Práctica 11 frentes de Pareto

Denisse Leyva

Mayo 12, 2021

1. Introducción

En optimización multicriterio, a un mismo conjunto de variables ocupa asignarse valores de tal forma que se optimizen dos o más funciones objetivo, que pueden contradecir una a otra, una mejora en una puede corresponder en una empeora en otra. Además hay que respetar potenciales restricciones, si es que haya.

Para estudiar este problema, vamos a primero implementar un generador de polinomios aleatorios. Estos polinomios los utilizaremos como funciones objetivo. Vamos a permitir solamente una variable por término y un término por grado por variable.

Para simplicidad de visualización, vamos a concentrarnos en el caso de dos funciones objetivo. Vamos a generar muchas soluciones al azar, también por simplicidad, los retos de la tarea permitirán mejorar este aspecto, y calculamos los valores de los objetivos para cada solución. Por simplicidad, vamos a suponer que no tenemos restricciones. Para cada objetivo, vamos a determinar si se va a minimizar (los marcamos con verdad) o maximizar (los marcamos con false). Marcamos con colores distintos las mejores soluciones de cada objetivo.

Como los dos mejores rara vez logran a coincidir y a veces son bastante opuestos, se ocupa una definición para qué en sí es una buena solución. Para eso usaremos la dominancia de Pareto: una solución domina a otra si no empeora ninguno de los objetivos y mejora a por lo menos uno. Vamos a calcular la dominancia en paralelo entre todas las soluciones y marcar aquellas que no son dominadas por ninguna otra solución [2].

2. Objetivo

Grafica el porcentaje de soluciones de Pareto (ojo, no es lo mismo que se grafica en el código ejemplo) como función del número de funciones objetivo para $k \in [2, 8]$ en pasos de dos con diagramas de violín combinados con diagramas de caja-bigote, verificando que diferencias observadas, cuando las haya, sean estadísticamente significativas. Razona en escrito a qué se debe el comportamiento observado [2].

3. Código

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

```
iteracion = 100 # cuantas iteraciones
porcentajes = []
for k in range(2, 9, 2): # cuantas funciones objetivo
    for it in range(iteracion):
        vc = 4
        md = 3
        tc = 5
        obj = [poli(md, vc, tc) for i in range(k)]
        minim = np.random.rand(k) > 0.5
        n = 250 # cuantas soluciones aleatorias
        sol = np.random.rand(n, vc)
        val = np.zeros((n, k))
        for i in range(n): # evaluamos las soluciones
            for j in range(k):
                val[i, j] = evaluate(obj[j], sol[i])
        sign = [1 + -2 * m for m in minim]
        no_dom = []
        for i in range(n):
            d = [domin_by(sign * val[i], sign * val[j]) for j in range(n)]
            no_dom.append(not np.any(d)) # si es cierto que ninguno es verdadero
        frente = val[no_dom, :]
        porcentaje = (len(frente)/n)*100
        porcentajes.append(porcentaje)
```

Código 1: Automatización para obtener las funciones objetivo de la 2 a la 8 de dos en dos.

4. Resultados

Como se puede observar en la gráfica dependiendo de las funciones objetivo entre menor sean, menor será el porcentaje de Pareto obtenido. Sin embargo, a más objetivos se puede observar que sube linealmente a cubrir casi el 100 % con el frente de Pareto. Esto se debe a que al momento de agregar más dimensiones el frente de Pareto por lógica va a ocupar mucho más espacio.

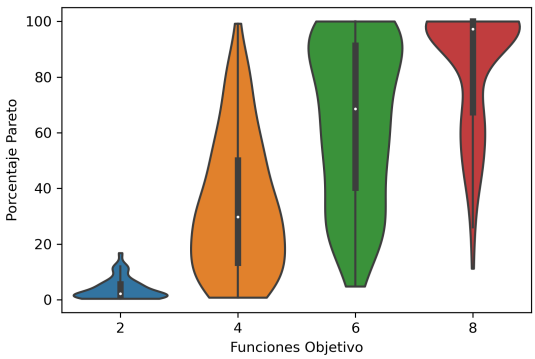


Figura 1: Gráfica violín de las funciones objetivo vs porcentaje Pareto.

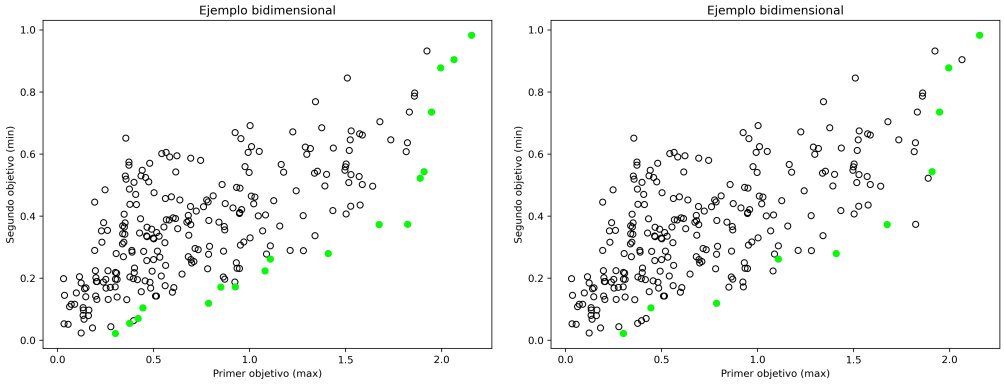
5. Reto 1

El primer reto es seleccionar un subconjunto (cuyo tamaño como un porcentaje del frente original se proporciona como un parámetro) del frente de Pareto de tal forma que la selección esté diversificada, es decir, que no estén agrupados juntos en una sola zona del frente las soluciones seleccionadas. Graficar los resultados de la selección, indicando con un color cuáles se incluyen en el subconjunto diverso [2].

Para hacer la reducción se buscó la distancia euclidiana mínima y luego se seleccionó la variable con la menor distancia entre los vecinos.

```
dm = max(x)
    for n in range(len(x)-1):
        dx = x[n] - x[n+1]
        dy = y[n] - y[n+1]
        dis = sqrt(dx*2 + dy*2)
        print(dis)
        if dm > dis:
            dm = dis
            m = n
            x1 = x[n]
            x2 = x[n+1]
            y1 = y[n]
            y2 = y[n+1]
    if m == 0:
        xd = x[m+1]
        yd = y[m+1]
        el = m+1
    elif m == max(x):
        xd = x[m-1]
        yd = y[m-1]
        el = m-1
    else:
        dx = x[m+1] - x[m+2]
        dy = y[m+1] - y[m+2]
        dis1 = sqrt(dx*2 + dy*2)
        dx = x[m] - x[m-1]
        dy = y[m] - y[m-1]
        dis2 = sqrt(dx*2 + dy*2)
        if dis1 < dis2:
            xd = x[m+1]
            yd = y[m+1]
            el = m+1
        else:
            xd = x[m]
            yd = y[m]
            el = m
    x = np.delete(x, el)
    y = np.delete(y, el)
```

Código 2: Reducción del frente de Pareto.



(a) Frente de Pareto Completo. (b) Frente de Pareto reducido en iteración 9 .

Figura 2: Gráficas que muestran la reducción del frente de Pareto.

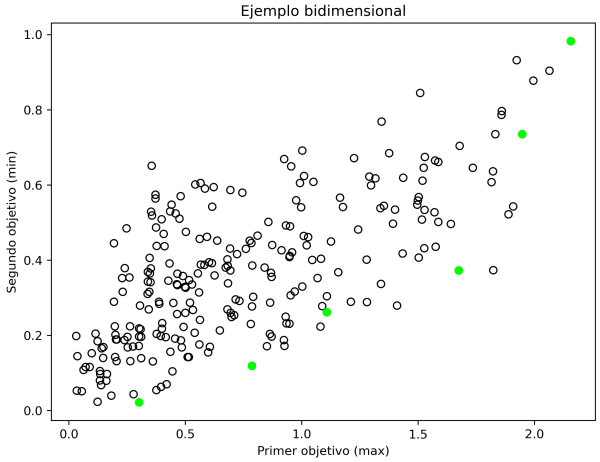


Figura 3: Frente de Pareto reducido en iteración 13.

Referencias

[1] *Denisse Leyva Repositorio*. URL: `https://github.com/Denisse251/Simulation/tree/main/Tarea.11`.

[2] *Elisa Schaeffer Práctica 7*. URL: `https://elisa.dyndns-web.com/teaching/comp/par/p11.html`.

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