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.
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, $x^{(k)} = a_j \in P$ (es uno de los puntos que nos dieron), T(x) se indefine, por lo que primero verificamos si a_j es solución óptima. Esto ocurre si se cumple:

$$\|R_j\| \leq w_j$$

En tal caso, a_j es un punto óptimo y se puede retornar como solución.

Si 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 $x^{(k)} = a_j \in P$, no se puede calcular el gradiente. Si $x^{(k)}$ es óptimo lo retornamos, y si no es óptimo iteramos utilizando el subgradiente, $x^{(k+1)} = S(a_j)$.
- $\alpha_k > 0$ 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. También agregamos como criterio de parada si el valor de la función ya no mejora entre las iteraciones.

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 3 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.

Los resultados obtenidos fueron los siguientes:

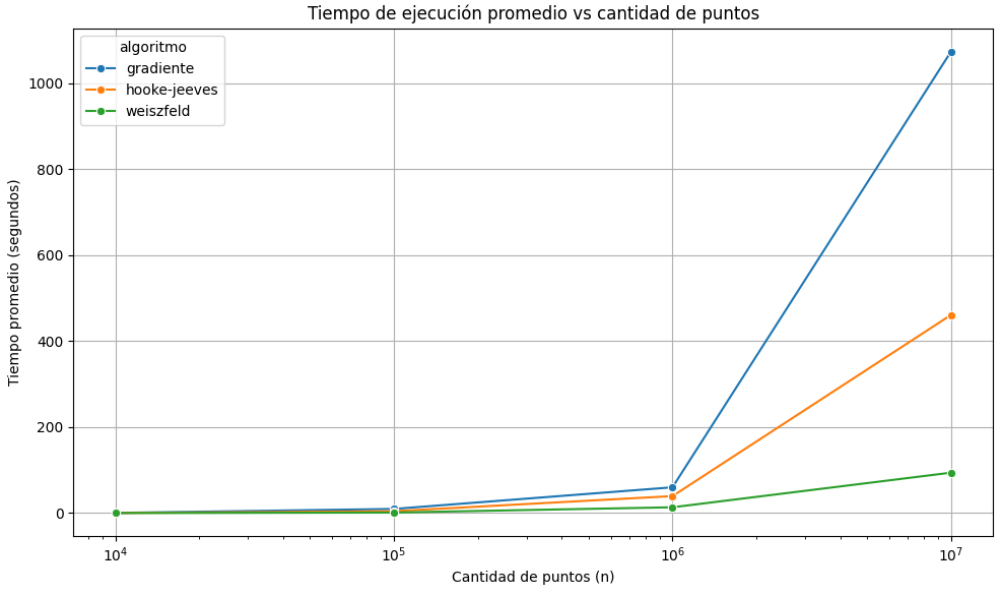

Figure 1: Tiempo de convergencia a medida que aumenta la cantidad de puntos

[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.

En este experimento[2] observamos que el algoritmo de **weiszfeld** es mucho más rápido que **Hooke-Jeeves** y que el **metodo del gradiente descendiente**. A medida que aumenta la cantidad de puntos se hace más evidente esta diferencia. También podemos ver que el **Hooke-Jeeves** es bastante más rápido que **gradiente descendente**, y más se nota cuanto más aumenta la cantidad de puntos.

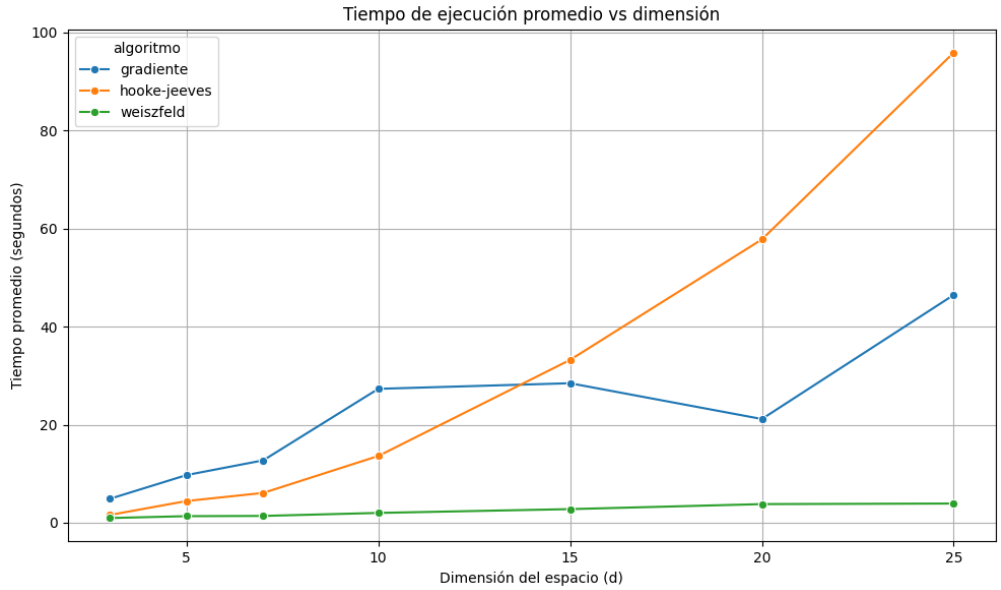

Figure 2: Tiempo de convergencia a medida que aumenta la dimensión

También observamos que si mantenemos fija la cantidad de puntos, a medida que aumenta la dimensión, el algoritmo de **weiszfeld** sigue siendo mucho más eficiente que los otros algoritmos. Podemos ver como a medida que aumenta la dimensión el **metodo del gradiente descendente** se hace más eficiente que **hooke-jeeves**.

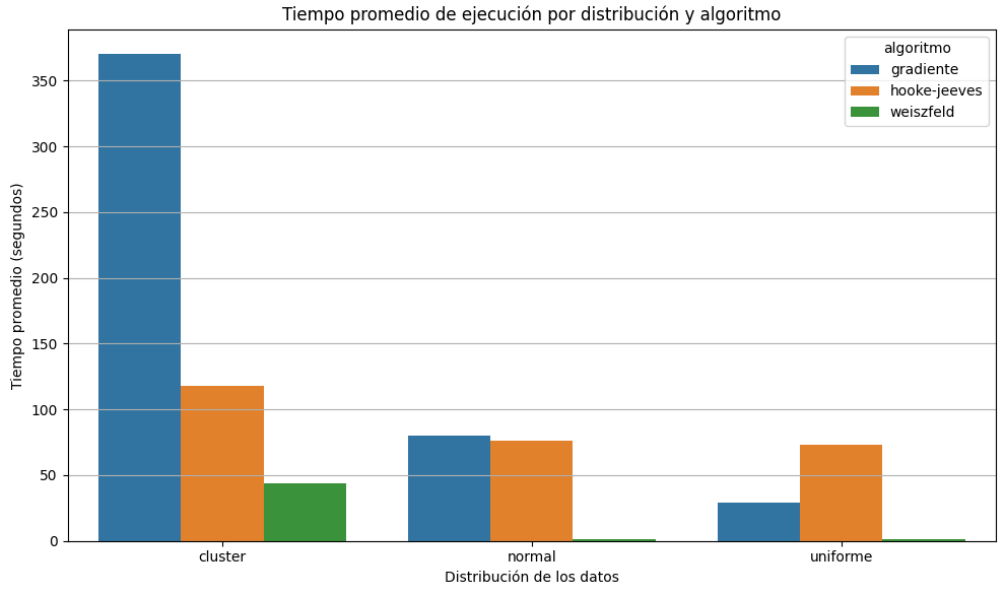

Figure 3: Tiempo de convergencia con distintas distribuciones de los datos

[2]Promediamos entre distintas instancias de dimensión 5, aumentando la cantidad de puntos

Por último vemos como, sin importar el algoritmo, el tiempo de convergencia aumenta mucho cuando los datos están distribuidos en clusters separados.[3]

Conclusión

En base a nuestros experimentos podemos concluir que para el problema de **Fermat-Weber**, sin importar la dimensión, cantidad de puntos o como estén agrupados los puntos, **Weiszfeld** es el algoritmo más eficiente para resolver este problema entre los algoritmos que comparamos. No solo mantiene su eficiencia a medida que aumenta la cantidad de puntos y la dimensión del problema, también es un algoritmo que nos asegura convergencia al óptimo global (a diferencia de **Hooke-Jeeves**). También podemos ver que sin importar el algoritmo, la cantidad de puntos ni su dimensión, es más difícil converger al mínimo global cuando los datos están agrupados en clusters separados.

[3]Promediamos los tiempos de todos los experimentos, de distintas dimensiones y distintas cantidades de puntos