Reentrega Informe TP3 Invop PNL

Juan I Catania, Felipe Pasquet

July 2025

1 Problema de Fermat-Weber

La formulación del problema de Fermat-Weber dice lo siguiente:
Dado un conjunto de puntos $P = \{a_1, a_2, \ldots, a_m\}, P \subseteq \mathbb{R}^d$ y un conjunto de pesos positivos $w_1, w_2, \ldots, w_m, w_i > 0$, queremos encontrar un punto $x \in \mathbb{R}^d$ que minimice la función objetivo:

$$W(x) = \sum_{i=1}^{m} w_i \|x - a_i\|$$

Para buscar el punto mínimo, buscamos puntos críticos. Cuando $x \neq a_i$ para todo i, el gradiente de $W(x)$ está dado por:

$$\nabla W(x) = \sum_{i=1}^{m} \frac{w_i(x - a_i)}{\|x - a_i\|}$$

Si $x = a_i$ para algún i, W no es diferenciable en x, pero es subdiferenciable, $\nabla j \in \partial W(pj)$, :

$$\nabla j = \sum_{i=1, i\neq j}^{m} w_i \frac{pj - pi}{\| pj - p \|} - w_j v$$

para todo $v \in R^n$ tal que $\|v\| \leq 1$.

Cuando los puntos no son colineales, y los pesos $w_i > 0$, la función W(x) es estrictamente convexa y **el problema de Fermat-Weber tiene una única solución óptima.**

Si $x^* \neq a_i$ para todo i, entonces:

$$\nabla W(x) = \sum_{i=1}^{m} \frac{w_i(x - a_i)}{\|x - a_i\|} = 0$$

a partir de eso podemos deducir:

$$\sum_{i=1}^{m} \frac{w_i(x - a_i)}{\|x - a_i\|} = x \sum_{i=1}^{m} \frac{w_i}{\|x - a_i\|} - \sum_{i=1}^{m} \frac{w_i a_i}{\|x - a_i\|} = 0 \longleftrightarrow x = \frac{\sum_{i=1}^{m} \frac{w_i a_i}{\|x - a_i\|}}{\sum_{i=1}^{m} \frac{w_i}{\|x - a_i\|}}$$

Si $x^* = a_j$ para algún j, entonces:

$$\|R_j\| \leq w_j, \textbf{donde } R_j = \sum_{i=1, i\neq j}^{m} w_i \frac{p_j - p_i}{\|p_j - p_i\|}$$

2 Algoritmos

2.1 Weiszfeld

Weiszfeld es un algoritmo iterativo para resolver el problema de Fermat-Weber. Sabemos que si x es mínimo de W si:

$$x = T(x) = \frac{\sum_{i=1}^{m} \frac{w_i a_i}{\|x-a_i\|}}{\sum_{i=1}^{m} \frac{w_i}{\|x-a_i\|}}$$

Así que buscamos el punto fijo de T, es decir, cuando se cumple lo que acabamos de decir. T(x) no está definido si $x = a_i$ para algún i, por lo que antes de iniciar el algoritmo iterativo, se verifica si alguno de los puntos a_k es solución óptima. Esto ocurre si se cumple:

$$\|R_k\| \leq w_k$$

En tal caso, a_k es un punto óptimo y se puede retornar como solución.
Si comprobamos que ninguno de los puntos de P es óptimo, tomamos como punto inicial el promedio ponderado de los puntos:

$$x^{(0)} = \frac{\sum_{i=1}^{m} w_i a_i}{\sum_{i=1}^{m} w_i}$$

Dado un punto $x^{(k)}$, se calcula la siguiente iteración como:

$$x^{(k+1)} = \frac{\sum_{i=1}^{m} \frac{w_i a_i}{\|x^{(k)} - a_i\|}}{\sum_{i=1}^{m} \frac{w_i}{\|x^{(k)} - a_i\|}} \xrightarrow{k\to\infty} x^*$$

Si llegamos a un $x^{(k)}$ tal que, $x^{(k)} = a_j \in P$ (y no es óptimo), iteramos de la siguiente manera:

$$x^{(k+1)} = S(a_j) = a_j + d_j t_j$$

Donde

$$d_j = -\frac{R_j}{\|R_j\|}, t_j = \frac{\|R_j\| - w_j}{\sum_{i\neq j} \frac{w_i}{\|a_i - a_j\|}}$$

Criterio de parada
El algoritmo termina inmediatamente si $x \in P$ y es óptimo.

Si no, el algoritmo se detiene cuando:

$$\|x^{(k)} - x^{(k-1)}\| \leq \epsilon$$

donde $\epsilon > 0$ es una tolerancia preestablecida. Esto, pues está cerca del mínimo, y se acerca tan lento que no vale la pena seguir iterando.

Resultado
El algoritmo retorna un punto x que minimiza la función $W(x)$ o que está suficientemente cerca del mínimo según el criterio de tolerancia.

2.2 Hooke - Jeeves

El método de Hooke y Jeeves es un algoritmo de optimización sin derivadas que busca minimizar una función $f : \mathbb{R}^n \to \mathbb{R}$ mediante una combinación de búsquedas locales (exploración) y saltos en direcciones prometedoras (movimientos de patrón).

Exploración local.
Dado un punto base $x \in \mathbb{R}^n$, se exploran sus coordenadas una por una:

- Se evalúa $f(x)$.
- Se intenta mover la coordenada x_i a $x_i + \delta$.
- Si no mejora, se prueba $x_i - \delta$.
- Si ninguna mejora, se vuelve al valor original.

Esta operación se repite para cada componente del vector. El resultado es un nuevo punto x' que mejora x, o se mantiene igual si no hay mejora.

Movimiento de patrón.
Si la exploración produce un punto mejor x_e respecto al anterior x_0, se realiza un salto en esa dirección:

$$x_p = x_e + \alpha(x_e - x_0)$$

donde $\alpha > 1$ es un factor de avance (por ejemplo, $\alpha = 2$). Este salto intenta aprovechar la dirección de mejora.
Luego se explora alrededor del punto x_p para confirmar si se mejora aún más.

Criterio de mejora.

- Si $f(x_e) < f(x_0)$, se acepta x_e como nuevo punto base.
- Si no hay mejora, se reduce el tamaño del paso: $\delta \leftarrow \delta/2$.

Condición de parada.
El algoritmo se detiene cuando el tamaño del paso δ es menor que una tolerancia ε. Es decir, si:

$$\delta \leq \epsilon$$

Resultado.
El algoritmo retorna un punto x tal que es cercano al mínimo dentro de la tolerancia especificada. Pero no garantiza la convergencia al óptimo exacto.

2.3 Gradiente Descendente

El método de gradiente descendente es un algoritmo iterativo para minimizar una función diferenciable $f : \mathbb{R}^n \to \mathbb{R}$.

Idea principal:
En cada iteración, se mueve en la dirección del **gradiente negativo** ya que moverse en la dirección opuesta al gradiente local garantiza la mayor reducción instantánea de la función, por lo que el algoritmo "desciende" hacia un mínimo local.

Algoritmo:
Dado un punto inicial $x^{(0)}$, se construye la sucesión

$$x^{(k+1)} = x^{(k)} - \alpha_k \nabla f\big(x^{(k)}\big)$$

donde:

- $\nabla f(x^{(k)})$ es el gradiente de f en el punto $x^{(k)}$. Al igual que en el algoritmo de Weiszfeld, si $x^{(k)} \in P$, no se puede calcular el gradiente. Si $x^{(k)}$ es óptimo lo retornamos, y si no es óptimo iteramos con $x^{(k+1)} = S(x^{(k)})$.
- $\alpha_k > 0$ es el tamaño del paso o *learning rate*, que puede ser constante o adapaptarse a alguna condicion dependiendo del criterio de paso.

Elección del tamaño del paso α_k:
Para determinar el tamaño del paso α_k usaremos el criterio de Armijo, este criterio asegura que cada paso proporcione una reducción suficiente en la función objetivo, obliga a que el decrecimiento sea proporcional al tamaño del paso, y evita pasos grandes con poco decrecimiento de f.

Condición de parada: El algoritmo termina cuando:

$$\|\nabla f(x^{(k)})\| < \varepsilon$$

donde $\varepsilon > 0$ es una tolerancia pequeña.

Resultado
El algoritmo, al igual que el de Weiszfeld, retorna un punto x que minimiza la función $f(x)$ o que está suficientemente cerca del mínimo según el criterio de tolerancia.

3 Análisis del tiempo de convergencia

Realizamos experimentos para comparar el tiempo de convergencia de los algoritmos *Weiszfeld*, *Hooke-Jeeves* y *Gradiente Descendente* utilizando distintas distribuciones de puntos en el plano [1] y pesos aleatorios; realizamos cada experimento 5 veces y promediamos los resultados.

- **Distribución uniforme:** Los puntos se generan de forma uniforme y dispersa en el espacio.
- **Distribución normal:** Los puntos se generan alrededor de una media central, formando un agrupamiento.
- **Distribución cluster:** Los puntos se distribuyen en dos grupos o clusters separados.

[1] La eleccion de estas distribuciones fueron meramente para ver como afecta a los algoritmos, mas alla de si pueden o no representar un escenario realista.

Los resultados obtenidos fueron los siguientes:

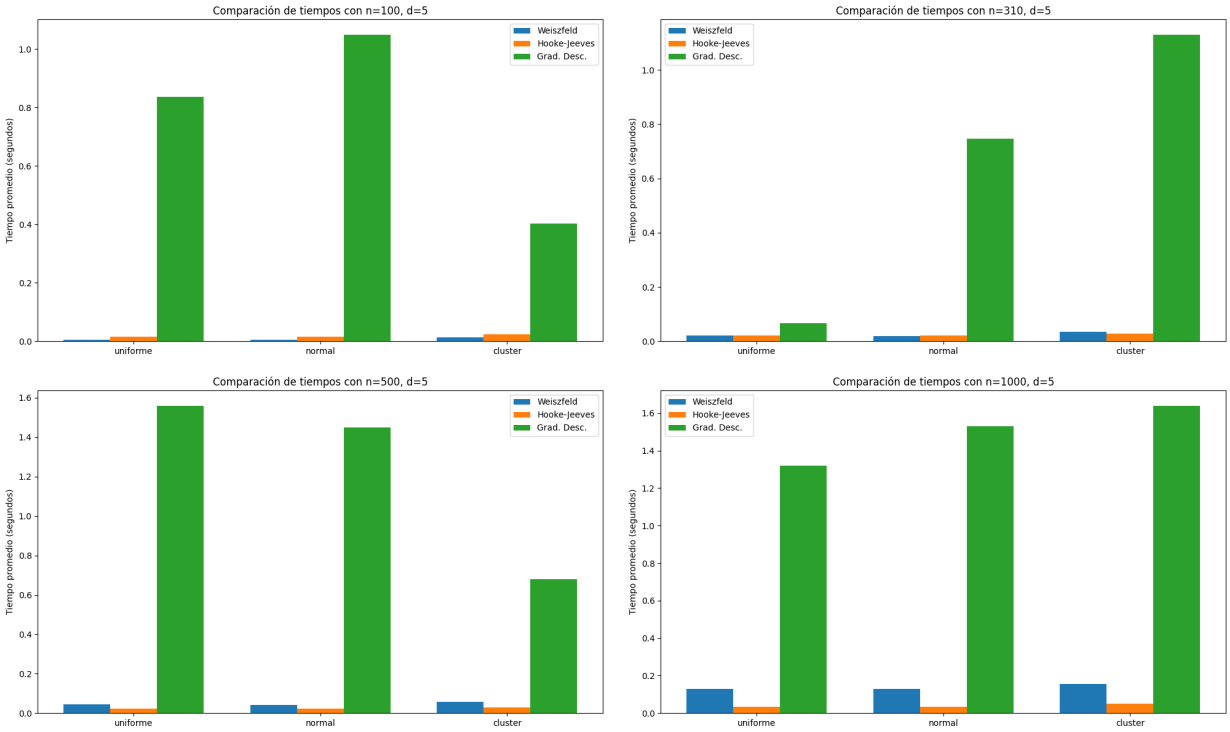

Figure 1: Tiempos a medida que crece la cantidad de puntos

Podemos ver que cuando la cantidad de puntos es baja, el algoritmo de Weiszfeld es el más rápido, pero a medida que aumenta, Hooke-Jeeves se hace el más rápido.

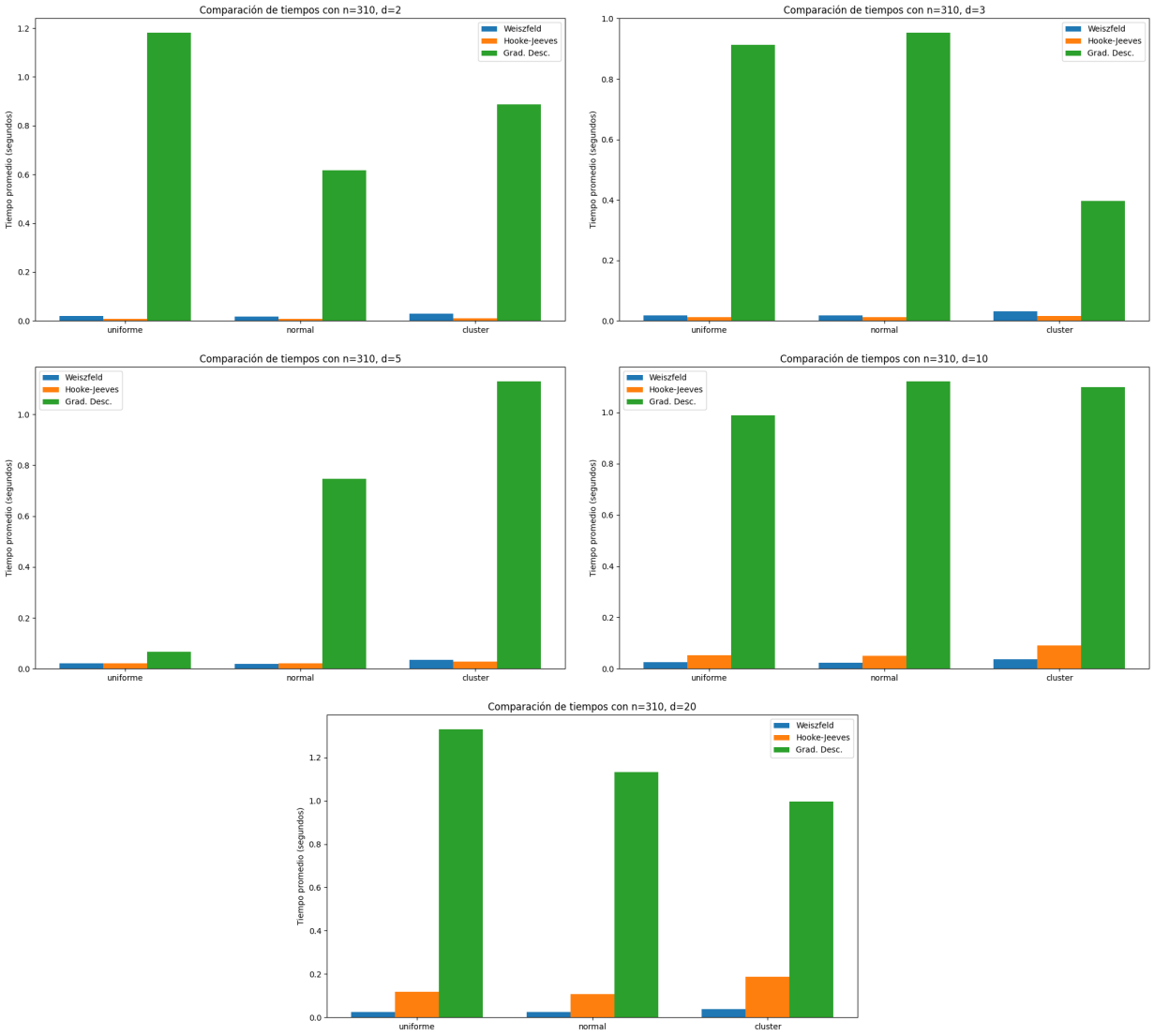

Figure 2: Tiempos a medida que crece la dimensión

Podemos ver que cuando la dimensión es baja el algoritmo de Hooke-Jeeves es el más rápido, pero a medida que aumenta, Weiszfeld se hace el más rápido.

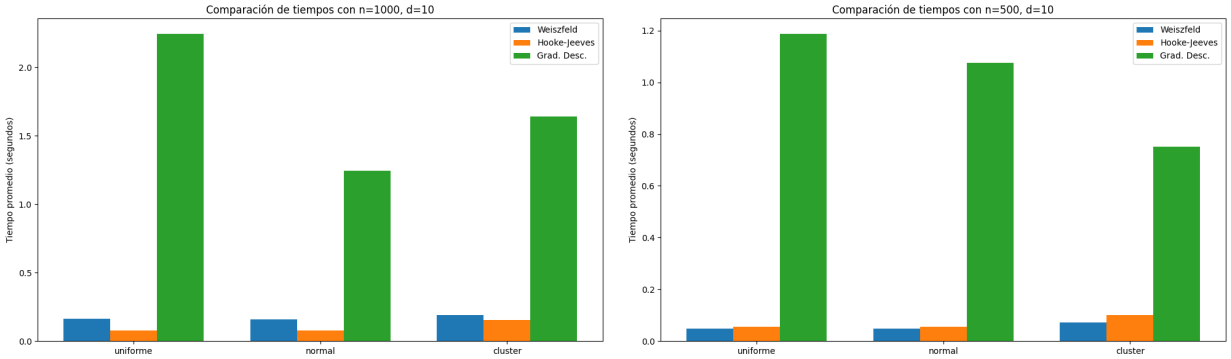

Figure 3: Vemos como afecta cambiar la dimensión y cantidad de puntos a experimentos que ya hicimos

Por último, vemos cómo en un caso de dimensión alta en el que el algoritmo de Weiszfeld se desempeñaba mejor, si aumentamos mucho la cantidad de puntos, el algoritmo de Hooke-jeeves de vuelta es más rápido que el de Weiszfeld. De manera contraria, en un caso de muchos puntos que antes era más rápido Hooke-Jeeves, si aumentamos la dimensión, Weiszfeld es más rápido.

Conclusiones

Observamos que los algoritmos de **Weiszfeld** y **Hooke-Jeeves** tienen un tiempo de ejecución bastante bueno. En algunos casos gana **Weiszfeld** en tiempo de ejecución y en otros casos **Hooke-Jeeves**. También observamos que en todos los casos el algoritmo más lento, por mucho, es el de **Gradiente Descendente**.
Podemos ver que sin importar cómo estén agrupados los puntos, a medida que crece la dimensión, manteniendo la cantidad de puntos fija, el algoritmo de **Weiszfeld** mejora el tiempo de ejecución respecto a **Hooke-Jeeves** y tarda menos en dimensiones altas.
Por otro lado, vemos que si mantenemos la dimensión fija y aumentamos la cantidad de puntos, el algoritmo de **Hooke-Jeeves** termina siendo más rápido que **Weiszfeld**.

En resumen, para aplicaciones donde la cantidad de puntos sea baja o la dimensión sea muy alta conviene el algoritmo de **Weiszfeld**, mientras que si la cantidad de puntos es muy alta o la dimensión es baja puede convenir usar **Hooke-Jeeves**. También vale la pena recordar que el algoritmo de **Hooke-Jeeves** no nos garantiza convergencia, a diferencia de los algoritmos de **Weiszfeld** y **Gradiente Descendente**.
Una explicación de por que nuestro algoritmo de Wiszfeld tarda más que Hooke-Jeeves a medida que aumenta la cantidad de puntos, es que verificamos todos los puntos dados para ver si son óptimos.