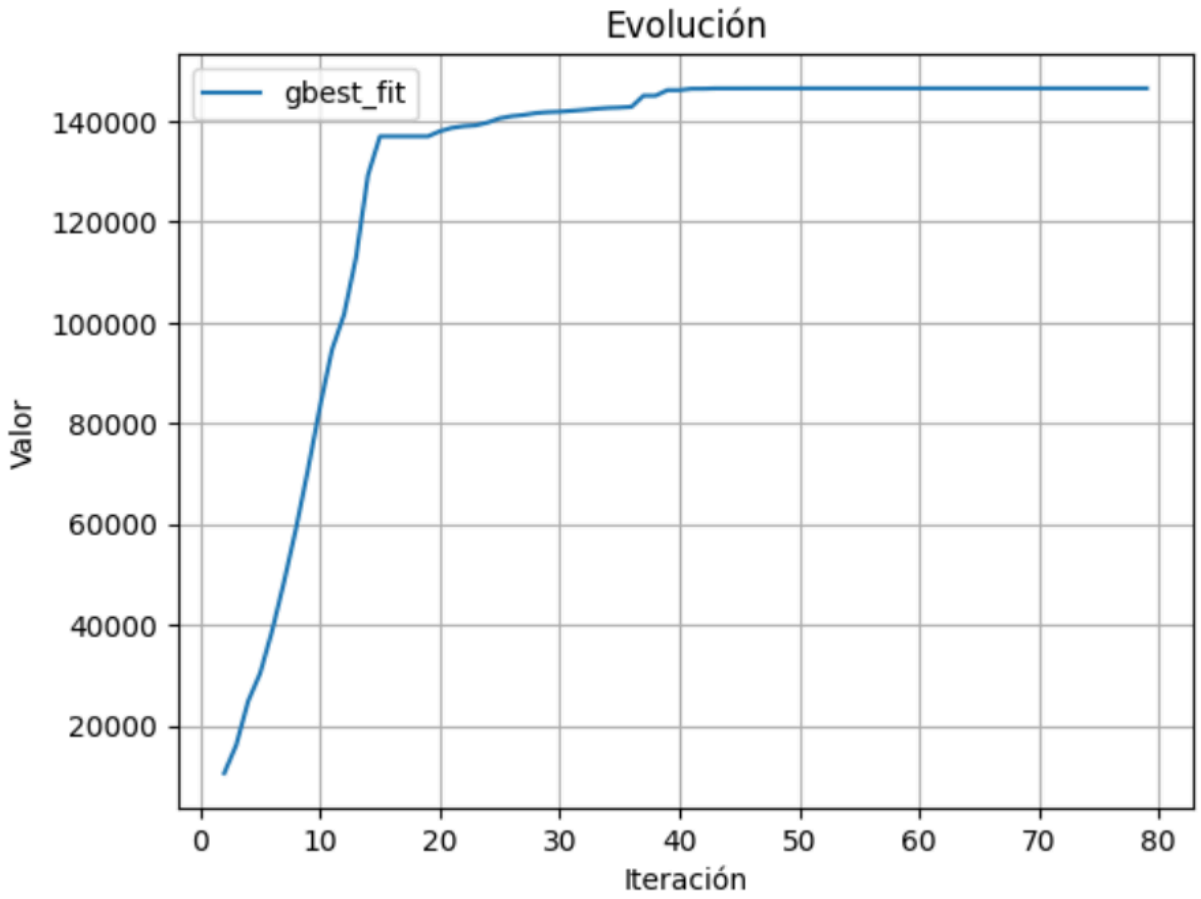



## Punto e:

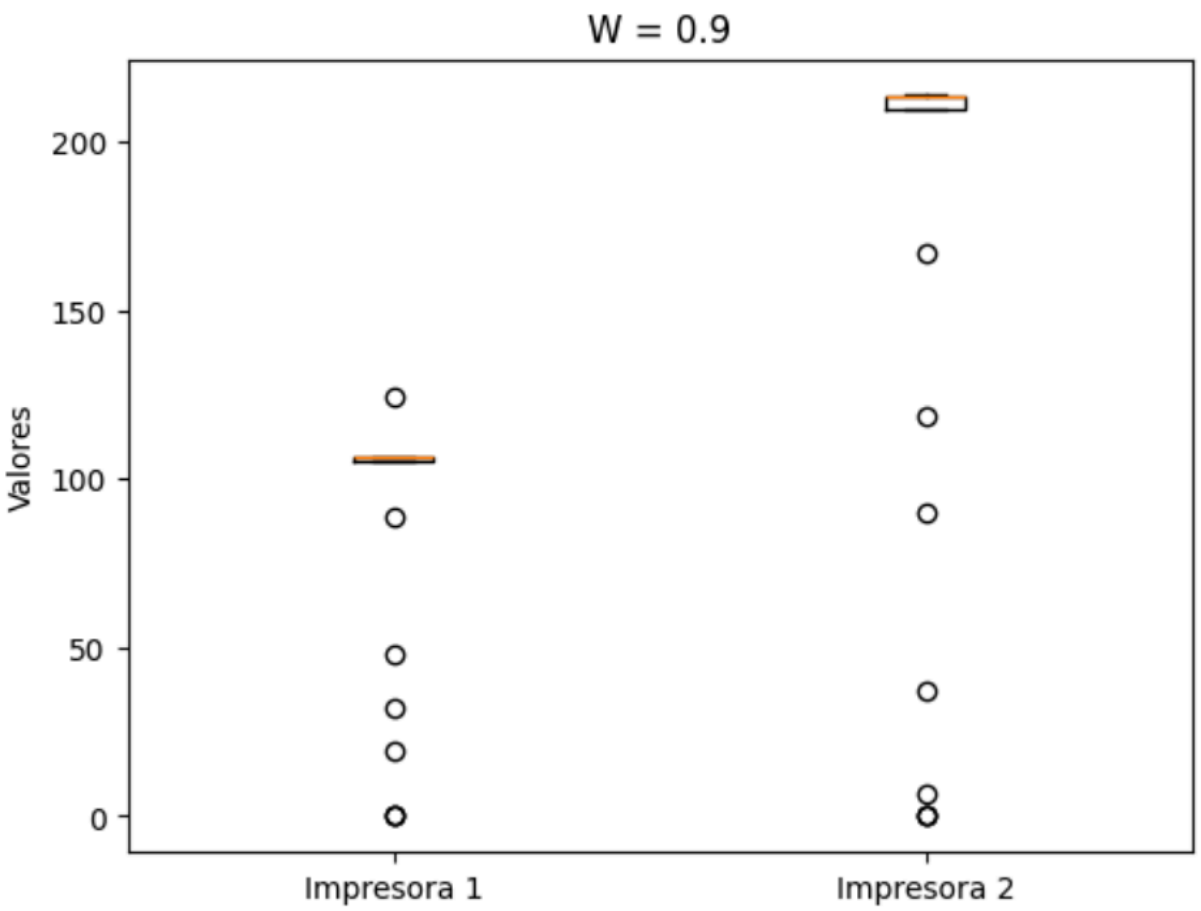
- Si se reduce en una unidad el tiempo de la impresora 2, significa que la producción de impresoras 2 se vuelve más eficiente. Como resultado, el algoritmo de optimización debería priorizar esta pieza, asignándole mayor importancia.

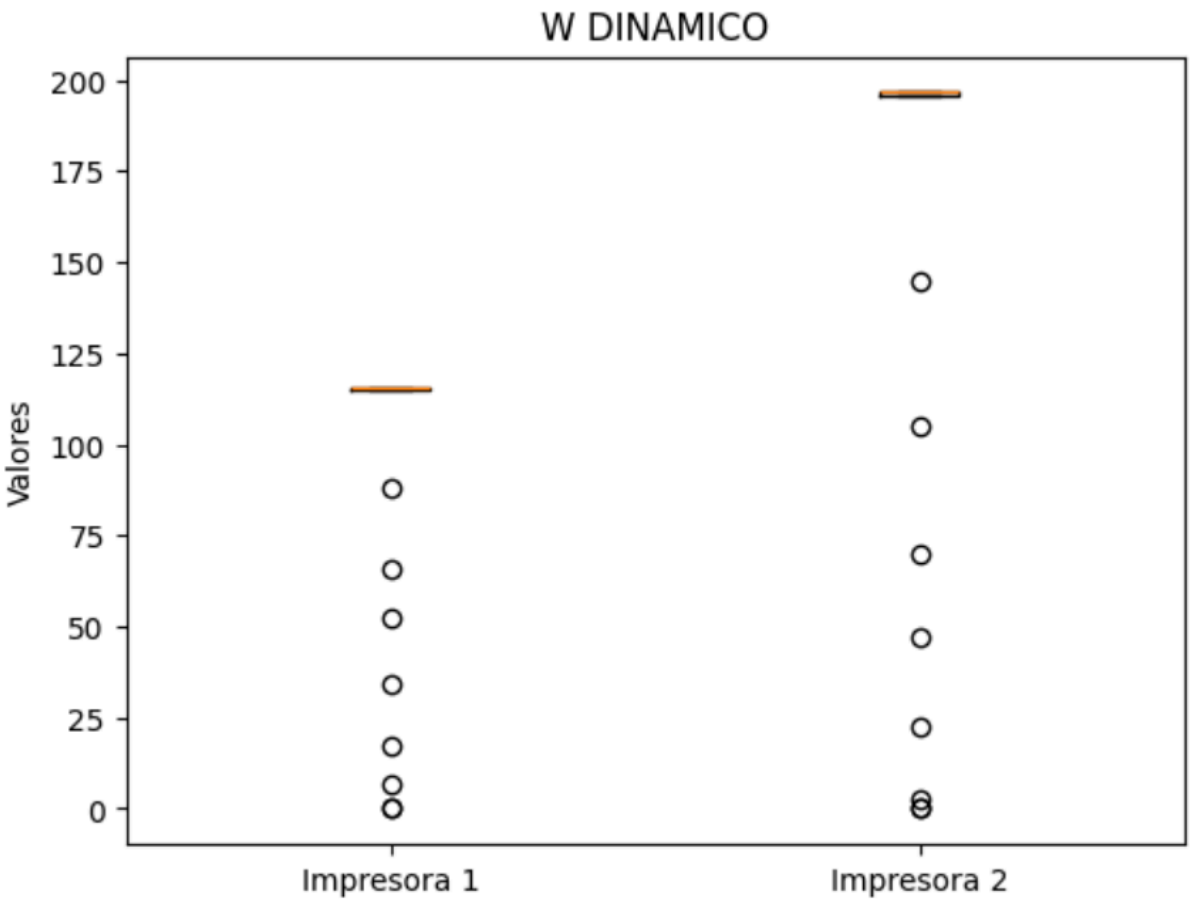
Mejor solución: [111.3115, 227.0855].
Valor óptimo: 146489.96614917566.

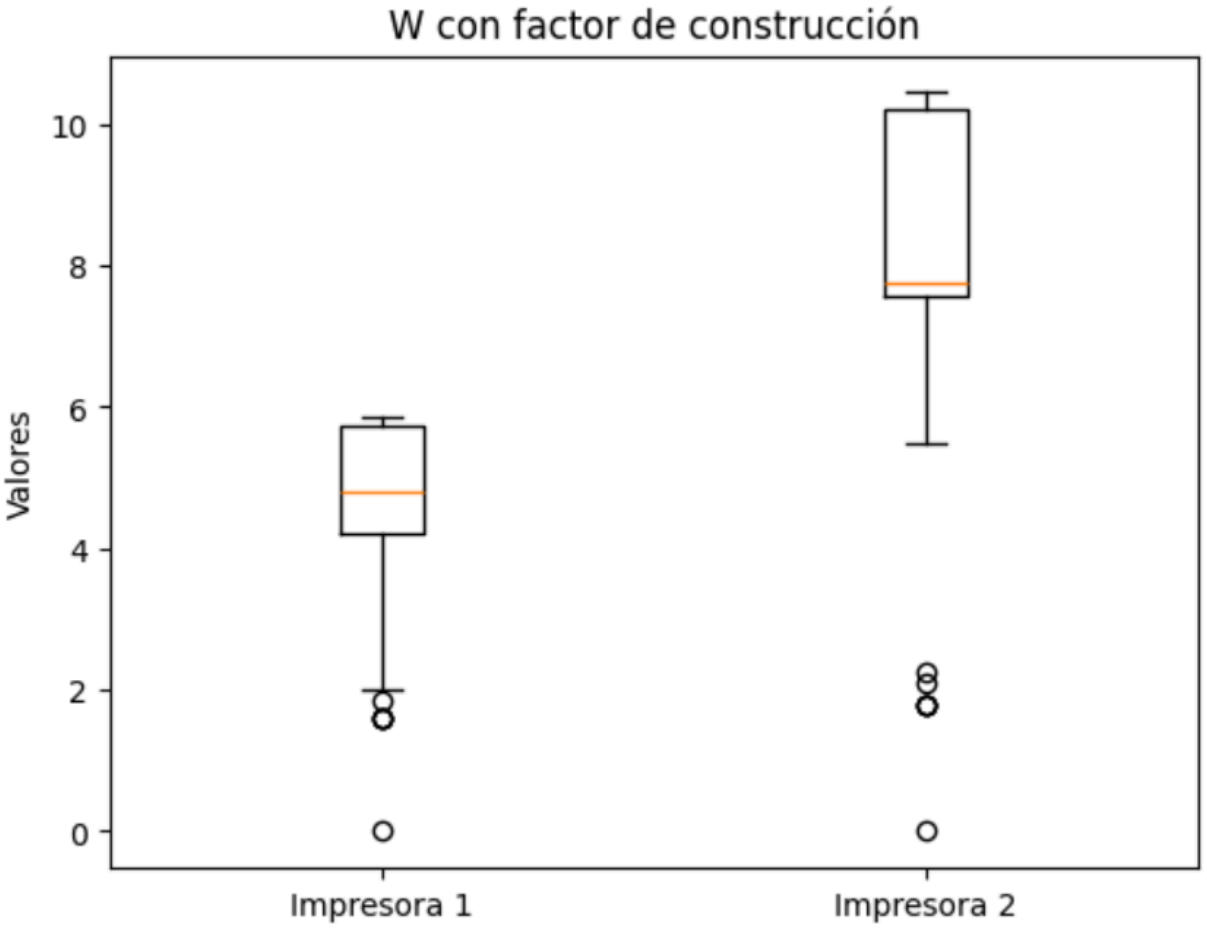

Evolución
Impresora 1
Impresora 2
Cantidad de Impresoras
Iteración
0
50
100
150
200
0
10
20
30
40
50
60
70
80

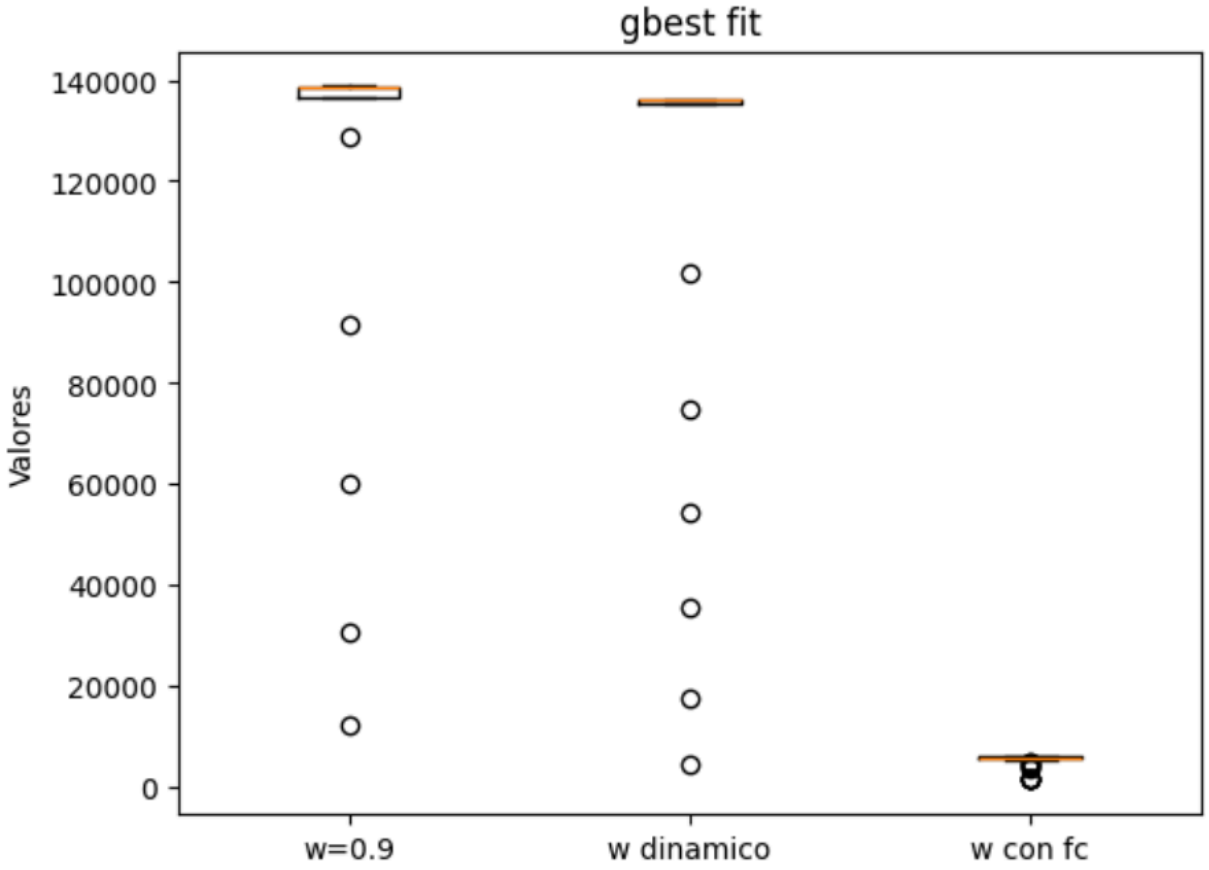

Evolución
gbest_fit
140000
120000
100000
80000
60000
40000
20000
Valor
0
10
20
30
40
50
60
70
80
Iteración

Punto f:


W = 0.9
Valores
200
150
100
50
0
Impresora 1
Impresora 2


W DINAMICO
Valores
200
175
150
125
100
75
50
25
0
Impresora 1
Impresora 2


W con factor de construcción
Valores
10
8
6
4
2
0
Impresora 1
Impresora 2


gbest fit
Valores
140000
120000
100000
80000
60000
40000
20000
0
w=0.9
w dinamico
w con fc

## Punto g:

**Primer gráfico (con w=0.9):**

- **Impresora 1**: Boxplot más bajo, con valores en torno a 100.
- **Impresora 2**: Boxplot más alto, alrededor de 200, y más ancho (mayor variabilidad en los resultados).
- **Conclusión**: La impresora 2 tiene un mejor rendimiento en términos de utilidad en comparación con la impresora 1 cuando se utiliza un factor de inercia constante (w=0.9). Además, la impresora 2 presenta una mayor variabilidad en sus resultados, lo que puede indicar que responde mejor a cambios en el enjambre. Esto sugiere que, para este valor de inercia constante, el algoritmo optimiza mejor la función de utilidad para la impresora 2.

**Segundo gráfico (con w dinámico):**

- **Impresora 1**: Boxplot más bajo, con valores cercanos a 120.
- **Impresora 2**: Boxplot más alto, alrededor de 200.
- **Conclusión**: Aunque ambos boxplots están más altos en comparación con el gráfico anterior, el rendimiento de la impresora 1 ha mejorado ligeramente con el uso de un factor de inercia dinámico (de 100 a 120). Sin embargo, la impresora 2 sigue mostrando un rendimiento significativamente mejor. El uso de w dinámico parece haber mejorado la optimización, pero no ha alterado la diferencia en el rendimiento relativo entre las dos impresoras.

**Tercer gráfico (sin w pero con factor de construcción):**

- **Impresora 1**: Boxplot con una mediana de aproximadamente 4.5.

- **Impresora 2**: Boxplot con una mediana de aproximadamente 7.8.
- **Conclusión**: Cuando se utiliza el factor de construcción en lugar del factor de inercia, ambos boxplots se acercan, y la diferencia en el rendimiento entre las dos impresoras disminuye. Esto sugiere que el factor de construcción puede ayudar a reducir la disparidad en la optimización entre las impresoras. Cuando el valor de phi es alto las partículas se mueven demasiado rápido sin dirigirse de manera óptima al máximo global. No hay un buen equilibrio entre exploración y explotación.

**Cuarto gráfico (comparación general entre w=0.9,w dinámico y sin w con factor de construcción):**

- **w=0.9**: Boxplot más alto y más grueso.
- **w dinámico**: Boxplot en segundo lugar, más bajo que w=0.9, pero más alto que el escenario sin w.
- **Sin w pero con factor de construcción**: Boxplot más bajo.
- **Conclusión**: El uso de un factor de inercia constante (w=0.9) proporciona los mejores resultados en términos de utilidad global, ya que el boxplot es el más alto y más grueso, lo que indica no sólo un mayor rendimiento, sino también una mayor variabilidad (posible mayor exploración). El uso de un factor de inercia dinámico es menos efectivo en términos de rendimiento máximo, aunque mejora en comparación con no usar inercia y utilizar el factor de construcción. El factor de construcción proporciona los resultados menos favorables.

## Punto h:

Las conclusiones son similares a las del punto h del ejercicio 1.