Práctica 10 algoritmo genético

Denisse Leyva

Mayo 12, 2021

1. Introducción

El problema de la mochila (inglés: knapsack) es un problema clásico de optimización, particularmente de programación entera, donde la tarea consiste en seleccionar un subconjunto de objetos de tal forma que (i) no se exceda la capacidad de la mochila en términos de la suma de los pesos de los objetos incluidos, y que (ii) el valor total de los objetos incluidos sea lo máximo posible. Este problema es pseudo-polinomial ya que existe un algoritmo de tabulación que determina la combinación óptima.

Aunque el algoritmo pseudo-polinomial sirve solamente para pesos enteros, nos servirá para esta décima práctica, donde probamos la implementación de un algoritmo genético en R de manera paralela. Los algoritmos genéticos se suelen utilizar en casos donde no existe ningún algoritmo exacto eficiente, pero para fines de aprendizaje, nos conviene comparar qué tan cerca a la solución óptima (que nos da el algoritmo pseudo-polinomial) logramos llegar con un algoritmo genético [2].

2. Objetivo

Cambia la selección de mutación y de los padres para reproducción a que use seleccion de ruleta: cada solución se selecciona como padre con una probabilidad que es linealmente proporcional a su valor de función objetivo y a su factibilidad, combinando los dos a alguna función que parezca conveniente e inversamente proporcional a alguna combinación de factibilidad y objectivo para la mutación (recomiendo aprovechar el parámetro prob en sample). Genere instancias con tres diferentes reglas:

- el peso y el valor de cada objeto se generan independientemente con una distribución normal.
- el valor de cada objeto se generan independientemente con una distribución exponencial y su peso es inversamente correlacionado con el valor, con un ruido normalmente distribuido de baja magnitud.
- el peso de cada objeto se generan independientemente con una distribución normal y su valor es (positivamente) correlacionado con el cuadrado del peso, con un ruido normalmente distribuido de baja magnitud.

Determina para cada uno de los tres casos a partir de qué tamaño de instancia el algoritmo genético es competitico con el algoritmo exacto en términos de valor total obtenido por segundo de ejecución y si la inclusión de la selección de ruleta produce una mejora estadísticamente significativa [2].

3. Código

El código base se sacó de Schaeffer [3]. El código completo se encuentra en el GitHub [1].

```
 obj = [objetivo(p[i], valores) for i in range(tam)]
            fac = [factible(p[i], pesos, capacidad) for i in range(tam)]
            peso_p = [((fac[i] + 1)*obj[i]) for i in range(tam)]
            peso_m = [(1/((fac[i] + 1)*(obj[i]))) for i in range(tam)]
            km = int(tam*pm)
            pop = [i for i in range(tam)]
            mut = choices(population=pop, weights=peso_m, k=km)
            pad = choices(population=pop, weights=peso_p, k=rep)

            for i in (mut): # mutarse con probabilidad pm
                p = np.vstack([p, mutacion(p[i], n)])

            for i in range(0,rep,2):  # reproducciones
                hijos = reproduccion(p[pad[i]], p[pad[i+1]], n)
                p = np.vstack([p, hijos[0], hijos[1]])
```

Código 1: Método de ruleta para reproducción y para mutación.

```
def generador_pesos(cuantos, low, high):
    return np.round(normalizar(np.random.normal(size = cuantos)) * (high - low) + low)

def generador_valores(pesos, low, high):
    return np.round(normalizar(np.random.normal(size = pesos)) * (high - low) + low)
```

Código 2: Instancia 1.

```
def generador_pesos2(valores, low, high):
    cant = 1 / valores
    return np.round(((normalizar(cant))) * (high - low) + low)

def generador_valores2(pesos, low, high):
    cant = np.arange(0, pesos)
    return np.round(normalizar(expon.pdf(cant))  * (high - low) + low)
```

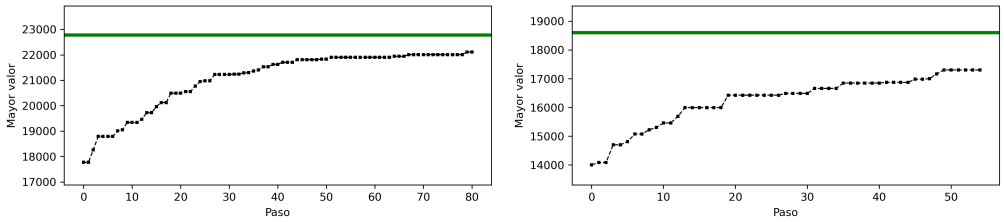
Código 3: Instancia 2.

```
def generador_pesos(cuantos, low, high):
    return np.round(normalizar(np.random.normal(size = cuantos)) * (high - low) + low)
def generador_valores3(pesos, low, high):
    return np.round((pesos**2) * (high - low) + low)
```

Código 4: Instancia 3.

4. Resultados

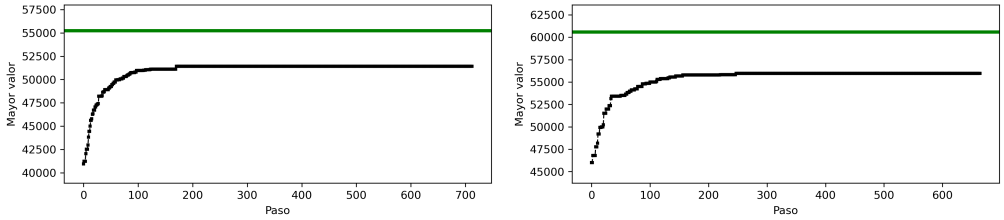
Como podemos observar en las siguientes gráficas el código genético es más eficiente en la segunda instancia ya que llega al valor optimo más rápido que el algoritmo knapsack.



(a) Gráfica de primera instancia sin ruleta a 100 objetivos.

(b) Gráfica de primera instancia con ruleta a 100 objetivos.

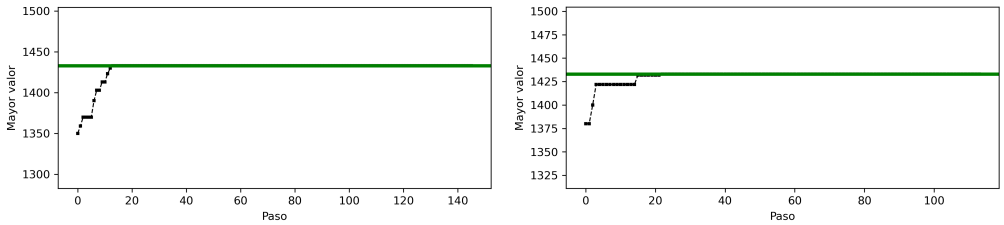
Figura 1: Gráficas comparando el mayor valor.



(a) Gráfica de primera instancia sin ruleta a 300 objetivos.

(b) Gráfica de primera instancia con ruleta a 300 objetivos.

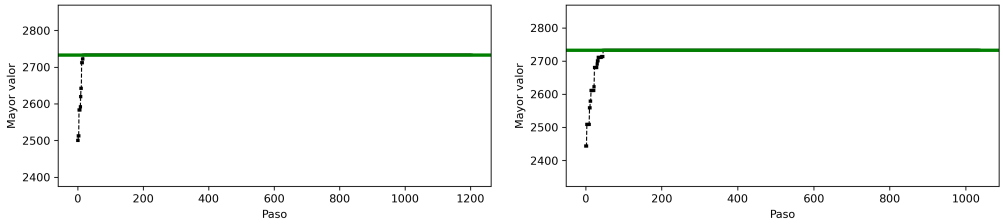
Figura 2: Gráficas comparando el mayor valor.



(a) Gráfica de segunda instancia sin ruleta a 100 objetivos.

(b) Gráfica de primera instancia con ruleta a 100 objetivos.

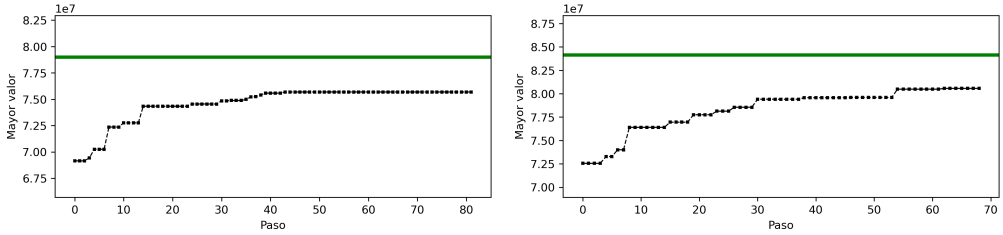
Figura 3: Gráficas comparando el mayor valor.



(a) Gráfica de segunda instancia sin ruleta a 300 objetivos.

(b) Gráfica de segunda instancia con ruleta a 300 objetivoss.

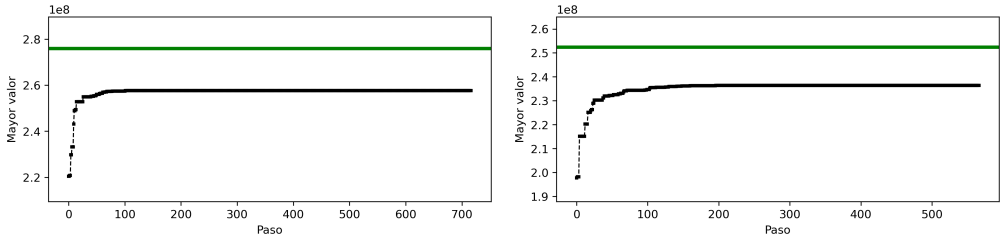
Figura 4: Gráficas comparando el mayor valor.



(a) Gráfica de tercer instancia sin ruleta a 100 objetivos.

(b) Gráfica de tercer instancia con ruleta a 100 objetivos.

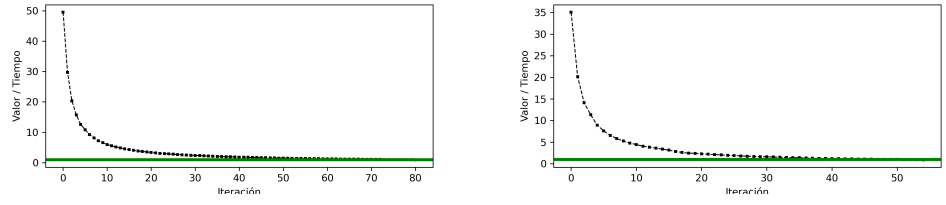
Figura 5: Gráficas comparando el mayor valor.



(a) Gráfica de tercer instancia sin ruleta a 300 objetivos.

(b) Gráfica de tercer instancia con ruleta a 300 objetivos.

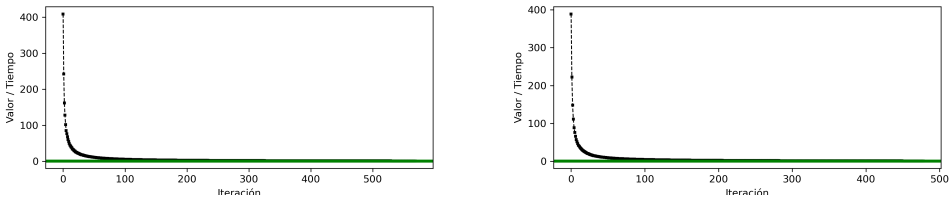
Figura 6: Gráficas comparando el mayor valor.



(a) Gráfica de primer instancia sin ruleta a 100 objetivos.

(b) Gráfica de primer instancia con ruleta a 100 objetivos.

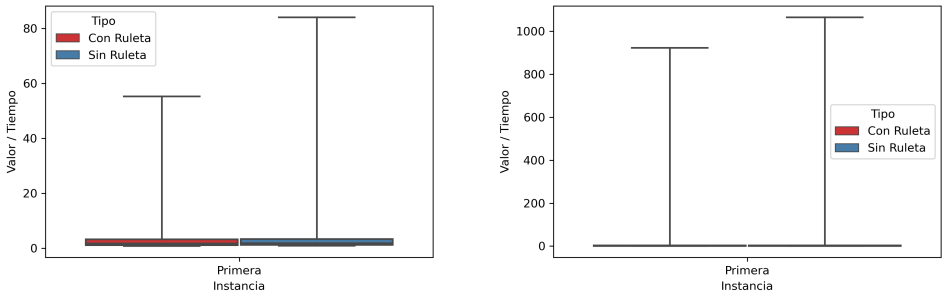
Figura 7: Gráfica comparando valor sobre tiempo.



(a) Gráfica de segunda instancia sin ruleta a 200 objetivos.

(b) Gráfica de segunda instancia con ruleta a 200 objetivos.

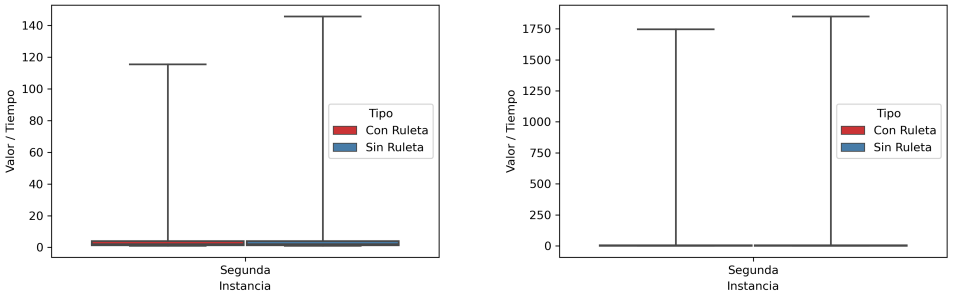
Figura 8: Gráfica comparando valor sobre tiempo.



(a) Caja bigote de primera instancia con 100 objetivos.

(b) Caja bigote de primera instancia con 400 objetivos.

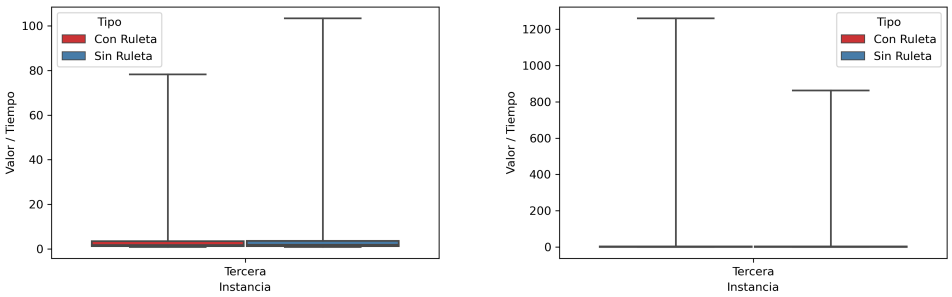
Figura 9: Caja bigote comparando valor sobre tiempo para cada instancia.



(a) Caja bigote de segunda instancia con 100 objetivos.

(b) Caja bigote de segunda instancia con 400 objetivos.

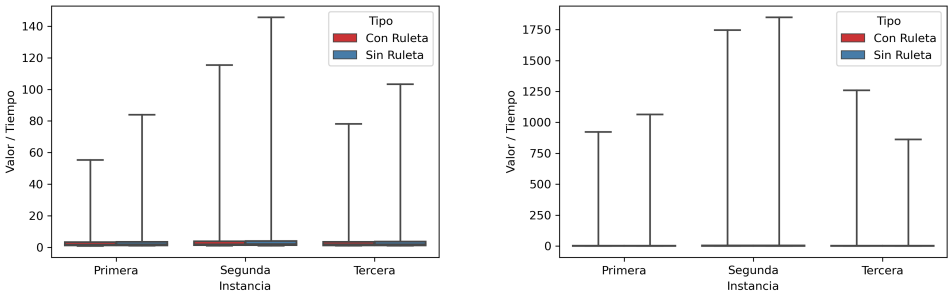
Figura 10: Caja bigote comparando valor sobre tiempo para cada instancia.



(a) Caja bigote de tercera instancia con 100 objetivos.

(b) Caja bigote de tercera instancia con 400 objetivos.

Figura 11: Caja bigote comparando valor sobre tiempo para cada instancia.



(a) Caja bigote de las tres instancias con 100 objetivos.

(b) Caja bigote de las tres instancias con 400 objetivos.

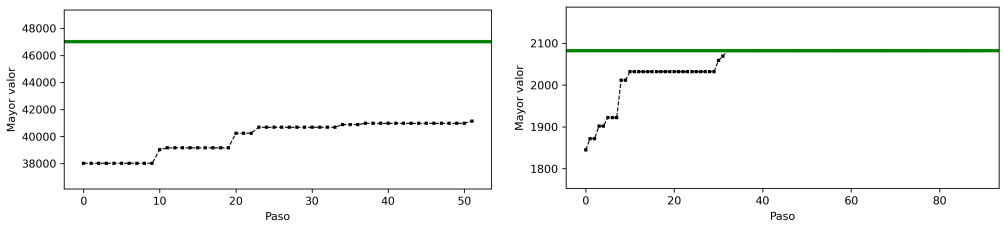
Figura 12: Caja bigote comparando valor sobre tiempo para cada instancia.

5. Reto 1

El primer reto es extender la selección de ruleta a la fase de supervivencia: en vez de quedarnos con las mejores soluciones, cada solución tiene una probablidad de entrar a la siguiente generación que es proporcional a su valor de la función objetivo, incorporando el sí o no es factible la solución en cuestión, permitiendo que los mejores entre las factibles entren siempre (donde k es un parámetro). Estudia nuevamente el efecto de este cambio en la calidad de la solución en los tres casos [2]. El código base se sacó de Schaeffer [3]. El código completo se encuentra en el GitHub [1].

```
d = []
k = int(init * 0.1) # porcentaje que siempre entra
#Usamos el acomodo para obtener el 10% mejor que son
#los que siempre van a entrar
for i in range(tam):
    d.append({'idx': i, 'obj': objetivo(p[i], valores),
              'fact': factible(p[i], pesos, capacidad)})
d = pd.DataFrame(d).sort_values(by = ['fact', 'obj'], ascending = False)
mantener = np.array(d.idx[:k])
popula = []
pesos_n = []
#Acomodamos los parametros para la Ruleta
for i in range(k, tam):
    vn = d.iloc[i]
    man = vn.idx
    o = vn.obj
    f = vn.fact
    popula.append(man)
    pesos_n.append((f+1)*o)
#Ruleta de suérvivencia
mantener2 = choices(population=popula, weights=pesos_n, k=(init-k))
#Guardamos valores de la ruleta
p2 = p[mantener2, :]
#Guardamos valores que siempre entran
p = p[mantener, :]
#Juntamos ambos arrays para tener el tamaño inicial
p = np.concatenate((p, p2), axis=0)
tam = p.shape[0]
```

Código 5: Genera la ruleta para la supervivencia.



(a) Gráfica de primera instancia con ruleta a 200 objetivos.

(b) Gráfica de segunda instancia con ruleta a 200 objetivos.

Figura 13: Gráficas comparando el mayor valor.

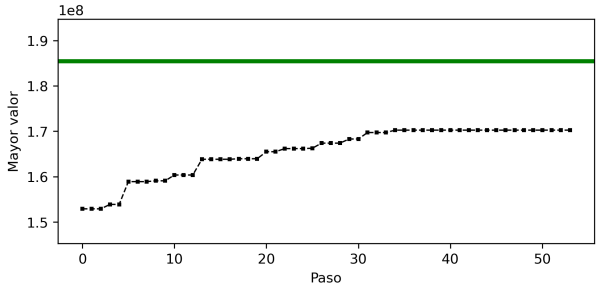


Figura 14: Gráfica de tercer instancia con ruleta a 200 objetivos.

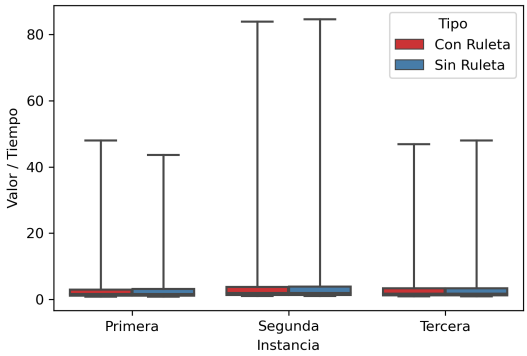
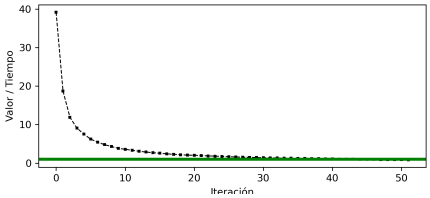
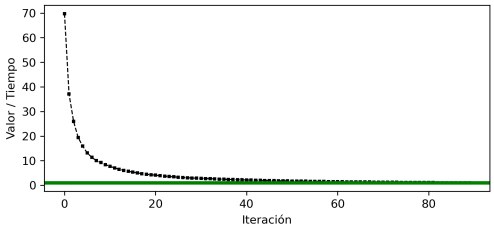


Figura 15: Caja bigote de las tres instancias con 200 objetivos.



(a) Gráfica de primera instancia con ruleta a 200 objetivos.



(b) Gráfica de segunda instancia con ruleta a 200 objetivoss.

Figura 16: Gráfica comparando valor sobre tiempo.

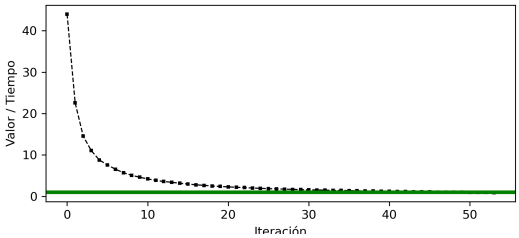


Figura 17: Gráfica de tercer instancia con ruleta a 200 objetivos.

Referencias

[1] *Denisse Leyva Repositorio*. URL: https://github.com/Denisse251/Simulation/tree/main/Tarea.10.

[2] *Elisa Schaeffer Práctica 7*. URL: https://elisa.dyndns-web.com/teaching/comp/par/p10.html.

[3] *Elisa Schaeffer Repositorio*. URL: https://github.com/satuelisa/Simulation.