Universidad Nacional de San Antonio Abad del Cusco
Facultad de Ingeniería Eléctrica, Electrónica, Informática y Mecánica
Escuela Profesional de Ingeniería Informática y de Sistemas

TESIS

# ALGORITMO DE OPTIMIZACIÓN MULTIOBJETIVO PARA EL PROBLEMA *CENTER-BASED CLUSTERING* PARA CONJUNTOS CON *OUTLIERS*

*Por:*

Br. Jared León Malpartida

*Bajo la asesoría de:*

M.Sc. Lauro Enciso Rodas

*Para optar al título profesional de:*

Ingeniero Informático y de Sistemas

Cusco – Perú

2019

*A mi familia.*

## Resumen

*Clustering* (agrupamiento) es usualmente considerado el problema más importante del aprendizaje automático no supervisado. Al igual que los problemas no supervisados, el problema del clustering consiste en descubrir patrones de agrupamiento. En particular, se busca agrupar un conjunto de datos no etiquetados en conjuntos llamados *clusters* (o grupos). Dada la naturaleza del problema, este aparece en multitud de áreas de investigación como: compresión de datos, análisis de imágenes, bioinformática, y minería de datos.

A la fecha, se han diseñado multitud de algoritmos y modelos de clustering. También, se ha generalizado el tipo de datos con los que se puede aplicar esta técnica. Uno de los modelos de clustering más ampliamente utilizados está relacionado con el conjunto de problemas *center-based*. Este conjunto de problemas es uno de los más recientemente estudiados debido a su eficiencia con grandes cantidades de datos. En general, un problema de este tipo busca particionar el conjunto inicial de elementos tomando como base algunos elementos centrales.

Con el objetivo de mejorar las técnicas actuales en esta rama; la presente investigación desarrolla y propone un nuevo algoritmo de clustering, denominado el algoritmo SSO-C. La metodología seguida para desarrollar el algoritmo consistió en la optimización de una función multiobjetivo que relaciona dos problemas formalmente definidos con el propósito de garantizar la robustez de la solución encontrada. Como búsqueda local para valores iniciales, se tomó soluciones con un cierto factor de aproximación para un problema de optimización combinatoria relacionado, el problema $k$-center. En la investigación también se desarrolla y propone un segundo algoritmo de clustering, denominado el algoritmo Emax. Este segundo algoritmo es derivado del caso más robusto de la función multiobjetivo. La convergencia del algoritmo Emax es demostrada.

Para efectos de comparación, se tomaron los algoritmos $k$-means y SSO. El primero es uno de los algoritmo más utilizados para hacer *clustering*, y el segundo es una adaptación del algoritmo de optimización *Social Spider Optimization* para clustering; ambos pertenecientes al modelo *center-based*. Se compararon los algoritmos mencionados junto con los propuestos (SSO-C y Emax) tomando un conjunto de 6 conjuntos de datos sintéticamente generados y 7 del mundo real tomados de la literatura. Los experimentos muestran con significación estadística que los algoritmos SSO-C y Emax dan los mejores resultados entre los algoritmos comparados.

Se espera que los algoritmos propuestos generen contribuciones significativas para estado-del-arte.



***Palabras clave**: Center-based Clustering, Social Spider Optimization, Optimización Multiobjetivo, Algoritmos de Aproximación, SSO-C, Emax.*

## Abstract

*Clustering* is usually considered as the most important unsupervised machine learning problem. As most unsupervised problems, clustering focuses on searching for patterns and groups. In particular, the objective is to group a set of unlabeled data into sets called *clusters*. Given the nature of this problem, it appears in a wide range of research areas such as: data compression, image analysis, bioinformatics, and data mining.

To the date, a wide range of algorithms and models of clustering have been developed. Also, the type of data that can be clustered have been generalized. One of the more widely used clustering models is related to the *center-based* set of problems. This set of problems is particularly efficient with a large amounts of data. This type of problems aims to partition an initial set of elements taking central elements as base.

With the objective of improving the current techniques on this field, in this work a new clustering algorithm is developed and proposed, namely the SSO-C algorithm. The followed methodology to develop the algorithm consisted on optimizing a multiobjective function that relates two well defined problems searching for robust solutions. As a local search for few initial values, some solutions having a certain approximation factor of a related combinatorial optimization problem called $k$-center were taken. In the present work, a second clustering algorithm is also developed and proposed, namely the Emax algorithm. This second algorithm is derived from the most robust case of the multiobjective function. The convergence of the Emax algorithm is proved.

For comparison purposes, two other algorithms were taken: $k$-means and SSO. The first one is one of the most used algorithms for performing clustering, and the second one is an adapted version of an optimization algorithm called *Social Spider Optimization* for clustering; both of them belong to the center-based model. The mentioned algorithms together with the proposed (SSO-C and Emax) were compared using a set of 6 synthetic and 7 real-world datasets. The experiments show with statistical significance that the SSO-C and Emax algorithms achieve the best results among the compared algorithms.

The proposed algorithms are expected to provide significant contributions to the state-of-the-art.



***Keywords**: Center-based Clustering, Social Spider Optimization, Multiobjective Optimization, Approximation Algorithms, SSO-C, Emax.*

# Índice

# Índice de figuras



# Índice de tablas

# Capítulo 1

# Aspectos generales

## 1.1. Problema de investigación

La presente investigación se centra en un problema dentro del aprendizaje automático no supervisado llamado *clustering*. Como cualquier otro problema de aprendizaje no supervisado, este involucra la búsqueda de patrones y estructuras en datos no etiquetados. En su idea más básica, el objetivo del clustering es agrupar objetos similares en conjuntos llamados *clusters*.

En esencia, clustering no es un problema estrictamente bien definido, más bien, una tarea muy general a ser resuelta. Por este motivo, muchos problemas y algoritmos bien definidos han sido propuestos para resolverla. Existe una gran variedad de conceptos y definiciones diferentes entre sí de qué es un *cluster* y cómo puede ser formado. También, no existe una convención general acerca de los tipos de datos que pueden ser *clusterizados*. Toda esa diversidad dio pie a multitud de modelos de clustering (Aggarwal & Reddy, 2013). Uno de los modelos mayor conocidos de clustering es el llamado *center-based model*.

*Center-based clustering* es el problema de particionar un conjunto de datos en clusters que sean representados, cada uno por un elemento llamado "centro" del cluster (Aggarwal & Reddy, 2013).

Uno de los algoritmos de clustering más conocidos y sencillos de comprender es el algoritmo $k$-means. Este algoritmo, pertenece a los modelos center-based. A pesar de la simpleza y potencia del algoritmo $k$-means, este suele presentar problemas de robustez con *outliers*, es decir, el algoritmo pierde precisión cuando el conjunto de datos con el que se trabaja presenta ruido considerable (Hautamäki, Cherednichenko, Kärkkäinen, Kinnunen & Fränti, 2005). Este problema se presenta mayormente cuando existe solapamiento entre clusters y cuando existen puntos alejados de los centros de concentración. Todos estos escenarios causan que el algoritmo en ocasiones no sea lo suficientemente *resistente* o *robusto*.

### 1.1.1. Descripción del problema

Se aproxima el problema del clustering para este trabajo de la siguiente forma: el problema, en un espacio métrico $(\mathbb{R}^d, L_2)$, donde $L_2$ denota el espacio Euclidiano, consiste en particionar un conjunto de $n$ puntos en $k$ subconjuntos o clusters teniendo como base alguna métrica de similitud o disimilitud. El conjunto de éstos $n$ puntos $\{x_1, x_2, \ldots, x_n\}$ estará representado por la letra $\mathcal{S}$ y los $k$ subconjuntos o clusters estarán representados por $C_1, C_2, \ldots, C_k$. El problema consiste en determinar los subconjuntos mencionados de modo que se cumplan las siguientes condiciones:

$$C_i \neq \varnothing \tag{1.1}$$

$$C_i \cap C_j = \varnothing \text{ para } i \neq j \tag{1.2}$$

$$\text{y } \bigcup_{i=1}^{k} C_i = \mathcal{S}. \tag{1.3}$$

Además de estas condiciones, los subconjuntos hallados también deben cumplir que; sea el clustering $C_1^1, C_2^1, \ldots, C_k^1$ un resultado producido por el algoritmo $k$-means, el algoritmo propuesto debe obtener el clustering $C_1, C_2, \ldots, C_k$ de manera que:

$$\text{AMI}[C_1, C_2, \ldots, C_k] > \text{AMI}[C_1^1, C_2^1, \ldots, C_k^1]. \tag{1.4}$$

Donde AMI representa la función *Adjusted Mutual Information* explicada en el marco teórico.

### 1.1.2. Identificación del problema

¿Existe un algoritmo de optimización multiobjetivo que solucione el problema center-based clustering sobre conjuntos con outliers?

## 1.2. Antecedentes

A continuación, se mencionan algunos antecedentes de la bibliografía revisada y que son relevantes para la investigación:

### 1.2.1. Algoritmo de optimización basado en el comportamiento social de las arañas para clustering

En este trabajo (Vera-Olivera, Soncco-Álvarez & Enciso-Rodas, 2016) se utilizó el algoritmo Social Spider Optimization (Cuevas, Díaz Cortés & Oliva Navarro, 2016) para hacer clustering agrupando los datos minimizando la suma de las distancias Euclidianas de cada punto al centro del cluster. Se hizo una implementación del algoritmo SSO y se adaptó para solucionar el problema del clustering. También se implementó un algoritmo genético para el mismo problema. Luego, se hizo comparaciones de los dos algoritmos mencionados y el algoritmo $k$-means, ya que este último es uno de los algoritmos más simples y potentes para hacer clustering. Las pruebas experimentales mostraron que el algoritmo SSO adaptado para clustering, al igual que el algoritmo genético, obtuvieron resultados competitivos con el algoritmo $k$-means.

### 1.2.2. Algoritmo de aproximación-2 para el problema *k*-center

En esta investigación (Gonzalez, 1985) se presenta un algoritmo de aproximación-2 con complejidad $O(nk)$ para el problema $k$-center. Siendo $n$ el número de puntos y $k$ el número de clusters a formar. El problema $k$-center está bastante relacionado con el problema del clustering en general, con la particularidad de ser un problema de optimización combinatoria. El algoritmo presentado garantiza una solución con función objetivo de alo más dos veces la solución óptima del problema $k$-center.

A pesar de la formulación cerrada del problema $k$-center, este trabajo lo introduce con un enfoque basado en grafos, en particular, introduce el problema de hallar un clique de tamaño $k + 1$ dentro de un grafo ponderado completo. El algoritmo asume la desigualdad triangular

entre los puntos. También muestra que el factor de aproximación-2 es el mejor posible si P $\neq$ NP. Es decir, lograr cualquier factor de aproximación-$(1+\varepsilon)$ con $0 \leq \varepsilon \leq 1$ es un problema NP-Hard.

La función objetivo tomada de la investigación minimiza la máxima distancia entre dos puntos que pertenezcan al mismo cluster (existen otras formulaciones equivalentes). En otras investigaciones, se adapta la solución presentada por este trabajo a una función objetivo más conveniente desde un punto de vista analítico para propósitos de análisis, facilitando la obtención de algoritmos más generales.

### 1.2.3. Algoritmos exactos y de aproximación para el problema *k*-center

El principal aporte de esta investigación (Agarwal & Procopiuc, 2002) es un algoritmo exacto para resolver el problema $k$-center de forma exacta en $\mathbb{R}^d$ con complejidad $n^{O\left(k^{l-1/d}\right)}$ bajo cualquier métrica-$L_p$. Este algoritmo puede ser generalizado para métricas más generales. El algoritmo es descrito en una métrica $L_\infty$ por simplicidad. La mayor parte de esta investigación se basa en la siguiente observación: si un conjunto $S$ puede ser cubierto por un conjunto $\mathcal{C}$ de $k$ cuadrados unitarios, entonces las únicas posibilidades son que $S$ se encuentra en una franja horizontal de longitud $\sqrt{k}+2$ o existe una línea horizontal $l$ que interseca a lo más $\sqrt{k}$ cuadrados de $\mathcal{C}$. Basándose en esto, se desarrolló un algoritmo de programación dinámica para encontrar la solución óptima. Este algoritmo también provee una complejidad $n^{O\left(k^{l-1/d}\right)}$ para el problema $k$-center discreto en $\mathbb{R}^d$. Este algoritmo asume las métricas $L_\infty$ y $L_2$. El algoritmo puede extenderse a otras métricas y al problema $k$-center discreto. Basándose en esto, la investigación también proporciona un algoritmo de aproximación-$(1+\varepsilon)$ para el problema $k$-center, con complejidad $O\left(n \log k\right) + (k/\varepsilon)^{O\left(k^{l-1/d}\right)}$. También se presenta un algoritmo con complejidad $n^{O\left(k^{l-1/d}\right)}$ para resolver el problema $k$-center-$L$-capacitado, suponiendo que $L = \Omega\left(n/k^{l-1/d}\right)$ o $L = O\left(1\right)$.

### 1.2.4. Algoritmo de *Swarm Intelligence* para optimización global

En este trabajo (Cuevas et al., 2016) se presenta un novedoso algoritmo de optimización global llamado *Social Spider Optimization* (SSO), el que está basado en *Swarm Intelligence*, que es un área de investigación que modela un comportamiento cooperativo en colonias de insectos y otros animales. Existen muchos de estos algoritmos propuestos para resolver un gran rango de problemas.

El algoritmo SSO, es un algoritmo para resolver problemas que involucran optimización, como lo será el problema formulado durante la investigación. El algoritmo está basado en el comportamiento cooperativo de las *arañas sociales*. En el algoritmo propuesto, los individuos simulan un conjunto de arañas que interactúan entre si basados en principios biológicos de la colonia. En este algoritmo se consideran dos tipos de agentes: arañas macho y arañas hembra. Dependiendo del género, cada individuo actúa por un conjunto de operadores evolutivos que imitan los comportamientos típicamente encontrados en la colonia.

Este algoritmo fue probado con un conjunto de funciones objetivo frente a dos algoritmos muy utilizados en la literatura: *Particle Swarm Optimization* (PSO) (Kennedy & Eberhart, 1995) y *Artificial Bee Colony* (ABC) (Karaboga & Basturk, 2007). Los resultados muestran un mejor desarrollo del algoritmo SSO al encontrar óptimos globales en las funciones benchmark.

## 1.3. Objetivos

### 1.3.1. Objetivo General

Proponer un algoritmo de optimización multiobjetivo que solucione el problema center-based clustering sobre conjuntos con outliers.

### 1.3.2. Objetivos Específicos

Los objetivos específicos incluyen:

- Formular el problema matemático center-based clustering como un problema de optimización multiobjetivo tomando en cuenta las características de una solución robusta.
- Solucionar el problema multiobjetivo utilizando el criterio de optimalidad de Pareto para hallar un algoritmo de optimización global efectuando una búsqueda local con algoritmos de aproximación.
- Diseñar un segundo algoritmo para el caso más robusto de la función multiobjetivo planteada.
- Generar conjuntos de datos que presenten la característica de múltiples outliers para efectuar las pruebas experimentales, implementar los algoritmos diseñados y probar estadísticamente su superioridad.

## 1.4. Alcances y limitaciones

Debido al interés que se tiene en desarrollar un algoritmo de clustering que funcione efectivamente con grandes cantidades de datos, es necesario limitar la investigación a una clase específica de algoritmos.

- Tanto los algoritmos propuestos, como los algoritmos de comparación, serán todos algoritmos de center-based clustering.
- Dado que el algoritmo $k$-means es el más utilizado para resolver el problema del clustering, y es también un algoritmo basado en centros; se tomará a este como estándar de comparación. Se utilizará también el algoritmo SSO (Vera-Olivera et al., 2016) al ser este último un algoritmo de center-based clustering y haber mostrado resultados competitivos con el algoritmo $k$-means en la investigación mencionada.
- Para obtener un factor de aproximación en la optimización de la función objetivo se utilizará un algoritmo de aproximación para el problema $k$-center, debido a que este es NP-hard. El algoritmo de aproximación que será utilizado tiene la ventaja de garantizar una solución óptima bajo Resistencia a la Perturbación (M.-F. Balcan, Haghtalab & White, 2016).
- Dado que existe una multitud de técnicas para acelerar el tiempo de ejecución de un algoritmo, la investigación se enfocará únicamente en desarrollar un algoritmo que logre una precisión superior.
- No se analizará la clase de complejidad a la que pertenece el algoritmo del caso extremo de la función objetivo. La hipótesis subsecuente a la investigación es que el problema propuesto es NP-Hard.

## 1.5. Metodología

El tipo de investigación realizado en el presente trabajo según su objetivo, corresponde con una investigación teórica. A continuación, se presenta la metodología utilizada para la investigación. Se tomó como referencia a (Brown & Porter, 1995).

1. **Diseño de los algoritmos.** En esta fase se hace realiza el diseño de los algoritmos que solucionarán el problema. Al finalizar esta etapa se debe contar con los algoritmos formulados y sus funcionamientos deben estar demostrados. El proceso realizado para el diseño de los algoritmos es mencionado a continuación:

   - **Diseño de la función objetivo.** En esta etapa se define matemáticamente el problema a resolver, esto permitirá tener un estricto marco de referencia bajo el cual desarrollar los algoritmos. El problema estará modelado por una función a ser optimizada, la función $\mathcal{F}$.
   - **Búsqueda local en $\mathcal{F}$.** En esta etapa se define el método de obtención de soluciones aproximadas para la función asumiendo un criterio razonable llamado Perturbation Resilience.
   - **Optimización multiobjetivo.** En esta etapa se cubren los últimos detalles necesarios para el diseño del primer algoritmo propuesto. Esto implica la utilización del criterio de Pareto para garantizar la obtención de la "mejor" solución. Al final de esta etapa, el algoritmo es presentado.
   - **Caso más robusto de $\mathcal{F}$.** En esta etapa se resuelve analíticamente un caso especial de la función objetivo con el fin de diseñar un algoritmo más apropiado para su caso más robusto. Se diseña teóricamente un algoritmo exacto (que no es especificado dado que demuestra no ser práctico); y luego se diseña un algoritmo heurístico para el problema. Se demuestra también que el algoritmo heurístico alcanza la convergencia.

2. **Generación de datasets sintéticos.** Para hacer la prueba de los algoritmos, es necesario contar con datasets con propiedades específicas. Particularmente, estos deben contar con valores atípicos generados especialmente de modo que un algoritmo no resistente a estos tenga una baja precisión de clasificación.

3. **Implementación de los algoritmos.** En esta etapa se utilizará el lenguaje de programación C para la implementación de los algoritmos. Esto permitirá pasar a la siguiente fase.

4. **Evaluación y análisis de los resultados.** En esta fase se utilizará la implementación de los algoritmos obtenida en el paso anterior y se efectuará una comparación con la función Adjusted Mutual Information (AMI). Posteriormente se analizará estadísticamente los resultados obtenidos para obtener las conclusiones.

## 1.6. Justificación

**Justificación científica.** Dentro de los algoritmos de clustering más utilizados en *data science*, el enfoque de center-based clustering es bastante eficiente en tiempo de ejecución y almacenamiento al ser aplicado a grandes cantidades de datos (Gan, Ma & Wu, 2007). Esto se debe, en parte, a que para identificar un cluster únicamente se requiere de un elemento representativo (centro del cluster). Así, si se tienen $k$ de estos elementos, se asigna cada elemento del conjunto

de datos original al elemento más próximo a él (de los $k$ centros representativos). Esto particiona el conjunto inicial en $k$ conjuntos disjuntos o clusters. Por lo tanto, un clustering sobre un conjunto $\mathcal{S}$ queda totalmente definido por $k$ elementos $\{c_1, c_2, \cdots, c_k\}$, con $c_j \in \mathbb{R}^d$.

El enfoque center-based consume muchos recursos de memoria (en tiempo de ejecución). Además, estos algoritmos no suelen reconocer formas sofisticadas ni patrones curvos dentro del conjunto de datos. A pesar de estos inconvenientes, existe una clara compensación en recursos temporales y de almacenamiento. Es por esto que el algoritmo de clustering más utilizado ($k$-means) es un algoritmo que utiliza el enfoque de center-based. La presente investigación se enfocará en diseñar un algoritmo de center-based clustering que obtenga una precisión superior a la del algoritmo $k$-means tomando como punto de referencia los conjuntos con outliers.

**Justificación tecnológica.** Clustering es tal vez, dentro de la Inteligencia Artificial actual, el intento más autónomo por imitar el comportamiento inteligente. El caso más frecuente dentro del análisis exploratorio se presenta cuando *nadie* sabe si los datos que están siendo analizados pueden ser condensados en un número pequeño de patrones representativos que pueden ser usados para resumir los datos en una representación más compacta. En este entorno, los algoritmos de clustering son capaces de "descubrir" los patrones ocultos de los datos tomando criterios de similitud/disimilitud sin la necesidad de información a priori de entrenamiento (muchas veces incluso, sin la necesidad del número de clusters a descubrir).

Es claro que el desempeño de un algoritmo de clustering es fundamental para la obtención de información a partir de datos. Con mucha frecuencia, los datos analizados en aplicaciones prácticas presentan "ruido" (o outliers) que pueden alterar el funcionamiento del algoritmo. Muchos de los algoritmos más utilizados de clustering (entre ellos el algoritmo $k$-means), presentan la tendencia de ser afectados frecuentemente por este tipo de elementos.

El diseño de algoritmos más robustos frente a datos con multitud de outliers sin duda es algo que mejoraría de sobremanera la calidad de información obtenida en aplicaciones prácticas. Este hecho tendría una relevancia significativa en campos como la minería de datos, análisis exploratorio, segmentación de imágenes, aplicaciones médicas, y aplicaciones financieras al poder los algoritmos ser más efectivos frente a datos variados.

## 1.7. Contribuciones

Las contribuciones de este trabajo serán para la comunidad de Teoría de la Computación y Aprendizaje Automático. Estas son:

- Un algoritmo de optimización multiobjetivo para el problema del clustering con parámetros de robusteza variables.
- Un algoritmo basado en centros para el problema del clustering más apropiado para el caso más extremo de valores atípicos permitido por la función objetivo diseñada (más robusto).
- Un conjunto de *datasets* sintéticos que permitan evaluar la robustez de algoritmos de center-based clustering.

# Capítulo 2

# Marco teórico

La presente investigación utiliza conceptos matemáticos de nivel universitario que incluyen: Teoría de Números elemental (Weil, 1974), Matemáticas Discretas (Rosen, 2002), Análisis Matemático (Stewart, 2012), Investigación Operativa (Hillier & Lieberman, 2001) y Estadística básica (DeGroot & Schervish, 2012). También se utilizan conceptos de Algorítmica (Cormen, Leiserson, Rivest & Stein, 2009) y Teoría de la Computación (Sipser, 2006).

Además de esto, algunos de los conceptos específicos requeridos se describen a continuación:

## 2.1. Espacios $L^p$

Los espacios $L^p$ son espacios funcionales definidos como una generalización natural de la norma-$p$ para espacios vectoriales de dimensión finita. Son también llamados espacios de Lebesgue por Henri Lebesgue (Dunford & Schwartz, 1971).

La longitud de un vector $x = (x_1, x_2, \ldots, x_n)$ en el espacio vectorial real $n$–dimensional $\mathbb{R}^n$ es usualmente dada por la norma Euclidiana:

$$\|x\|_2 = \left(x_1^2 + x_2^2 + \cdots + x_n^2\right)^{1/2}$$

La distancia Euclidiana entre dos puntos $x$ y $y$ es la longitud $\|x - y\|_2$ de la línea recta entre los dos puntos. En muchas situaciones, la distancia Euclidiana es insuficiente para capturar la distancia real en un espacio dado. Una analogía a esto es sugerida por un conductor de taxi en un conjunto de calles con forma de cuadrícula. Este mediría las distancias no en términos de la longitud de la línea recta entre los dos puntos, más bien en términos de las distancias rectilineares (las componentes de la recta), lo que tiene en cuenta que las calles son perpendiculares o paralelas unas con otras. Las clase de norma-$p$ generaliza ambos ejemplos de la siguiente manera:

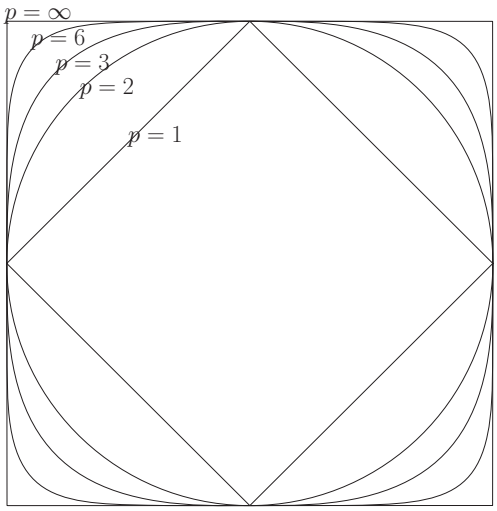
**Definición 2.1.1.** Para un número real $p \geq 1$, la norma-$p$ o norma-$L^p$ de $x$ está definida por:

$$\|x\|_p = (|x_1|^p + |x_2|^p + \cdots + |x_n|^p)^{1/p} \tag{2.1}$$

Las barras de valor absoluto son innecesarias cuando $p$ es un número racional y, en su forma reducida, tiene un numerador par. De lo anterior, la norma Euclidiana se encuentra en esta clase y es la norma-2, y la norma-1 corresponde con la distancia rectilineal (*Manhattan distance*) La Figura 2.1 muestra cómo varía la forma de una circunferencia dependiendo de la norma que se utiliza para la distancia.

La norma $L^\infty$ o norma máxima (o norma uniforme) es el límite de las normas $L^p$ cuando $p \to \infty$. Este límite es equivalente a la siguiente definición:

Figura 2.1: Circunferencia definida en los espacios $L^p$ con $p = 1, 2, 3, 6$ y $\infty$.




***Fuente:*** *Elaboración propia.*

$$\|x\|_\infty = \lim_{p\to\infty} \|x\|_p = \max\{|x_1|, |x_2|, \ldots, |x_n|\} \tag{2.2}$$

Para todos los $p \geq 1$, las normas-$L^p$ y $L^\infty$ satisfacen las propiedades de una "función de longitud" (o norma), que son:

- Solo el vector cero tiene longitud cero,
- La longitud de un vector es positiva y homogénea con respecto a la multiplicación por un escalar (homogeneidad positiva), y
- La longitud de la suma de dos vectores no es mayor a la suma de longitudes de los vectores (desigualdad triangular).

La distancia rectilineal entre dos puntos nunca es menor que la distancia del segmento entre ellos. Formalmente, esto significa que la norma Euclidiana de cualquier vector está acotada por su norma-1:

$$\|x\|_2 \leq \|x\|_1$$

Esta propiedad se generaliza a las normas-$p$ en que la norma-$p$ $\|x\|_p$ de cualquier vector $x$ no crece con $p$:

$$\|x\|_{p+a} \leq \|x\|_p \text{ para cualquier vector } x \text{ y números reales } p \geq 1 \text{ y } a \geq 0$$

En la dirección opuesta, se tiene una relación entre la norma-1 y la norma-2:

$$\|x\|_1 \leq \sqrt{n}\|x\|_2$$

Esta desigualdad depende de la dimensión $n$ del espacio vectorial subyacente y es consecuencia directa de la desigualdad de Cauchy-Schwarz.

## 2.2. Problema de Optimización multiobjetivo

La presente investigación se modelará como un problema de optimización multiobjetivo. A continuación, se define los conceptos fundamentales útiles para el caso.

En su versión más simple, un problema de optimización multiobjetivo requiere encontrar un vector variable $x$ en el dominio $\mathcal{X}$ que optimice el vector objetivo $f(x)$ (Du & Swamy, 2016).

**Definición 2.2.1** (Problema de optimización multiobjetivo)**.** Un problema de optimización multiobjetivo consiste en optimizar un sistema con $k$ objetivos en conflicto.

$$\text{mín } f(x) = (f_1(x), f_2(x), \ldots, f_k(x))^T, x \in \mathcal{X} \tag{2.3}$$

sujeto a:

$$g_i(x) \leq 0, i = 1, 2, \ldots, m, \tag{2.4}$$

$$h_i(x) = 0, i = 1, 2, \ldots, p, \tag{2.5}$$

donde $x = (x_1, x_2, \ldots, x_n)^T \in \mathbb{R}^n$, las funciones objetivo $f_i \colon \mathbb{R}^n \to \mathbb{R}, i = 1, \ldots, k$, y $g_i, h_j \colon \mathbb{R}^n \to \mathbb{R}, i = 1, \ldots, m, j = 1, \ldots, p$ son las funciones de condición del problema.

Los objetivos en conflicto son el caso en el que mejorar la solución de algún objetivo tiende a simultáneamente empeorar la solución de otro. La solución a un problema de optimización multiobjetivo no es una única solución óptima, si no un conjunto de soluciones representando las mejores propuestas de entre los objetivos.

Para optimizar un sistema con funciones objetivo en conflicto, se suele tomar la suma ponderada de estas funciones como función global del sistema

$$F(x) = \sum_{i=1}^{k} w_i \overline{f_i}(x), \tag{2.6}$$

donde $\overline{f_i}(x) = \dfrac{f_i(x)}{|\max(f_i(x))|}$ son objetivos normalizados, y $\displaystyle\sum_{i=1}^{k} w_i = 1$.

El criterio de Pareto es un método muy popular para la optimización multiobjetivo. Está basado en el principio de la no dominancia. El óptimo de Pareto da un conjunto de soluciones para el cual no hay forma de mejorar una solución sin empeorar otra. En los problemas de optimización multiobjetivo, el concepto de no dominancia provee un significado bajo el cual múltiples soluciones pueden ser comparadas y subsecuentemente ordenadas. A continuación, se define estos conceptos formalmente.

**Definición 2.2.2** (Dominancia de Pareto)**.** Se dice que un vector variable $x_1 \in \mathbb{R}^n$ domina a otro vector $x_2 \in \mathbb{R}^n$, denotado por $x1 \succ x2$, si y solo si $x_1$ es mejor o igual que $x_2$ en todos sus atributos, y estrictamente mejor en al menos un atributo, es decir, $\forall i : f_i(x_1) \geq f_i(x_2)$ y existe un $j$ tal que $f_j(x_1) > f_j(x_2)$.

Para dos soluciones $x_1, x_2$, si $x_1$ es mejor en todos los objetivos que $x_2$, se dice que $x_1$ domina fuertemente a $x_2$. Si $x_1$ no es peor que $x_2$ en todos los objetivos y mejor en al menos un objetivo, se dice que $x_1$ domina a $x_2$. Un conjunto no–dominado de soluciones es un conjunto de soluciones que no son débilmente dominadas por otra solución en el conjunto.

**Definición 2.2.3** (No dominancia)**.** Un vector variable $x_1 \in \mathcal{X} \subset \mathbb{R}^n$ es no–dominado con respecto a $\mathcal{X}$, si no existe otro vector $x_2 \in \mathcal{X}$ tal que $x_2 \prec x_1$.

**Definición 2.2.4** (Optimalidad de Pareto)**.** Un vector variable $x^* \in \mathcal{F} \subset \mathbb{R}^n$ ($\mathcal{F}$ es la región factible) es Pareto–óptimo si es no–dominado con respecto a $\mathcal{F}$.

**Definición 2.2.5** (Frontera óptima de Pareto)**.** La frontera óptima de pareto $\mathcal{P}^*$ es definida por el espacio en $\mathbb{R}^n$ formado por todas las soluciones Pareto–óptimas $\mathcal{P}^* = \{x \in \mathcal{F} \mid x \text{ es Pareto–óptimo}\}$.

La frontera óptima de Pareto es un conjunto de soluciones óptimas no–dominadas, pudiendo ser infinito.

**Definición 2.2.6** (Frontera de Pareto)**.** La frontera de Pareto $\mathcal{PF}^*$ es definida por

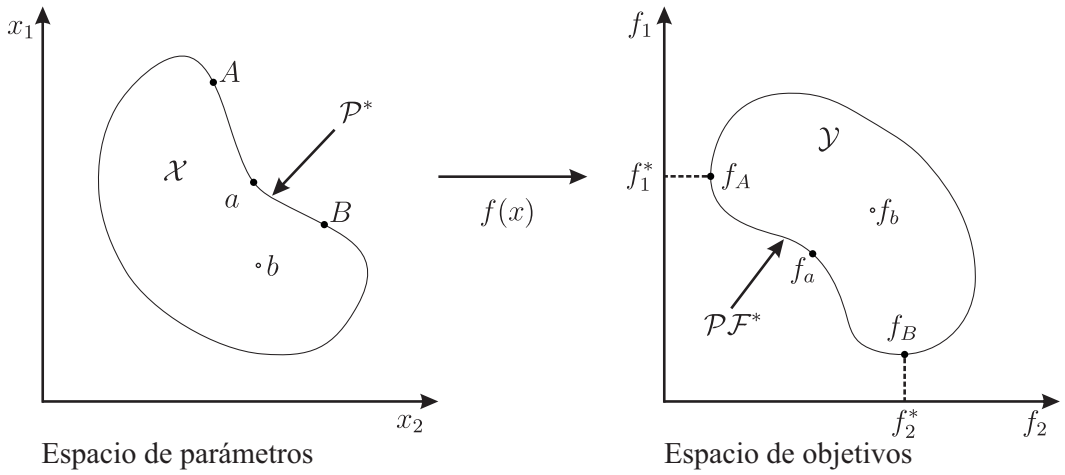
$$\mathcal{PF}^* = \{f(x) \in \mathbb{R}^k \mid x \in \mathcal{P}^*\} \tag{2.7}$$

La frontera de Pareto es el conjunto imagen de la frontera óptima de Pareto mapeada en el espacio objetivo.

El obtener la frontera de Pareto de un problema de optimización multiobjetivo es el objetivo principal de la optimización multiobjetivo. Una buena solución debe contener un número limitado de puntos, que deben estar lo más cerca posible a la frontera exacta de Pareto, como también deben estar uniformemente esparcidos de tal forma que no se dejen regiones sin explorar.

La Figura 2.2 ilustra las soluciones óptimas de Pareto para un problema en dos dimensiones con dos objetivos. El borde superior de los puntos desde $A$ hasta $B$ del dominio $\mathcal{X}$, denotado $\mathcal{P}^*$, contiene todas las soluciones Pareto–óptimas. La frontera de los puntos desde $f_A$ a $f_B$ a través del borde inferior del dominio $\mathcal{Y}$, denotado $\mathcal{PF}^*$, contiene toda la frontera de Pareto en el espacio objetivo. Para dos puntos $a$ y $b$, sus mapeos $f_a$ dominan a $f_b$, denotado $f_a \prec f_b$. Por lo tanto, el vector de decisión $x_a$ es una solución no–dominada. Las fronteras de Pareto pueden ser convexas, cóncavas o discontínuas.

Figura 2.2: Ilustración de soluciones óptimas de Pareto para un problema en dos dimensiones con dos objetivos. $\mathcal{X} \subset \mathbb{R}^n$ es el dominio de $x$, y $\mathcal{Y} \subset \mathbb{R}^m$ es el dominio de $f(x)$.




***Fuente:*** *Elaboración propia.*

**Definición 2.2.7** ($\varepsilon$–dominancia)**.** Se dice que un vector variable $x_1 \in \mathbb{R}^n$ $\varepsilon$–domina a otro vector $x_2 \in \mathbb{R}^n$, denotado $x_1 \succ_\varepsilon x_2$, si y solo si $x_1$ es mejor o igual a $\varepsilon x_2$ en todos los atributos, y estrictamente mejor en al menos un atributo, es decir, $\forall i\colon f_i(x_1) \geq f_i(\varepsilon x_2)$ y existe un $j$ tal que $f_j(x_1) > f_j(\varepsilon x_2)$ (Zitzler, Thiele, Laumanns, Fonseca & da Fonseca, 2003).

Si $\varepsilon = 1$, la $\varepsilon$–dominancia es la misma que Pareto dominancia; de otra forma, el área dominada por $x_i$ es alargada o encogida. Entonces, $\varepsilon$–dominancia relaja el área de la dominancia de Pareto por un factor de $\varepsilon$.

## 2.3. Algoritmo *Social Spider Optimization*

El comportamiento inteligente colectivo de insectos o grupos de animales en la naturaleza como bandadas de aves, colonias de hormigas, cardúmenes de peces, etc. han atraído la atención de los investigadores para resolver problemas complejos de la vida real. Esta rama de la inteligencia artificial que lidia con el comportamiento colectivo de colonias a través de interacciones complejas de los individuos es frecuentemente llamada *swarm intelligence*.

Muchos algoritmos han sido desarrollados como una combinación de reglas determinísticas y aleatoriedad, imitando el comportamiento de insectos o grupos de animales en la naturaleza. Tales métodos incluyen el comportamiento social de aves y peces en el algoritmo *Particle Swarm Optimization* (PSO), el comportamiento de enjambres de abejas en el algoritmo *Artificial Bee Colony* (ABC), el comportamiento de las bacterias en *Bacterial Foraging Optimization Algorithm* (BFOA), entre otros. Un algoritmo reciente de *swarm intelligence* basado en el comportamiento de las arañas para tareas de optimización es el algoritmo *Social Spider Optimization* (SSO) (Cuevas et al., 2016). Este algoritmo demuestra tener un rendimiento a la altura de algoritmos en el estado del arte de *swarm intelligence* para optimización global.

El algoritmo considera dos tipos diferentes de agentes (arañas sociales): machos y hembras. Dependiendo del género, cada individuo se comporta mediante un conjunto de operadores que imitan diferentes comportamientos cooperativos típicos en una colonia. Una característica interesante es el alto porcentaje de arañas hembra en la colonia (alrededor del 70 %). Todas las arañas de la colonia mantienen interacciones con las demás, estas interacciones pueden darse indirectamente por medio de la telaraña o directamente como en el apareamiento. La información transmitida por la telaraña es codificada a través de vibraciones. La intensidad de una vibración depende del peso y la distancia de la araña que la produjo. Los comportamientos sociales de las arañas se clasifican en interacciones sociales y apareamiento. Las arañas hembra presentan un instinto de atracción o repulsión sobre otras arañas, independientemente de su género. Para estas arañas, tal instinto se desarrolla comúnmente acorde a las vibraciones emitidas a través de la telaraña. La decisión de atracción o repulsión se determina de acuerdo a un proceso aleatorio interno de la araña. El comportamiento de las arañas macho está orientado a la reproducción. Estas arañas se clasifican en dos grupos: dominantes y no dominantes. Las arañas macho dominantes tienen un peso más grande en comparación con las arañas macho no dominantes. En un escenario típico, las arañas dominantes son atraídas a las arañas hembra más cercanas. En contraste, las arañas no dominantes tienen a concentrarse en el centro de la población de machos como una estrategia para obtener los recursos desperdiciados por las arañas dominantes. El apareamiento se da entre arañas macho dominantes y las hembras, cuando una araña dominante encuentra una o más arañas hembra dentro de un radio específico, esta se aparea con todas las arañas dentro del rango para producir crías.

El algoritmo que simula este comportamiento de arañas sociales es el siguiente:

1. Considerando $N$ como el número total de miembros $n$ dimensionales, el número de arañas macho $N_m$ y hembras $N_f$ en toda la población S:

$$N_f = \lfloor (0.9 - U[0,1) \cdot 0.25) \cdot N \rfloor \text{ y } N_m = N - N_f$$

2. Inicializar aleatoriamente los miembros hembra (F = $\{f_1, f_2, \ldots, f_{N_f}\}$) y macho (M = $\{m_1, m_2, \ldots, m_{N_m}\}$) donde (S = $\{\mathrm{s}_1 = f_1, \mathrm{s}_2 = f_2, \ldots, \mathrm{s}_{N_f} = f_{N_f}, \mathrm{s}_{N_f+1} = m_1, \mathrm{s}_{N_f+2} = m_2, \ldots, \mathrm{s}_N = m_{N_m}\}$) y calcular el radio de apareamiento:

$$r = \frac{\sum_{j=1}^{n} \left(p_j^{\text{máx}} - p_j^{\text{mín}}\right)}{2 \cdot n}$$

$$f_{ij}^0 = p_j^{\text{mín}} + U[0,1) \cdot \left(p_j^{\text{máx}} - p_j^{\text{mín}}\right)$$

$$m_{kj}^0 = p_j^{\text{mín}} + U[0,1) \cdot \left(p_j^{\text{máx}} - p_j^{\text{mín}}\right)$$

3. Calcular el peso de cada araña de S (asumiendo un problema de maximización con función objetivo $J$).

$$w_i = \frac{J(\mathrm{s}_i) - peor_{\mathrm{s}}}{mejor_{\mathrm{s}} - peor_{\mathrm{s}}}$$

$$\text{donde } mejor_{\mathrm{s}} = \max_{k \in \{1,2,\ldots,N\}} J(\mathrm{s}_k) \text{ y } peor_{\mathrm{s}} = \min_{k \in \{1,2,\ldots,N\}} J(\mathrm{s}_k)$$

4. Mover las arañas hembra de acuerdo al operador cooperativo de la hembra. Calcular $Vibc_i$, la vibración captada por el individuo $i$ transmitida por el individuo $c$, el cual es el miembro más cercano con mayor peso que $i$:

$$Vibc_i = w_c \cdot e^{-d_{i,c}^2}$$

Calcular $Vibb_i$, la vibración captada por el individuo $i$ transmitida por el individuo $b$, el cual es el miembro con mayor peso en toda la colonia.

$$Vibb_i = w_b \cdot e^{-d_{i,b}^2}$$

$$f_i^{k+1} = f_i^k + \alpha \cdot Vibc_i \cdot \left(\mathrm{s}_c - f_i^k\right) + \beta \cdot Vibb_i \cdot \left(\mathrm{s}_b - f_i^k\right) + \delta \text{ con probabilidad } PF$$

$$f_i^{k+1} = f_i^k - \alpha \cdot Vibc_i \cdot \left(\mathrm{s}_c - f_i^k\right) - \beta \cdot Vibb_i \cdot \left(\mathrm{s}_b - f_i^k\right) + \delta \text{ con probabilidad } 1 - PF$$

5. Mover las arañas macho de acuerdo al operador cooperativo del macho. Calcular $Vibf_i$, la vibración captada por el individuo $i$ transmitida por el individuo $c$, el cual es la araña hembra más cercana a $i$.

$$Vibf_i = w_f \cdot e^{-d_{i,f}^2}$$

Encontrar la araña macho media ($w_{N_f+m}$) de $M$.

$$m_i^{k+1} = m_i^k + \alpha \cdot Vibf_i \cdot \left(\mathrm{s}_f - m_i^k\right) + \delta \cdot \left(U[0,1) - \frac{1}{2}\right) \text{ si es dominante}$$

$$m_i^{k+1} = m_i^k + \alpha \cdot \left(\frac{\sum_{h=1}^{N_m} m_h^k \cdot w_{N_f+h}}{\sum_{h=1}^{N_m} w_{N_f+h}} - m_i^k\right) \text{ si no es dominante}$$

6. Para las arañas dominantes, efectuar el apareamiento. Encontrar a las hembras más cercanas y formar una cría con cada una, si el peso de la criatura resultante es mejor que el peso del menos pesado de la colonia, reemplazarlo.

7. Si el criterio de finalización es alcanzado, el proceso es finalizado; de otro modo, volver al paso 3.

Los detalles técnicos del algoritmo pueden ser encontrados en (Cuevas et al., 2016). En esta investigación, se asume que el algoritmo *Social Spider Optimization* es invocado con los siguientes parámetros, asumiendo un problema de minimización:

- $J$, una función objetivo real de variable vectorial.
- s, el conjunto inicial de arañas, si alguna de estas es el vector cero, se le asignará un valor aleatorio de la forma descrita anteriormente.
- $n_s$, el número total de arañas de la colonia ($N$).
- $n_{it}$, el número de iteraciones a efectuar por el algoritmo. Este es el criterio de finalización.

El valor devuelto por el algoritmo es único: $x$, la posición de la araña con mejor peso en la colonia. Este vector representa el parámetro solución del problema.

## 2.4. P, NP, NP-Hard y NP-Completo

Muchos algoritmos de búsqueda y ordenamiento en arreglos son algoritmos de complejidad temporal polinomial. Esto es, con un tamaño de entrada $n$, su peor tiempo de ejecución es $O(n^k)$ para alguna constante $k$. Una pregunta natural concerniente a la complejidad temporal de los algoritmos es si todos los problemas pueden ser resueltos en tiempo polinomial. La respuesta es no. Por ejemplo, existen problemas como el *Halting Problem* formulado por Alan Turing, el cual no puede ser resuelto por ninguna computadora, sin importar el tiempo que se le de para su ejecución. Existen también problemas que pueden ser resueltos, pero no en tiempo $O(n^k)$ para alguna constante $k$. Generalmente, se denomina a los problemas que pueden ser resueltos por un algoritmo de complejidad temporal polinomial como tratables, o problemas fáciles, y a los problemas que requieren un tiempo superpolinomial como intratables, o difíciles. Los algoritmos existentes para solucionar estos últimos problemas tienen una cota inferior en el tiempo de ejecución, la cual es $\Omega(k^n)$, donde $k > 1$ es una constante y $n$ es el tamaño de la entrada.

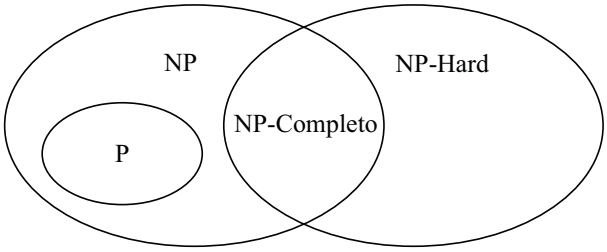
La clase P (complejidad temporal polinomial) consiste en aquellos problemas que pueden ser resueltos en tiempo polinomial. Más específicamente, son problemas que pueden ser resueltos en tiempo $O(n^k)$ para alguna constante $k$, donde $n$ es el tamaño de la entrada al problema. Esta clase está definida como el conjunto de problemas de decisión que pueden ser resueltos por una máquina de Turing determinística utilizando un algoritmo de complejidad temporal polinomial para el peor caso.

La clase NP (complejidad temporal polinomial no determinística) consiste en aquellos problemas que son "verificables" en tiempo polinomial. Está definida como el conjunto de problemas de decisión que pueden ser verificados por una máquina de Turing no determinística utilizando un algoritmo no determinístico de complejidad temporal polinomial para el peor caso. A pesar de que los algoritmos no determinísticos no pueden ser ejecutados directamente en computadoras convencionales, este concepto es importante y de ayuda para el análisis de la complejidad computacional de los problemas. Todos los problemas en P también pertenecen a NP, esto es, $P \subseteq NP$.

Informalmente, la clase NP-Hard es el conjunto de los problemas que son al menos tan difíciles como el problema más difícil de NP. Estos problemas no necesariamente deben pertenecer a NP.

El conjunto de problemas NP-Completo es un subconjunto de NP (Cook, 1971). Se dice que un problema de decisión $A$ es NP-Completo, si $A$ está en NP y $A$ también está en NP-Hard. Los problemas NP-Completos son los problemas más difíciles de NP. Todos ellos tienen la misma complejidad. Estos son difíciles ya que no se conocen algoritmos de tiempo polinomial que los resuelvan. La relación entre todas estas clases está ilustrada en la Figura 2.3, suponiendo que P $\neq$ NP.

Figura 2.3: Relación más probable entre las clases P, NP, NP-completo y NP-hard.




***Fuente:*** *Elaboración propia.*

Los problemas de optimización bajo ciertas condiciones, son todos NP-Completos o NP-Hard. Al momento, ningún algoritmo con complejidad temporal polinomial puede garantizar que una solución óptima será encontrada. El tratamiento formal de las clases de complejidad mencionadas requiere un entorno en los Lenguajes Formales (Cormen et al., 2009).

## 2.5. *K*-means

$K$-means es una técnica de *center based clustering* que busca particionar un conjunto de $n$ elementos en $k$ conjuntos en los cuales cada punto pertenece al *cluster* con el promedio más cercano. Esto resulta en un particionamiento del espacio de datos en un Diagrama de Voronoi.

El problema es computacionalmente difícil, de cualquier forma, existen algoritmos heurísticos que son comúnmente utilizados y que convergen rápidamente a un óptimo local. El algoritmo está relacionado al clasificador *K-Nearest-Neighbors*, una técnica popular del aprendizaje automático para la clasificación que es comúnmente confundida con el algoritmo $k$-means.

El algoritmo más común utiliza una técnica de refinamiento de la solución. Dada su popularidad, es comúnmente llamado el *algoritmo k-means*; también es conocido como el algoritmo de Lloyd, particularmente en la comunidad de ciencias de la computación.

Dado un conjunto inicial de $k$ promedios $m_1^{(1)}, m_2^{(1)}, \ldots, m_k^{(1)}$, el algoritmo procede por alternar los siguientes pasos:

- **Paso de asignación:** Asignar cada observación al *cluster* cuyo promedio tiene la menor distancia Euclidiana al cuadrado, esto es intuitivamente el promedio más cercano (Matemáticamente, esto significa particionar las observaciones de acuerdo con el Diagrama de Voronoi generado por los promedios).

$$S_i^{(t)} = \{x_p : \|x_p - m_i^{(t)}\|^2 \leq \|x_p - m_j^{(t)}\|^2 \forall j, 1 \leq j \leq k\},$$

  donde cada $x_p$ es asignado exactamente a un $S^{(t)}$, incluso si este pudiera ser asignado a dos o más de ellos.

- **Paso de actualización:** Calcular los nuevos promedios como los centroides de las observaciones en los nuevos *clusters*.

$$m_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{x_j \in S_i^{(t)}} x_j$$

El algoritmo es normalmente presentado como una asignación de los objetos al *cluster* más cercano (en distancia). Al utilizar una función diferente al cuadrado de la distancia Euclidiana, el algoritmo puede no converger. Se han presentado multitud de modificaciones al algoritmo $k$-means como el algoritmo $k$-means esférico y $k$-medoides, estos utilizando otras medidas de distancia.

## 2.6. Mutual Information

Una métrica bastante utilizada para la evaluación de un *clustering* es la llamada *Mutual Information* (Vinh, Epps & Bailey, 2009).

Dado un conjunto $S$ con $N$ elementos $S = \{s_1, s_2, \ldots, s_N\}$, considérese dos particiones de $S$, $U = \{U_1, U_2, \ldots, U_R\}$ con $R$ *clusters*, y $V = \{V_1, V_2, \ldots, V_C\}$ con $C$ *clusters*. Supóngase que un objeto es escogido aleatoriamente de $S$, la probabilidad de que ese objeto esté en el *cluster* $U_i$ (y similarmente para cada elemento de $V$) es:

$$P(i) = \frac{|U_i|}{N}$$

.

La entropía asociada a la partición $U$ (y similarmente a $V$) es:

$$H(U) = -\sum_{i=1}^{R} P(i) \log P(i)$$

.

Luego, la métrica *Mutual Information* (MI) entre las dos particiones:

$$\mathrm{MI}(U, V) = \sum_{i=1}^{R} \sum_{j=1}^{C} P(i, j) \log \frac{P(i, j)}{P(i)P'(j)}, \tag{2.8}$$

donde $P(i, j)$ denota la probabilidad de que un punto pertenezca a ambos *clusters* $U_i$ en $U$ y al *cluster* $V_j$ en $V$. El valor base de *Mutual Information* entre dos *clusterings*, no está basado en un valor constante, y tiende a ser más grande cuando las dos particiones tienen un número mayor de *clusters* (con un número fijo de elementos en $N$). Esto significa que el valor de *Mutual Information* y *Normalized Mutual Information* no está ajustado para coincidencias por casualidad y crecerá con el número de etiquetas (clusters) sin importar el valor real de "información mutua" entre las asignaciones. La métrica *Adjusted Mutual Information* está definida como:

$$\mathrm{AMI}(U, V) = \frac{\mathrm{MI}(U, V) - E[\mathrm{MI}(U, V)]}{\mathrm{máx}(H(U), H(V)) - E[\mathrm{MI}(U, V)]}, \tag{2.9}$$

donde $E[\mathrm{MI}(U, V)]$ es el valor esperado de $\mathrm{MI}(U, V)$. AMI toma el valor de 1 cuando las dos particiones son idénticas y 0 cuando el valor de *Mutual Information* entre las dos particiones es igual al valor esperado solamente por coincidencia. También es posible que esta métrica tome un valor negativo bajo ciertas circunstancias.

# Capítulo 3

# Desarrollo del proyecto

## 3.1. Función objetivo

Para solucionar el problema del *center-based clustering*, es necesario diseñar la función objetivo del problema. El diseño de esta función comprende dos problemas dentro de center-based clustering que buscan encontrar soluciones que se ajusten a ciertos criterios bien definidos. A continuación, se define el primero de estos problemas.

**Problema 1** (El problema $k$-means)**.** Sea $\mathcal{S}$ un conjunto de $n$ puntos $\{x_1, x_2, \ldots, x_n\}$ en un espacio métrico $(\mathbb{R}^d, L^2)$, y sea $k > 1$ un número entero. Encontrar un conjunto $C = \{c_1, c_2, \ldots, c_k\}$ de $k$ elementos y una matriz $A = [a_{ij}]$ de tamaño $n \times k$ de tal forma que se minimice:

$$J_1(C, A) = \sum_{i=1}^{n} \sum_{j=1}^{k} a_{ij} \|x_i - c_j\|_2^2 \tag{3.1}$$

sujeto a:

$$c_j \in \mathbb{R}^d \text{ para todo } j \tag{3.2}$$

$$a_{ij} \in \{0, 1\} \text{ para todos } i, j \tag{3.3}$$

$$\sum_{j=1}^{k} a_{ij} = 1 \text{ para todo } i. \tag{3.4}$$

La primera entrada del Problema $k$-means es un conjunto $\mathcal{S}$ de $n$ puntos. Este es el conjunto de puntos sobre el que se pretende hacer clustering. Cada uno de los puntos de $\mathcal{S}$ debe pertenecer a $\mathbb{R}^d$, es decir, debe ser un vector con $d$ componentes. Un ejemplo de un conjunto $\mathcal{S}$ en $\mathbb{R}^2$ es $\{(-2, -1), (1, 2.5), (2, 1), (-1, 2), (1, 0)\}$. En este caso, se tienen puntos en dos dimensiones, de la misma forma, se puede formar conjuntos con puntos en cualquier dimensión $d$[1]. El segundo valor que recibe el problema es un número entero $k$. Este número representa la cantidad de clusters que se pretende formar. Para fines prácticos, se considera que $k$ debe ser mayor a 1 y menor a la cantidad de puntos en $\mathcal{S}$ ya que formar un único cluster con todos los puntos, o $n$ clusters asignando a cada cluster un único punto es trivial y no aporta conocimiento significativo.

Para evaluar la concentración de los clusters se define una función $J_1$ que mide la calidad del clustering considerando los parámetros $C$ y $A$ que representan cómo está efectuado el clustering sobre el conjunto $\mathcal{S}$.

[1]Todos los puntos de $\mathcal{S}$ deben tener la misma dimensión

El primer parámetro de la función $J_1$ es un conjunto $C$ con la misma cantidad de puntos como número de clusters que se quiere formar. La condición (3.2) indica que todos los puntos de $C$ deben pertenecer a $\mathbb{R}^d$, es decir, deben tener dimensión $d$. Cada punto del conjunto $C$ representará a un cluster como su "centro". Por lo tanto, los elementos del conjunto $C$ son llamados *centros*.

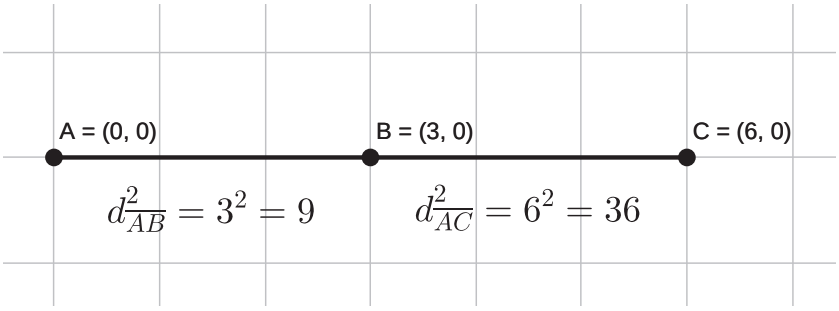
El segundo parámetro de la función $J_1$ es una matriz $A$ de tamaño $n \times k$. La condición (3.3) indica que los elementos de esta matriz deben ser unos o ceros. El propósito de esta matriz es asignar puntos de $\mathcal{S}$ a puntos de $C$. Se dice que el punto $x_i$ está asignado al centro $c_j$ si y solo si $a_{ij} = 1$. La condición (3.4) indica que cada punto del conjunto $\mathcal{S}$ debe ser asignado a exactamente un punto del conjunto $C$. Es decir, cada punto de $\mathcal{S}$ debe pertenecer a un único cluster.

Con el conjunto $C$ y la matriz $A$, se toma las distancias Euclidianas de cada punto al elemento de $C$ al cual fue asignado. El valor devuelto por la función $J_1$ es la suma de los cuadrados de tales distancias. El objetivo de este problema es encontrar el conjunto $C$ y la matriz $A$ que hagan que el valor devuelto por $J_1$ sea el mínimo posible. El Teorema 3.1.1 indica que el problema es computacionalmente difícil.

***Teorema* 3.1.1** (Mahajan, Nimbhorkar y Varadarajan, 2012)**.** *El problema k-means es NP-Hard.*

Una observación clave para el diseño de la función objetivo es que la función $J_1$ toma una suma sobre el cuadrado de distancias Euclidianas entre dos puntos. Esto hace que las distancias más grandes afecten más en proporción al resultado que las distancias pequeñas. La "penalización" que recibe un punto crece con el cuadrado de su distancia a un punto fijo. Este hecho es ilustrado en la Figura 3.1 donde se tiene tres puntos colineales con el punto $A$ como punto fijo. A pesar de que el punto $C$ está al doble de distancia de $A$ que el punto $B$, este recibe una penalización mucho mayor que el punto $B$.

Figura 3.1: La penalización crece con el cuadrado de la distancia.

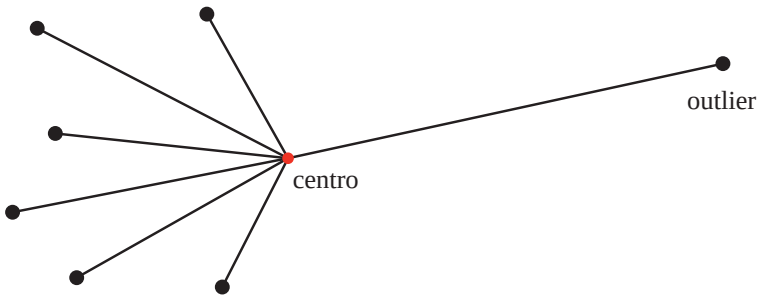



***Fuente:*** *Elaboración propia.*

Este comportamiento cobra mayor importancia cuando el conjunto $\mathcal{S}$ contiene multitud de outliers, pues generalmente un outlier se encuentra anormalmente alejado de los puntos "típicos". Si a este punto se le asigna un elemento $c_j$ de $C$ que se encuentra entre muchos puntos típicos, este causará que su penalización sea grande, haciendo que la posición de $c_j$ cambie para reducir la penalización. Este ejemplo se ilustra en la Figura 3.2 en la que se puede ver que un punto alejado genera el reajuste de un punto de $C$ para la función $J_1$ decrezca.

Una mejora a este comportamiento para incrementar la robustez frente a los outliers es hacer que la penalización para estos puntos crezca proporcionalmente a la distancia Euclidiana a su punto asignado. De esta forma se define el segundo problema necesario para el diseño de la función objetivo.

Figura 3.2: Un outlier genera la necesidad de reajuste de un centro.




***Fuente:*** *León Malpartida et al. (2019).*

**Problema 2** (El problema Emax)**.** Sea $\mathcal{S}$ un conjunto de $n$ puntos $\{x_1, x_2, \ldots, x_n\}$ en un espacio métrico $(\mathbb{R}^d, L^2)$, y sea $k > 1$ un número entero. Encontrar un conjunto $C = \{c_1, c_2, \ldots, c_k\}$ de $k$ elementos y una matriz $A = [a_{ij}]$ de tamaño $n \times k$ de tal forma que se minimice:

$$J_2(C, A) = \sum_{i=1}^{n} \sum_{j=1}^{k} a_{ij} \|x_i - c_j\|_2 \tag{3.5}$$

sujeto a:

$$c_j \in \mathbb{R}^d \text{ para todo } j \tag{3.6}$$

$$a_{ij} \in \{0, 1\} \text{ para todos } i, j \tag{3.7}$$

$$\sum_{j=1}^{k} a_{ij} = 1 \text{ para todo } i. \tag{3.8}$$

Como se puede observar, la función $J_2$ del Problema Emax toma una suma sobre las *distancias Euclidianas* de cada punto de $\mathcal{S}$ a un punto de $C$. Las condiciones (3.6), (3.7) y (3.8) son idénticas a las del Problema 1.

Para el diseño de la función objetivo se tomarán las funciones (3.1) y (3.5) con el fin de minimizar ambas funciones de costo simultáneamente. Para optimizar un sistema con funciones en conflicto, se suele tomar la suma ponderada de las funciones como representación (Du & Swamy, 2016, p. 17). De esta forma, la función objetivo que se tomará para la optimización multiobjetivo es una combinación lineal de $J_1$ y $J_2$, es decir:

$$\alpha J_1 + \beta J_2,$$

donde $\alpha + \beta = 1$ y las condiciones (3.6), (3.7) y (3.8) se mantienen para $C$ y $A$. El objetivo, nuevamente, es encontrar el conjunto de centros $C$ y la matriz $A$ que minimizan el objetivo.

## 3.2. Búsqueda local en $\mathcal{F}$

Se efectuará la minimización de la función $\mathcal{F}$ utilizando un algoritmo de Optimización Global de *Swarm Intelligence* llamado *Social Spider Optimization*. Este algoritmo simula arañas artificiales que se mueven a través del espacio de búsqueda. Es deseable que la posición inicial de las arañas cubra una gran parte del espacio de búsqueda aleatoriamente con una distribución uniforme. Dependiendo de la función objetivo, algunas arañas tienen mayor probabilidad que otras de llegar a óptimos locales. Es posible que el inicializar el valor de pocas arañas cerca de óptimos locales de la función $\mathcal{F}$, ayude a mejorar la convergencia del algoritmo. Para hacer esto, se asumirá que el conjunto $\mathcal{S}$ posee una propiedad natural en su estructura para resolver un problema útil en clustering. Esta propiedad es llamada *Perturbation Resilience*. Para definir esta propiedad, es necesario definir primero el problema $k$-center.

### 3.2.1. Problema $k$-center

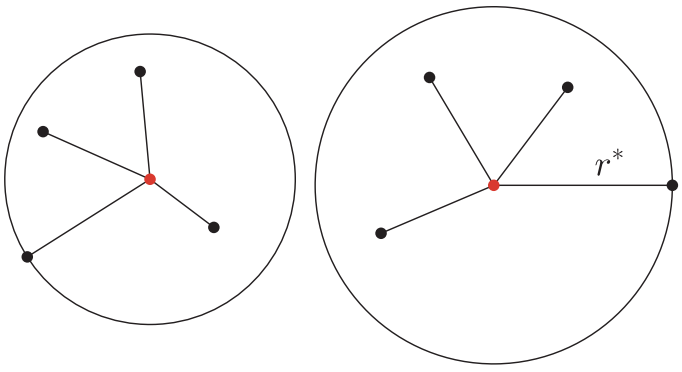
Similar a los Problemas $k$-means y Emax, el objetivo del problema $k$-center es localizar $k$ centros de tal forma que se minimice una función de costo. Esto es definido a continuación.

**Problema 3** (El problema $k$-center)**.** Sea $\mathcal{S}$ un conjunto de $n$ puntos $\{x_1, x_2, \ldots, x_n\}$ en un espacio métrico $(\mathbb{R}^d, L^2)$, y sea $k > 1$ un número entero. Encontrar un conjunto $C = \{c_1, c_2, \ldots, c_k\} \subseteq \mathcal{S}$ de $k$ elementos de $\mathcal{S}$ de tal forma que se minimice:

$$J_3(C) = \max_{x \in \mathcal{S}} \left( \min_{c \in C} \|x - c\|_2 \right). \tag{3.9}$$

La función objetivo del problema $k$-center depende únicamente del conjunto de centros. El objetivo es seleccionar de forma óptima $k$ elementos de $\mathcal{S}$[2] de tal forma que se minimice la máxima distancia de un punto de $\mathcal{S}$ a su centro más cercano, se denota $r^*$ a este valor (ver la Figura 3.3).

Figura 3.3: El problema $k$-center: minimizar el máximo radio.




***Fuente:*** *Elaboración propia.*

Una observación para problema $k$-center es que cada conjunto de centros $C$ induce una partición de Voronoi $C_1, C_2, \ldots, C_k$ en $\mathcal{S}$. Se denota $\mathcal{OPT}(\mathcal{S})$ a la partición de Voronoi inducida por el conjunto $C \subseteq \mathcal{S}$ que minimiza $J_3$, es decir, el clustering óptimo.

Se han diseñado algoritmos exactos y de aproximación para el problema $k$-center (Agarwal & Procopiuc, 2002). También se han encontrado algoritmos de aproximación óptimos en factor

[2]La restricción de seleccionar centros que pertenezcan a $\mathcal{S}$ convierte al problema $k$-center en un problema de optimización combinatoria.

de aproximación y complejidad temporal. El siguiente teorema indica que no existe un algoritmo con factor de aproximación menor a 2 con complejidad temporal polinomial a menos que P = NP.

***Teorema* 3.2.1** (Gonzalez, 1985)**.** *Aproximar el problema k-center con un factor de aproximación–$(2-\varepsilon)$ para algún $\varepsilon > 0$ es NP-Completo.*

Un resultado directo del teorema anterior es el siguiente corolario.

***Corolario* 3.2.1.1.** *El problema k-center es NP-Hard.*

### 3.2.2. Perturbation Resilience

Perturbation Resilience (Bilu & Linial, 2012) es una noción de estabilidad que se asume para el conjunto $\mathcal{S}$. Esta propiedad implica que, si las distancias entre los puntos cambian por una pequeña perturbación, el clustering óptimo para el problema $k$-center se mantiene. Formalmente, $\rho$ es llamada una función de distancia $\alpha$–perturbation del espacio $L^2$, si para todo $p, q \in \mathcal{S}$,

$$\|p - q\|_2 \leq \rho(p, q) \leq \alpha\|p - q\|_2. \tag{3.10}$$

La propiedad de Perturbation Resilience es definida formalmente a continuación (M.-F. Balcan et al., 2016).

**Definición 3.2.1.** El par ordenado $(\mathcal{S}, L^2)$ satisface la propiedad $\alpha$–perturbation resilience para el problema $k$-center, si para cada $\alpha$–perturbation $\rho$ de $L^2$, el clustering óptimo del problema $k$-center bajo $\rho$ es único e igual a $\mathcal{OPT}(\mathcal{S})$.

Es importante aclarar que a pesar de que los centros óptimos puedan cambiar, la partición de Vornoi inducida por estos debe ser la misma. En (M. F. Balcan & Liang, 2012) se da un algoritmo exacto para el problema $k$-center bajo $\left(1+\sqrt{2}\right)$–perturbation resilience. Sin embargo, es posible encontrar soluciones óptimas para factores menores de perturbation resilience. El siguiente teorema es clave para encontrar dichos algoritmos.

***Teorema* 3.2.2** (M.-F. Balcan et al., 2016)**.** *Dado el par ordenado $(\mathcal{S}, L^2)$ que satisface la propiedad $\alpha$–perturbation resilience para el problema k-center y dado un conjunto $C$ de $k$ centros siendo una solución con factor de aproximación $\alpha$ para el problema k-center, esto es, para todo $p \in \mathcal{S}$; existe un $c \in C$ tal que $\|c - p\|_2 \leq \alpha r^*$. Entonces la partición de Voronoi inducida por $C$ es el clustering óptimo.*

El Teorema 3.2.2 implica que cualquier algoritmo de aproximación–$\alpha$ para el problema $k$-center encuentra la solución óptima para instancias de clustering que satisfacen la propiedad de $\alpha$–perturbation resilience. Por este motivo es posible utilizar un algoritmo de aproximación–2 para el problema $k$-center y obtener una solución óptima para conjuntos $\mathcal{S}$ cuyo clustering óptimo no cambie bajo pequeñas perturbaciones.

### 3.2.3. Algoritmo de aproximación–2

Existe un algoritmo simple con factor de aproximación 2 para el problema $k$-center (Gonzalez, 1985). Por conveniencia, se re-define la forma en que se mide la solución del problema. Se considera que un clustering es una partición de $\mathcal{S}$ en $k$ conjuntos $(C_1, C_2, \ldots, C_k)$, donde cada conjunto $C_i$ es llamado un cluster.

La función objetivo $J_3 \colon C_1, C_2, \ldots, C_k \to \mathbb{R}_{\geq 0}$ es definida como la máxima distancia (Euclidiana) entre dos puntos dentro del mismo cluster. Sea $\mathcal{OPT}(\mathcal{S})$ el menor valor de la función $J_3$ para todos los posibles clusterings de $\mathcal{S}$. El Algoritmo 1 genera un clustering con factor de aproximación 2 para el problema $k$-center.

**Algoritmo 1:** Greedy $k$-center (Gonzalez, 1985)

**Input:** $\mathcal{S} \in (\mathbb{R}^d, L^2), k$

```
1 c_1 ← u, donde u es un elemento aleatorio de S.
2 Colocar todos los puntos de S en C_1.
3 for i = 2 to k do
4 |   c_i ← u, donde u es el punto más lejano del conjunto actual de centros.
5 |   for v ∈ C_j con j < i do
6 |   |   if ||v − c_i||_2 ≤ ||v − c_j||_2 then
7 |   |   |   Mover v de C_j a C_i.
```

**Output:** $(C_1, C_2, \ldots, C_k)$

El algoritmo puede implementarse con una complejidad espacial de $O(n)$ si en cada momento se mantiene para cada punto de $\mathcal{S}$, la distancia al centro más cercano. El algoritmo tiene éxito en encontrar una solución con factor de aproximación–2 en tanto el conjunto $\mathcal{S}$ satisfaga la desigualdad triangular. Para demostrar el factor de aproximación del Algoritmo 1, los siguientes lemas serán de utilidad.

***Lema* 3.2.3.** *Si existe un conjunto $K \subseteq \mathcal{S}$ de $k+1$ puntos tal que para cada $u, v \in K$ con $u \neq v$, $\|u - v\|_2 \geq h$. Entonces $\mathcal{OPT}(\mathcal{S}) \geq h$.*

*Demostración.* Sea $C_1, C_2, \ldots, C_k$ un clustering de $\mathcal{S}$ tal que $J_3(C_1, \ldots, C_k) = \mathcal{OPT}(\mathcal{S})$. Como el conjunto $K$ contiene $k+1$ elementos que serán asignados a $k$ clusters, algún cluster $C_i$ debe contener al menos dos elementos $u, v \in K$ (por el principio del palomar). Entonces $\|u - v\|_2 \geq h \implies J_3(C_1, \ldots, C_k) \geq h \implies \mathcal{OPT}(\mathcal{S}) \geq h$. □

***Lema* 3.2.4.** *Sea $C_1, C_2, \ldots, C_k$ un clustering de $\mathcal{S}$ generado por el Algoritmo 1. Sea $u \in C_j$ un elemento cuya distancia a $c_j$ es maximal, llámese a esa distancia $h$. Entonces $J_3(C_1, \ldots, C_k) \leq 2h$.*

*Demostración.* Con el objetivo de encontrar una contradicción, supóngase que existen dos elementos $u, v \in C_i$ tal que $\|u - v\|_2 > 2h$. Por la desigualdad triangular, se tiene que

$$\begin{aligned} \|u - c_i\|_2 + \|c_i - v\|_2 &\geq \|u - v\|_2 \\ &> 2h. \end{aligned} \tag{3.11}$$

Se asume sin pérdida de generalidad que $\|u - c_i\|_2 \leq \|v - c_j\|_2$. Entonces

$$\|u - c_j\|_2 = \|v - c_j\|_2 - \varepsilon, \text{ para algún } \varepsilon \geq 0.$$

Reemplazando $\|u - c_j\|_2$ en (3.11) se tiene,

$$(\|v - c_j\|_2 - \varepsilon) + \|c_i - v\|_2 > 2h.$$

Despejando $\|v-c_j\|_2$ se llega a que $\|v-c_j\|_2 > h+\varepsilon/2$, lo cual es claramente una contradicción ya que se sabe que la máxima distancia entre dos puntos del mismo cluster es $h$. Por lo tanto $J_3(C_1, \ldots, C_k) \leq 2h$.

□

Finalmente, es posible utilizar los Lemas 3.2.3 y 3.2.4 para demostrar el factor de aproximación que alcanza el Algoritmo 1.

***Teorema* 3.2.5.** *El Algoritmo 1 es un algoritmo de aproximación–2 si la función objetivo satisface la desigualdad triangular.*

*Demostración.* Sea $C_1, \ldots, C_k$ un clustering de $\mathcal{S}$ computado por el Algoritmo 1. Sea $u \in C_j$ un elemento cuya distancia a $c_j$ es maximal, llámese a esa distancia $h$. La línea 6 del algoritmo garantiza que la distancia de $u$ a su centro asignado en todo momento solamente puede mantenerse o disminuir. Entonces $\|u - c_i\|_2 \geq h$ para todo $1 \leq i \leq k$. Dado que $u$ nunca se convirtió en el centro de un cluster, cada vez que se definió un nuevo cluster, la distancia de su centro a cualquier otro centro definido anteriormente fue al menos $h$. Esto es, $\|c_p - c_q\|_2 \geq h$ para $p \neq q$.

Entonces, existe un conjunto $\{u, c_1, c_2, \ldots, c_k\} \in \mathcal{S}$ de $k+1$ elementos en el que todos los pares de elementos tienen distancia de al menos $h$. Luego, por el Lema 3.2.3:

$$\mathcal{OPT}(\mathcal{S}) \geq h.$$

También, por el Lema 3.2.4 e introduciendo el resultado anterior:

$$\begin{aligned} J_3(C_1, \ldots, C_k) &\leq 2h \\ &\leq 2 \cdot \mathcal{OPT}(\mathcal{S}). \end{aligned}$$

□

El tiempo de ejecución del peor caso del Algoritmo 1 está dado por:

$$T(n,k,d) = \Theta(n) + \sum_{i=2}^{k} \left( \Theta(n) + \sum_{j=1}^{n} \sum_{l=1}^{k} \sum_{m=1}^{d} \Theta(1) \right) + \Theta(k) \tag{3.12}$$

$$= \Theta(n) + \Theta(n \cdot k \cdot d) + \Theta(k) \tag{3.13}$$

$$= \Theta(n \cdot k \cdot d), \tag{3.14}$$

donde $d$ es la dimensión del espacio en el que se opera.

## 3.3. Optimización multiobjetivo

### 3.3.1. Problema multiobjetivo

Una vez establecida la noción de la función objetivo, se define formalmente el problema multiobjetivo a continuación.

**Problema 4.** Sea $\mathcal{S}$ un conjunto de $n$ puntos $\{x_1, x_2, \ldots, x_n\}$ en un espacio métrico $(\mathbb{R}^d, L^2)$, y sea $k > 1$ un número entero. Encontrar un conjunto $C = \{c_1, c_2, \ldots, c_k\}$ de $k$ elementos y una matriz $A = [a_{ij}]$ de tamaño $n \times k$ de tal forma que; dados un $\alpha \geq 0$ y un $\beta \geq 0$ con $\alpha + \beta = 1$, se minimice:

$$\mathcal{F}(C, A) = \alpha \left( \sum_{i=1}^{n} \sum_{j=1}^{k} a_{ij} \|x_i - c_j\|_2^2 \right) + \beta \left( \sum_{i=1}^{n} \sum_{j=1}^{k} a_{ij} \|x_i - c_j\|_2 \right) \tag{3.15}$$

sujeto a:

$$c_j \in \mathbb{R}^d \text{ para todo } j \tag{3.16}$$

$$a_{ij} \in \{0, 1\} \text{ para todos } i, j \tag{3.17}$$

$$\sum_{j=1}^{k} a_{ij} = 1 \text{ para todo } i. \tag{3.18}$$

Como se puede ver, las condiciones de los Problemas $k$-means y Emax se mantienen en (3.16), (3.17) y (3.18).

Se pretende minimizar la función $\mathcal{F}$ definida en (3.15) con respecto al conjunto $C$ y la matriz $A$. Nótese que por la condición (3.17), la matriz $A$ debe contener valores discretos y por la condición (3.18) cada fila debe contener exactamente un uno. Este problema no es apropiado para un algoritmo de optimización global ya que el espacio de búsqueda no es continuo y por lo tanto, no es diferenciable en todo su dominio. Muchos algoritmos de optimización global requieren calcular el gradiente de la función objetivo en cada iteración y otros requieren obtener el valor de la función en puntos que podrían no estar definidos para la función $\mathcal{F}$. Es posible redefinir la función objetivo de tal forma que sea continua en todo su dominio. El siguiente teorema será de utilidad para corregir esto y será bastante utilizado durante toda la investigación:

***Teorema* 3.3.1.** *Dado el Problema 4, sea $\hat{C} = \{c_j\}$ fijado. Entonces la función $\mathcal{F}(\hat{C}, A)$ es minimizada si y solo si a cada punto de $\mathcal{S}$ se le asigna el punto de $\hat{C}$ más cercano. Esto es, únicamente cuando la matriz $A$ tiene las siguientes entradas:*

$$a_{ij} = \begin{cases} 1 & \text{si } j = \arg\min_k \|x_i - c_k\|_2 \\ 0 & \text{en caso contrario} \end{cases} \quad \text{para todos } i, j. \tag{3.19}$$

*Demostración.* ( $\Longrightarrow$ ) Si $\mathcal{F}(\hat{C}, A)$ es mínima, entonces para cada punto $x_i$, la siguiente expresión es mínima (de lo contrario, existiría una asignación diferente que reduciría el costo de la función):

$$\alpha \left( \sum_{j=1}^{k} a_{ij} \|x_i - c_j\|_2^2 \right) + \beta \left( \sum_{j=1}^{k} a_{ij} \|x_i - c_j\|_2 \right).$$

Por las condiciones (3.17) y (3.18), la expresión anterior puede escribirse como:

$$\alpha\|x_i - c_j\|_2^2 + \beta\|x_i - c_j\|_2.$$

Como $\alpha\|x_i - c_j\|_2^2$ es una función monótona creciente, minimizarla es equivalente a minimizar $\alpha\|x_i - c_j\|_2$. Entonces, el objetivo es minimizar:

$$\begin{aligned}\alpha\|x_i - c_j\|_2 + \beta\|x_i - c_j\|_2 &= (\alpha + \beta)\|x_i - c_j\|_2 \\ &= \|x_i - c_j\|_2.\end{aligned}$$

( $\Longleftarrow$ ) Si se minimiza $\|x_i - c_j\|_2$, también se minimiza $\alpha\|x_i - c_j\|_2^2 + \beta\|x_i - c_j\|_2$. Por las condiciones (3.17) y (3.18), esta expresión puede ser escrita como:

$$\alpha\left(\sum_{j=1}^{k} a_{ij}\|x_i - c_j\|_2^2\right) + \beta\left(\sum_{j=1}^{k} a_{ij}\|x_i - c_j\|_2\right),$$

minimizando $\mathcal{F}(\hat{C}, A)$ para cada punto $x_i$.

□

El Teorema 3.3.1 puede interpretarse de la siguiente forma: si se tiene un conjunto fijo de centros $C$, entonces es posible encontrar la matriz $A$ de forma óptima asignando a cada punto $x_i$ de $\mathcal{S}$ el punto $c_j$ de $C$ al que esté más cerca en distancia Euclidiana.

De esta forma, la función $\mathcal{F}$ queda libre de $A$ y se redefine como:

$$\mathcal{F}_1(C) = \alpha\left(\sum_{i=1}^{n}\sum_{j=1}^{k} a_{ij}\|x_i - c_j\|_2^2\right) + \beta\left(\sum_{i=1}^{n}\sum_{j=1}^{k} a_{ij}\|x_i - c_j\|_2\right) \tag{3.20}$$

sujeto a:

$$c_j \in \mathbb{R}^d \text{ para todo } j \in \{1, 2, \ldots, k\}, \tag{3.21}$$

donde cada entrada de la matriz $A = [a_{ij}]$ está dada por:

$$a_{ij} = \begin{cases} 1 & \text{si } j = \arg\min\limits_{k} \|x_i - c_k\|_2 \\ 0 & \text{en caso contrario} \end{cases} \quad \text{para todos } i, j.$$

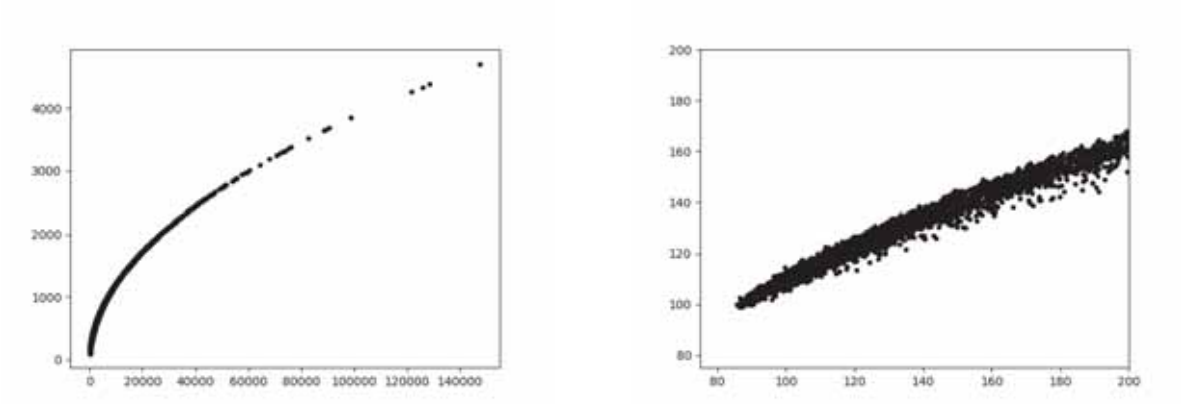
### 3.3.2. Optimización global

Antes de utilizar el criterio de Pareto en este problema de optimización multiobjetivo, véase el comportamiento la función $\mathcal{F}$ cuando se analiza el valor de $J_1$ y $J_2$ en la Figura 3.4. Como se puede ver, existe una correlación entre las funciones $J_1$ y $J_2$. Esto sugiere que la frontera óptima de Pareto está bastante "cerca" del mínimo de la función $\mathcal{F}$. Esto es así debido a que uno de los puntos que minimiza la función domina a la mayoría de los demás puntos. En la práctica, esto es bastante aceptable y se puede suponer que este punto (perteneciente a los que minimizan $\mathcal{F}$) es Pareto–óptimo y que, por lo tanto, pertenece a la frontera óptima de Pareto $\mathcal{P}^*$. Es de esperarse que exista una relación entre los Problemas $k$-means y Emax, ya que el Problema $k$-means busca minimizar la varianza y el Problema Emax busca minimizar la desviación estándar. Estas dos cantidades están estrechamente relacionadas.

Figura 3.4: Comportamiento de $\mathcal{F}$ de acuerdo a los valores de $J_1$ y $J_2$.

(a) Comportamiento a distancia.

(b) Comportamiento cerca del mínimo.




***Fuente:*** *León Malpartida et al., 2019.*

Por este motivo, para solucionar el problema multiobjetivo hace falta únicamente minimizar la función $\mathcal{F}$ (optimizar la función).

Para la optimización de esta función se utilizará el algoritmo Social Spider Optimization (SSO) (Cuevas et al., 2016) tomando las siguientes consideraciones:

- $\mathcal{F}_1 \colon \mathbb{R}^{kd} \to \mathbb{R}_{\geq 0}$ es la función objetivo definida en (3.20) y es esta la función de la cual se buscará la convergencia. Cada araña pertenecerá a $\mathbb{R}^{kd}$, es decir, será un punto en dimensión $kd$. Esto es, la estructura de una araña será:

$$(\underbrace{c_{11}, c_{12}, \ldots, c_{1d}}_{c_1}, \underbrace{c_{21}, c_{22}, \ldots, c_{2d}}_{c_2}, \ldots, \underbrace{c_{k1}, c_{k2}, \ldots, c_{kd}}_{c_k}) \in \mathbb{R}^{kd}.$$

- Al ser este un problema de minimización, los valores $mejor_{\mathrm{s}}$ y $peor_{\mathrm{s}}$ son definidos de la siguiente manera:

$$mejor_{\mathrm{s}} = \min_{i=1}^{N} \mathcal{F}_1(\mathrm{s}_i), \tag{3.22}$$

$$peor_{\mathrm{s}} = \max_{i=1}^{N} \mathcal{F}_1(\mathrm{s}_i). \tag{3.23}$$

  Siendo $N$ la cantidad de arañas de la colonia.

- En cada momento del cálculo de $\mathcal{F}_1$, el valor de $a_{ij}$ estará dado por (3.19).

- Si c es un parámetro de $\mathcal{F}_1$ en $\mathbb{R}^{kd}$, el valor mínimo de la componente $i$ de c, $p_i^{\min}$ será el valor mínimo de la componente $((i-1) \mod k) + 1$ entre todos los puntos $x_k$ de $\mathcal{S}$. Similarmente, el valor máximo de la componente $i$ de c, $p_i^{\max}$ será el valor máximo de la componente $((i-1) \mod k) + 1$ entre todos los puntos $x_k$ de $\mathcal{S}$.

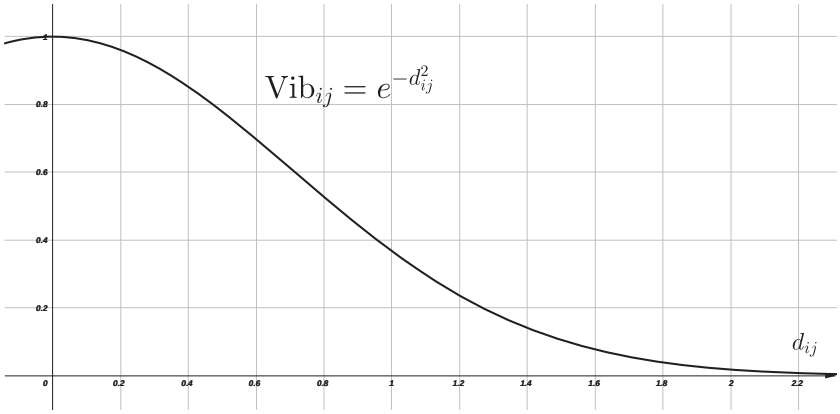
Durante la ejecución del algoritmo, las arañas artificiales generan vibraciones que son captadas por toda la colonia. La vibración generada por la araña $\mathrm{s}_j$ y captada por la araña $\mathrm{s}_i$ se modela mediante la siguiente ecuación:

$$\mathrm{Vib}_{i,j} = w_j \cdot e^{-d_{ij}^2},$$

donde $w_j$ es el peso de la araña $\mathrm{s}_j$ y $d_{ij} = \|\mathrm{s}_i - \mathrm{s}_j\|_2$ es la distancia Euclidiana entre la araña $\mathrm{s}_i$ y la araña $\mathrm{s}_j$.

Como se puede ver en la Figura 3.5, la vibración que recibe cada araña decrece con el cuadrado de la distancia Euclidiana a la araña que la generó. Para la función $\mathcal{F}_1$, las distancias Euclidianas entre dos soluciones pueden crecer significativamente dependiendo de las distancias entre los puntos de $\mathcal{S}$. Esto puede ocasionar que no exista comunicación entre las arañas ya que las vibraciones captadas serían efectivamente cero.

Figura 3.5: Comportamiento de la vibración en función a la distancia.




***Fuente:*** *León Malpartida et al., 2019.*

Para solucionar este problema se considera que el valor de la máxima distancia que se usará para el cálculo de las vibraciones será 1.5. Esta consideración se justifica debido a que aproximadamente el 80 % del área de la función entre 0 y $\infty$ se encuentra entre 0 y 1.5. Para esto, cada distancia $d_{ij}$ se sustituye con $d'_{ij}$ mediante la siguiente ecuación:

$$d'_{ij} = 1.5 \cdot \frac{d_{ij}}{d_{\max}}, \tag{3.24}$$

donde $d_{\max}$ es la máxima distancia entre la araña $\mathrm{s}_i$ y las demás arañas de la colonia. Con esto, se tiene ya todo lo necesario para definir el algoritmo de optimización para el problema; el algoritmo SSO-C es definido a continuación:

**Algoritmo 2:** SSO-C (León Malpartida et al., 2019)

**Input:** $\mathcal{S} \in (\mathbb{R}^d, L^2), k, \alpha, n_s, n_{it}$

$\mathrm{s} \leftarrow (\underbrace{\overrightarrow{0}, \overrightarrow{0}, \ldots, \overrightarrow{0}}_{n_s})$

**for** $i = 1$ **to** $k$ **do**

  $(C_1, \ldots, C_k) \leftarrow$ Greedy $k$-center$(\mathcal{S}, k)$

  $\mathrm{c} = \left\{ \frac{1}{|C_i|} \sum_{u \in C_i} u \mid 1 \leq i \leq k \right\}$

  $\mathrm{s}_i \leftarrow \overrightarrow{\mathrm{c}}$

$\beta \leftarrow 1 - \alpha$

$\mathcal{F}_1(C) = \alpha \left( \sum_{i=1}^{n} \sum_{j=1}^{k} a_{ij} \|x_i - c_j\|_2^2 \right) + \beta \left( \sum_{i=1}^{n} \sum_{j=1}^{k} a_{ij} \|x_i - c_j\|_2 \right)$

$x \leftarrow SSO(\mathcal{F}_1(C), \mathrm{s}, n_s, n_{it})$

Obtener el conjunto $C = \{c_1, c_2, \ldots, c_k\}$ de $x$

**forall** $i, j$ *tal que* $a_{ij} = 1$ **do**

  Colocar $x_i$ en $C_j$

**Output:** $(C_1, C_2, \ldots, C_k)$

Los parámetros del Algoritmo 2 son:

- $\mathcal{S}$, el conjunto de puntos sobre el cual se hará clustering.
- $k$, el número de clusters en los que se separará $\mathcal{S}$.
- $\alpha$, el parámetro de ponderación para la optimización multiobjetivo.
- $n_s$, el número total de arañas de la colonia que se utilizará.
- $n_{it}$, el número de iteraciones que se efectuarán.

La salida del algoritmo es una partición de $\mathcal{S}$, representando los $k$ clusters. El algoritmo tiene un tiempo de ejecución en el peor caso de:

$$\begin{aligned} T(n,k,d,n_s,n_{it}) &= \sum_{i=1}^{k} \Theta(n \cdot k \cdot d) + \sum_{j=1}^{n_{it}} \left( O(\lambda \cdot n \cdot k \cdot d) + O(n_s^2 \cdot d) \right) + \Theta(n \cdot d + k) \\ &= \Theta(n \cdot k^2 \cdot d) + \sum_{j=1}^{n_{it}} \left( O\left(\lambda \cdot n \cdot k \cdot d\right) + O\left(n_s^2 \cdot d\right) \right) + \Theta\left(n \cdot d + k\right) \\ &= \Theta\left(n \cdot k^2 \cdot d\right) + O\left(n_{it} \left(\lambda \cdot n \cdot k \cdot d + n_s^2 \cdot d\right)\right) \\ &= O\left(n \cdot k^2 \cdot d + n_{it} \cdot \lambda \cdot n \cdot k \cdot d + n_{it} \cdot n_s^2 \cdot d\right) \\ &= O\left(nkd\left(k^2 + n_{it}\lambda\right) + n_{it} n_s^2 d)\right). \end{aligned} \tag{3.25}$$

## 3.4. Caso más robusto de $\mathcal{F}$

Como recordatorio, la función objetivo $\mathcal{F}$ definida en (3.15) es

$$\mathcal{F}(C, A) = \alpha \left( \sum_{i=1}^{n} \sum_{j=1}^{k} a_{ij} \|x_i - c_j\|_2^2 \right) + \beta \left( \sum_{i=1}^{n} \sum_{j=1}^{k} a_{ij} \|x_i - c_j\|_2 \right).$$

Supongamos que se quiere minimizar la función $\mathcal{F}$ tomando $\alpha = 1$. Para este caso, el valor de $\beta$ sería 0, por lo que es posible escribir la función de la siguiente manera:

$$\mathcal{F}(C, A) = \sum_{i=1}^{n} \sum_{j=1}^{k} a_{ij} \|x_i - c_j\|_2^2.$$

Como se puede observar, Esta es la función objetivo del problema $k$-means definida en (3.1). Esto significa que minimizar la función $\mathcal{F}$ para este caso, resulta en solucionar el problema $k$-means. Aunque es posible utilizar el algoritmo SSO-C para minimizar $\mathcal{F}$, existe una solución más apropiada. Esta solución es una heurística que consiste en resolver iterativamente los siguientes subproblemas (Huang, 1998):

**Subproblema** $P_1$ : Fijar $A = \hat{A}$ y resolver el problema reducido $\mathcal{F}(\hat{A}, C)$.

**Subproblema** $P_2$ : Fijar $C = \hat{C}$ y resolver el problema reducido $\mathcal{F}(A, \hat{C})$.

El Teorema 3.3.1 introducido anteriormente es útil para resolver el subproblema $P_2$ (este funciona para todo $\alpha$ y $\beta$), por lo que el paso 2 consiste en asignar cada punto de $\mathcal{S}$ a su centro más cercano.

El siguiente teorema será de utilidad para resolver el subproblema $P_1$.

***Teorema* 3.4.1.** *Dado el Problema k-means, sea $\hat{A} = \{a_{ij}\}$ fijado. Entonces la función $\mathcal{F}(\hat{A}, C)$ es minimizada si y solo si se definen los elementos de $C$ como la media de los puntos de $\mathcal{S}$ a los cuales se asignó. Esto es, únicamente si*

$$c_j = \frac{\sum_{i=1}^{n} a_{ij} x_i}{\sum_{i=1}^{n} a_{ij}} \text{ para todo } j. \tag{3.26}$$

*Demostración.* ( $\Longrightarrow$ ) Re escribiendo la función objetivo:

$$\mathcal{F}(\hat{A}, C) = \sum_{i=1}^{n} \sum_{j=1}^{k} a_{ij} (x_i - c_j)^T (x_i - c_j).$$

Si $\mathcal{F}$ es mínimo cuando $\hat{A}$ está fijado, entonces $\nabla_{cj} \mathcal{F} = 0$. Se obtendrá el gradiente de $\mathcal{F}$ con respecto a $c_j$:

$$\nabla_{c_j} \mathcal{F} = \sum_{i=1}^{n} \sum_{j'=1}^{k} a_{ij'} \nabla_{c_j} (x_i - c_{j'})^T (x_i - c_{j'}).$$

Se puede obviar la sumatoria sobre $j'$ ya que todos los términos serán iguales a 0 excepto cuando $j' = j$.

$$\begin{aligned}\nabla_{c_j}\mathcal{F} &= \sum_{i=1}^{n} a_{ij}\nabla_{c_j}(x_i - c_j)^T(x_i - c_j)\\ &= \sum_{i=1}^{n} a_{ij}\nabla_{c_j}(x_i^T x_i - 2c_j^T x_i + c_j^T c_j)\\ &= \sum_{i=1}^{n} a_{ij}(\nabla_{c_j} x_i^T x_i - \nabla_{c_j} 2c_j^T x_i + \nabla_{c_j} c_j^T c_j)\\ &= \sum_{i=1}^{n} a_{ij}(-2x_i + 2c_j).\end{aligned}$$

Luego, resolviendo $\nabla_{cj}\mathcal{F} = 0$ para $c_j$:

$$2c_j \sum_{i=1}^{n} a_{ij} = 2\sum_{i=1}^{n} a_{ij}x_i,$$

$$c_j = \frac{\sum_{i=1}^{n} a_{ij}x_i}{\sum_{i=1}^{n} a_{ij}}.$$

( $\Longleftarrow$ ) Si $c_j = \frac{\sum_{i=1}^{n} a_{ij}x_i}{\sum_{i=1}^{n} a_{ij}}$ para todo $j$, se comprobará que $c_j$ es el único punto crítico mínimo de $\mathcal{F}$. Para esto se tomará la segunda derivada de $\mathcal{F}$ con respecto a $c_j$ en su $h$-ésima componente.

$$\frac{\partial}{\partial c_{jh}}\nabla_{cj}\mathcal{F} = 2\left(\sum_{i=0}^{n} a_{ij}\right)\mathrm{e}_h,$$

donde $\mathrm{e}_1 = (1, 0, \ldots, 0), \mathrm{e}_2 = (0, 1, \ldots, 0)$, etc. representan las bases estándar o bases naturales. Esto es debido a que se está diferenciando un vector y el resultado será 0 para los elementos en posición diferente a $h$, y 1 para los elementos en posición igual a $h$. Por lo tanto, la matriz de segundas derivadas (matriz hessiana) es:

$$\nabla^2_{cj}\mathcal{F} = 2\left(\sum_{i=0}^{n} a_{ij}\right)\mathrm{I},$$

donde I es la matriz identidad. Mientras la expresión anterior sea no nula[3], se concluye que $\nabla^2_{cj}\mathcal{F} > 0$, y por lo tanto $c_j$ es mínimo.

□

Tomando esta consideración, la heurística para minimizar la función objetivo cuando $\alpha = 1$ (el problema $k$-means) consiste en efectuar iterativamente los dos siguientes pasos (asumiendo una inicialización de $C$ aleatoria):

- Asignar cada punto de $\mathcal{S}$ al punto más cercano de $C$ según el Teorema 3.3.1.
- Definir cada punto de $C$ como la media de los puntos de $S$ a los cuales fue asignado según el Teorema 3.4.1.

[3]Esto es así porque la suma sobre $i$ de $a_{ij}$ da el número de puntos de $\mathcal{S}$ asignados a $c_j$. Para fines prácticos, debe asumirse que no existe un punto de $C$ al cual no existe un punto de $\mathcal{S}$ asignado.

Como se puede observar, esta heurística corresponde con el algoritmo $k$-means.

De la misma forma, es posible diseñar una heurística para minimizar el caso más robusto de la función objetivo de una forma más apropiada[4]. Para tal efecto, consideraremos que $\alpha = 0$, haciendo que $\beta = 1$ y $\mathcal{F}$ sea:

$$\mathcal{F}(C, A) = \sum_{i=1}^{n} \sum_{j=1}^{k} a_{ij} \|x_i - c_j\|_2.$$

La función objetivo ahora se convierte en la del problema Emax, por lo tanto, minimizarla implica resolver el problema Emax. Similarmente al caso anterior, la heurística consistirá en solucionar iterativamente los siguientes subproblemas:

**Subproblema** $P_1$ : Fijar $A = \hat{A}$ y resolver el problema reducido $\mathcal{F}(\hat{A}, C)$.

**Subproblema** $P_2$ : Fijar $C = \hat{C}$ y resolver el problema reducido $\mathcal{F}(A, \hat{C})$.

Como en el caso anterior, el Teorema 3.3.1 sirve para cualquier valor de $\alpha$ y $\beta$, por lo tanto este es útil para resolver el subproblema $P_2$. Sin embargo, resolver el subproblema $P_1$ no es tan simple como en el caso anterior (a pesar de la similitud de la función objetivo). Para solucionar $P_1$, se introducirá uno de los problemas no triviales más antiguos de la geometría computacional.

***Teorema* 3.4.2.** *Dado el Problema Emax, sea $\hat{A} = \{a_{ij}\}$ fijado. Entonces la función $\mathcal{F}(\hat{A}, C)$ es minimizada si y solo si se definen los elementos de $C$ como un punto geometric median de los puntos de $\mathcal{S}$ a los cuales se asignó. Esto es, únicamente si*

$$c_j \in \arg\min_{c \in \mathbb{R}^d} \sum_{i=1}^{n} a_{ij} \|x_i - c\|_2 \text{ para todo } j. \tag{3.27}$$

*Demostración.* ( $\Longrightarrow$ ) La ubicación de un centro no afecta a los puntos que no están asignados a él, por lo que cada centro es independiente de cualquier otro. Si $J_2(C, \hat{A})$ es mínimo, entonces para cada centro $c_j$ la siguiente expresión es también mínima:

$$\sum_{i=1}^{n} a_{ij} \|x_i - c_j\|_2. \tag{3.28}$$

Por lo tanto, $c_j$ debe ser un punto que minimiza (3.28).

( $\Longleftarrow$ ) Si $c_j$ es un centro que minimiza (3.28), entonces la función objetivo es también mínima al tratarse de la suma de mínimos independientes. Entonces $J_2(C, \hat{A})$ es mínimo para cada $c_j$. □

Como se puede ver, resolver el subproblema $P_1$ requiere de encontrar un punto que minimice la suma de las distancias Euclidianas de él a un conjunto de puntos fijos. Este punto es conocido como el *geometric median*. Se llamará problema geometric median al problema de encontrar el punto del mismo nombre.

A pesar de la naturaleza antigua de este problema, existen pocas garantías teóricas para resolverlo. El problema geometric median fue primeramente formulado para el caso de tres puntos a inicios del siglo XVII por Pierre de Fermat (Krarup & Vajda, 1997). A pesar de la construcción de una solución elegante por medio de regla y compás para el caso de 3 puntos por

[4] Anteriormente se utilizó una forma modificada del algoritmo SSO-C para este caso de la función objetivo (Vera-Olivera et al., 2016), dando pie al algoritmo SSO modificado para clustering.

Evangelista Torricelli, no existe una construcción similar para un número más grande de puntos. De hecho, se demostró que incluso para 5 puntos, no existe solución por radicales en el campo de los racionales (Bajaj, 1988). Se estudió el problema de aproximación $(1+\varepsilon)$ para valores grandes de $n$. Muchos de autores han propuesto algoritmos de complejidad temporal polinomial en $n$, $d$ y $1/\varepsilon$. En (Cohen, Lee, Miller, Pachocki & Sidford, 2016) se hace un recuento de estos algoritmos. También se proponen dos algoritmos con complejidad temporal casi lineal para el problema geometric median. Sin embargo, el algoritmo más utilizado para resolver el problema es el algoritmo de Weiszfeld propuesto en 1937 (Weiszfeld, 1937). Un aspecto desfavorable de este algoritmo es que, bajo ciertas circunstancias, el algoritmo podría no converger; sin embargo, en un caso práctico funciona bastante bien. En general, todas las propuestas y variaciones del algoritmo de Weiszfeld no proveen una cota asintótica sobre la complejidad temporal.

El algoritmo de Weiszfeld define un conjunto de pesos que son inversamente proporcionales a las distancias desde la estimación actual a los puntos, y crea una nueva estimación que resulta del promedio ponderado de los puntos de acuerdo a tales pesos. Esto es,

$$c_{k+1} = \left(\sum_{i=1}^{n} \frac{x_i}{\|x_i - c_k\|_2}\right) \Bigg/ \left(\sum_{i=1}^{n} \frac{1}{\|x_i - c_k\|_2}\right). \tag{3.29}$$

Este algoritmo converge casi para todas las posiciones iniciales, pero falla en converger cuando una de las estimaciones cae en uno de los puntos dados. Este caso es fácilmente modificable de tal manera que la convergencia para todos los puntos está asegurada.

### 3.4.1. Algoritmo exacto

Dejando de lado temporalmente la heurística mencionada, el Teorema 3.4.2 brinda suficiente información para desarrollar un algoritmo exacto para el Problema Emax. Para diseñar un algoritmo exacto básico, es posible utilizar el hecho de que, dada una partición del conjunto $\mathcal{S}$ en $k$ subconjuntos no vacíos, es posible encontrar el centro de cada partición (asumiendo que esta forma un cluster) gracias al Teorema 3.4.2. Primeramente, se determinará cotas sobre el número de posibles clusterings que podrían formarse con el fin de verificar que este no es infinito.

Se asume un conjunto $\mathcal{S}$ de $n$ puntos $\{x_1, x_2, \ldots, x_n\}$ en un espacio métrico $(\mathbb{R}^d, L^2)$ y un entero $k > 1$.

Sea $A_i$ el conjunto de $k$ clusterings posibles a formar con el conjunto $\mathcal{S}$ dado que el cluster $i$ está vacío. Es fácil ver que:

$$|A_1| = |A_2| = \cdots = |A_k| = (k-1)^n. \tag{3.30}$$

Dado que existen $k$ de estos conjuntos, la suma sobre las cardinalidades es:

$$\sum_{1 \le i \le k} |A_i| = \binom{k}{1}(k-1)^n. \tag{3.31}$$

En este ámbito, $A_i \cap A_j$ es el conjunto de $k$ clusterings posibles a formar con el conjunto $\mathcal{S}$ dado que el cluster $i$ y el cluster $j$ están vacíos. Al igual que en el caso anterior, la cardinalidad de estos conjuntos para $i \neq j$ es:

$$|A_1 \cap A_2| = |A_1 \cap A_3| = \cdots = |A_{k-1} \cap A_k| = (k-2)^n. \tag{3.32}$$

También, la suma sobre estas cardinalidades es:

$$\sum_{1\leq i<j\leq k} |A_i \cap A_j| = \binom{k}{2}(k-2)^n. \tag{3.33}$$

En general, $A_i \cap A_j \cap \cdots \cap A_z$ es el conjunto de $k$ clusterings posibles a formar con el conjunto $\mathcal{S}$ dado que los $m$ clusters $i, j, \ldots, z$ están vacíos. La suma sobre estas cardinalidades es:

$$\sum_{1\leq i<j<\cdots<z\leq k} |A_i \cap A_j \cap \cdots \cap A_z| = \binom{k}{m}(k-m)^n. \tag{3.34}$$

Se denotará $k! \cdot c(n,k)$ al número de posibles $k$ clusterings no vacíos de $n$ elementos redundando en permutaciones de clusterings. Es claro que este número es igual al número total de clusterings (vacíos o no) menos el número de clusterings que contienen algún (uno o más) cluster vacío:

$$k! \cdot c(n,k) = n^k - |A_1 \cup A_2 \cup \cdots \cup A_k|. \tag{3.35}$$

Es posible calcular el segundo miembro de la ecuación usando el principio de inclusión-exclusión:

$$\begin{aligned} |A_1 \cup A_2 \cup \cdots \cup A_k| = &\sum_{1\leq i\leq k} |A_i| - \sum_{1\leq i<j\leq k} |A_i \cap A_j| + \cdots \\ &+ (-1)^{k+1} \sum_{1\leq i<j<\cdots<z\leq k} |A_i \cap A_j \cap \cdots \cap A_z| \\ =&\binom{k}{1}(k-1)^n - \binom{k}{2}(k-2)^n + \cdots + (-1)^{k+1}\binom{k}{k}(k-k)^n. \end{aligned} \tag{3.36}$$

De esta forma, $k! \cdot c(n,k)$ queda como:

$$\begin{aligned} k! \cdot c(n,k) &= n^k - \binom{k}{1}(k-1)^n + \binom{k}{2}(k-2)^n - \cdots + (-1)^k\binom{k}{k}(k-k)^n \\ &= \binom{k}{0}k^n - \binom{k}{1}(k-1)^n + \binom{k}{2}(k-2)^n - \cdots + (-1)^k\binom{k}{k}(k-k)^n \\ &= \sum_{i=0}^{k}(-1)^i\binom{k}{i}(k-i)^n. \end{aligned} \tag{3.37}$$

Luego, el número de posibles $k$ clusterings no vacíos de $\mathcal{S}$ es:

$$c(n,k) = \frac{1}{k!}\sum_{i=0}^{k}(-1)^i\binom{k}{i}(k-i)^n. \tag{3.38}$$

Este resultado corresponde con los números de Stirling de segunda especie. En (Rennie & Dobson, 1969) se demostró que, si $n > 2$ y $1 \leq k \leq n-1$, entonces:

$$\frac{1}{2}\left(k^2+k+2\right)k^{n-k-1} - 1 \leq c(n,k) \leq \frac{1}{2}\binom{n}{k}k^{n-k}. \tag{3.39}$$

Esto no solo es importante por las cotas halladas en el número de clusterings; también es útil para, conociendo la definición, poder hallar un algoritmo de programación dinámica que encuentre todos los posibles clusterings.

Es posible ver que $c(n,k)$ puede definirse recursivamente de la siguiente forma:

$$c(n,k) = \begin{cases} 1 & \text{si } k = 0 \text{ y } n = 0 \\ 0 & \text{si } k = 0 \text{ y } n > 0 \\ k \cdot c(n-1,k) + c(n-1,k-1) & \text{si } k > 0 \text{ y } n > 0, \end{cases} \tag{3.40}$$

lo que con un poco de análisis puede dar pie al algoritmo exacto. Este algoritmo debería calcular los $O(k^n)$ posibles clusterings y para cada uno, encontrar los centros de los clusters según el Teorema 3.4.2. Con esta información es posible calcular el valor de $\mathcal{F}(C,A)$ para cada posible clustering, guardando en cada etapa el clustering para el cual $\mathcal{F}$ tenga el valor mínimo visto.

Es claro que, dadas las asíntotas mostradas en (3.39), el algoritmo descrito tendría una complejidad temporal en el peor caso de $\Omega(k^{n-k})$. Un algoritmo con dicha complejidad no es práctico en absoluto incluso para conjuntos pequeños. Intentar hacer clustering en 5 grupos sobre un conjunto $\mathcal{S}$ con 30 elementos utilizando este algoritmo en una computadora promedio supondría un tiempo de ejecución de aproximadamente 100 años. Es por este motivo que, es necesario encontrar otra solución para resolver el problema, sacrificando la solución óptima en pocos casos.

### 3.4.2. Heurística

La heurística para el algoritmo aproximado, como se indicó anteriormente, consistirá en resolver los subproblemas $P_1$ y $P_2$ iterativamente hasta alcanzar la convergencia. En $P_1$ se desea calcular el punto geometric median punto para todos los puntos de $\mathcal{S}$ que comparten el mismo centro. De la misma forma, en el problema $P_2$ es posible calcular las asignaciones teniendo los centros fijados. Por lo tanto, la heurística para el problema Emax consiste en iterativamente efectuar los siguientes pasos hasta la convergencia:

- Asignar cada punto de $\mathcal{S}$ al punto más cercano de $C$ según el Teorema 3.3.1.
- Definir cada punto de $C$ como el punto geometric median de los puntos de $S$ a los cuales fue asignado re-calculándolo iterativamente según (3.29).

El procedimiento final para el caso más robusto de la función objetivo es presentado en el Algoritmo 3:

**Algoritmo 3:** Emax (León Malpartida et al., 2019)

**Input:** $\mathcal{S} \in (\mathbb{R}^d, L^2), k$

**for** $i = 1$ **to** $k$ **do**
  $c_i \leftarrow x_i \cdot U(0,1)$
**while** *no se alcanzó la convergencia* **do**
  Asignar cada punto $x_i \in \mathcal{S}$ a $c_j$ si $j = \arg\min_k \|x_i - c_k\|_2$
  **for** $j = 1$ **to** $k$ **do**
    Sea $A = \{a_1, a_2, \ldots, a_m\}$ los puntos de $\mathcal{S}$ asignados a $c_j$
    **while** *no se alcanzó la convergencia* **do**
      $c_j \leftarrow \left(\sum_{i=1}^{m} \frac{a_i}{\|a_i - c_j\|_2}\right) \Big/ \left(\sum_{i=1}^{m} \frac{1}{\|a_i - c_j\|_2}\right)$
**forall** $i, j$ *tal que* $a_{ij} = 1$ **do**
  Colocar $x_i$ en $C_j$

**Output:** $(C_1, C_2, \ldots, C_k)$

En el algoritmo Emax, $U(0,1]$ representa un valor aleatorio mayor a cero y menor a uno.

Cada iteración del algoritmo consiste en un paso de asignación y un paso de definición del geometric median. Un criterio para fijar la parada del algoritmo es asignar un número máximo de iteraciones a realizar, y si el algoritmo llega a este punto, se devuelve el mejor resultado encontrado. Sin embargo, como se especifica en el algoritmo, el criterio de parada es la convergencia del mismo. Por lo tanto, es necesario demostrar que el algoritmo alcanza la convergencia en cada caso. Para demostrar este hecho, los siguientes lemas serán de utilidad.

***Lema* 3.4.3.** *Dado el algoritmo Emax; cada vez que $P_1$ es resuelto, $\mathcal{F}(C, A)$ decrece o se mantiene.*

*Demostración.* Sea $C^* = \{c_j^*\}$ el conjunto de centros luego de haber resuelto $P_1$ en alguna iteración del algoritmo, y sea $C = \{c_j\}$ el conjunto de centros de la iteración previa. Luego, el cambio en $\mathcal{F}$ para la iteración actual es:

$$\begin{aligned}\mathcal{F}(C^*, \hat{A}) - \mathcal{F}(C, \hat{A}) &= \sum_{i=1}^{n}\sum_{j=1}^{k} a_{ij}\|x_i - c_j^*\|_2 - \sum_{i=1}^{n}\sum_{j=1}^{k} a_{ij}\|x_i - c_j\|_2 \\ &= \sum_{j=1}^{k}\left(\sum_{i=1}^{n} a_{ij}\|x_i - c_j^*\|_2 - \sum_{i=1}^{n} a_{ij}\|x_i - c_j\|_2\right) \\ &\leq 0. \end{aligned} \tag{3.41}$$

La inecuación es válida ya que por el Teorema 3.4.2, $c_j^*$ está definido como:

$$c_j^* \in \arg\min_{c \in \mathbb{R}^d} \sum_{i=1}^{n} a_{ij}\|x_i - c\|_2.$$

□

***Lema* 3.4.4.** *Dado el algoritmo Emax; cada vez que $P_2$ es resuelto, $\mathcal{F}(C, A)$ decrece o se mantiene.*

*Demostración.* Sea $A^* = \{a^*_{ij}\}$ la matriz de asignaciones luego de haber resuelto $P_2$ en alguna iteración del algoritmo, y sea $A = \{a_{ij}\}$ la matriz de asignaciones en la iteración previa. Luego, el cambio en $\mathcal{F}$ para la iteración actual es:

$$\begin{aligned}\mathcal{F}(\hat{C}, A^*) - \mathcal{F}(\hat{C}, A) &= \sum_{i=1}^{n}\sum_{j=1}^{k} a^*_{ij}\|x_i - c_j\|_2 - \sum_{i=1}^{n}\sum_{j=1}^{k} a_{ij}\|x_i - c_j\|_2 \\ &= \sum_{i=1}^{n}\sum_{j=1}^{k} \|x_i - c_j\|_2 \left(a^*_{ij} - a_{ij}\right) \\ &\leq 0. \end{aligned} \tag{3.42}$$

La inecuación es válida ya que por el Teorema 3.3.1, $a^*_{ij}$ está definido como

$$a^*_{ij} = \begin{cases} 1 & \text{si } j = \arg\min\limits_k \|x_i - c_k\|_2 \\ 0 & \text{en caso contrario} \end{cases} \quad \text{para todos } i, j.$$

□

Con estos dos lemas, es posible demostrar fácilmente la convergencia del algoritmo Emax.

***Teorema* 3.4.5.** *El algoritmo Emax converge.*

*Demostración.* El algoritmo Emax resuelve iterativamente $P_1$ y $P_2$. Por los Lemas 3.4.3 y 3.4.4, la función $\mathcal{F}$ decrece monótonamente en cada iteración. Dado que existen $O(k^n)$ formas de particionar el conjunto $\mathcal{S}$ en $k$ clusters; cada forma puede ser llamada un clustering. En cada iteración del algoritmo, se produce un nuevo clustering basado únicamente en el clustering anterior. En cada iteración, existen posibles dos casos:

1. Si el nuevo clustering es igual al anterior, entonces la función $\mathcal{F}$ se mantiene y el siguiente clustering será nuevamente el mismo.
2. Si el nuevo clustering es diferente al anterior, entonces el nuevo tiene un costo menor, i.e., la función $\mathcal{F}$ decrece.

Dado que el algoritmo itera sobre una función cuyo dominio es un conjunto finito, la iteración debe eventualmente entrar en un ciclo. El ciclo no puede tener una longitud mayor a 1, ya que por el caso 2, se encontraría un clustering con menor costo que el mismo, siendo este caso imposible. Por este motivo el ciclo debe tener longitud de exactamente 1 y por lo tanto, el algoritmo Emax converge en un número finito de iteraciones. □

El tiempo de ejecución en el peor caso del algoritmo Emax es:

$$\begin{aligned} T(n,k,d) &= \Theta\left(k \cdot d\right) + \sum_{i=1}^{\gamma}\left(\Theta(n) + \sum_{j=1}^{k}\Theta\left(\lambda \cdot n \cdot d\right)\right) + \Theta(n) \\ &= \sum_{i=1}^{\gamma}\left(\Theta(n) + \Theta\left(\lambda \cdot n \cdot k \cdot d\right)\right) + \Theta(n + k \cdot d) \\ &= \Theta\left(\gamma \cdot \lambda \cdot n \cdot k \cdot d\right) + \Theta(n + k \cdot d) \\ &= \Theta\left(\gamma \cdot \lambda \cdot n \cdot k \cdot d\right), \end{aligned} \tag{3.43}$$

donde $\gamma$ es el número de iteraciones necesarias para la convergencia del algoritmo, $\lambda$ es el número de iteraciones necesarias para hallar el geometric median, $n$ es el número de elementos, $k$ es el número de clusters a formar, y $d$ es la dimensión del espacio en el que se opera.

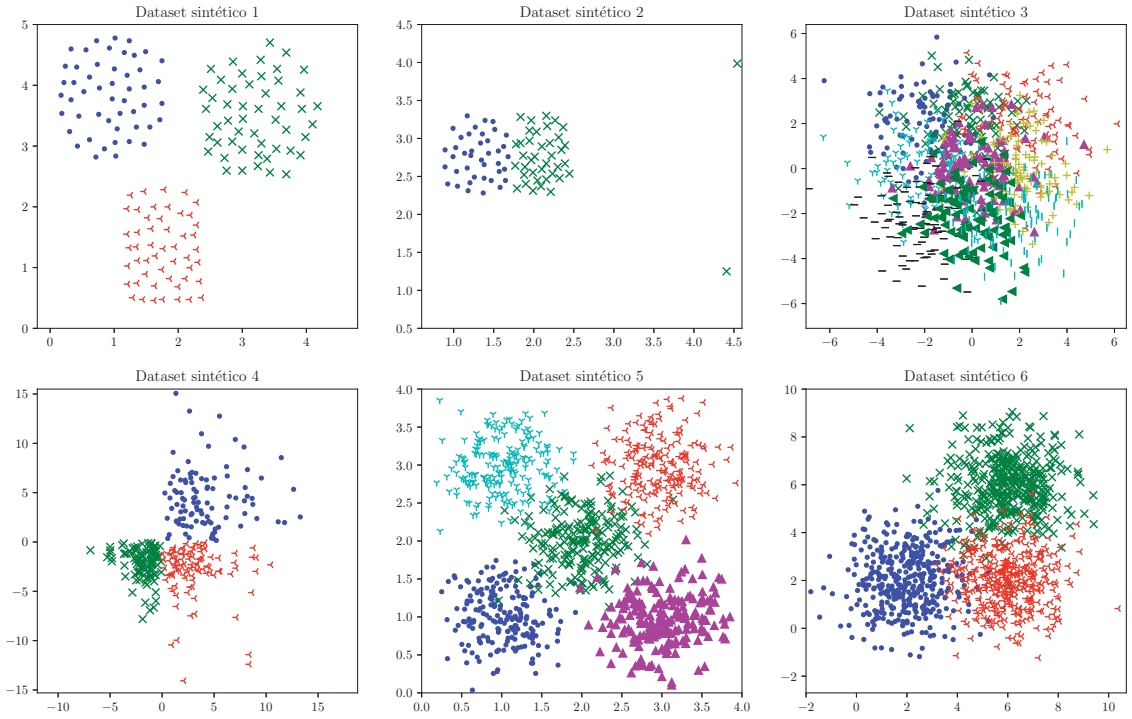
## 3.5. Conjuntos de prueba

Para observar los resultados de los algoritmos propuestos, se realizaron algunas pruebas experimentales. Es necesario utilizar conjuntos de datos que presenten múltiples outliers, y también algunos conjuntos estándares de la bibliografía.

### 3.5.1. Conjuntos de datos sintéticos

Se generaron seis conjuntos sintéticos para propósitos del experimento. La generación de estos conjuntos y sus características son detallados a continuación. La Figura 3.6 muestra gráficamente cada uno de estos datasets. En la Tabla 3.1 se brinda información clave sobre ellos.

Figura 3.6: Datasets sintéticos utilizados en las pruebas experimentales.




***Fuente:*** *León Malpartida et al., 2019.*

- **Dataset sintético 1**: El primer dataset es relativamente simple para un algoritmo de clustering básico. Contiene tres clusters claramente separados y con dimensiones similares

- **Dataset sintético 2**: Este dataset contiene dos clusters apegados. La característica de este, es que el segundo cluster posee dos outliers alejados considerablemente de su centro de masa.

- **Dataset sintético 3**: Este dataset contiene nueve clusters. A diferencia de los datasets anteriores, en este se generó cada cluster de acuerdo a una distribución de probabilidad Gaussiana Multivariable.

  La distribución Gaussiana Multivariable o Normal Multivariable es la distribución conjunta más utilizada para variables continuas. La función de densidad de probabilidad de la distribución para una dimensión $D$ está definida por la siguiente expresión:

$$\mathcal{N}(\mathrm{x}|\mu,\Sigma) = \frac{1}{(2\pi)^{D/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathrm{x}-\mu)^T\Sigma^{-1}(\mathrm{x}-\mu)\right), \tag{3.44}$$

donde $\mu = \mathbb{E}[\mathrm{x}] \in \mathbb{R}^D$ es el vector media, y $\Sigma = \mathrm{cov}[\mathrm{x}]$ es la matrix de covarianza $D \times D$. La constante de normalización $(2\pi)^{D/2}|\Sigma|^{1/2}$ se asegura que la función de densidad integre a 1.

- **Dataset sintético 4**: Este dataset contiene tres clusters. Como en el caso anterior, en este los clusters también fueron generados de acuerdo a una distribución de probabilidad. La distribución utilizada fue la Distribución Gamma.

  La distribución gamma es una distribución continua para variables aleatorias $x > 0$. Esta está definda en términos de dos parámetros, llamados "shape" $a > 0$ y "rate" $b > 0$:

$$\mathrm{Ga}(T|\mathrm{shape} = a, \mathrm{rate} = b) = \frac{b^a}{\Gamma(a)} T^{a-1} e^{-Tb} \tag{3.45}$$

  donde $\Gamma(a)$ es la función gamma:

$$\Gamma(x) = \int_0^\infty u^{x-1} e^{-u} du \tag{3.46}$$

  Esta distribución tiene algunas propiedades, entre ellas:

$$\mathrm{media} = \frac{a}{b}, \mathrm{moda} = \frac{a-1}{b}, \mathrm{varianza} = \frac{a}{b^2}, \mathrm{desviación\ estándar} = \sqrt{\frac{a}{b^2}} \tag{3.47}$$

  Cada componente de cada punto del dataset fue generada utilizando la distribución gamma independientemente y con los mismos valores de $a$ y $b$.

- **Dataset sintéticos 5 y 6**: Estos datasets, al igual que el dataset 3, fueron generados usando una distribución Gaussiana Multivariable. El objetivo de estos datasets es evaluar el comportamiento del algoritmo frente a grandes cantidades de datos. El dataset 5 contiene 1000 puntos y el dataset 6, 1200. Cada dataset fue generado con parámetros $\mu$ y $\Sigma$ independientes entre sí.

Tabla 3.1: Información de los datasets sintéticos.

| | | | | |
|---|---|---|---|---|
| Data 1 | **Espacio** | **Puntos por cluster** | | |
| | $\mathbb{R}^2$ | Cluster 1 50 puntos | | |
| | | Cluster 2 50 puntos | | |
| | | Cluster 3 50 puntos | | |
| Data 2 | **Espacio** | **Puntos por cluster** | | |
| | $\mathbb{R}^2$ | Cluster 1 40 puntos | | |
| | | Cluster 2 40 puntos | | |
| Data 3 | **Espacio** | **Puntos por cluster** | $\boldsymbol{\mu}$ | $\boldsymbol{\Sigma}$ |
| | $\mathbb{R}^2$ | Cluster 1 100 puntos | $(-2, 2)$ | $\begin{bmatrix} 1.69 & 0 \\ 0 & 1.69 \end{bmatrix}$ |
| | | Cluster 2 100 puntos | $(0, 2)$ | |
| | | Cluster 3 100 puntos | $(2, 2)$ | |
| | | Cluster 4 100 puntos | $(-2, 0)$ | |
| | | Cluster 5 100 puntos | $(0, 0)$ | |
| | | Cluster 6 100 puntos | $(2, 0)$ | |
| | | Cluster 7 100 puntos | $(-2, -2)$ | |
| | | Cluster 8 100 puntos | $(0, -2)$ | |
| | | Cluster 9 100 puntos | $(2, -2)$ | |
| Data 4 | **Espacio** | **Puntos por cluster** | $\boldsymbol{a}$ | $\boldsymbol{b}$ |
| | $\mathbb{R}^2$ | Cluster 1 100 puntos | 2 | $1/2$ |
| | | Cluster 2 100 puntos | 2 | 1 |
| | | Cluster 3 100 puntos | 2 | $2/3$ |
| Data 5 | **Espacio** | **Puntos por cluster** | $\boldsymbol{\mu}$ | $\boldsymbol{\Sigma}$ |
| | $\mathbb{R}^2$ | Cluster 1 200 puntos | $(1, 1)$ | $\begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix}$ |
| | | Cluster 2 200 puntos | $(2, 2)$ | |
| | | Cluster 3 200 puntos | $(3, 3)$ | |
| | | Cluster 4 200 puntos | $(1, 3)$ | |
| | | Cluster 5 200 puntos | $(3, 1)$ | |
| Data 6 | **Espacio** | **Puntos por cluster** | $\boldsymbol{\mu}$ | $\boldsymbol{\Sigma}$ |
| | $\mathbb{R}^2$ | Cluster 1 400 puntos | $(2, 2)$ | $\begin{bmatrix} 1.44 & 0 \\ 0 & 1.44 \end{bmatrix}$ |
| | | Cluster 2 400 puntos | $(6, 6)$ | |
| | | Cluster 3 400 puntos | $(6, 2)$ | |

***Fuente:*** *León Malpartida et al., 2019.*

### 3.5.2. Conjuntos de datos reales

Se tomó siete datasets comúnmente utilizados para pruebas experimentales y con propósitos de enseñanza. Los datasets fueron tomados de UCI Machine Learning Repository (Dua & Karra Taniskidou, 2017). Estos datasets tienen la tarea asociada de clasificación y la información presentada no contiene valores faltantes. Algunas características de estos datasets se encuentran en la Tabla 3.2.

Tabla 3.2: Información acerca de los datasets reales.

| **Dataset** | **Iris** | **Vowel** | **Oil** | **Balance** | **Cancer** | **Wine** | **Glass** |
|---|---|---|---|---|---|---|---|
| **Dimensión** | 4 | 3 | 5 | 4 | 30 | 13 | 8 |
| **Clusters** | 3 | 6 | 3 | 3 | 2 | 3 | 6 |
| **Puntos** | 150 | 871 | 56 | 625 | 569 | 178 | 214 |

***Fuente:*** *León Malpartida et al., 2019.*

## 3.6. Experimentación

Para propósitos de la experimentación se utilizó los algoritmos SSO adaptado para clustering por (Vera-Olivera et al., 2016), $k$-means (algoritmo de Lloyd), Emax, y el algoritmo SSO-C. La salida de los cuatro algoritmos fue modificada de la siguiente manera: sea $n$ el número de puntos del dataset de entrada, la salida de cada algoritmo es una secuencia de $n$ elementos $a_1, a_2, \ldots, a_n$, siendo $a_i$ un número entero entre 0 y $k-1$ representando el número del cluster al que el punto fue asignado como predicción del algoritmo. Esta secuencia de $n$ elementos, junto con la secuencia $b_1, b_2, \ldots, b_n$ de clusters verdaderos a los que los puntos pertenecen son luego analizados con la función Adjusted Mutual Information (AMI), la cual devuelve un valor real entre $-1$ y $1$[5].

Se implementó los algoritmos SSO, $k$-means, Emax, y SSO-C en el lenguaje de programación C (gcc versión 8.1.0 compilado por MinGW-W64) en un procesador Intel Core i7-6700. Se utilizó una memoria RAM de 16GB y Windows 10 version 1809 como sistema operativo. Los datsets fueron utilizados como archivos (.woz) en entrada estándar. El código fuente de los algoritmos propuestos, Emax y SSO-C están disponibles respectivamente en (León, 2019a, 2019b).

Teniendo seis conjuntos de datos sintéticos y siete reales, la primera tarea realizada fue ejecutar los algoritmos: SSO, $k$-means (algoritmo de Lloyd), Emax, y SSO-C sobre todos los conjuntos de datos. Se realizó 50 ejecuciones de cada algoritmo sobre cada conjunto de datos. Información sobre el máximo, la mediana y la media de los resultados es indicada en la Tabla 3.3, cada fila resalta al mejor resultado de entre los cuatro algoritmos.

Tabla 3.3: Resultados de los experimentos realizados en los cuatro algoritmos.

| **Datasets sintéticos** | | | | | | **Datasets reales** | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Dataset** | | **SSO** | ***k*-means** | **Emax** | **SSO-C** | **Dataset** | | **SSO** | ***k*-means** | **Emax** | **SSO-C** |
| | Máximo | **1.000** | **1.000** | **1.000** | **1.000** | | Máximo | 0.793 | 0.748 | 0.787 | **0.829** |
| Dataset S. 1 | Mediana | **1.000** | **1.000** | **1.000** | **1.000** | Iris | Mediana | 0.713 | 0.748 | 0.757 | **0.762** |
| | Media | **1.000** | **1.000** | **1.000** | **1.000** | | Media | 0.716 | 0.743 | 0.763 | **0.765** |
| | Máximo | **1.000** | 0.854 | **1.000** | **1.000** | | Máximo | 0.517 | 0.477 | 0.509 | **0.517** |
| Dataset S. 2 | Mediana | **1.000** | 0.802 | **1.000** | **1.000** | Vowel Indian | Mediana | 0.451 | 0.459 | **0.490** | 0.453 |
| | Media | 0.964 | 0.813 | 0.664 | **1.000** | | Media | 0.445 | 0.463 | **0.493** | 0.452 |
| | Máximo | 0.353 | 0.341 | 0.352 | **0.366** | | Máximo | 0.250 | 0.236 | **0.272** | 0.250 |
| Dataset S. 3 | Mediana | 0.336 | 0.340 | 0.342 | **0.342** | Crude Oil | Mediana | 0.204 | 0.182 | **0.272** | 0.182 |
| | Media | 0.336 | 0.338 | **0.343** | 0.343 | | Media | 0.201 | 0.190 | **0.249** | 0.189 |
| | Máximo | 0.806 | 0.770 | 0.813 | **0.814** | | Máximo | **0.274** | 0.160 | 0.134 | 0.253 |
| Dataset S. 4 | Mediana | 0.761 | 0.770 | **0.813** | 0.790 | Balance Scale | Mediana | **0.214** | 0.115 | 0.117 | 0.109 |
| | Media | 0.719 | 0.770 | **0.813** | 0.784 | | Media | **0.223** | 0.106 | 0.114 | 0.113 |
| | Máximo | 0.888 | 0.885 | 0.882 | **0.898** | | Máximo | 0.453 | 0.422 | **0.458** | 0.440 |
| Dataset S. 5 | Mediana | 0.863 | 0.881 | 0.882 | **0.884** | Breast Cancer | Mediana | 0.351 | 0.422 | **0.458** | 0.374 |
| | Media | 0.852 | 0.882 | **0.882** | 0.882 | | Media | 0.322 | 0.422 | **0.458** | 0.369 |
| | Máximo | 0.770 | 0.756 | 0.758 | **0.779** | | Máximo | **0.432** | 0.423 | 0.427 | **0.432** |
| Dataset S. 6 | Mediana | 0.752 | 0.754 | 0.755 | **0.755** | Wine | Mediana | 0.413 | **0.423** | 0.419 | **0.423** |
| | Media | 0.748 | 0.754 | 0.755 | **0.757** | | Media | **0.419** | 0.408 | 0.414 | 0.417 |
| | | | | | | | Máximo | 0.301 | 0.316 | 0.319 | **0.389** |
| | | | | | | Glass | Mediana | 0.177 | 0.274 | **0.319** | 0.318 |
| | | | | | | | Media | 0.165 | 0.290 | 0.319 | **0.320** |

***Fuente:*** *León Malpartida et al., 2019.*

Los análisis estadísticos fueron realizados (de acuerdo a (Demšar, 2006), ver también (García & Herrera, 2008) y (Derrac, García, Molina & Herrera, 2011)) sobre los resultados de los algoritmos (muestras de 50 elementos) de la siguiente forma:

1. Primero, el Test de Friedman fue realizado para obtener una conclusión sobre la hipótesis

[5]Cuanto más alto sea el valor devuelto por esta función, mejor es el clustering encontrado. Un valor de 1 indica un clustering perfecto, y un valor de 0 indica un cluster aleatorio. Es posible obtener valores negativos bajo circunstancias específicas.

nula (sobre los algoritmos usados):

$H_0$: *Todos los algoritmo tienen el mismo desempeño.*

Rechazar esta hipótesis implicaría que al menos un algoritmo es superior a los demás.

2. Segundo, cuando la hipótesis nula sea rechazada de acuerdo al Test de Friedman, entonces el test de Holm es realizado para obtener una conclusión sobre la nueva hipótesis nula (sobre los algoritmos usados):

$H_0$: *Un algoritmo de control tiene el mismo desempeño con respecto a cualquier otro algoritmo.*

Rechazar esta hipótesis implicaría que el algoritmo de control es superior a los demás.

En ambas pruebas, se tomó un nivel de significación de $\alpha = 0.05$ por defecto[6]. Dados los resultados en el Dataset Sintético 1 (todos los valores perfectos), este dataset no se utilizó para el análisis estadístico.

La hipótesis nula del Test de Friedman fue rechazada en todos los datasets, sugiriendo que existe al menos un algoritmo superior a los demás. La Tabla 3.4 presenta los resultados del Test de Holm, donde se indica el algoritmo de control y los algoritmos en los que la hipótesis nula fue rechazada (resaltados en negrita).

Tabla 3.4: Resultados del Test de Holm.

| **Dataset** | **Control** | $i$ | **Algoritmo** | **Rank** | **p-value** | $\alpha/i$ | **Dataset** | **Control** | $i$ | **Algoritmo** | **Rank** | **p-value** | $\alpha/i$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | SSO-C | 3 | ***k*-means** | 3.63 | 1.29E-14 | 0.017 | | Emax | 3 | **SSO** | 3.18 | 1.48E-15 | 0.017 |
| Dataset S. 2 | (Rank: | 2 | **Emax** | 2.49 | 9.95E-04 | 0.025 | Vowel | (Rank: | 2 | **SSO-C** | 2.96 | 1.03E-12 | 0.025 |
| | 1.64) | 1 | **SSO** | 2.24 | 2.00E-02 | 0.050 | | 1.12) | 1 | ***k*-means** | 2.74 | 3.51E-10 | 0.050 |
| | SSO-C | 3 | **SSO** | 3.08 | 4.04E-05 | 0.017 | | Emax | 3 | ***k*-means** | 3.17 | 4.30E-17 | 0.017 |
| Dataset S. 3 | (Rank: | 2 | ***k*-means** | 2.76 | 4.00E-03 | 0.025 | Oil | (Rank: | 2 | **SSO-C** | 3.17 | 4.30E-17 | 0.025 |
| | 2.02) | 1 | Emax | 2.14 | 6.40E-01 | 0.050 | | 1.00) | 1 | **SSO** | 2.66 | 1.28E-10 | 0.050 |
| | Emax | 3 | **SSO** | 3.23 | 1.59E-16 | 0.017 | | SSO | 3 | ***k*-means** | 3.34 | 2.58E-19 | 0.017 |
| Dataset S. 4 | (Rank: | 2 | ***k*-means** | 3.22 | 2.20E-16 | 0.025 | Balance | (Rank: | 2 | **SSO-C** | 3.02 | 9.49E-15 | 0.025 |
| | 1.10) | 1 | **SSO-C** | 2.45 | 1.71E-07 | 0.050 | | 1.02) | 1 | **Emax** | 2.62 | 5.76E-10 | 0.050 |
| | Emax | 3 | **SSO** | 3.56 | 4.58E-11 | 0.017 | | Emax | 3 | **SSO** | 3.32 | 2.58E-19 | 0.017 |
| Dataset S. 5 | (Rank: | 2 | ***k*-means** | 2.54 | 8.00E-03 | 0.025 | Cancer | (Rank: | 2 | **SSO-C** | 3.23 | 5.78E-18 | 0.025 |
| | 1.86) | 1 | SSO-C | 2.04 | 4.80E-01 | 0.050 | | 1.00) | 1 | ***k*-means** | 2.45 | 1.96E-08 | 0.050 |
| | Emax | 3 | ***k*-means** | 2.87 | 3.16E-04 | 0.017 | | SSO | 3 | ***k*-means** | 3.09 | 6.54E-04 | 0.017 |
| Dataset S. 6 | (Rank: | 2 | **SSO** | 2.86 | 3.66E-04 | 0.025 | Wine | (Rank: | 2 | Emax | 2.47 | 3.10E-01 | 0.025 |
| | 1.94) | 1 | SSO-C | 2.33 | 1.30E-01 | 0.050 | | 2.21) | 1 | SSO-C | 2.23 | 9.30E-01 | 0.050 |
| | SSO-C | 3 | **SSO** | 3.51 | 2.76E-15 | 0.017 | | Emax | 3 | **SSO** | 3.92 | 3.39E-21 | 0.017 |
| Iris | (Rank: | 2 | ***k*-means** | 3.22 | 1.22E-11 | 0.025 | Glass | (Rank: | 2 | ***k*-means** | 2.66 | 4.87E-06 | 0.025 |
| | 1.47) | 1 | Emax | 1.80 | 2.00E-01 | 0.050 | | 1.48) | 1 | SSO-C | 1.94 | 7.00E-02 | 0.050 |

***Fuente:*** *León Malpartida et al., 2019.*

[6]Este valor no es la ponderación $\alpha$ del algoritmo SSO-C. El valor 0.05 es estándar en la comunidad científica para considerar una hipótesis aceptada.

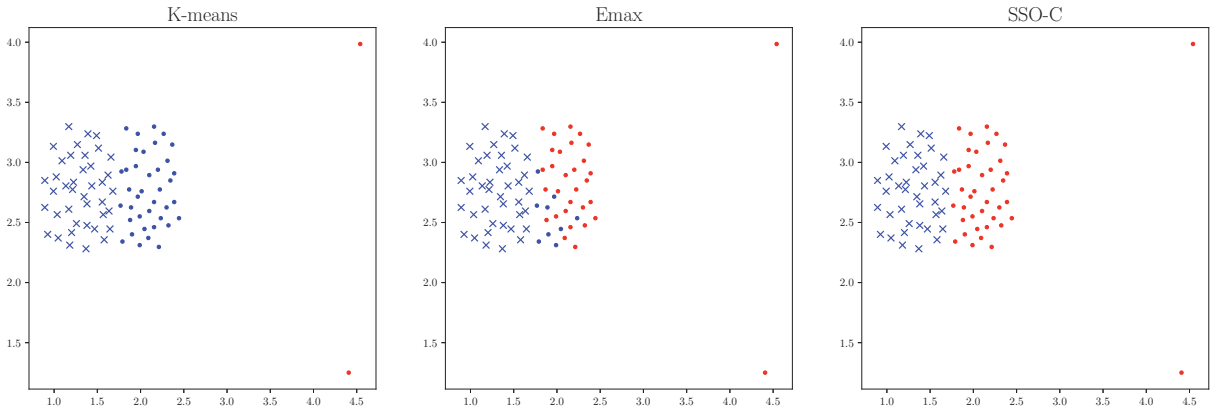
# Capítulo 4

# Discusión de resultados

Como se indicó previamente, para cada dataset, se ejecutó 50 veces cada algoritmo, esto junta 200 resultados por dataset. Las ejecuciones fueron resumidas en los valores máximos (la mejor precisión de las ejecuciones), la mediana (o valor promedio) y la media. La información se presenta en la Tabla 3.3. Sobre esta, lo siguiente puede ser observado:

Debido a que el dataset sintético 1 tiene los clusters bien separados y definidos (ver la Figura 3.6), todos los algoritmos obtienen una precisión de 1 para este. Esto es, todos encuentran el clustering perfecto. En el dataset sintético 2, el algoritmo $k$-means obtiene una precisión promedio de 0.813, mientras que los demás algoritmos obtienen como valor máximo y mediana un clustering perfecto, el algoritmo SSO-C obtiene un valor perfecto en todas las ejecuciones. En el dataset sintético 3, nuevamente el algoritmo $k$-means obtiene las precisiones más bajas, seguido a este está el algoritmo SSO y luego Emax, siendo este último el de mejor ejecución media. El algoritmo SSO-C obtiene nuevamente los mejores valores máximos y mediana. La situación es similar con los datasets sintéticos 4 y 5. Los valores más bajos varían entre el algoritmo $k$-means y SSO. Mientras que los mejores máximos los tiene el algoritmo SSO-C y los mejores valores medianas y medias, el algoritmo Emax. En el dataset sintético 6 el algoritmo SSO-C tiene los mejores valores máximos, medianas y medias. Luego, en mediana y media, el algoritmo Emax va por debajo por pocas décimas. La Figura 4.1 muestra el comportamiento de los algoritmos $k$-means, Emax y SSO-C frente al Dataset sintético 2.

Figura 4.1: Comportamiento de $k$-means, Emax y SSO-C frente al Dataset sintético 2.




***Fuente:*** *Elaboración propia.*

Como se puede observar, en los datasets sintéticos (con outliers), el algoritmo SSO-C obtiene siempre los mejores máximos. Los mejores valores medianas y medias se alternan entre el

algoritmo SSO-C y Emax. Los algoritmos $k$-means y SSO tienen los valores más bajos en casi todos los casos.

En el dataset Iris, el algoritmo SSO-C tiene los mejores valores máximos, medianas y medias. Emax tiene los segundos mejores valores medianas y medias. La situación es parecida en el dataset Vowel Indian, en el que SSO-C tiene los mejores valores máximos y Emax los mejores valores medianas y medias. En el dataset Crude Oil, el algoritmo Emax tiene los mejores valores máximos, medianas y medias, seguido por el algoritmo SSO-C por pocas décimas. En el dataset Balance Scale, el algoritmo SSO tiene los mejores valores máximos, medianas y medias, seguido por el algoritmo SSO-C. Nuevamente, en el dataset Breast Cancer, el algoritmo Emax tiene los mejores valores máximos, medianas y medias, seguido por el algoritmo SSO-C. En el dataset Wine, el algoritmo SSO-C tiene los mejores valores máximos y medianas, mientras que $k$-means tiene el mejor valor mediana. Este es el único caso en el que el algoritmo $k$-means tiene el mejor valor tomando en cuenta la mediana de las ejecuciones. En el último dataset; Glass, el algoritmo SSO-C tiene los mejores valores máximos y medias, mientras que Emax tiene el mejor valor mediana.

Se puede ver que en los datasets reales, los algoritmos SSO-C y Emax suelen tener los mejores valores máximos. Los mejores valores medianas y medias suele tenerlos el algoritmo Emax. Sin embargo, los resultados no son tan específicos como en el caso de los datasets sintéticos, siendo la naturaleza de cada problema real muy variada. Tomando en cuenta todos los datasets (sintéticos y de la vida real) el algoritmo SSO-C suele tener el mejor valor máximo, mientras que el algoritmo Emax suele tener los mejores valores medianas y medias.

A pesar de los resultados mostrados en la Tabla 3.3. Los resultados completos (50 ejecuciones para cada algoritmo sobre cada dataset) fueron sometidos a los test estadísticos ya especificados. Primero, el test de Friedman logró rechazar la hipótesis nula sobre todos los datasets exceptuando el primero ya que el desempeño de todos los algoritmos frente a este fue perfecto. Este resultado indica que, en todos los demás casos, existe al menos un algoritmo que sobresale del resto. Con este resultado, se realizó el test de Holm para los resultados completos.

Es importante notar que Emax es el algoritmo de control en 7 de 12 casos, siendo la mejor opción en más de la mitad de los casos. En estas 7 veces, es posible decir con suficiente evidencia que Emax es mejor que $k$-means y SSO. Sin embargo, solo en 4 casos Emax es significativamente mejor que SSO-C. En los demás casos, SSO-C es el algoritmo de control en 3 ocasiones y es posible decir que éste es mejor que $k$-means y SSO. Sin embargo, solo es significativamente mejor que Emax en 1 de 3 veces.

Dados estos resultados, se puede concluir con significación estadística que que SSO-C y Emax obtienen mejores resultados que $k$-means y SSO. Siendo Emax la mejor opción en más de la mitad de los casos y SSO-C en la mayoría de los restantes. Sin embargo, no es posible determinar estadísticamente el mejor algoritmo entre SSO-C y Emax.

# Conclusiones

1. La conclusión en base al objetivo general es:

   Se logró proponer un algoritmo de optimización multiobjetivo para el problema *center-based clustering* llamado SSO-C, que alcanza una alta precisión en conjuntos de datos con y sin *outliers*.

   Se demostró con suficiente fundamento matemático que el algoritmo SSO-C pertenece al modelo *center-based* y es capaz de correctamente crear una partición de un conjunto de datos en un tiempo de ejecución para el peor caso en $O\left(nkd\left(k^2 + n_{it}\lambda\right) + n_{it}n_s^2 d)\right)$ con el fin de hacer *clustering*.

   Se comprobó estadísticamente que el algoritmo SSO-C alcanza una precisión superior al algoritmo $k$-means y SSO en conjuntos con y sin *outliers* con un nivel de significancia de 5 %, valor estándar en la comunidad científica. El algoritmo SSO-C se mostró capaz de encontrar los mejores *clusterings* en casi todos los casos con un número suficiente de ejecuciones.

2. La conclusión en base al primer objetivo específico es:

   La formulación del problema matemático del cual se desarrolla el algoritmo SSO-C es más robusta que la del algoritmo $k$-means. También, existe un rango continuo de valores en los que la formulación puede variar su robustez.

3. La conclusión en base al segundo objetivo específico es:

   El criterio de Pareto brinda una justificación razonable para solucionar el problema formulado como un problema de optimización global ponderando funciones sobre la optimización multiobjetivo.

   La búsqueda local realizada por un algoritmo de aproximación–2 para un problema de optimización combinatoria relacionado como proceso inicial del algoritmo SSO-C ofrece candidatos importantes con garantías teóricas de no haber sido afectadas por *outliers*.

   La equivalencia de una función discreta a una función real continua es una ventaja importante para el proceso de búsqueda local y global de la mejor solución.

   Debido a que el algoritmo SSO-C busca optimizar globalmente una función, este necesita tener definido un número máximo de iteraciones o cualquier otro criterio de parada.

4. La conclusión en base al tercer objetivo específico es:

   El algoritmo propuesto para el caso más robusto de la función objetivo llamado Emax, demuestra ser una heurística competitiva con el algoritmo SSO-C, superándolo en más de la mitad de los casos.

   Se demostró que el algoritmo Emax genera particiones de conjuntos en tiempo de ejecución para el peor caso de $\Theta\left(\gamma\lambda nkd\right)$.

A diferencia del algoritmo SSO-C, se demostró formalmente la convergencia del algoritmo Emax en un número finito de iteraciones.

5. La conclusión en base al cuarto objetivo específico es:

   La generación de conjuntos de datos con presencia de *outliers* requirió de variables aleatorias pertenecientes a las distribuciones Gaussiana multivariable y Gamma multivariable.

   El análisis estadístico mostró la superioridad de los algoritmos SSO-C y Emax frente a los algoritmos $k$-means y SSO (el algoritmo adaptado para *clustering*).

   Cuando todos los algoritmos son comparados, el algoritmo SSO-C se mostró como el más apropiado para encontrar valores máximos; sin embargo, cuando se trata de valores medianas y medias, el algoritmo Emax encuentra los mejores.

   El análisis estadístico entre los algoritmos SSO-C y Emax muestran que Emax es un algoritmo superior en 7 de 12 casos, en los que, se encuentra significación estadística frente a SSO-C en solamente 4 casos; motivo por el cual no es posible determinar el mejor algoritmo entre ambos.

# Trabajos futuros

1. Se sabe que el problema $k$-means es NP-Hard; sin embargo, no se sabe la clase de problemas a la que pertenece el problema del que deriva el algoritmo SSO-C. Del mismo modo, no se sabe a qué clase de problemas pertenece el problema del que deriva el algoritmo Emax. Se deja como trabajo futuro analizar la clase de problemas al que pertenecen, se teoriza que ambos problemas son NP-Hard.

2. Se aboradará también el diseño de un algoritmo que tenga buen desempeño en formas de conjuntos más sofisticadas no elípticas.

3. Es recomendable, utilizando técnicas más avanzadas, generar datasets más complejos con el fin de evaluar los límites de los algoritmos propuestos.

4. El método de *interior-point* ha demostrado ser el estado del arte frente a problemas de búsqueda similares al problema *geometric median*. Se utilizará el algoritmo de (Cohen et al., 2016) para evaluar si el cambio es suficientemente beneficioso para la el algoritmo Emax.

# Bibliografía


Agarwal, P. K. & Procopiuc, C. M. (2002). Exact and Approximation Algorithms for Clustering. *Algorithmica*, *33*(2), 201-226. doi:10.1007/s00453-001-0110-y

Aggarwal, C. C. & Reddy, C. K. (2013). *Data Clustering: Algorithms and Applications* (1st). Chapman & Hall/CRC.

Bajaj, C. (1988). The algebraic degree of geometric optimization problems. *Discrete & Computational Geometry*, *3*(2), 177-191. doi:10.1007/BF02187906

Balcan, M. F. & Liang, Y. (2012). Clustering under Perturbation Resilience. En A. Czumaj, K. Mehlhorn, A. Pitts & R. Wattenhofer (Eds.), *Automata, Languages, and Programming* (pp. 63-74). Berlin, Heidelberg: Springer Berlin Heidelberg.

Balcan, M.-F., Haghtalab, N. & White, C. (2016). k-Center Clustering Under Perturbation Resilience. En I. Chatzigiannakis, M. Mitzenmacher, Y. Rabani & D. Sangiorgi (Eds.), *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)* (Vol. 55, 68:1-68:14). Leibniz International Proceedings in Informatics (LIPIcs). doi:10.4230/LIPIcs.ICALP.2016.68

Bilu, Y. & Linial, N. (2012). Are Stable Instances Easy? *Comb. Probab. Comput. 21*(5), 643-660. doi:10.1017/S0963548312000193

Brown, R. & Porter, T. (1995). The Methodology of Mathematics. *The Mathematical Gazette*, *79*(485), 321-334. doi:10.2307/3618304

Cohen, M. B., Lee, Y. T., Miller, G., Pachocki, J. & Sidford, A. (2016). Geometric Median in Nearly Linear Time. En *Proceedings of the Forty-eighth Annual ACM Symposium on Theory of Computing* (pp. 9-21). STOC '16. doi:10.1145/2897518.2897647

Cook, S. A. (1971). The Complexity of Theorem-proving Procedures. En *Proceedings of the Third Annual ACM Symposium on Theory of Computing* (pp. 151-158). STOC '71. doi:10.1145/800157.805047

Cormen, T. H., Leiserson, C. E., Rivest, R. L. & Stein, C. (2009). *Introduction to Algorithms, Third Edition* (3rd). The MIT Press.

Cuevas, E., Díaz Cortés, M. A. & Oliva Navarro, D. A. (2016). A Swarm Global Optimization Algorithm Inspired in the Behavior of the Social-Spider. En *Advances of Evolutionary Computation: Methods and Operators* (pp. 9-33). doi:10.1007/978-3-319-28503-0_2

DeGroot, M. & Schervish, M. (2012). *Probability and Statistics*. Addison-Wesley. Recuperado desde https://books.google.com.pe/books?id=4TlEPgAACAAJ

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, *7*, 1-30.

Derrac, J., García, S., Molina, D. & Herrera, F. (2011). A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, *1*(1), 3-18.

Du, K.-L. & Swamy, M. N. S. (2016). Introduction. En *Search and Optimization by Metaheuristics: Techniques and Algorithms Inspired by Nature* (pp. 1-28). doi:10.1007/978-3-319-41192-7_1

Dua, D. & Karra Taniskidou, E. (2017). UCI Machine Learning Repository. University of California, Irvine, School of Information y Computer Sciences. Recuperado desde http://archive.ics.uci.edu/ml

Dunford, N. & Schwartz, J. (1971). *Linear Operators: Vol.: 3. : Spectral Operators*. Pure and Applied Mathematics - Wiley. Wiley-Interscience. Recuperado desde https://books.google.com.pe/books?id=PGNHtAEACAAJ

Gan, G., Ma, C. & Wu, J. (2007). *Data clustering - theory, algorithms, and applications*. SIAM.

García, S. & Herrera, F. (2008). An Extension on "Statistical Comparisons of Classifiers over Multiple Data Sets" for all Pairwise Comparisons. *Journal of Machine Learning Research*, *9*, 2677-2694.

Gonzalez, T. F. (1985). Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, *38*, 293-306. doi:10.1016/0304-3975(85)90224-5

Hautamäki, V., Cherednichenko, S., Kärkkäinen, I., Kinnunen, T. & Fränti, P. (2005). Improving K-Means by Outlier Removal. En H. Kalviainen, J. Parkkinen & A. Kaarna (Eds.), *Image Analysis* (pp. 978-987). Berlin, Heidelberg: Springer Berlin Heidelberg.

Hillier, F. S. & Lieberman, G. J. (2001). *Introduction to Operations Research* (Seventh). New York, NY, USA: McGraw-Hill.

Huang, Z. (1998). Extensions to the k-Means Algorithm for Clustering Large Data Sets with Categorical Values. *Data Min. Knowl. Discov. 2*(3), 283-304. doi:10.1023/A:1009769707641

Karaboga, D. & Basturk, B. (2007). Artificial Bee Colony (ABC) Optimization Algorithm for Solving Constrained Optimization Problems. En *Proceedings of the 12th International Fuzzy Systems Association World Congress on Foundations of Fuzzy Logic and Soft Computing* (pp. 789-798). IFSA '07. doi:10.1007/978-3-540-72950-1_77

Kennedy, J. & Eberhart, R. C. (1995). Particle swarm optimization. En *Proceedings of the IEEE International Conference on Neural Networks* (pp. 1942-1948).

Krarup, J. & Vajda, S. (1997). On Torricelli's geometrical solution to a problem of Fermat. *IMA Journal of Management Mathematics*, *8*(3), 215-224. doi:10.1093/imaman/8.3.215. eprint: /oup/backfile/content_public/journal/imaman/8/3/10.1093/imaman/8.3.215/2/8-3-215.pdf

León Malpartida, J., Chullo Llave, B., Enciso Rodas, L. & Soncco Álvarez, J. L. (2019). A multiobjective optimization algorithm for center-based clustering. En *2019 XLV Latin American Computer Conference (CLEI) (accepted)*. Symposium on Theory of Computation, Panama City, Panama.

León, J. (2019a). Emax. (Version 1.0.0). Zenodo (MIT License). doi:10.5281/zenodo.2858409

León, J. (2019b). SSO-C. (Version 1.0.0). Zenodo (MIT License). doi:10.5281/zenodo.3237925

Mahajan, M., Nimbhorkar, P. & Varadarajan, K. (2012). The planar k-means problem is NP-hard. *Theoretical Computer Science*, *442*, 13-21. Special Issue on the Workshop on Algorithms and Computation (WALCOM 2009). doi:10.1016/j.tcs.2010.05.034

Rennie, B. & Dobson, A. (1969). On stirling numbers of the second kind. *Journal of Combinatorial Theory*, *7*(2), 116-121. doi:10.1016/S0021-9800(69)80045-1

Rosen, K. H. (2002). *Discrete Mathematics and Its Applications* (5th). McGraw-Hill Higher Education.

Sipser, M. (2006). *Introduction to the Theory of Computation* (Second). Course Technology.

Stewart, J. (2012). *Calculus : early transcendentals*. Belmont, Cal.: Brooks/Cole, Cengage Learning.

Vera-Olivera, H., Soncco-Álvarez, J. L. & Enciso-Rodas, L. (2016). Social Spider Algorithm Approach for Clustering. En *Proceedings of the 3rd Annual International Symposium on Information Management and Big Data - SIMBig 2016, Cusco, Peru, September 1-3, 2016.* (pp. 114-121). Recuperado desde http://ceur-ws.org/Vol-1743/paper14.pdf

Vinh, N. X., Epps, J. & Bailey, J. (2009). Information Theoretic Measures for Clusterings Comparison: Is a Correction for Chance Necessary? En *Proceedings of the 26th Annual International Conference on Machine Learning* (pp. 1073-1080). ICML '09. doi:10.1145/1553374.1553511

Weil, A. (1974). *Basic number theory.* Grundlehren der mathematischen Wissenschaften. Springer.

Weiszfeld, E. (1937). Sur le point pour lequel la Somme des distances de n points donnes est minimum. *Tohoku Mathematical Journal, First Series*, *43*, 355-386.

Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C. M. & da Fonseca, V. G. (2003). Performance assessment of multiobjective optimizers: an analysis and review. *IEEE Transactions on Evolutionary Computation*, *7*(2), 117-132. doi:10.1109/TEVC.2003.810758

# Apéndice A:

# Publicación

León Malpartida, J., Chullo Llave, B., Enciso Rodas, L. & Soncco Álvarez, J. L. (2019). A multi-objective optimization algorithm for center-based clustering. En *2019 XLV Latin American Computer Conference (CLEI) (accepted). Symposium on Theory of Computation. Panama City, Panama.*