UNIVERSIDAD DE GRANADA

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA INFORMÁTICA

Mejorando metaheurísticas para el problema del clustering semi-supervisado

Autor
Teresa Córdoba Lillo

Directores
Daniel Molina Cabrera
Francisco Javier Rodríguez Díaz

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, Junio de 2025

Mejorando metaheurísticas para el problema del clustering semi-supervisado

Autor
Teresa Córdoba Lillo

Directores
Daniel Molina Cabrera
Francisco Javier Rodríguez Díaz

Mejorando metaheurísticas para el problema del clustering semi-supervisado

Teresa Córdoba Lillo

Palabras clave: *clustering* semi-supervisado, metaheurística, restricciones Must-Link/Cannot-Link, Evolución Diferencial, algoritmo memético

Resumen

El *clustering* es una técnica de aprendizaje no supervisado que agrupa elementos según su similitud. El *clustering* semi-supervisado extiende esta idea incorporando una pequeña cantidad de conocimiento experto, a menudo en forma de restricciones *Must-Link* y *Cannot-Link*, que indican si dos elementos deben o no estar en el mismo grupo. Esta información permite obtener soluciones más coherentes, interpretables y de mayor calidad, sin necesidad de contar con datos completamente etiquetados. Por ello, se ha convertido en una herramienta útil con aplicación en diversos ámbitos como la biomedicina o la clasificación de textos, entre otros.

Recientemente se ha propuesto un algoritmo exacto para el *clustering* semi-supervisado con restricciones *Must-Link* y *Cannot-Link*, pero resulta inviable su uso con conjuntos de datos grandes debido a los altos tiempos de ejecución que requiere. Como alternativa, han surgido numerosos algoritmos aproximados, muchos de ellos basados en metaheurísticas, entre los que destaca un algoritmo memético reciente basado en Evolución Diferencial, menos costoso computacionalmente que el exacto, pero que no garantiza encontrar la solución óptima.

El objetivo principal de este trabajo es analizar y mejorar el rendimiento de dicho algoritmo memético. Se estudiará su comportamiento y se propondrán diversas modificaciones orientadas a mejorarlo, tanto en términos de eficiencia computacional como de calidad de las soluciones generadas. Además, se realizará un análisis experimental exhaustivo ejecutando los algoritmos en múltiples conjuntos de datos, lo que permitirá evaluar la calidad de las propuestas y compararlas con el algoritmo original.

Improving metaheuristics for the semi-supervised clustering problem

Teresa Córdoba Lillo

Keywords: semi-supervised clustering, metaheuristics, Must-Link/Cannot-Link constraints, Differential Evolution, memetic algorithm

Abstract

Clustering is an unsupervised learning technique that groups elements according to their similarity. Semi-supervised clustering extends this idea by incorporating a small amount of expert knowledge, often in the form of Must-Link and Cannot-Link constraints, which indicate whether or not two items should be in the same group. This information allows for more consistent, interpretable and higher quality solutions, without the need for fully labeled data. Therefore, it has become a useful tool with application in various fields such as biomedicine or text classification, among others.

Recently, an exact algorithm for semi-supervised clustering with Must-Link and Cannot-Link constraints has been proposed, but it is infeasible to use with large datasets due to the high execution times required. As an alternative, numerous approximate algorithms have emerged, many of them based on metaheuristics, including a recent memetic algorithm based on Differential Evolution, which is less computationally expensive than the exact one, but does not guarantee finding the optimal solution.

The main objective of this project is to analyse and improve the performance of this memetic algorithm. Its behaviour will be studied and several modifications aimed at improving it will be proposed, both in terms of computational efficiency and the quality of the solutions generated. In addition, an exhaustive experimental analysis will be carried out by running the algorithms on multiple data sets, which will allow us to evaluate the quality of the proposals and compare them with the original algorithm.

Yo, **Teresa Córdoba Lillo**, alumna de la titulación Grado en Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 52022806A, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Teresa Córdoba Lillo

Granada a 15 de junio de 2025.

D. **Daniel Molina Cabrera**, Profesor del Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada.

D. **Francisco Javier Rodríguez Díaz**, Profesor del Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado ***Mejorando metaheurísticas para el problema del clustering semi-supervisado***, ha sido realizado bajo su supervisión por **Teresa Córdoba Lillo**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 15 de junio de 2025

Los directores:

Daniel Molina Cabrera **Francisco Javier Rodríguez Díaz**

Agradecimientos

A mi familia, y en especial a mis padres por su apoyo incondicional a lo largo de estos años, por enseñarme el valor del esfuerzo y por estar siempre ahí, celebrando mis logros y animándome en cada paso del camino.

A mis tutores, por su guía, paciencia y dedicación durante el desarrollo de este trabajo.

A Laura, por acompañarme en este proceso, confiar en mí y animarme en los momentos más difíciles.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Conceptos básicos y motivación

El agrupamiento (*clustering*) busca agrupar un conjunto de instancias en grupos llamados *clusters*, de forma que las instancias que pertenecen a un mismo *cluster* sean más similares entre sí que con las que pertenecen a *clusters* distintos. De forma habitual, el *clustering* se utiliza cuando no se tiene información *a priori* sobre la pertenencia de los datos a clases predefinidas. Por esta razón, tradicionalmente se considera una técnica propia del aprendizaje no supervisado. Sin embargo, en ocasiones podemos disponer de cierta información parcial o conocimiento experto que podemos utilizar para guiar la búsqueda de soluciones hacia soluciones más significativas y más cercanas a la partición real de los datos. De esta forma, estamos incorporando un cierto grado de supervisión a la tarea, lo que da lugar al *clustering* semi-supervisado.

El *clustering* semisupervisado presenta ventajas claras frente al enfoque no supervisado, ya que al aprovechar la información parcial puede mejorar la calidad de las soluciones obtenidas y hacerlas más coherentes con el conocimiento experto del dominio del problema [1]. Además, favorece la interpretabilidad, ya que las soluciones reflejan las relaciones conocidas o deseadas dentro de los datos.

A lo largo de los años, el *clustering* semisupervisado se ha utilizado en una amplia variedad de campos. Entre sus aplicaciones se encuentran el análisis de datos biológicos, la segmentación de imágenes, vídeo y sonido, la clasificación de textos y el análisis de datos en la *web* [2]. Incluso se ha utilizado en tareas como la detección de carriles en carreteras a partir de datos GPS de coches. Esta última fue la primera aplicación propuesta por Wagstaff et al. [3]. Estos ejemplos muestran su versatilidad y utilidad, especialmente en contextos donde una supervisión total no es viable, pero

sí se puede disponer de conocimiento parcial.

Existen numerosos modelos de *clustering*, cada uno con diferentes enfoques y criterios de similitud, así como distintas formas de incorporar conocimiento experto al proceso [4]. En este trabajo nos centraremos en el *Euclidean Minimum Sum of Squares Clustering* (MSSC) con restricciones *Must-Link* (ML) y *Cannot-Link* (CL), también conocido como *constrained clustering*. En este enfoque, las instancias se representan como vectores del espacio euclídeo d-dimensional $\mathbb{R}^d$, y se busca encontrar una partición de los datos en *clusters* de forma que se minimice la suma de las distancias euclídeas al cuadrado entre cada instancia y el centroide del *cluster* al que pertenece, respetando las restricciones y siendo el centroide la media de todas las instancias asignadas a dicho *cluster*. Las restricciones ML indican que un par de instancias deben pertenecer al mismo *cluster*, mientras que las restricciones CL obligan a que estén en *clusters* distintos. Resolver este problema es computacionalmente complejo; si bien el MSSC no supervisado es NP-hard, como se demuestra en [5], resolver su variante con restricciones es al menos tan difícil como su versión no supervisada.

Debido a su complejidad, a pesar de que existen algoritmos exactos para el MSSC con restricciones, su aplicación en conjuntos de datos grandes no es factible. Es por ello por lo que han surgido en los últimos años multitud de algoritmos basados en metaheurísticas [6], que buscan encontrar una solución próxima a la óptima globalmente, pero en un tiempo mucho menor. Entre ellos, se encuentra el algoritmo S-MDEClust (Mansueto y Schoen, 2024) [7], que es un algoritmo memético basado en Evolución Diferencial. Este algoritmo, que describiremos en más detalle en el Capítulo 5, es comparado en el artículo original con un algoritmo *branch-and-cut exacto* [8] con buenos resultados: S-MDEClust consigue igualar al exacto en la calidad de las soluciones en términos de la función objetivo que se busca minimizar (la suma de distancias al cuadrado entre instancias y centroides) en la mayoría de los conjuntos de datos en los que se prueba, y además logra encontrar la solución en un tiempo bastante menor en promedio. Sin embargo, el tiempo de ejecución en ciertos casos sigue siendo prohibitivo, especialmente en aquellos conjuntos de datos con un gran número de instancias o alta dimensionalidad, donde el coste computacional aumenta significativamente.

En este trabajo buscamos proponer modificaciones en uno o varios componentes del algoritmo S-MDEClust con el propósito de mejorarlo, ya sea en términos de la calidad de las soluciones que encuentra, como en cuanto a eficiencia computacional.

1.2. Objetivos

El objetivo principal de este trabajo es estudiar y mejorar la optimización del *clustering* semi-supervisado con restricciones mediante el uso de metaheurísticas, centrándose en mejorar la eficiencia. En concreto, se parte de un algoritmo de referencia basado en una estrategia memética con Evolución Diferencial: S-MDEClust [7], sobre el que se proponen y evalúan distintas modificaciones encaminadas a mejorar su eficiencia y la calidad de las soluciones obtenidas.

En resumen, los objetivos de este trabajo son:

- Analizar en profundidad el funcionamiento del algoritmo de referencia: S-MDEClust.
- Proponer mejoras que incrementen su rendimiento computacional y la calidad de las particiones generadas.
- Diseñar e implementar una alternativa metaheurística con un enfoque distinto.
- Evaluar experimentalmente las propuestas utilizando múltiples conjuntos de datos.
- Comparar los resultados obtenidos frente al algoritmo de referencia y extraer conclusiones sobre su utilidad práctica.

Capítulo 2

Planificación y presupuesto

En este capítulo se aborda la planificación seguida para el desarrollo del proyecto, desglosando el trabajo realizado en tareas y detallando el número de horas dedicadas a cada una. A partir de esta información, se realiza una estimación del presupuesto total del proyecto, teniendo en cuenta el tiempo total invertido.

2.1. Tareas realizadas

A continuación se detallan las tareas realizadas.

- **Comprensión del problema**: Revisión del material inicial proporcionado por los tutores, así como la búsqueda y lectura de información complementaria con el objetivo de comprender el problema del *clustering* semi-supervisado.
- **Búsqueda de información**: Búsqueda y revisión artículos, publicaciones y otros documentos necesarios para la realización del proyecto.
- **Estudio de los algoritmos de referencia**: análisis y estudio detallado del funcionamiento de los algoritmos usados como base comparativa en el proyecto.
- **Implementación**: Incluye tanto la implementación de los componentes de la propuesta, como el desarrollo de otros *scripts* necesarios (automatización de ejecuciones, creación de gráficas y tablas, generación de restricciones y formateo de datasets).
- **Obtención de los resultados**: Ejecución de los algoritmos de referencia y de las propuestas implementadas sobre los distintos conjuntos de datos.

- **Análisis de los resultados**: Análisis e interpretación de los resultados obtenidos.
- **Realización de la memoria**: Redacción de la memoria y revisión de la misma.
- **Reuniones con los tutores**: Reuniones casi semanales en las que se discutieron las modificaciones a implementar, los resultados obtenidos y en las tomaron decisiones clave del proyecto, como la elección de los conjuntos de datos, las métricas de evaluación y otros aspectos.

2.2. Planificación

A continuación se muestra una tabla con el tiempo dedicado a cada una de las tareas especificadas en el apartado anterior.

Tarea	**Duración (horas)**
Comprensión del problema	10
Búsqueda de información	20
Estudio de los algoritmos de referencia	15
Implementación	120
Obtención de los resultados	191.63
Análisis de los resultados	10
Realización de la memoria	150
Reuniones con los tutores	15
TOTAL	340 + 191.63

Tabla 2.1: Tiempo dedicado

El tiempo total dedicado al proyecto ha sido 340 horas de trabajo, más 191.63 horas de cómputo para la obtención de los resultados, lo que hace un total de 531.63 horas.

La planificación seguida se detalla en la Figura 2.1 haciendo uso de un diagrama de Gantt, que permite visualizar de forma clara la distribución temporal de las tareas.

Como podemos ver, hay varias tareas que se han realizado en paralelo, como la implementación, la obtención de resultados y su análisis, así como la búsqueda de información necesaria para la realización del proyecto y las reuniones periódicas con los tutores.

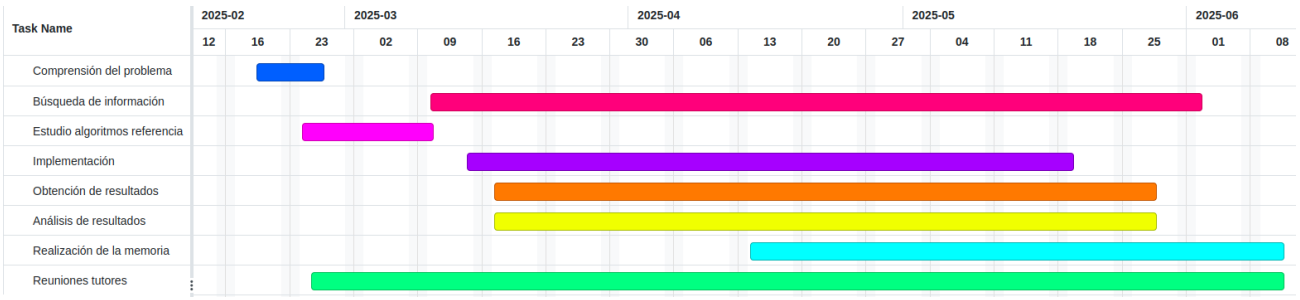


Figura 2.1: Planificación seguida

2.3. Presupuesto

Para realizar una estimación económica del proyecto, debemos tener en cuenta tres aspectos principales: el precio de la mano de obra, el precio de cómputo y el precio del ordenador utilizado para realizar el proyecto.

El precio de la mano de obra se ha estimado en 20€/hora. Por otro lado, el ordenador utilizado ha sido un ASUS TUF Dash F15, con un procesador Intel Core i7 12650H y 16 GB de RAM. El precio dicho portátil actualmente es de 1100 €. Para estimar el coste asociado al proyecto, se ha supuesto una vida útil de cinco años para el portátil, y dado que el desarrollo ha durado aproximadamente un año, se ha considerado un 20 % del precio total, lo que equivale a 220 €.

Para calcular el coste del precio de cómputo, se ha utilizado la calculadora de precios de *Amazon Web Services* (AWS). Al seleccionar el servicio EC2 e introducir las especificaciones del equipo utilizado, se obtiene que la instancia más similar es la c6g.4xlarge, con un coste por hora bajo demanda de 0.51 €.

Con estos datos, ya es posible calcular el presupuesto final del proyecto:

$$20\frac{€}{h} \cdot 340h + 0.51\frac{€}{h} \cdot 191.63h + \frac{1100€}{5} = 7117.73€$$

Concepto	**Precio Base**	**Duración**	**Precio Total**
Mano de obra	20 €/hora	340 horas	6 800 €
Cómputo	0.51 €/hora	191.63 horas	97.73 €
Ordenador	1 100 €	1 año de 5	220 €
Total			**7117.73 €**

Tabla 2.2: Estimación del presupuesto del proyecto

Finalmente, el presupuesto del proyecto queda fijado en 7117.73 €.

Capítulo 3

Revisión de la literatura

En los últimos años, el interés por el problema del *clustering* semi-supervisado con restricciones ha crecido notablemente, como muestra la Figura 3.1. Se ha producido un aumento en el número de publicaciones por año, de menos de 10 al principio de los 2000s a casi 100 en los últimos años, que refleja la importancia y la investigación creciente sobre este tema. Como vimos en el capítulo anterior, incorporar conocimiento experto al *clustering* permite obtener particiones más útiles y ajustadas a las necesidades reales, lo que ha motivado el desarrollo de una amplia variedad de algoritmos.

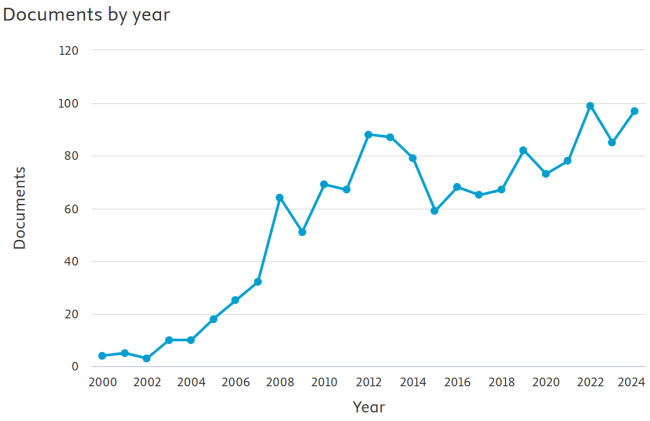


Figura 3.1: Número de publicaciones anuales sobre el clustering semi-supervisado con restricciones extraído de Scopus.

En este capítulo haremos un repaso por algunos de los algoritmos que se han propuesto para el *clustering* semi-supervisado con restricciones.

3.1. Algoritmos no basados en metaheurísticas

Multitud de los algoritmos existentes para *clustering* semi-supervisado son adaptaciones del clásico algoritmo para *clustering* no supervisado K-means (MacQueen, 1967) [9]. En este grupo se encuentra uno de los primeros algoritmos propuestos para este problema: COP-KMEANS (Wagstaff et al. 2001) [3]. COP-KMEANS parte de una solución inicial y repite los siguientes dos pasos hasta alcanzar la convergencia, es decir, hasta que deja de haber cambios en la solución.

1. Asignar cada instancia a un *cluster* de la solución.
2. Modificar los centroides de la solución para que sean la media de las instancias asignadas a él.

La diferencia con K-means es que en el paso 1, en lugar de asignar cada instancia al centroide más cercano, se asigna al más cercano tal que se satisfagan todas las restricciones ML y CL. Este algoritmo sigue un enfoque voraz (*greedy*), en el que las decisiones de asignación se toman de forma inmediata sin realizar *backtracking*. Esto hace que sea muy rápido pero presenta un gran inconveniente, y es que aunque exista una solución factible que satisfaga todas las restricciones, COP-KMEANS no garantiza encontrarla, ya que el éxito depende en gran medida del orden en el que se asignan las instancias. Además, incluso en los casos en los que logra encontrar una solución factible, esta podría corresponder únicamente a un óptimo local, lejos de la mejor solución posible en términos de la función objetivo.

Para tratar el problema de la dependencia del orden de asignación en COP-KMEANS, se han propuesto distintas variantes. Una de ellas es ICOP-k-means [10], que precalcula el orden de asignación basado en la certeza de cada instancia, estimada mediante técnicas de *clustering ensemble*. Así, se asignan primero las instancias más seguras, reduciendo el riesgo de violar restricciones en asignaciones posteriores.

CLC-Kmeans [11] es otra variante de COP-KMEANS que también modifica el orden de asignación, agrupando las instancias relacionadas por restricciones CL y asignándolas de forma conjunta a *clusters* compatibles. De este modo, se minimizan los conflictos durante la asignación. Ambas propuestas mejoran la robustez de COP-KMEANS, incrementando la probabilidad de encontrar soluciones factibles.

Entre las variantes más recientes se encuentra BLPKM_{CC} [12], que aborda el paso de asignación 1 formulándolo como un problema de programación binaria, garantizando así el cumplimiento de todas las restricciones. Profundizaremos más en este enfoque en el Capítulo 5, ya que es empleado por el algoritmo memético de referencia S-MDEClust.

Los algoritmos mencionados hasta ahora tienen en común que buscan cumplir todas las restricciones de forma estricta. Sin embargo, también existen otros enfoques que permiten incumplir restricciones a cambio de una penalización en la función objetivo. Un ejemplo de esto es PCK-Means [13], que al asignar las instancias a los *clusters* en el paso 1 busca minimizar una versión modificada de la función objetivo del MSSC (suma de distancias euclídeas al cuadrado entre instancias y centroides), a la que se añade un término de penalización que depende del número de restricciones incumplidas.

Por otro lado, también se han propuesto métodos exactos para resolver el problema del *clustering* semi-supervisado con restricciones. El primero de ellos fue propuesto por Xia en 2009 [14], que es una extensión para incluir restricciones ML y CL del método exacto para MSSC propuesto unos años antes en [15]. Este método logra encontrar la solución óptima en conjuntos de datos muy pequeños, aproximadamente de 25 instancias según Aloise en [16], aunque también puede proporcionar soluciones aproximadas deteniendo el algoritmo antes de alcanzar el óptimo global.

Posteriormente, en 2014, Babaki, Guns y Nijssen propusieron un enfoque basado en generación de columnas [17], donde el problema se resuelve de forma iterativa añadiendo progresivamente variables (columnas) que tienen el potencial de mejorar la solución actual. Este método permite manejar restricciones ML y CL, así como restricciones más generales, aunque su aplicación práctica se limita a conjuntos de datos de menos de 200 instancias.

Más tarde, en 2016, Guns et al. propusieron el algoritmo CPRBBA [18], que sigue una estrategia *branch-and-bound*. CPRBBA también permite optimizar varios objetivos a la vez, como minimizar la función objetivo del MSSC y maximizar la separación entre *clusters*, pero, al igual que métodos anteriores, su aplicabilidad está restringida a problemas de tamaño reducido, de alrededor de 200 instancias.

Recientemente, en 2022, Piccialli et al. propusieron el algoritmo PC-SOS-SDP [8]. Este algoritmo ha supuesto un gran avance en cuanto a métodos exactos para el MSSC semi-supervisado. Se trata de un algoritmo *branch-and-cut* que, para el cálculo de la cota inferior, utiliza una relajación por programación semidefinida (SDP) del MSSC y para la cota superior el algoritmo BLPKM_{CC} [12] mencionado con anterioridad, con una inicialización más sofisticada. Este algoritmo ha demostrado ser significativamente más eficiente que sus predecesores, siendo capaz de encontrar la solución óptima en conjuntos de datos de hasta 800 instancias, con más de 20000 dimensiones y un número de restricciones aproximadamente igual a la mitad del tamaño del conjunto de datos. Además, destaca por su buena escalabilidad con respecto a la dimensionalidad, manteniendo su eficacia incluso en problemas de alta dimensionalidad donde otros métodos exactos no resultan prácticos.

3.2. Algoritmos basados en metaheurísticas

A pesar de los avances logrados con los métodos exactos, su aplicabilidad práctica sigue estando limitada a problemas de tamaño reducido debido a su elevado coste computacional. Por ello, se han desarrollado múltiples algoritmos aproximados, como los que hemos visto al inicio del capítulo. Dentro de este enfoque aproximado, una parte importante de las propuestas se basa en metaheurísticas,

Las metaheurísticas son algoritmos de optimización de propósito general que, en muchas ocasiones, están inspirados en procesos naturales, como la evolución biológica, el comportamiento social de animales o fenómenos físicos. Su objetivo principal es encontrar soluciones de buena calidad en un tiempo reducido mediante un equilibrio entre la exploración del espacio de soluciones (buscar en regiones diversas) y la explotación (intensificar la búsqueda en áreas del espacio de soluciones prometedoras).

Un ejemplo de metaheurística para *clustering* con restricciones ML y CL es el algoritmo MCLA [19] (Vu, Labroche y Bouchon-Meunier, 2009), extensión del algoritmo para *clustering Leader Ant* (LA) que se basa en el comportamiento de las colonias de hormigas. En este enfoque, cada hormiga artificial está caracterizada por un genoma, que corresponde a una instancia específica del conjunto de datos, y una etiqueta que identifica la colonia (*cluster*) a la que pertenece. En cada iteración, se elige aleatoriamente una hormiga que no haya sido ya asignada a una colonia y se calcula su distancia media (similitud) con un determinado número de hormigas de cada colonia viable, tomando en cuenta únicamente aquellas colonias que cumplen las restricciones. La hormiga se asigna a la colonia con la menor distancia media, siempre que esta distancia supere un umbral establecido. Si no se alcanza el umbral o no hay colonias existentes, la hormiga crea una nueva colonia y se convierte en su líder. En [19] también se proponen dos modificaciones del algoritmo con el mismo esquema pero adaptadas para tratar con otros tipos de restricciones.

Otro ejemplo de algoritmo basado en colonias de hormigas es CAC [20], basado en el algoritmo RWAC, inspirado en el comportamiento de las hormigas al buscar un lugar para dormir. Este algoritmo tiene en cuenta las restricciones, modificando fuerzas atractivas y repulsivas entre hormigas (instancias) involucradas en las restricciones.

Dentro de las metaheurísticas, los algoritmos genéticos (AGs) han demostrado ser especialmente efectivos para abordar problemas complejos de optimización como el *clustering* con restricciones. Los AGs, inspirados en la selección natural, trabajan con una población de soluciones candidatas y aplican operadores de cruce y mutación para generar nuevas soluciones que compiten con las existentes en la población para formar parte de esta.

Un ejemplo destacado es el algoritmo propuesto por Gribel et al. en 2022 [21], una adaptación de HG-MEANS [22], algoritmo considerado parte del estado del arte en MSSC no supervisado. Una característica innovadora de esta propuesta es la incorporación de un parámetro que mide la precisión de las restricciones, lo que permite gestionar posibles errores en su definición. Otro algoritmo destacado es S-MDEClust [7], una propuesta reciente que sigue una estrategia memética basada en Evolución Diferencial en la que profundizaremos más en el Capítulo 5.

Por otro lado, más recientemente, en 2020, González-Almagro et al. propusieron el algoritmo DILS_{CC} [23], que sigue una estrategia DILS, variante del método clásico de búsqueda local iterativa (*Iterative Local Search*, ILS). Esta estrategia tiene como objetivo explorar el espacio de búsqueda para encontrar la solución que optimice una función objetivo determinada, que en este caso es el producto de la suma de distancias media *intra-cluster* y el número de restricciones incumplidas por la solución.

A diferencia de los algoritmos ILS, los DILS mantienen dos soluciones en memoria en todo momento en lugar de una: la mejor solución (m_b) y la peor (m_w) de la iteración actual del algoritmo, evaluadas según la función objetivo. El algoritmo DILS_{CC} comienza inicializando m_b y m_w aleatoriamente. En cada iteración, se combinan ambas soluciones para generar una nueva solución, a la que posteriormente se aplica un operador de mutación y una fase de búsqueda local. Si la nueva solución generada (m_t) mejora a m_w, esta última se actualiza con m_t. A pesar de que el operador de mutación introduzca diversidad en la búsqueda, existe el riesgo de caer en un óptimo local tras un número reducido de iteraciones. Para tratar este problema, cuando se detecta estancamiento, medido en términos de la diferencia de función objetivo de m_b y m_w, se reinicializa m_w aleatoriamente manteniendo la mejor solución m_b. DILS_{CC} ha sido comparado con varios algoritmos del estado del arte, mostrando una alta similitud entre las particiones obtenidas y las particiones reales, gracias a su equilibrio entre exploración y explotación en la búsqueda de soluciones óptimas.

Esta revisión ofrece una visión general del estado actual del *clustering* semi-supervisado con restricciones, proporcionando una base útil para el desarrollo posterior del trabajo.

Capítulo 4

Formalización del problema del clustering semi-supervisado

El problema del agrupamiento por suma mínima de cuadrados (*minimum sum-of-squares clustering*, MSSC) con restricciones puede ser formulado como un problema de optimización global con restricciones, en el que se tiene como objetivo minimizar una función objetivo: la suma de las distancias euclídeas al cuadrado entre cada instancia y el centroide al que está asignado, asegurando el cumplimiento de las restricciones ML y CL.

Formalmente, dado un conjunto de datos con n instancias $D = \{p_1, p_2, ..., p_n\}$, donde p_i es un vector real de d dimensiones, es decir, $p_i \in \mathbb{R}^d \quad \forall i \in \{1, ..., n\}$, y unos conjuntos de restricciones ML y CL denotados por $\mathcal{ML}$, $\mathcal{CL} \subset \{1, ..., n\} \times \{1, ..., n\}$ respectivamente, resolver el problema de *clustering* consiste en encontrar una partición de los datos en K *clusters*, $C = \{C_1, C_2, ...C_K\}$.

En esta formulación, se asume que el número de *clusters* K es conocido de antemano. Sin embargo, en muchas aplicaciones prácticas, el número óptimo de *clusters* no se conoce *a priori*, y encontrar dicho valor constituye en sí mismo un problema complejo, que suele requerir técnicas adicionales, como métodos basados en información estadística. Estudiar estos métodos no es el objetivo de este trabajo, así que asumiremos en todo momento que K es conocido.

El centroide del *cluster* j, denotado $\mu_j \in \mathbb{R}^d$, se define como el promedio de las instancias asignadas a dicho *cluster*:

$$\mu_j = \frac{\sum_{i=1}^{n} x_{ij} p_i}{\sum_{i=1}^{n} x_{ij}} \tag{4.1}$$

donde, $x_{ij} \in \{0, 1\}$ son variables binarias tales que

$$x_{ij} = \begin{cases} 1 & \text{si la instancia } p_i \text{ está asignada al } \textit{cluster } j \\ 0 & \text{en caso contrario} \end{cases}$$

Se busca que la partición C encontrada minimice la siguiente función objetivo:

$$f(C) = \sum_{i=1}^{n} \sum_{j=1}^{K} x_{ij} \left\| p_i - \mu_j \right\|^2 \tag{4.2}$$

sujeto a las siguientes restricciones:

- Cada instancia debe pertenecer exactamente a un único *cluster*.

$$\sum_{j=1}^{K} x_{ij} = 1 \quad \forall i \in \{1, \ldots, n\} \tag{4.3}$$

- Cada *cluster* debe tener al menos una instancia asignada.

$$\sum_{i=1}^{n} x_{ij} \geq 1 \quad \forall j \in \{1, \ldots, K\} \tag{4.4}$$

- Si dos instancias están involucradas en la misma restricción *must-link*, deben pertenecer al mismo *cluster*.

$$x_{ik} = x_{jk} \quad \forall (i, j) \in \mathcal{ML},\ \forall k \in \{1, \ldots, K\} \tag{4.5}$$

- Si dos instancias están involucradas en la misma restricción *cannot-link*, no pueden pertenecer al mismo *cluster*.

$$x_{ik} + x_{jk} \leq 1 \quad \forall (i, j) \in \mathcal{CL},\ \forall k \in \{1, \ldots, K\} \tag{4.6}$$

Con esta formulación, el MSSC semi-supervisado es un problema de programación mixta, es decir, un problema de optimización que involucra tanto variables enteras (las variables $x_{ij} \in \{0, 1\}$, que indican la asignación de instancias a *clusters*) como variables reales continuas (los centroides $\mu_j \in \mathbb{R}^d$, que son vectores de d dimensiones).

Este problema y su versión no supervisada son NP-hard [24]. En parte, esta elevada complejidad se debe a dos factores: por un lado, las restricciones ML y CL se imponen como restricciones duras, es decir, se impone que la solución encontrada debe necesariamente respetar estas restricciones, lo

que reduce el espacio de soluciones factibles y complica significativamente la exploración de soluciones viables. Por otro lado, la función objetivo es no lineal y no convexa, y está caracterizada por la presencia de múltiples óptimos locales, lo que dificulta encontrar el óptimo global. Estas dificultades hacen que el MSSC semi-supervisado sea un problema extremadamente complejo de resolver de forma exacta.

Además, al imponer restricciones duras surge un problema adicional que no se daba en el MSSC no supervisado, y es que puede ocuirrir que no exista ninguna solución que satisfaga todas las restricciones. Determinar si dado un conjunto de datos y un conjunto de restricciones ML y CL, existe una solución que satisfaga todas las restricciones es a lo que se conoce como problema de factibilidad (referido en la literatura como *feasibility problem*), y fue demostrado en [25] que es NP-completo.

Debido a estas dificultades, gran parte de la investigación en *clustering* semi-supervisado se ha centrado en el desarrollo de algoritmos heurísticos y metaheurísticos capaces de encontrar soluciones de buena calidad en un tiempo razonable.

Capítulo 5

Algorimos de referencia

En este capítulo describiremos el algoritmo de referencia a partir del cual se desarrollarán las propuestas de mejora presentadas en los siguientes capítulos: el algoritmo S-MDEClust [7]. Asimismo, se presenta el algoritmo exacto PC-SOS-SDP [8], que se utilizará para obtener las soluciones óptimas.

5.1. S-MDEClust

S-MDEClust es una adaptación de un algoritmo para MSSC no supervisado de los mismos autores [26], para incorporar restricciones ML y CL. Como ya hemos comentado con anterioridad, S-MDEClust es un algoritmo memético. Este tipo de algoritmos son poblacionales, es decir, en lugar de tener una única solución, trabajan con un conjunto de soluciones que forman una población. En cada iteración del algoritmo se generan nuevas soluciones combinando de alguna forma las soluciones de la población actual y se evalúa si dichas soluciones deben entrar a formar parte de la población, reemplazando a alguna solución actual, o no. Además, antes de este proceso de reemplazamiento se puede aplicar un procedimiento de búsqueda local para refinar la solución, utilizando conocimiento específico del problema.

A continuación presentamos el esquema general del algoritmo S-MDEClust y posteriormente definiremos en detalle cada uno de sus componentes.

Algorithm 1 Algoritmo S-MDEC1ust

Require: conjunto de datos (*dataset*) $D = \{p_1, \ldots, p_n\} \subset \mathbb{R}^d$, número de *clusters* $K \in \mathbb{N}$, conjuntos de restricciones *must-link* y *cannot-link* $\mathcal{M}_L, \mathcal{C}_L \subset \{1, \ldots, n\} \times \{1, \ldots, n\}$, tamaño de la población $P \in \mathbb{N}$, $N_{MAX} \in \mathbb{N}$, $tol \in \mathbb{R}^+$.

Inicializar población $\mathcal{P} = \{S_1, \ldots, S_P\}$
Determinar $S^\star \in \mathcal{P}$ tal que $f(S^\star) \leq f(S), \forall S \in \mathcal{P}$
Inicializar $n^\star_{it} = 0$
while $(n^\star_{it} < N_{MAX}) \wedge (\sum_{s=1}^{P} \sum_{\bar{s}>s} |f(S_s) - f(S_{\bar{s}})| > tol)$ **do**
  for all $S_s \in \mathcal{P}$ **do**
    Seleccionar aleatoriamente $S_1, S_2, S_3 \in \mathcal{P}$, todos diferentes entre sí y distintos de S_s
    Aplicar operador de cruce con S_1, S_2, S_3 para generar un descendiente O_s
    Aplicar o no el operador de mutación a O_s para obtener $\tilde{O}_s$, con un probabilidad que disminuye a lo largo de las iteraciones
    Aplicar búsqueda local a $\tilde{O}_s$ para obtener una solución O'_s
    if $f(O'_s) < f(S_s)$ **then**
      $S_s = O'_s$
      if $f(O'_s) < f(S^\star)$ **then**
        $S^\star = O'_s$
        $n^\star_{it} = 0$
      else
        $n^\star_{it} = n^\star_{it} + 1$
      end if
    end if
  end for
end while
return $S^\star$

5.1.1. Representación de la solución

Cada individuo S de la población $\mathcal{P} = \{S_1, \ldots, S_P\}$ se representa mediante dos estructuras de datos:

- Vector de pertenencia $\phi^S \in \mathbb{N}^n$, tal que $\phi^S_i = k \Leftrightarrow x_{ik} = 1$, con $k \in \{1, ..., K\}$. Es decir, la posición i-ésima de este vector indica el número del *cluster* al que está asignado la instancia i-ésima.
- Matriz de coordenadas $\psi^S = [\mu_1, \mu_2, ..., \mu_K]^T \in \mathbb{R}^{K \times d}$, es decir, una matriz en la que la fila i-ésima contiene las coordenadas del *cluster* i-ésimo.

A continuación se muestra un ejemplo para ilustrar esto.

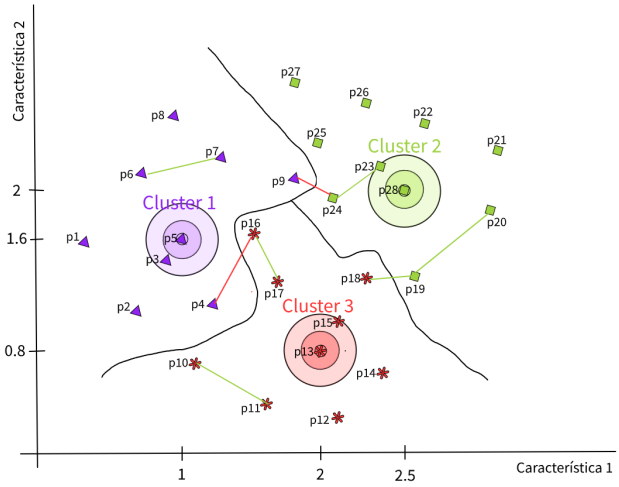


Figura 5.1: Ejemplo de 2 dimensiones con K=3 y n=28. Las restricciones ML se representan con líneas verdes y las CL con líneas rojas

Matriz de coordenadas

1	1.6
2.5	2
2	0.8

Vector de pertenencia

1	1	1	1	1	1	1	1	1	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2

Con esta notación, podemos expresar la función objetivo 4.2 de la siguiente manera.

$$f(S) = \sum_{i=1}^{n} \left\| p_i - \psi^S_{\phi^S_i} \right\|^2 . \tag{5.1}$$

Cabe destacar que cada estructura, por sí sola, permite caracterizar completamente la solución; sin embargo, es conveniente mantener las dos en memoria para reducir el tiempo de ejecución. A partir del vector de pertenencia ϕ, podemos obtener ψ en $O(nKd)$ recordando que el centroide de cada *cluster* es la media de las instancias que pertenecen a dicho *cluster*, como se indica en la Ecuación 4.1. Obtener ϕ desde la matriz de coordenadas ψ es más complejo. No basta con asignar cada instancia a su centroide más cercano, como sucede en el MSSC no supervisado; ahora deben considerarse las restricciones ML y CL.

Este hecho se puede apreciar en la figura del ejemplo anterior 5.1. Si nos fijamos en la instancia p_{16}, vemos que en lugar de asignarse al *cluster* 1, que sería el más cercano, se asigna al *cluster* 3 para respetar las restricciones.

5.1.2. Paso de asignación

Entendemos como paso de asignación a la asignación de las instancias a los centroides, es decir, a la obtención de ϕ^S a partir de ψ^S. En [7] se proponen dos métodos de asignación: una asignación exacta y una asignación *greedy* más eficiente pero que no garantiza el cumplimiento de todas las restricciones ML y CL. La asignación *greedy* será útil en partes intermedias del algoritmo, en las que no es indispensable que las soluciones cumplan todas las restricciones, para reducir el coste computacional.

Asignación exacta

El método de asignación exacta consiste en resolver el siguiente problema de optimización.

$$\text{mín} \sum_{i=1}^{N} \sum_{k=1}^{K} x_{ik} \|p_i - \psi_k^S\|^2, \quad \text{sujeto a } 4.3, 4.4, 4.5, 4.6 \tag{5.2}$$

Asignación *Greedy*

Primeramente, se dividen todas las instancias en grupos basados en las restricciones *must-link*, de forma que las instancias que están relacionadas mediante este tipo de restricciones se encuentren en el mismo grupo. Una vez establecidos estos grupos, para cada grupo G, se busca un *cluster* al que se asignarán todas las instancias de dicho grupo, de forma que no exista otro grupo $\hat{G}$ asignado al mismo *cluster* y al que pertenezca una instancia que tenga una restricción *cannot-link* con una instancia de G. Además, entre los *clusters* que cumplan esta condición, se elegirá aquel que minimice la suma de distancias de las instancias del grupo G a su centroide. Dependiendo del orden en el que se realicen las asignaciones de los grupos, podría ocurrir que, aunque exista una solución factible, no sea posible encontrar ningún *cluster* para algún grupo G sin incumplir ninguna restricción *cannot-link*, en cuyo caso se asignará al *cluster* más cercano a los puntos de G sin considerar ninguna restricción.

A continuación, se muestra el pseudocódigo de la asignación *greedy*.

Algorithm 2 Asignación *Greedy*

Require: *dataset* $D = \{p_1, \ldots, p_n\} \subset \mathbb{R}^d$, número de *clusters* $K \in \mathbb{N}$, conjuntos de restricciones *must-link* y *cannot-link* $\mathcal{ML}, \mathcal{CL} \subset \{1, \ldots, n\} \times \{1, \ldots, n\}$, matriz de centroides ψ^S.

Definir $\mathcal{G}_{\mathcal{ML}} = \{G \subset \{1, \ldots, N\} \mid \forall i, j \in G$,sujeto a $i \neq j, (i, j) \in \mathcal{ML}\}$

Inicializar $\mathcal{A} = \emptyset$
for all $G \in \mathcal{G}_{\mathcal{ML}}$ **do**
  Determinar

$$k_G \in \arg\min_k \sum_{i \in G} \|p_i - \psi_k^S\|^2$$

  sujeto a:

$$k \in \{1, \ldots, K\} \mid \nexists(\hat{G}, k) \in \mathcal{A} \text{ tal que } \exists(i_c, j_c) \in \mathcal{CL} \text{ con } i_c \in G \wedge j_c \in \hat{G}$$

  if k_G no está definido **then**
    Determinar

$$k_G \in \arg\min_{k \in \{1,\ldots,K\}} \sum_{i \in G} \|p_i - \psi_k^S\|^2$$

  end if
  $\mathcal{A} = \mathcal{A} \cup \{(G, k_G)\}$
  for all $i \in G$ **do**
    $\phi_i^S = k_G$
  end for
end for

return ϕ^S

Este algoritmo no garantiza que la solución encontrada satisfaga todas las restricciones. Sin embargo, se puede usar, en lugar del método exacto, durante el cruce y mutación. En estos pasos intermedios no es necesario que se satisfagan todas las restricciones y de esta forma se reduce el coste computacional.

5.1.3. Operador de cruce

Dada una solución $S_s = (\phi^{S_s}, \psi^{S_s}) \in \mathcal{P}$, con $s \in \{1, ..., n\}$. Se eligen tres soluciones aleatorias $S_1, S_2, S_3 \in \mathcal{P}$, de forma que sean diferentes entre sí y diferentes a S_s. La solución descendiente $O_s = (\phi^{O_s}, \psi^{O_s})$ se obtiene de la

siguiente manera.

$$\psi^{O_s} = \psi^{S_1} + F(\psi^{S_2} - \psi^{S_3}) \tag{5.3}$$

con $F \in (0.5, 0.8)$. ϕ^{O_s} se obtiene realizando un paso de asignación 5.1.2.

Algo a tener en cuenta es que, dada una solución S del problema de *clustering*, la permutación de las filas de su matriz de coordenadas ψ^S da lugar a otra solución que es totalmente equivalente. Por ello, antes de realizar el cruce 5.3 se debe de aplicar alguna estrategia para emparejar los centroides de las soluciones S_1, S_2 y S_3, como la descrita en [27].

5.1.4. Operador de mutación

El operador de mutación tiene como objetivo introducir diversidad en la población, evitando una convergencia prematura hacia soluciones subóptimas. Consiste en una relocalización aleatoria de uno de los centroides de la solución. A medida que avanzan las iteraciones, la probabilidad de aplicar esta operación disminuye progresivamente, ya que se pretende favorecer la intensificación en fases más avanzadas de la búsqueda, centrándose en refinar las soluciones prometedoras en lugar de explorar nuevas regiones del espacio de búsqueda.

Partiendo de la solución resultante del cruce $O_s = (\phi^{O_s}, \psi^{O_s})$, el operador de mutación se aplica de la siguiente forma:

1. Elegir un centroide $\bar{k} \in \{1, ...K\}$ con probabilidad uniforme y eliminarlo de la solución.

2. Asignar cada instancia p_i, con $i \in \{1, ..., n\}$ a uno de los $K-1$ *clusters* restantes, dando lugar al vector de pertenencia temporal $\tilde{\phi}$.

3. Se elige una instancia $p_{\bar{i}}$, $\bar{i} \in \{1, ..., n\}$ como nuevo centroide, basada en la siguiente probabilidad.

$$P(\bar{i}) = \left(\frac{1-\alpha}{n}\right) + \left(\frac{\alpha \cdot d_{\bar{i}}}{\sum_{j=1}^{n} d_j}\right) \tag{5.4}$$

   donde d_j es la distancia entre la instancia p_j y su centroide asignado, es decir, $d_j = ||p_j - \psi^{O_s}_{\tilde{\phi}_j}||$. Además, el parámetro $\alpha \in [0, 1]$ juega un papel fundamental en la elección del nuevo centroide. Cuando $\alpha = 0$, todas las instancias tienen la misma probabilidad de ser elegidas como centroides. En cambio, si $\alpha = 1$, las instancias más alejadas de su centroide asignado tienen una mayor probabilidad de ser seleccionadas.

Es necesario hacer dos aclaraciones con respecto al paso 2, ya que su comportamiento depende del tipo de asignación que se emplee.

- **Asignación exacta**: Aunque pueda existir una solución factible que satisfaga todas las restricciones para K clusters, al reducir el número de clusters a $K-1$ es posible que no exista ninguna partición que cumpla todas las restricciones. Si se da este caso, la asignación exacta no será viable. En consecuencia, se establecerá $\alpha = 0$, es decir, todas las instancias tendrán la misma probabilidad de ser seleccionadas como centroides en el paso 3.
- **Asignación Greedy**: En la asignación solo se deben de considerar las instancias que estuvieran asignadas al centroide $\bar{k}$ eliminado en el primer paso 1.

5.1.5. Búsqueda local

La búsqueda local es el algoritmo BLPKM$_{CC}$, una variante de K-Means diseñada para *clustering* semi-supervisado, propuesta en [12]. Este algoritmo parte de una configuración inicial de centroides ψ y repite los siguientes dos pasos hasta cumplir un determinado criterio de parada.

1. Se aplica la asignación exacta, detallada en la Ecuación 5.2, para determinar los valores de ϕ.
2. Se actualizan los centroides ψ de modo que cada ψ_i se convierta en el centro geométrico de las instancias asignadas al i-ésimo *cluster*, conforme a la ecuación 4.1.

El criterio de parada es que se alcance la convergencia, es decir, que los centroides no se actualicen en dos iteraciones sucesivas.

La búsqueda local cumple una doble función: por un lado, permite refinar las soluciones generadas tras las fases de cruce y mutación; por otro, actúa como mecanismo de reparación, ya que al utilizar la asignación exacta se garantiza que la solución resultante satisfaga todas las restricciones ML y CL.

5.1.6. Inicialización de la población

Cada individuo de la población inicial $\mathcal{P} = \{S_1, \ldots, S_P\}$ se genera aplicando la búsqueda local explicada en la Subsección 5.1.5 sobre una solución aleatoria. Los centroides de dicha solución aleatoria se obtienen eligiendo K instancias (sin reemplazo) del conjunto de datos $\mathcal{D}$.

5.2. Algoritmo exacto: PC-SOS-SDP

En esta sección, describimos el funcionamiento del algoritmo exacto PC-SOS-SDP, desarrollado por Piccialli et al. [8]. Este algoritmo pertenece a la familia de métodos *branch and cut*, que combinan la exploración del espacio de soluciones mediante ramificación (*branching*), dividiendo el problema original en subproblemas más pequeños, con el uso de desigualdades válidas (*cutting planes*) que permiten descartar regiones del espacio de soluciones no factibles o subóptimas y acelerar la convergencia. Utilizamos este algoritmo para obtener la solución óptima del problema y compararla con los resultados alcanzados por nuestras propuestas.

5.2.1. Notación

A continuación, describimos brevemente la notación que se va a usar en esta sección.

- $\mathcal{N} = \{1, \ldots, n\}$ es el conjunto de índices de las instancias del conjunto de datos.
- $\mathcal{K} = \{1, \ldots, K\}$ es el conjunto de índices de los *clusters*.
- $\mathcal{ML}$, $\mathcal{CL} \subseteq \mathcal{N} \times \mathcal{N}$ son los conjuntos de restricciones ML y CL respectivamente.
- $\mathcal{S}^n$ es el conjunto de todas las matrices reales simétricas de tamaño $n \times n$.
- $M \succeq 0$ denota que la matriz M es semidefinida positiva.
- $\mathcal{S}^n_+$ es el conjunto de todas las matrices semidefinidas positivas de tamaño $n \times n$.
- $\langle \cdot, \cdot \rangle$ denota el producto escalar de traza. Es decir, para cualquier $A, B \in \mathbb{R}^{m \times n}$, definimos $\langle A, B \rangle := \text{trace}(B^\top A)$.
- Dada una matriz A, denotamos por A_i la i-ésima fila de A.
- Denotamos por e_n el vector de unos de longitud n y por I_n la matriz identidad de tamaño $n \times n$. Se omitirá el subíndice en caso de que la dimensión sea clara por el contexto.
- Denotamos por $\mathcal{F} = \{x_{ij} \in \{0, 1\}$ sujeto a 4.3, 4.4, 4.5 y 4.6$\}$ a la región discreta factible del problema del MSSC con restricciones 4.2.

5.2.2. Reformulación del problema

Podemos expresar el problema del MSSC con restricciones 4.2, sustituyendo los centroides μ_j por la expresión 4.1, de la siguiente forma.

$$\text{mín} \sum_{i=1}^{n}\sum_{j=1}^{K} x_{ij} \left\| p_i - \frac{\sum_{l=1}^{n} x_{lj} p_l}{\sum_{l=1}^{n} x_{lj}} \right\|^2 \tag{5.5}$$

$$\text{sujeto a} \quad x_{ij} \in \mathcal{F}, \quad \forall i \in \mathcal{N}, \quad \forall j \in \mathcal{K}.$$

Este problema, cuando $\mathcal{ML} = \mathcal{CL} = \emptyset$, fue demostrado por Peng y Wei [28] que es equivalente a un problema SDP (*semidefinite programming*) no lineal, llamado 0-1 SDP. En este tipo de problemas, la variable principal es una matriz semidefinida positiva sobre la cual se optimiza una función objetivo, sujeta a restricciones que introducen no linealidad. Cualquier solución del MSSC sin restricciones está asociada a una solución del siguiente problema.

$$\begin{aligned} &\text{mín} \quad \text{tr}(W(I - Z)) \\ &\text{sujeto a} \quad Ze = e, \quad \text{tr}(Z) = k, \quad Z \geq 0, \quad Z^2 = Z, \quad Z = Z^\top. \end{aligned} \tag{5.6}$$

donde:

- W_p es la matriz con el conjunto de datos, en la que cada fila corresponde con una instancia p_i.
- $W = W_p W_p^\top$ es la matriz de productos escalares de las instancias, en la que $W_{ij} = p_i^\top p_j \; \forall i, j \in \mathcal{N}$.
- X es una matriz de tamaño $n \times K$ que contiene las variables de decisión x_{ij}.
- La matriz Z se define como $Z = X(X^\top X)^{-1} X^\top$. Se puede comprobar que Z es una matriz simétrica sin elementos negativos, que cumple $Z = Z^2$. Además, las suma de sus filas y columnas es 1 y su traza K.

Piccialli et al. [8] extienden esta formulación para contemplar las restricciones ML y CL, comprobando que:

- Si las instancias p_i y p_j pertenencen al mismo *cluster* C, se cumple $Z_i = Z_j$. Además, los elementos distintos de cero de las columnas i y j de Z son iguales a $\frac{1}{|C|}$, donde $|C|$ es el número de elementos del *cluster* C.

- Si las instancias p_i y p_j pertenencen al *clusters* distintos, entonces $Z_{ij} = 0$.

Esto permite expresar el problema del MSSC con restricciones 4.2 mediante la siguiente formulación 0-1 SDP.

$$\text{mín} \quad \text{tr}(W(I - Z)) \tag{5.7}$$

$$\begin{aligned} \text{sujeto a} \quad & Ze = e, \quad \text{tr}(Z) = k, \quad Z \geq 0, \quad Z^2 = Z, \quad Z = Z^\top, \\ & Z_{ih} = Z_{jh} \quad \forall h \in \mathcal{N}, \quad \forall (i,j) \in \mathcal{ML}, \\ & Z_{ij} = 0 \quad \forall (i,j) \in \mathcal{CL}. \end{aligned} \tag{5.8}$$

5.2.3. Cálculo de la cota inferior

A diferencia del problema original 4.2, el problema 5.8 tiene una función objetivo lineal, y además, todas sus restricciones excepto $Z = Z^2$ son también lineales. Relajando esta restricción a $Z \succeq 0$ obtenemos la siguiente relajación SDP, con la que podemos obtener una buena cota inferior para el problema original.

$$\text{mín} \quad \text{tr}(W(I - Z)) \tag{5.9}$$

$$\begin{aligned} \text{sujeto a} \quad & Ze = e, \quad \text{tr}(Z) = k, \quad Z \geq 0, \quad Z = Z^\top, \quad Z \in \mathcal{S}_+^n \\ & Z_{ih} = Z_{jh} \quad \forall h \in \mathcal{N}, \quad \forall (i,j) \in \mathcal{ML}, \\ & Z_{ij} = 0 \quad \forall (i,j) \in \mathcal{CL}. \end{aligned}$$

Además, las restricciones ML cumplen una serie de propiedades que permiten reducir la dimensionalidad del problema. A partir del conjunto inicial $\mathcal{ML}$ se puede formar el conjunto de restricciones ML ampliado $\mathcal{ML}'$ aplicando las siguientes propiedades.

- **Simetría**: Si existe la restricción $(i,j) \in \mathcal{ML}'$, también debe cumplirse $(j,i) \in \mathcal{ML}' \quad \forall i,j \in \mathcal{N}$.
- **Reflexividad**: Cada instancia está relacionada consigo misma, es decir, $(i,i) \in \mathcal{ML}' \quad \forall i \in \mathcal{N}$.
- **Transitividad**: Si $(i,h) \in \mathcal{ML}'$ y $(h,j) \in \mathcal{ML}'$, entonces $(i,j) \in \mathcal{ML}', \quad \forall i,h,j \in \mathcal{N}$.

A partir de $\mathcal{ML}'$ podemos agrupar las instancias del conjunto de datos en s grupos $G_1, \ldots G_s$, con $s \leq n$, tal que $\forall (i,j) \in G_l \quad (i,j) \in \mathcal{ML}', \quad \forall l \in \{1, \ldots, s\}$. Podemos representar cada grupo G_i mediante un único punto $\bar{p_i}$, calculado como la suma de todas las instancias de G_i.

$$\bar{p_i} = \sum_{p_j \in G_i} p_j$$

Así, el problema se transforma de encontrar una partición para las instancias originales $p_1, \dots p_n$ a encontrar una partición para los puntos $\bar{p_1}, \dots, \bar{p_s}$. La relajación SDP 5.2.3 es equivalente al siguiente problema.

$$\begin{aligned} \text{mín } & \langle I, W \rangle - \langle \mathcal{T}^s W (\mathcal{T}^s)^\top, Z \rangle \\ \text{sujeto a } & Ze^s = e, \\ & \langle \text{Diag}(e^s), Z \rangle = K, \\ & Z_{ij} = 0 \quad \forall (i,j) \in \overline{\mathcal{CL}}, \\ & Z \geq 0, \quad Z \in \mathcal{S}_+^s. \end{aligned} \tag{5.10}$$

donde

- $\mathcal{T}$ es una matriz $s \times n$, tal que $\mathcal{T}_{ij}^s = \begin{cases} 1 & \text{si } j \in G_i \\ 0 & \text{si } j \notin G_i \end{cases}$
- $\overline{\mathcal{CL}}$ es el conjunto de restricciones CL entre los puntos $\bar{p_i}$, tal que $(i,j) \in \overline{\mathcal{CL}}$ si $(l,m) \in \mathcal{CL}$, $p_l \in G_i$, $p_m \in G_j$.

El algoritmo PC-SOS-SDP produce un árbol binario, en el que el nodo raíz corresponde con el problema 0-1 SDP 5.8. En cada ramificación del árbol, se eligen dos instancias p_i y p_j y se generan dos nodos hijos: en el hijo izquierdo se impone una restricción ML entre ellas, mientras que en el hijo derecho se impone una CL. De este modo, el conjunto de soluciones cada nodo padre se divide en dos subconjuntos disjuntos.

Para elegir el par de instancias p_i y p_j se eligen aquellas tales que:

$$\arg\max_{i,j} \left\{ \text{mín} \left\{ Z_{ij}, \|Z_i - Z_j\|^2 \right\} \right\}$$

5.2.4. Desigualdades válidas

El algoritmo PC-SOS-SDP utiliza tres tipos de desigualdades para reforzar la cota.

- **Desigualdades de pares**. para toda solución factible, se cumple lo siguiente:

$$Z_{ij} \leq Z_{ii}, \quad Z_{ij} \leq Z_{jj}, \quad \forall i, j \in \mathcal{N},\ i \neq j. \tag{5.11}$$

- **Desigualdades triangulares**. Si las instancias p_i y p_j están en el mismo *cluster*, y las instancias p_j y p_h también están en el mismo *cluster*, entonces p_i y p_h deben de estar en el mismo *cluster*.

$$Z_{ij} + Z_{ih} \leq Z_{ii} + Z_{jh} \quad \forall i, j, h \in \mathcal{N}, \text{ siendo } i, j, h \text{ distintos entre sí.} \tag{5.12}$$

- **Desigualdades de clique**. Si el número de *clusters* es igual a K, en cualquier subconjunto de instancias de tamaño $K+1$ deben de haber el menos dos instancias que pertenezcan al mismo *cluster*.

$$\sum_{(i,j)\in Q, i<j} Z_{ij} \geq \frac{1}{n-K+1} \quad \forall Q \subseteq \mathcal{N}, |Q| = K+1. \tag{5.13}$$

Estas desigualdades se añaden a la relajación SDP mediante un procedimiento de plano de corte únicamente si son incumplidas. Es decir, no se añaden todas desde el inicio, sino que solo se añaden aquellas que no se satisfacen en la solución actual del problema relajado. Tras añadirlas, se vuelve a calcular la solución del problema con las restricciones incorporadas. Además, después de cada iteración del plano de corte se eliminan las restricciones que no estén activas en la solución actual, es decir, aquellas que no estén limitando directamente la solución óptima encontrada. De esta forma, se evita el aumento excesivo del número de restricciones y se mantiene el tamaño del problema dentro de unos límites computacionalmente manejables. Por otro lado, las desigualdades que se incluyeron al problema de un nodo padre en la última iteración son directamente heredadas por los nodos hijos desde el inicio.

5.2.5. Cálculo de la cota superior

El algoritmo utilizado para calcular la cota superior en cada nodo es una variante del algoritmo BLPKM$_{CC}$ [12]. Este algoritmo, como vimos anteriormente en la Sección 5.1.5, garantiza que la solución obtenida cumpla todas las restricciones, siempre que exista una solución factible. Sin embago, presenta un inconveniente: es altamente sensible a la elección de los centroides iniciales. Para tratar este problema, Piccialli et al. [29] proponen una técnica de inicialización basada en la solución de la relajación SDP, utilizada previamente para hallar la cota inferior. La idea es que, si la solución Z de la relajación es ajustada, es decir, es una solución factible del problema original 5.8, entonces se pueden recuperar los centroides iniciales a partir de la matriz Z. Si no lo es, se busca una matriz $\hat{Z}$, lo más cercana posible a Z en términos de la norma de Frobenius, es decir, se minimiza la suma de los cuadrados de las diferencias entre las entradas de ambas matrices. Una vez hallada $\hat{Z}$ se obtienen los centroides a partir de esta matriz.

La modificación de BLPKM$_{CC}$ que incorpora la inicialización descrita se llama IPC-k-means, y es la heurística utilizada para calcular la cota superior en cada nodo del algoritmo PC-SOS-SDP. El pseudocódigo de IPC-k-means se presenta a continuación.

Algorithm 3 IPC-k-means

Require: *Dataset* $D = \{p_1, \ldots, p_n\}$, número de *clusters* K, conjuntos de restricciones $\mathcal{ML}$ y $\mathcal{CL}$, matriz de datos W_p, la solución óptima $\hat{Z}$ de la relajación SDP con restricciones $\mathcal{ML}$ y $\mathcal{CL}$

1: Resolver $\hat{Z} = \arg\min\{\|\hat{Z} - Z\|_F$ sujeto a rango$(Z) = K\}$.
2: Calcular la aproximación de la matriz de centroides $\hat{M} = \hat{Z}W_p$.
3: Agrupar las filas de $\hat{M}$ con k-means para obtener los centroides iniciales de los *clusters* $\mu_1, \ldots, \mu_K$.
4: **repeat**
5: Calcular las asignaciones óptimas de *clusters* x_{ij}^* resolviendo:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{K} x_{ij} \|p_i - \mu_j\|^2 \quad \text{sujeto a } x_{ij} \in \mathcal{F}, \quad \forall i \in \mathcal{N}, \forall j \in \mathcal{K} \tag{5.14}$$

6: Definir $C_j \leftarrow \{p_i : x_{ij}^* = 1\}$ para cada $j = 1, \ldots, K$.
7: Actualizar los centroides de los *clusters* $\mu_1, \ldots, \mu_K$ calculando la media de los puntos asignados a cada *cluster* $C_1, \ldots, C_K$.
8: **until** convergencia;
9: **Salida:** *Clusters* $C_1, \ldots, C_K$

5.2.6. Esquema general del algoritmo

Una vez explicados los distintos componentes que forman el algoritmo PC-SOS-SDP, presentamos a continuación el pseudcódigo completo del mismo.

Algorithm 4 Algoritmo PC-SOS-SDP

Entrada: Conjuntos de restricciones $\mathcal{ML}$ y $\mathcal{CL}$, número de *clusters* K, matriz W
Construir $\mathcal{T}^s$, e^s y $\overline{\mathcal{CL}}$ a partir de $\mathcal{ML}$ y $\mathcal{CL}$.
Sea P_0 el problema 0-1 SDP inicial 5.8, establecer $Q = \{P_0\}$
Inicializar la mejor solución $X^* = \text{null}$ y su valor de la función objetivo $v^* = \infty$
while Q no esté vacío **do**
  Seleccionar y extraer un problema P de Q.
  Comprobar si el problema P es factible. Si no lo es, ir al Paso 5.
  Resolver la relajación SDP 5.10 para obtener una cota inferior LB y la solución óptima Z .
  if $LB \geq v^*$ **then**
    Ir al Paso 5.
  end if
  Buscar desigualdades de pares 5.11, triangulares 5.12 y de clique 5.13 incumplidas por Z. Si se encuentran, añadirlas a la relajación SDP actual y volver al Paso 8
  Ejecutar el algoritmo Algorithm 3 para obtener una solución X y la cota superior UB
  if $UB < v^*$ **then**
    Actualizar $v^* \leftarrow UB$, $X^* \leftarrow X$
  end if
  Seleccionar un par de instancias p_i y p_j y dividir el problema P en dos subproblemas. Para cada problema actualizar $\mathcal{T}^s$, e^s y $\overline{\mathcal{CL}}$ en consecuencia, añadir ambos problemas a Q e ir al Paso 5.
end while
Salida: Matriz de la solución óptima X^* y su valor de la función objetivo v^*

Capítulo 6

Descripción de la propuesta

A lo largo de este capítulo se presentarán los diversos componentes que se han ido incorporando al modelo, partiendo del algoritmo S-MDEClust. [7].

6.1. Asignación

Recordemos que, en el algoritmo original, se propone un método *greedy* para asignar cada instancia a un *cluster*, una vez fijados los centroides de la solución. Como vimos en la Sección 5.1.2, este método consiste en agrupar las instancias de acuerdo a las restricciones ML y luego asignar iterativamente cada grupo de instancias al mejor *cluster* disponible. Se considera como mejor aquel *cluster* que, sin incumplir ninguna restricción, minimiza la suma de las distancias al cuadrado entre las instancias del grupo y el centroide correspondiente. Si no existe ningún *cluster* que satisfaga todas las restricciones, se elige entonces el que minimice las distancias al cuadrado sin tenerlas en cuenta.

No obstante, podría ser interesante no escoger siempre el mejor *cluster*, para favorecer la exploración de diferentes regiones del espacio de soluciones. Teniendo en cuenta que la solución obtenida se refinará posteriormente mediante una búsqueda local en la que sí se aplica la asignación exacta explicada en la Sección 5.1.2, puede resultar beneficioso introducir cierta aleatoriedad en esta etapa inicial. Con esta motivación, se han desarrollado dos variantes del método *greedy*: la asignación *greedy* aleatorizada y la asignación *greedy* aleatorizada con penalización.

6.1.1. Asignación *greedy* aleatorizada

En esta variante, se parte del mismo procedimiento que en el enfoque *greedy* original: se agrupan las instancias según las restricciones ML y se identifican los *clusters* viables que no incumplen ninguna restricción. Sin embargo, en lugar de asignar el grupo al *cluster* más cercano de forma determinista, se introduce un componente aleatorio. Concretamente, para cada *cluster* viable se calcula la suma de distancias al cuadrado entre su centroide y las instancias del grupo que se desea asignar. Estas distancias se invierten y se normalizan para construir una distribución de probabilidad. El grupo se asignará a un *cluster* de forma aleatoria según esta distribución. De esta forma, los *clusters* más cercanos tendrán más probabilidad de ser seleccionados, aunque los *clusters* más lejanos también tendrán opción de ser elegidos. En el caso de que la asignación del grupo de instancias a cualquiera de los *clusters* provoque el incumplimiento de alguna restricción, se considerarán todos los *clusters* posibles, sin tener en cuenta las restricciones, y se aplicará el mismo procedimiento probabilístico para realizar la asignación.

Algorithm 5 Asignación *Greedy* Aleatorizada

Require: *dataset* $D = \{p_1, \ldots, p_n\} \subset \mathbb{R}^d$, número de *clusters* $K \in \mathbb{N}$, conjuntos de restricciones *must-link* y *cannot-link* $\mathcal{ML}, \mathcal{CL} \subset \{1, \ldots, n\} \times \{1, \ldots, n\}$, matriz de centroides ψ^S.

Definir $\mathcal{G}_{\mathcal{ML}} = \{G \subset \{1, \ldots, N\} \mid \forall i, j \in G$,sujeto a $i \neq j, (i, j) \in \mathcal{ML}\}$

Inicializar $\mathcal{A} = \emptyset$
for all $G \in \mathcal{G}_{\mathcal{M}_\mathcal{L}}$ **do**
  Construir la lista $ksPosibles \leftarrow [k \in \{1, \ldots, K\} \mid \nexists(\hat{G}, k) \in \mathcal{A}$ tal que $\exists(i_c, j_c) \in \mathcal{CL},\ i_c \in G \wedge j_c \in \hat{G}]$
  if $|ksPosibles| == 0$ **then**
    $ksPosibles \leftarrow [1 \ldots, K]$
  end if
  Calcular la lista $dist \leftarrow \left[\sum_{i \in G} \|p_i - \psi_k^S\|^2 \ \forall\ k \in ksPosibles\right]$
  Calcular la lista $invDist \leftarrow \left[\frac{1}{d+\epsilon} \ \forall\ d \in dist\right]$
  $probs \leftarrow \left[\frac{x}{\sum invDist} \ \forall\ x \in invDist\right]$
  Elegir aleatoriamente k_G de $ksPosibles$ según la distribución $probs$
  $\mathcal{A} = \mathcal{A} \cup \{(G, k_G)\}$
  for all $i \in G$ **do**
    $\phi_i^S = k_G$
  end for
end for

return ϕ^S

El pseudocódigo de la asignación *greedy* aleatorizada se muestra en el Algoritmo 5. En el paso 9 se utiliza $\epsilon > 0$ para evitar la división entre 0.

6.1.2. Asignación *greedy* aleatorizada con penalización

Esta variante mantiene el esquema general del enfoque *greedy* aleatorizado, pero añade una penalización adicional que desincentiva la asignación de grupos a *clusters* que incumplan restricciones. En lugar de determinar qué *clusters* son viables, todos los *clusters* se consideran candidatos. Para cada uno, se calcula la suma de distancias al cuadrado entre su centroide y las instancias del grupo, y se añade una penalización si existen conflictos de restricciones CL con grupos ya asignados a ese cluster. Esta penalización es la mitad de la media del rango de los valores del conjunto de datos en cada dimensión, multiplicada por el número de dimensiones. Las distancias con la penalización añadida se transforman en una distribución de probabilidad, invirtiéndolas y normalizándolas, y se asigna el grupo a un *cluster* seleccionado aleatoriamente según esa distribución.

Algorithm 6 Asignación *Greedy* Aleatorizada con Penalización

Require: *dataset* $D = \{p_1, \ldots, p_n\} \subset \mathbb{R}^d$, número de *clusters* $K \in \mathbb{N}$, restricciones *must-link* y *cannot-link* $\mathcal{ML}, \mathcal{CL}$, matriz de centroides ψ^S

1: Calcular penalización $\lambda = \text{media}\,(\text{máx}(D) - \text{mín}(D)) \cdot d/2$
2: Definir $\mathcal{G}_{\mathcal{ML}} = \{G \subset \{1, \ldots, n\} \mid \forall i, j \in G,\ i \neq j \Rightarrow (i, j) \in \mathcal{ML}\}$
3: Inicializar $\mathcal{A} = \emptyset$
4: **for all** $G \in \mathcal{G}_{\mathcal{ML}}$ **do**
5: $\quad ksPosibles \leftarrow \{1, \ldots, K\}$
6: $\quad ksConflictivos \leftarrow [k \in \{1, \ldots, K\} \mid \exists(\hat{G}, k) \in \mathcal{A}$ tal que $\exists(i_c, j_c) \in \mathcal{CL},\ i_c \in G \wedge j_c \in \hat{G}]$
7: $\quad$ Calcular la lista $dist \leftarrow [\sum_{i \in G} \|p_i - \psi_k^S\|^2 + \lambda \cdot \mathbf{1}_{k \in ksConflictivos}\ \ \forall\, k \in ksPosibles]$
8: $\quad$ Calcular $invDist \leftarrow \left[\frac{1}{c+\epsilon}\ \forall\, c \in dist\right]$ $\{\epsilon > 0$ para evitar división por cero$\}$
9: $\quad$ Calcular $probs \leftarrow \left[\frac{x}{\sum invDist}\ \forall\, x \in invDist\right]$
10: $\quad$ Elegir aleatoriamente k_G de $ksPosibles$ según la distribución $probs$
11: $\quad \mathcal{A} = \mathcal{A} \cup \{(G, k_G)\}$
12: $\quad$ **for all** $i \in G$ **do**
13: $\quad\quad \phi_i^S = k_G$
14: $\quad$ **end for**
15: **end for**
16:
17: **return** ϕ^S

El pseudocódigo de la asignación *greedy* aleatorizada con penalización se muestra en el algoritmo 6. Donde $\mathbf{1}_{k \in conflictivos}$ es una función booleana que toma el valor 1 si se cumple la condición $k \in conflictivos$ y 0 en caso contrario. Así, la penalización λ se añade únicamente para los *clusters* con conflictos.

6.2. Búsqueda Local

6.2.1. Método Solis Wets

Con el objetivo de mejorar la calidad de las soluciones antes de aplicar la búsqueda local del algoritmo original (Subsección 5.1.5), se ha introducido una fase previa de búsqueda local basada en el método de Solis Wets [30]. Para evitar una sobrecarga computacional excesiva y no perjudicar a la diversidad de la población, esta estrategia se aplica únicamente a un individuo en cada iteración, el que se considere más prometedor. Para seleccionar a dicho individuo se han considerado dos opciones:

- **Versión 1**: se aplica sobre el mejor individuo de la población, siempre y cuando no se haya aplicado ya en iteraciones previas sobre dicho individuo sin obtener mejora.
- **Versión 2**: se aplica sobre el mejor individuo de la población.

La búsqueda local de Solis Wets es un algoritmo con un enfoque de ascensión de colinas aleatorizado que mantiene en todo momento una única solución en memoria: la mejor encontrada hasta el momento. Las soluciones con las que trabaja el algoritmo son las matrices de centroides. En cada iteración del algoritmo, se parte de una solución actual ψ^S y se genera una diferencia aleatoria dif a partir de una distribución normal centrada en un *array* que llamaremos *bias*, de las mismas dimensiones que la solución, es decir, $K \times d$; y una desviación típica controlada por el parámetro ρ. Se evalúan dos posibles nuevas soluciones: $\psi^S + dif$, y si no mejora, $\psi^S - dif$. Si alguna de ellas mejora a la mejor solución actual, evaluadas según la función objetivo, se actualiza la mejor solución y se registra un éxito; en caso contrario, se registra un fallo.

Para evaluar una solución ψ^S, es necesario realizar primero un paso de asignación (Subsección 5.1.2) para obtener el vector de pertenencia ϕ^S y así construir la solución completa $S = (\phi^S, \psi^S)$, que puede evaluarse mediante la función objetivo presentada en la Ecuación 5.1. Sin embargo, si no utilizamos la asignación exacta explicada en la Sección 5.1.2, existe la posibilidad de que la solución no satisfaga todas las restricciones.

Para tratar de evitar que las nuevas soluciones incumplan un gran número de restricciones, proponemos una función objetivo penalizada, en la que se introduce una penalización proporcional al número de restricciones incumplidas por la solución. De este modo, se pueden considerar dos opciones para la evaluación de soluciones dentro del algoritmo de Solis Wets:

- Usar la función objetivo original (Ecuación 5.1).
- Utilizar una función objetivo con penalización, definida como:

$$f_p(S) = f(S) \cdot (1 + \delta \cdot infeasibility) \tag{6.1}$$

  donde $f(S)$ es la función objetivo original (sin penalización), δ es una constante que controla el peso del término de penalización, e $infeasibility$ representa el tanto por uno de restricciones incumplidas respecto al total.

Por otro lado, como ya hemos mencionado brevemente, este método emplea dos parámetros: *bias* y ρ que se van adaptando de forma dinámica y permiten aprender progresivamente cómo guiar la búsqueda de manera más eficiente. El parámetro *bias* actúa como estimación de la dirección más prometedora hacia el óptimo local. Se inicializa a 0, ya que en un inicio no disponemos de información acerca de cuál es la mejor dirección de búsqueda y se actualiza tras cada éxito o fallo. De este modo, el algoritmo va aprendiendo progresivamente hacia dónde conviene desplazar las soluciones, favoreciendo la exploración en direcciones más prometedoras. Por su parte, el parámetro ρ se adapta en función del comportamiento reciente de la búsqueda: tras varios éxitos consecutivos, se incrementa ρ para explorar regiones más amplias y tras varios fallos consecutivos, se reduce para afinar más la búsqueda.

El pseudocódigo completo del método de Solis Wets se presenta en el Algoritmo 7.

6.2.2. Selección de individuos sobre los que se aplica la Búsqueda Local

En la versión original del algoritmo, la búsqueda local se aplica sobre todos los individuos de la población en cada iteración. Sin embargo, este enfoque resulta computacionalmente muy costoso, ya que cada búsqueda local requiere realizar múltiples veces el paso de asignación exacta, el cual implica resolver un problema de optimización para asignar las instancias a los centroides. Esta operación es uno de los cuellos de botella del algoritmo, especialmente en problemas de gran tamaño.

Para reducir este coste computacional y a la vez favorecer la diversidad de la población, proponemos aplicar la búsqueda únicamente sobre una fracción de la población. En concreto, se aplicará en cada iteración al 10 % de las mejores soluciones, con el fin de refinar aquellas soluciones más prometedoras, y un 10 % adicional elegido aleatoriamente entre el resto de la población, para favorecer la diversidad de la población y disminuir la probabilidad de quedar estancados en óptimos locales de forma prematura. Para el resto de individuos no seleccionados, se omite la búsqueda local y, en su lugar, se realiza un único paso de asignación exacta como mecanismo de reparación; de esta forma, se garantiza que todas las soluciones de la población cumplan todas las restricciones.

Algorithm 7 Algoritmo de Solis-Wets

Require: Matriz de centroides de la solución inicial $centers$, número máximo de evaluaciones $maxevals$, ρ, $dataset$ D, conjunto de restricciones ML, CL, función objetivo $fobj$

$bias \leftarrow \mathbf{0}$
Inicializar $evals \leftarrow 0$, $failures \leftarrow 0$, $successes \leftarrow 0$, $num\_mejoras \leftarrow 0$

$labels \leftarrow$ asignar instancias de D a $centers$
$fitness\_sol \leftarrow fobj(D, centers, labels, ML, CL)$
while $evals < maxevals$ **and** $\rho > \epsilon$ **do**
  $dif \leftarrow \mathcal{N}(bias, \rho)$
  $newcenters \leftarrow centers + dif$
  $newlabels \leftarrow$ asignar instancias de D a $newcenters$
  $fitness\_new \leftarrow fobj(D, newcenters, newlabels, ML, CL)$
  $evals \leftarrow evals + 1$
  if $fitness\_new < fitness\_sol$ **then**
    $num\_mejoras \leftarrow num\_mejoras + 1$
    $failures \leftarrow 0$, $successes \leftarrow successes + 1$
    $bias \leftarrow 0.4 \cdot dif + 0.2 \cdot bias$
    Actualizar $centers$, $labels$, $fitness\_sol$ con $newcenters$, $newlabels$ y $fitness\_new$ respectivamente
  else if $evals < maxevals$ **then**
    $newcenters \leftarrow centers - dif$
    $newlabels \leftarrow$ asignar instancias de D a $newcenters$
    $fitness\_new \leftarrow fobj(D, newcenters, newlabels, ML, CL)$
    $evals \leftarrow evals + 1$
    if $fitness\_new < fitness\_sol$ **then**
      $num\_mejoras \leftarrow num\_mejoras + 1$
      $failures \leftarrow 0$, $successes \leftarrow successes + 1$
      $bias \leftarrow bias - 0.4 \cdot dif$
      Actualizar $centers$, $labels$, $fitness\_sol$ con $newcenters$, $newlabels$ y $fitness\_new$ respectivamente
    else
      $failures \leftarrow failures + 1$
      $successes \leftarrow 0$
      $bias \leftarrow 0.5 \cdot bias$
    end if
  end if
  if $successes \geq 5$ **then**
    $successes \leftarrow 0$
    $\rho \leftarrow 2 \cdot \rho$
  else if $failures \geq 3$ **then**
    $failures \leftarrow 0$
    $\rho \leftarrow \rho / 2$
  end if
end while
return $labels$, $centers$

Donde ϵ, en la línea 5, es un número positivo muy cercano a 0. Con esto se pretende detener la búsqueda cuando ρ alcanza valores muy pequeños, lo que indica que las modificaciones en las soluciones son mínimas y el algoritmo ha quedado estancado.

6.3. Evolución Diferencial

6.3.1. SHADE

El operador de cruce del algoritmo original (Subsección 5.1.3) no hace uso del parámetro CR. Este parámetro, habitual en algoritmos de Evolución Diferencial, controla la probabilidad de aplicar el cruce a cada individuo de la población. En caso de no aplicar el cruce, el descendiente del individuo sería él mismo. El algoritmo original tampoco contempla mecanismos de adaptación de parámetros, lo que podría limitar su capacidad de exploración. Con el objetivo de mejorar la eficacia del proceso evolutivo y guiar la búsqueda de manera más informada, haremos uso del mecanismo de adaptación de parámetros SHADE (*Success-History based Adaptive Differential Evolution*) [31].

La principal ventaja de SHADE es que ajusta dinámicamente los valores de F y CR en cada iteración, en función del historial de configuraciones que han dado buenos resultados en iteraciones anteriores. SHADE ha demostrado ser efectivo en una amplia variedad de problemas *benchmark*, superando en muchos casos al rendimiento de algoritmos de Evolución Diferencial sin adaptación de parámetros y a otras estrategias de adaptación dinámica de parámetros [31].

SHADE selecciona de forma dinámica los parámetros CR y F para cada individuo de la población. Para ello, mantiene dos listas temporales, S_{CR} y S_F, en las que se almacenan los valores de CR y F utilizados por aquellos individuos que han logrado mejorar respecto a su predecesor en términos de la función objetivo. Estas listas se reinician al comienzo de cada iteración. Por otro lado, se utilizan dos memorias de tamaño H, una para CR y otra para F, ambas inicializadas con valores iguales a 0.5 en todas sus posiciones.

En cada iteración, los parámetros CR_i y F_i para un individuo S_i se generan del siguiente modo:

$$CR_i = \text{randn}(M_{CR,r_i}, 0.1) \tag{6.2}$$

$$F_i = \text{randc}(M_{F,r_i}, 0.1) \tag{6.3}$$

donde $\text{randn}(\mu, \sigma^2)$ representa un valor aleatorio tomado de una distribución normal con media μ y varianza σ^2, y $\text{randc}(\mu, \gamma)$ representa un valor aleatorio

obtenido de una distribución de Cauchy con media μ y escala γ. El índice $r_i \in \{1, \ldots, H\}$ se selecciona aleatoriamente, y M_{CR,r_i} y M_{F,r_i} hacen referencia al elemento r_i-ésimo de las memorias M_{CR} y M_F, respectivamente. Además, si los valores generados para CR_i o F_i se encuentran fuera del intervalo $[0, 1]$, se truncan a dicho rango.

Al final de cada iteración, se actualiza la posición k de las memorias M_{CR} y M_F utilizando la información almacenada en S_{CR} y S_F. El índice k se inicializa a 1 y se incrementa en cada actualización, reiniciándose a 1 cuando supera H. La actualización de la posición k para la iteración $i + 1$ se realiza conforme a las siguientes expresiones:

$$M_{CR,k,i+1} = \begin{cases} \text{mean}_{W_A}(S_{CR}) & \text{si } S_{CR} \neq \emptyset \\ M_{CR,k,i} & \text{en caso contrario} \end{cases} \tag{6.4}$$

$$M_{F,k,i+1} = \begin{cases} \text{mean}_{W_L}(S_F) & \text{si } S_F \neq \emptyset \\ M_{F,k,i} & \text{en caso contrario} \end{cases} \tag{6.5}$$

donde $\text{mean}_{W_A}(S_{CR})$ y $\text{mean}_{W_L}(S_F)$ son medias ponderadas que se calculan de acuerdo a las siguientes ecuaciones.

$$\text{mean}_{WA}(S_{CR}) = \sum_{k=1}^{|S_{CR}|} w_k \cdot S_{CR,k} \tag{6.6}$$

$$w_k = \frac{\Delta f_k}{\sum_{k=1}^{|S_{CR}|} \Delta f_k} \tag{6.7}$$

$$\text{mean}_{W_L}(S_F) = \frac{\sum_{k=1}^{|S_F|} w_k \cdot S_{F,k}^2}{\sum_{k=1}^{|S_F|} w_k \cdot S_{F,k}}$$

donde Δf_k es la diferencia en valor absoluto entre el valor objetivo del k-ésimo individuo de la solución y su predecesor.

En el Algoritmo 8 se muestra el pseudocódigo completo del algoritmo S-MDEClust con la incorporación del método de adaptación de parámetros SHADE que acabamos de describir.

Algorithm 8 Algoritmo S-MDEC1ust con SHADE

Require: Dataset $D = \{p_1, \ldots, p_n\} \subset \mathbb{R}^d$, número de clusters K, restricciones $\mathcal{M}_L, \mathcal{C}_L$, tamaño de población P, número máximo de iteraciones N_{MAX}, tolerancia tol.

Inicializar población $\mathcal{P}_0 = \{S_{1,0}, \ldots, S_{P,0}\}$ aleatoriamente
Inicializar los H valores de M_{CR} y M_F a 0.5
Inicializar índice $k = 1$
Determinar $S^\star$ tal que $f(S^\star) \leq f(S)$ para todo $S \in \mathcal{P}_0$
Inicializar $n^\star_{it} = 0$
while $(n^\star_{it} < N_{MAX}) \wedge (\sum_{s=1}^{P} \sum_{\bar{s}>s} |f(S_{s,G}) - f(S_{\bar{s},G})| > tol)$ **do**
  Inicializar $S_{CR} = \emptyset$, $S_F = \emptyset$
  for $s = 1$ **to** P **do**
    Seleccionar $r_i \in [1, H]$ aleatoriamente
    $CR_s = \text{randn}(M_{CR,r_i}, 0.1)$
    $F_s = \text{randc}(M_{F,r_i}, 0.1)$
    if rand(0,1) $< CR_s$ **then**
      Seleccionar aleatoriamente $S_1, S_2, S_3 \in \mathcal{P}$, todos diferentes entre sí y distintos de S_s
      Aplicar operador de cruce utilizando F_s con S_1, S_2, S_3 para generar el descendiente O_s
    else
      $O_s = S_s$
    end if
    Aplicar o no el operador de mutación a O_s para obtener $\tilde{O}_s$, con un probabilidad que disminuye a lo largo de las iteraciones
    Aplicar búsqueda local a $\tilde{O}_s$ para obtener una solución O'_s
    if $f(O'_s) < f(S_s)$ **then**
      $S_s = O'_s$
      Añadir CR_s a S_{CR}
      Añadir F_s a S_F
      if $f(O'_s) < f(S^\star)$ **then**
        $S^\star = O'_s$
        $n^\star_{it} = 0$
      else
        $n^\star_{it} = n^\star_{it} + 1$
      end if
    end if
  end for
  if $S_{CR} \neq \emptyset$ y $S_F \neq \emptyset$ **then**
    Actualizar $M_{CR,k}$ y $M_{F,k}$ de acuerdo a las ecuaciones 6.6 y 6.3.1
    $k = (k + 1) \bmod H$
  end if
end while
return $S^\star$

6.3.2. Cambios en el operador de cruce

En la versión original del algoritmo, el operador de cruce utilizado es el cruce clásico de la Evolución Diferencial, que genera un descendiente v_i a partir de tres individuos distintos mediante la expresión $v_i = x_{r_1} + F \cdot (x_{r_2} - x_{r_3})$, donde x_{r_1}, x_{r_2} y x_{r_3} son individuos aleatorios seleccionados de la población y F es el parámetro que controla la amplitud de la perturbación (Subsección 5.1.3). Este operador no incorpora información sobre la calidad de los individuos, lo que podría limitar su capacidad para guiar la búsqueda de manera eficiente hacia regiones prometedoras del espacio de soluciones.

Con el objetivo de mejorar el rendimiento del algoritmo, proponemos sustituir el cruce original por dos nuevas variantes basadas en la estrategia *current-to-pbest* [32], que introducen un sesgo hacia individuos de alta calidad.

Para ambas variantes, que llamaremos pbest1 y pbest2, el descendiente v_i de cada individuo x_i de la población se genera según la siguiente expresión:

$$v_i = x_i + F \cdot (x_b - x_i) + F \cdot (x_{r_1} - x_{r_2}) \tag{6.8}$$

donde x_{r_1} y x_{r_2} son individuos seleccionados de forma aleatoria de la población distintos de x_i.

La diferencia entre las dos variantes radica en la elección de x_b.

- **Cruce pbest1**: x_b es un individuo aleatorio seleccionado entre el $p\,\%$ de los mejores individuos de la población en términos del valor de la función objetivo.
- **Cruce pbest2**: x_b es el mejor individuo, según el valor de la función objetivo, entre un $p\,\%$ aleatorio del total de la población.

Estas variantes introducen una presión selectiva hacia soluciones de alta calidad, pero conservando cierta diversidad mediante la aleatoriedad en la elección de x_b.

6.4. Reinicio de la población

Uno de los principales desafíos al trabajar con algoritmos evolutivos es la pérdida de diversidad en la población, que puede limitar la capacidad de explorar el espacio de soluciones de forma efectiva y provocar una convergencia prematura hacia óptimos locales. Una forma común de abordar este problema es introducir un mecanismo de reinicio en el algoritmo.

El reinicio consiste en volver a generar parcial o totalmente la población cuando se detecta una situación de estancamiento. De esta forma, se reintroduce diversidad a la población, lo que permite al algoritmo escapar de óptimos locales y continuar la búsqueda hacia mejores soluciones.

A la hora de realizar el reinicio, es importante tener en cuenta varios elementos que determinan cuándo y cómo se lleva a cabo:

- **Condición de activación:** el reinicio se activa tras un número determinado de iteraciones consecutivas sin mejora de la mejor solución de la población. Esta condición busca detectar situaciones de estancamiento en la búsqueda.
- **Número de reinicios:** para evitar un uso excesivo de los reinicios, lo que podría implicar un consumo excesivo de recursos computacionales, se establece un número de reinicios a lo largo de la ejecución del algoritmo.
- **Generación de la nueva población:** tras el reinicio, se conserva la mejor solución encontrada hasta el momento, mientras que el resto de la población se genera de forma aleatoria, de la forma explicada en Subsección 5.1.6. Esto permite mantener la mejor solución obtenida y, al mismo tiempo, recuperar diversidad.
- **Tamaño de la población:** se han considerado dos estrategias:
  1. Mantener el tamaño de la población constante.
  2. Disminuir el tamaño de la población a la mitad en cada reinicio: se parte de un tamaño de población más grande, concretamente el doble, y se disminuye el tamaño a la mitad después de cada reinicio. Esto puede favorecer la diversidad en las primeras fases, cuando la población es más grande, y acelarar la convergencia en etapas posteriores, cuando la población es más pequeña.

6.5. Otro enfoque: GRASP

Saliendo de la línea de modificaciones sobre el algoritmo original, se ha desarrollado un nuevo algoritmo basado en la estrategia GRASP (*Greedy Randomized Adaptive Search Procedure*)

Los algoritmos GRASP son métodos multiarranque, en los que en cada iteración tienen dos fases: una fase de construcción de una solución mediante un procedimiento *greedy* aleatorizado, y una fase de mejora, en la que se aplica una búsqueda local sobre la solución obtenida en la iteración. Al terminar el algoritmo, se devuelve la mejor solución encontrada.

Hay varios aspectos que se han tenido en cuenta al diseñar el algoritmo:

- **Criterio de parada:** el algoritmo se detendrá cuando se alcance un número máximo de iteraciones $maxIter$, o un número máximo de iteraciones consecutivas sin mejoras en la mejor solución encontrada $Nmax$.
- **Procedimiento para construir la solución *greedy* aleatorizada:** debe ser capaz de generar soluciones iniciales razonables que sirvan como punto de partida para la fase de mejora, con un componente de aleatoriedad controlada para favorecer la exploración del espacio de soluciones. El pseudocódigo completo del algoritmo diseñado se muestra en Algoritmo 9
- **Procedimiento de Búsqueda Local:** se empleará el algoritmo BLPKM$_{CC}$ [12], el mismo empleado en la búsqueda local del algoritmo S-MDEClust, ya que ha demostrado ser efectivo y garantiza el cumplimiento de todas las restricciones.

Algorithm 9 Solución *Greedy* Aleatorizada

Require: *dataset* $D = \{p_1, \ldots, p_n\} \subset \mathbb{R}^d$, número de *clusters* $K \in \mathbb{N}$, conjuntos de restricciones $\mathcal{ML}, \mathcal{CL}$, función objetivo $fobj$

Inicializar $idxCenters \leftarrow \emptyset$ {índices de las instancias elegidas como centroides}
for $c = 1$ **to** K **do**
  $candidates \leftarrow \emptyset$
  for all $i \in \{1, \ldots, n\} \setminus idxCenters$ **do**
    $idxCenters \leftarrow idxCenters \cup \{i\}$
    asignar las instancias de D a $idxCenters$ para obtener el vector de pertenencia $labels$, teniendo en cuenta las restricciones $\mathcal{ML}$ y $\mathcal{CL}$
    $totalDist \leftarrow fobj(D, idxCenters, labels)$
    Añadir $(totalDist, i)$ a $candidates$
    $idxCenters \leftarrow idxCenters \setminus \{i\}$
  end for
  Ordenar $candidates$ por $totalDist$ de menor a mayor
  $RCL\_size \leftarrow$ máx$(2, \lfloor 0.1 \cdot |candidates| \rfloor)$
  Elegir $newCenter$ aleatoriamente de entre los RCL_size primeros elementos de $candidates$
  $idxCenters \leftarrow idxCenters \cup \{newCenter\}$
end for
$centers \leftarrow D[idxCenters]$

return $centers$

Para construir la solución *greedy* aleatorizada, se parte de un conjunto

de centroides de la solución vacío, y en cada iteración, una instancia del conjunto de datos es elegida como nuevo centroide de un *cluster*. Para ello, se crea una lista formada por los índices de todas las instancias que aún no han sido seleccionadas, y se ordenan según el valor de la función objetivo de la solución que se obtendría al considerar esa instancia candidata como centroide, además de las que ya forman parte de la solución. A partir de esa lista, se crea la RLC (*Restricted List of Candidates*) con el 10 % de los mejores candidatos. Por último, se selecciona aleatoriamente una instancia de esta RCL y se añade al conjunto de centroides.

En cuanto a la línea 6 del algoritmo, hay que tener en cuenta que no se puede utilizar la asignación exacta vista en la Sección 5.1.2, ya que el número de centroides de la solución, excepto en la última iteración, será menor que el número de *clusters* K, debido a que la solución se va construyendo paso a paso y aún no está completa. Así que, aunque exista una solución factible del problema para K *clusters*, podría no existir una solución factible que cumpla todas las restricciones para un número de *clusters* menor que K. Por esto, se debe usar un enfoque *greedy* para la asignación, como el que comentamos en la Sección 5.1.2.

Por otro lado, en la línea 12 se establece el tamaño de la RCL mediante la expresión $RCL\_size \leftarrow \text{máx}(2, \lfloor 0.1 \cdot |candidates| \rfloor)$. De esta forma, se garantiza que la RCL contenga al menos dos candidatos, incluso cuando el número total de candidatos sea pequeño. Esto se hace para mantener la aleatoriedad en la selección.

Finalmente, en el Algoritmo 10, se muestra el pseudocódigo completo de la propuesta GRASP.

Algorithm 10 GRASP

Require: *dataset* $D = \{p_1, \ldots, p_n\} \subset \mathbb{R}^d$, número de *clusters* $K \in \mathbb{N}$, conjuntos de restricciones $\mathcal{ML}, \mathcal{CL}$, número máximo de iteraciones $maxIter$, número máximo de iteraciones consecutivas sin mejora $Nmax$

Inicializar contadores: $nIter \leftarrow 0$, $nIterWOImpr \leftarrow 0$
$bestScore \leftarrow \infty$
while $nIterWOImpr < Nmax$ **and** $nIter < maxIter$ **do**
  $centers \leftarrow SolGreedyAleatorizada(D, K, \mathcal{ML}, \mathcal{CL}, fobj)$
  Aplicar búsqueda local a $centers$ para obtener el vector de pertenencia $labels$, la matriz de centroides $centers$ y el valor de la función objetivo de la solución $score$
  if $bestScore > score$ **then**
    $bestLabels \leftarrow labels$
    $bestCenters \leftarrow centers$
    $bestScore \leftarrow score$
    $nIterWOImpr \leftarrow 0$
  else
    $nIterWOImpr \leftarrow nIterWOImpr + 1$
  end if
  $nIter \leftarrow nIter + 1$
end while
return $bestLabels$, $bestCenters$

Capítulo 7

Diseño experimental

En este capítulo se detallan todos los aspectos relacionados con el diseño experimental seguido. Al ser la mayoría de nuestras propuestas modificaciones sobre el algoritmo memético S-MDEClust [7], los resultados obtenidos se compararán directamente con los resultados proporcionados por la versión original de dicho algoritmo. Sin embargo, como este algoritmo es aproximado, no podemos saber qué tan próximas están las soluciones obtenidas del óptimo global. Para obtener una estimación de esto, ejecutaremos también el algoritmo exacto PC-SOS-SDP [8], explicado en el Capítulo 5.

7.1. Conjuntos de datos utilizados

Los conjuntos de datos utilizados en este trabajo son, en su mayoría, los mismos empleados en el artículo en el que se presenta el algoritmo exacto [8]. Además, se han incorporado dos nuevos conjuntos de datos: *Movement Libras*, para contar con un ejemplo con un número elevado de clases, y *Toxicity*, con el objetivo de incluir otro caso con alta dimensionalidad.

En total, son 14 conjuntos de datos reales de problemas de clasificación, con un número de instancias que varía entre 150 y 801, un número de características entre 4 y 20531, y un número de clases que oscila entre 2 y 15. Esta variedad permite evaluar el rendimiento de las propuestas en escenarios con diferentes grados de complejidad. En concreto, los conjuntos de datos utilizados son:

1. ***Iris***: conjunto clásico de clasificación de flores en tres especies distintas a partir de medidas de sépalos y pétalos.

2. ***Wine***: contiene características químicas de vinos para clasificarlos según su calidad.

3. ***Connectionist***: señales de sonar utilizadas para distinguir entre ecos reflejados por cilindros metálicos y por rocas con forma cilíndrica.

4. ***Seeds***: medidas de propiedades geométricas de granos pertenecientes a tres variedades distintas de trigo.

5. ***Heart***: diagnóstico de enfermedades cardíacas basado en medidas clínicas.

6. ***Vertebral***: conjunto de datos con características biomecánicas usadas para clasificar pacientes ortopédicos como normales o con alguna patología.

7. ***Computers***: recoge series temporales de consumo eléctrico en hogares británicos, registradas cada dos minutos durante 24 horas. El objetivo es clasificar si el dispositivo es un ordenador de sobremesa o un portátil.

8. ***Gene***: conjunto de datos con expresiones genéticas de pacientes con distintos tipos de tumores.

9. ***Movement Libras***: contiene 15 clases de movimientos de mano en LIBRAS, la lengua de signos brasileña.

10. ***Toxicity***: propiedades moleculares para predecir la toxicidad de compuestos químicos.

11. ***ECG5000***: contiene latidos cardíacos extraídos de un paciente con insuficiencia cardíaca congestiva, con el objetivo de clasificar los latidos en cinco categorías.

12. ***Ecoli***: conjunto de datos para predecir la localización subcelular de la bacteria *Escherichia coli* (E. coli) en función de propiedades fisicoquímicas.

13. ***Glass***: conjunto de datos que clasifica muestras de vidrio en seis tipos distintos, definidos según su composición en óxidos.

14. ***Accent***: conjunto de datos con grabaciones de palabras en inglés pronunciadas por hablantes de seis países distintos, utilizado para la detección y clasificación de acentos.

Estos conjuntos de datos pueden ser encontrados en los repositorios UCI [33] y UCR [34]. En la Tabla 7.1 se muestran las características de cada uno de ellos. En concreto, se detalla:

- n: número de instancias del conjunto de datos.
- d: número de atributos del conjunto de datos.

Dataset	n	d	k	r
Iris	150	4	3	50 y 100
Wine	178	13	3	50 y 100
Connectionist	208	60	2	50 y 100
Seeds	210	7	3	50 y 100
Heart	299	12	2	100 y 150
Vertebral	310	6	2	100 y 150
Computers	500	720	2	150 y 250
Gene	801	20531	5	200 y 400
Movement Libras	360	90	15	100 y 150
Toxicity	171	1203	2	50 y 100
ECG5000	500	140	5	150 y 250
Ecoli	336	7	8	100 y 150
Glass	214	9	6	50 y 100
Accent	329	12	6	100 y 150

Tabla 7.1: Información datasets

- k: número de clases del conjunto de datos.
- r: número de restricciones utilizado.

7.2. Conjunto de restricciones utilizados

Para ejecutar los algoritmos, tanto el exacto como el memético y nuestras propuestas, es necesario disponer de los conjuntos de restricciones ML y CL. En el caso de los conjunto de datos empleados en el artículo que introduce el algoritmo exacto [8], se utilizarán exactamente los mismos conjuntos de restricciones utilizados en dicho trabajo. Para los dos conjuntos de datos adicionales, se generarán restricciones siguiendo el mismo procedimiento descrito en ese artículo.

Para cada conjunto de datos con n instancias, se consideran seis configuraciones distintas de restricciones: tres con un número de restricciones aproximadamente igual a $n/2$ y otros tres con $n/4$ restricciones. Dentro de cada grupo se incluyen tres variantes: una con un 50 % de restricciones ML y 50 % de CL, otra con solo restricciones ML, y la última con solo restricciones CL.

Para cada una de las seis configuraciones de restricciones descritas, se generan cinco conjuntos diferentes de restricciones de forma aleatoria siguiendo el procedimiento clásico para generación de restricciones descrito en [3], lo que da lugar a un total de 30 variantes distintas para cada conjunto de datos. Dicho procedimiento consiste en elegir un par de instancias

del conjunto de datos de forma aleatoria y generar una restricción ML si las instancias pertenecen a la misma clase o una restricción CL si son de clases distintas. El procedimiento continúa hasta que se haya generado el número deseado de restricciones ML y CL.

7.3. Detalles de la implementación y parámetros de los algoritmos

Todas las pruebas experimentales se han realizado en un portátil ASUS TUF Dash F15, con un procesador Intel Core i7 12650H con 10 núcleos, 16 GB de RAM y Ubuntu 22.04.

7.3.1. Algoritmo exacto: PC-SOS-SDP

El algoritmo PC-SOS-SDP [8] está implementado en C++ y tiene algunas rutinas implementadas en MATLAB. Para resolver la relajación SDP, se emplea el software SDPNAL+ de MATLAB [35]. Además, recordemos que para calcular la cota superior, es necesario resolver el problema de optimización formulado en la ecuación 5.14; para ello, se hace uso de Gurobi [36]. El código fuente del algoritmo se puede encontrar en `https://github.com/antoniosudoso/pc-sos-sdp`.

Por otro lado, PC-SOS-SDP tiene varios parámetros de configuración. Estos parámetros no se deben especificar explícitamente en cada ejecución, sino que sus valores están recogidos en el archivo `clustering_c++/config.txt`. Para nuestras ejecuciones, se han mantenido todos los parámetros por defecto, excepto `SDP_SOLVER_FOLDER`, que se ha ajustado para indicar la ruta local de instalación de SDPNAL+.

Uno de estos parámetros es `BRANCH_AND_BOUND_MAX NODES`, que indica el número máximo de nodos en el árbol de búsqueda del algoritmo. Este parámetro está establecido en 200, y una vez que se alcanza este número máximo, se detiene el algoritmo y se devuelve la mejor solución encontrada. Otro parámetro es `BRANCH_AND_BOUND_VISITING_STRATEGY`, que define la estrategia utilizada para explorar el árbol de búsqueda. En este caso, su valor está fijado en 0, lo que implica que se utiliza la estrategia *best-first*, es decir, se exploran primero los nodos más prometedores.

La totalidad de estos parámetros, sus valores y su descripción se especifican en la Tabla A.1 del Apéndice.

Los parámetros de ejecución se detallan a continuación, especificando el orden en el que se deben proporcionar:

1. `DATASET`: ruta del conjunto de datos.

2. `K`: número de *clusters*
3. `CONSTRAINTS`: ruta del archivo de restricciones.
4. `LOG`: ruta del archivo de `log`.
5. `RESULT`: ruta del archivo con la solución encontrada.

7.3.2. Algoritmo memético: S-MDEClust (versión original)

El algoritmo S-MDEClust está implementado enteramente en Python. El código fuente de la versión original del algoritmo, es decir, sin ninguna de las modificaciones explicadas en el Capítulo 6, se encuentra en `https://github.com/pierlumanzu/s_mdeclust`.

Para obtener los resultados de este algoritmo, se emplea la asignación *greedy* y se hace uso del operador de mutación, lo cual se indica mediante los correspondientes parámetros de ejecución. El resto de parámetros se mantienen con sus valores por defecto, que son los recomendados en el artículo original donde se presenta el algoritmo [7]. En la Tabla 7.2 se detallan todos los parámetros del algoritmo, incluyendo el valor empleado en nuestra ejecución, su descripción y, cuando corresponde, los valores posibles que pueden tomar.

7.3.3. Algoritmos propuestos

Se han implementado dos programas: uno que incorpora todas las modificaciones sobre el algoritmo S-MDEClust presentadas en el capítulo anterior, y otro que implementa el enfoque GRASP descrito en la Sección 6.5. Todo el código fuente está disponible en `https://github.com/teresaCL/TFG`.

S-MDEClust modificado

Todos los parámetros descritos en la subsección anterior se siguen utilizando, y sus valores en las pruebas experimentales serán los mismos, salvo que se indique lo contrario. Para incorporar las modificaciones propuestas al algoritmo original, se han añadido nuevos parámetros y, en algunos casos, ampliado los valores permitidos de ciertos parámetros ya existentes.

Los nuevos parámetros y los parámetros modificados se muestran en la Tabla 7.3, junto a su descripción y su valor por defecto, que serán los que se utilicen para las pruebas experimentales a no ser que se indique lo contrario.

Parámetro	Valor	Descripción
`--dataset`	-	Ruta del archivo con el conjunto de datos.
`--constraints`	-	Ruta al archivo con las restricciones.
`--K`	-	Número de *clusters*.
`--seed`	42	Semilla para el generador pseudoaleatorio.
`--verbose`	False	Muestra el progreso del algoritmo si se especifica.
`--assignment`	greedy	Método de asignación. Valores posibles: ▪ `exact`: asignación exacta. ▪ `greedy`: asignación *greedy*.
`--mutation`	True	Emplea el operador de mutación si se especifica.
`--P`	20	Tamaño de la población.
`--Nmax`	5000	Número máximo de evaluaciones consecutivas sin mejora de la mejor solución.
`--max_iter`	∞	Número máximo de iteraciones del algoritmo.
`--tol_pop`	10^{-4}	Mínimo de diversidad de la población antes de detener el algoritmo.
`--Nmax_ls`	1	Número máximo de iteraciones consecutivas sin mejora en la búsqueda local.
`--max_iter_ls`	∞	Número máximo de iteraciones en la búsqueda local.
`--tol_sol`	10^{-6}	Tolerancia para comparar si una solución es mejor que otra.
`--F`	mdeclust	Parámetro F usado en el cruce. Valores posibles: ▪ `random`: valor aleatorio en el intervalo (0, 2). ▪ `mdeclust`: valor aleatorio en el intervalo (0.5, 0.8). ▪ valor numérico específico en el intervalo (0, 2).
`--alpha`	0.5	Parámetro α usado en la mutación.

Tabla 7.2: Parámetros de ejecución del algoritmo S-MDEClust

GRASP

En la Tabla 7.4 se detallan los parámetros utilizados en la ejecución del algoritmo GRASP, incluyendo su descripción y los valores empleados en los experimentos.

7.4. Métricas empleadas

Para evaluar el rendimiento de los algoritmos se han utilizado dos métricas principales:

- ***Score***: valor de la función objetivo del problema MSSC con restricciones, presentada en la ecuación 4.2.
- **Tiempo de ejecución**: tiempo necesario (en segundos) para completar la ejecución del algoritmo. Esta métrica permite comparar la eficiencia computacional de las distintas propuestas.

Además, en el caso del algoritmo exacto, se considera una métrica adicional: ***Gap***, que mide la diferencia relativa entre la cota superior y la inferior.

$$Gap_{\%} = \frac{CS - CI}{CS} \cdot 100 \tag{7.1}$$

donde CS hace referencia a la cota superior y CI a la cota inferior. Cuanto menor sea el valor de esta métrica, más próxima estará la solución del óptimo global.

7.5. Librerías y dependencias

La ejecución del algoritmo S-MDEClust y de las propuestas desarrolladas requiere de un entorno `Anaconda` configurado con las siguientes librerías y versiones:

- `Python` 3.11.9
- `pip` 24.0
- `NumPy` 2.0.0
- `SciPy` 1.14.0
- `pandas` 2.2.2
- `gurobipy` 11.0.2
- `NetworkX` 3.3

Algunas partes del código hacen uso del *solver* Gurobi. Para su correcta ejecución es necesario tener instalado el `Gurobi Optimizer`, así como disponer de una licencia válida. En el caso de uso académico, Gurobi ofrece licencias gratuitas que pueden obtenerse desde su página oficial [36].

Parámetro	**Valor por defecto**	**Descripción**
`--assignment`	greedy	Método de asignación. Valores posibles: ▪ `exact`: asignación exacta. ▪ `greedy`: asignación *greedy*. ▪ `greedy_rand`: asignación *greedy* aleatorizada. ▪ `greedy_rand_penalty`: asignación *greedy* aleatorizada con penalización.
`--F`	mdeclust	Parámetro F usado en el cruce. Valores posibles: ▪ `random`: valor aleatorio en el intervalo (0, 2). ▪ `mdeclust`: valor aleatorio en el intervalo (0.5, 0.8). ▪ `half_mdeclust`: valor aleatorio en el intervalo (0.25, 0.4). ▪ valor numérico específico en el intervalo (0, 2).
`--crossover`	original	Tipo de operador de cruce utilizado. Valores posibles: ▪ `original`: cruce original. ▪ `pbest_v1`: cruce pbest1. ▪ `pbest_v2`: cruce pbest2.
`--solis`	no	Indica si aplicar el algoritmo de Solis Wets. Valores posibles: ▪ `no`: no se aplica Solis Wets. ▪ `wo_penalty`: se aplica Solis Wets usando función objetivo sin penalización. ▪ `w_penalty`: se aplica Solis Wets usando función objetivo con penalización.
`--apply_LS_all`	True	Indica si se debe de aplicar la búsqueda local a todos los individuos de la población o no
`--restart`	0	Indica el número de reinicios
`--decrease_pop_size_reset`	False	Indica si disminuir el tamaño de la población en cada reinicio o no
`--shade`	False	Indica si utilizar SHADE o no.

Tabla 7.3: Parámetros de ejecución del algoritmo S-MDEClust, incluyendo las propuestas

Parámetro	**Valor**	**Descripción**
`--dataset`	-	Ruta del archivo con el conjunto de datos.
`--constraints`	-	Ruta al archivo con las restricciones.
`--K`	-	Número de *clusters*.
`--seed`	42	Semilla para el generador pseudoaleatorio.
`--verbose`	False	Muestra el progreso del algoritmo si se especifica.
`--assignment`	greedy	Método de asignación. Valores posibles: ▪ `greedy`: asignación *greedy*. ▪ `greedy`: asignación *greedy*. ▪ `greedy_rand`: asignación *greedy* aleatorizada.
`--Nmax`	3	Número máximo de iteraciones consecutivas sin mejora.
`--max_iter`	30	Número máximo de iteraciones del algoritmo.
`--Nmax_ls`	1	Número máximo de iteraciones consecutivas sin mejora en la búsqueda local.
`--max_iter_ls`	∞	Número máximo de iteraciones en la búsqueda local.
`--tol_sol`	10^{-6}	Tolerancia para comparar si una solución es mejor que otra.

Tabla 7.4: Parámetros de ejecución del algoritmo GRASP

Capítulo 8

Experimentación y resultados

En este capítulo se presentan los experimentos realizados para evaluar el rendimiento de las propuestas desarrolladas. Se muestran los resultados obtenidos, comparándolos con los del algoritmo de referencia, con el objetivo de analizar la efectividad de las modificaciones introducidas y justificar su utilidad a través de métricas objetivas.

8.1. Resultados del algoritmo referencia y del algoritmo exacto

El algoritmo de referencia es el algoritmo memético S-MDEClust [7]. Recordamos que uno de los objetivos principales de este proyecto es mejorar el rendimiento de dicho algoritmo. Sin embargo, dado que se trata de un método aproximado, con el fin de obtener soluciones óptimas que permitan evaluar la calidad de los resultados, también se utilizará el algoritmo exacto PC-SOS-SDP [8].

En la Tabla 8.1 se recogen los resultados del algoritmo exacto. Para cada conjunto de datos, se muestran los valores medios de las 30 ejecuciones de la función objetivo (*Score*), el tiempo de ejecución en segundos (*Time*) y del $Gap_{\%}$, la métrica presentada en la Ecuación 7.1. Cabe señalar que faltan los resultados de algunos conjuntos de datos, ya que no fue posible ejecutar el algoritmo exacto sobre ellos debido a las elevadas demandas de recursos computacionales.

Por otro lado, en la Tabla 8.2 se muestran los resultados obtenidos con la versión original del algoritmo S-MDEClust, es decir, sin ninguna de las modificaciones propuestas en el Capítulo 6. Al igual que antes, para cada

Dataset	Score	Gap	Time(s)
Iris	84.756	0.023 %	15.9
Wine	3421828.000	0.028 %	45.5
Connectionist	314.774	0.005 %	69.4
Seeds	624.257	0.045 %	80.0
Heart	3347.195	0.004 %	151.9
Vertebral	559533.900	0.022 %	178.9
Computers	320716.600	0.003 %	822.5
Gene	17808460.000	0.007 %	706.4

Tabla 8.1: Resultados algoritmo exacto

conjunto de datos se presentan los valores medios tanto de *Score*, como de *Time*.

Dataset	Score	Time(s)
Iris	84.756	1.7
Wine	3421827.432	2.2
Connectionist	314.774	2.7
Seeds	624.257	2.5
Heart	3347.200	6.8
Vertebral	559533.960	1.6
Computers	320716.689	12.9
Gene	17808477.337	225.6
Movement Libras	381.828	89.2
Toxicity	4.926e15	0.8
ECG5000	12734.756	39.0
Ecoli	15.627	39.9
Glass	90.755	13.9
Accent	29545.555	28.7

Tabla 8.2: Resultados algoritmo S-MDEClust original

Los resultados mostrados en la Tabla 8.2 se utilizarán como referencia para comparar el rendimiento de las propuestas desarrolladas. Por tanto, en las tablas que se presentan a lo largo de este capítulo no se mostrarán los valores absolutos de *Score* y *Time*, sino su diferencia absoluta con respecto a los resultados de referencia, es decir, la resta directa entre ambos valores (*Score* propuesta - *Score* original). Cabe recordar que el MSSC con restricciones es un problema de minimización, por lo que las diferencias negativas indican un mejor desempeño de nuestra propuesta. Cuanto mayor es el valor absoluto de dichas diferencias, mayor es la mejora obtenida.

Además, en las tablas presentadas se incluirá la media de estas diferencias y las veces que mejora, se mantiene constante o empeora el *Score* en

comparación con el algoritmo de referencia.

Por otro lado, podemos ver que en la mayoría de los casos las diferencias de *Score* entre las soluciones obtenidas por el algoritmo S-MDClust original y el exacto son de tan solo unas pocas décimas o centésimas. En algunos conjuntos de datos más sencillos, como *Iris*, *Wine*, *Connectionist* o *Seeds*, estas diferencias incluso desaparecen por completo, lo que indica que el algoritmo es capaz de alcanzar soluciones prácticamente óptimas en dichos casos. Esto indica que las mejoras en *Score* que pueden lograrse con las propuestas no serán de gran magnitud. No obstante, esto no implica que dichas mejoras carezcan de relevancia, especialmente si vienen acompañadas de reducciones en el tiempo de ejecución.

8.2. Acrónimos

Para simplificar la referencia a cada una de las propuestas realizadas, se utilizarán acrónimos a lo largo del capítulo. A continuación, se muestra la lista con todos los acrónimos empleados y su correspondiente descripción.

- AGR-RAND: Asignación *Greedy* aleatorizada (Subsección 6.1.1).
- AGR-RAND-P: Asignación *Greedy* aleatorizada con penalización (Subsección 6.1.2).
- SHADE: Uso de la estrategia para adaptación de parámetros SHADE (Subsección 6.3.1).
- PBEST1-F: Cruce pbest1 con $F \in [0.5, 0.8]$.(Subsección 6.3.2)
- PBEST2-F: Cruce pbest2 con $F \in [0.5, 0.8]$.(Subsección 6.3.2)
- PBEST1-F/2: Cruce pbest1 con $F \in [0.25, 0.4]$.(Subsección 6.3.2)
- PBEST2-F/2: Cruce pbest2 con $F \in [0.25, 0.4]$.(Subsección 6.3.2)
- SW-V1-WO-PEN: Versión 1 del Algoritmo Solis Wets usando la función objetivo sin penalización. (Subsección 6.2.1)
- SW-WO-PEN o SW: Versión 2 del Algoritmo Solis Wets usando la función objetivo sin penalización. (Subsección 6.2.1)
- SW-W-PEN: Versión 2 del Algoritmo Solis Wets usando la función objetivo con penalización. (Subsección 6.2.1)
- SEL-BL: Elegir a qué individuos de la población aplicar la búsqueda local, según lo explicado en la Subsección 6.2.2

- R-XIT-Y: Aplicar la estrategia de reinicio detallada en la Sección 6.4 manteniendo el tamaño de la población constante. X e Y toman valores numéricos: X indica el máximo de iteraciones sin mejora antes de activar el reinicio, mientras que Y especifica el número de reinicios. Por ejemplo, R-3IT-2 indica que el máximo de iteraciones seguidas sin mejora es 3 y que se realizan 2 reinicios.
- R-XIT-Y-DIS: similar al anterior. La única diferencia es que se disminuye el tamaño de la población después de cada reinicio.
- GRASP: enfoque GRASP propuesto en la Sección 6.5

Para los algoritmos que combinan varios componentes, se encadenarán los acrónimos de los componentes involucrados.

8.3. Enfoque GRASP

En la Tabla 8.3 se muestran los resultados de la propuesta GRASP, presentada en la Sección 6.5.

Dataset	Score	Time (s)
Iris	+0.025	-0.3
Wine	+24461.053	-0.1
Connectionist	+0.041	-0.6
Seeds	+0.008	+0.5
Heart	+1.681	-1.6
Vertebral	+10.438	+2.0
Computers	+134.362	+2.0
Gene	+101.498	-68.1
Movement Libras	+6.344	+72.2
Toxicity	-6.710e+12	+0.5
ECG5000	+130.177	+18.6
Ecoli	+0.172	+4.4
Glass	+2.320	-0.8
Accent	+93.915	+1.6
Promedio	-4.793e+11	+2.2
Veces mejor	1	6
Veces igual	0	0
Veces peor	13	8

Tabla 8.3: Resultados GRASP

Se observa que, aunque el promedio del *Score* sea menor que cero, esto se debe únicamente a la mejora obtenida en el conjunto de datos *Toxicity*.

Sin embargo, en el resto de los conjuntos de datos se aprecia un claro empeoramiento de la calidad de las soluciones en términos de *Score*. Además, el tiempo de ejecución promedio también empeora con respecto al algoritmo S-MDEClust.

8.4. Variantes de la asignación *greedy*

Las primeras modificaciones sobre el algoritmo S-MDEClust propuestas fueron acerca de la asignación *greedy*. En concreto, se plantearon dos variantes: la asignación *greedy* aleatorizada y la asignación *greedy* aleatorizada con penalización, descritas en la Subsección 6.1.1 y Subsección 6.1.2, respectivamente. Estas modificaciones se introdujeron con el objetivo de fomentar una mayor exploración del espacio de soluciones al añadir aleatoriedad durante el proceso de asignación.

En la Tabla 8.4 se muestran los resultados del algoritmo con la asignación *greedy* aleatorizada y en la Tabla 8.5 con la asignación *greedy* aleatorizada con penalización.

Dataset	Score	Time (s)
Iris	+0.000	+0.0
Wine	+0.000	+0.1
Connectionist	+0.000	+0.4
Seeds	+0.000	+0.1
Heart	-0.002	+0.5
Vertebral	+0.000	+0.1
Computers	-0.017	+1.4
Gene	+0.000	-16.6
Movement Libras	-0.069	-0.4
Toxicity	+0.000	+0.1
ECG5000	+0.023	+1.6
Ecoli	+0.000	-1.4
Glass	-0.021	+1.4
Accent	+0.501	+0.1
Promedio	+0.030	-0.9
Veces mejor	4	3
Veces igual	8	1
Veces peor	2	10

Tabla 8.4: Resultados AGR-RAND

Dataset	Score	Time (s)
Iris	+0.000	+0.0
Wine	+39.893	+0.1
Connectionist	+0.000	+0.4
Seeds	+0.000	+0.2
Heart	-0.004	+0.0
Vertebral	+0.000	+0.1
Computers	-0.017	+1.4
Gene	+0.000	-15.5
Movement Libras	+0.030	-2.2
Toxicity	+0.000	+0.1
ECG5000	-0.044	+1.6
Ecoli	+0.002	+3.2
Glass	-0.035	+3.5
Accent	-0.070	+2.9
Promedio	+2.840	-0.3
Veces mejor	5	2
Veces igual	6	2
Veces peor	3	10

Tabla 8.5: Resultados AGR-RAND-P

Si comparamos las dos tablas anteriores, podemos ver que el rendimiento del algoritmo con asignación *greedy* aleatorizada (AGR-RAND) parece ligeramente superior con respecto a la versión con penalización (AGR-RAND-P).

En términos de *Score*, aunque AGR-RAND mejora una vez menos que la versión con penalización, también empeora una vez menos. Además, cuando se producen empeoramientos, estos son menos acusados en la versión sin penalización. Por otro lado, aunque en ambas se consigue una reducción del promedio del tiempo de ejecución, con AGR-RAND esta reducción es más pronunciada, consiguiendo mejorar en los tres conjuntos de datos en los que el algoritmo original tardaba más: *Gene*, *Movement Libras* y *Ecoli*. Además, aunque el tiempo de ejecución empeora más veces de las que mejora, estos incrementos son en la mayoría de los casos de unas pocas décimas de segundo.

Por lo anterior, si tuviéramos que elegir entre una de las dos variantes, optaríamos por AGR-RAND. Sin embargo, no está claro si supone o no una mejora respecto al algoritmo original: el tiempo sí se reduce en promedio, pero el *Score*, aunque mejore en más ocasiones de las que empeora, estas mejoras son muy pequeñas.

8.5. Evolución Diferencial

8.5.1. SHADE

En la Subsección 6.3.1 describimos la estrategia de adaptación de parámetros SHADE, que adapta los parámetros F y CR en función de los valores de estos que dieron buenos resultados en iteraciones anteriores. Esta modificación se introdujo con el objetivo de guiar la búsqueda de forma más informada, con la esperanza de mejorar los resultados obtenidos.

En la Tabla 8.6 se muestran los resultados obtenidos al añadir esta estrategia al algoritmo original.

Como podemos ver, el *Score* de las soluciones obtenidas solo empeora respecto al algoritmo original en dos conjuntos de datos, mientras que mejora en cinco. Además, también se obtiene una reducción promedio en el tiempo de ejeucución, destancando especialmente la reducción obtenida en el conjunto de datos *Gene*, que es el que más tiempo de ejecución requería con la versión original del algoritmo. Sin embargo, los empeoramientos en términos de *Score* son, en general, de mayor magnitud que las mejoras, lo que provoca que el promedio de las diferencias del *Score* sea muy cercano a cero.

Dataset	Score	Time (s)
Iris	+0.000	+0.2
Wine	+0.000	-0.1
Connectionist	+0.000	+0.2
Seeds	+0.000	+0.0
Heart	-0.004	-0.4
Vertebral	+0.000	-0.1
Computers	-0.149	+1.6
Gene	+0.000	-24.2
Movement Libras	+0.129	+4.6
Toxicity	+0.000	+0.1
ECG5000	-0.084	-1.7
Ecoli	-0.002	-0.8
Glass	-0.006	+0.9
Accent	+0.091	-1.4
Promedio	-0.002	-1.5
Veces mejor	5	7
Veces igual	7	1
Veces peor	2	6

Tabla 8.6: Resultados SHADE

8.5.2. Cambios en el operador de cruce

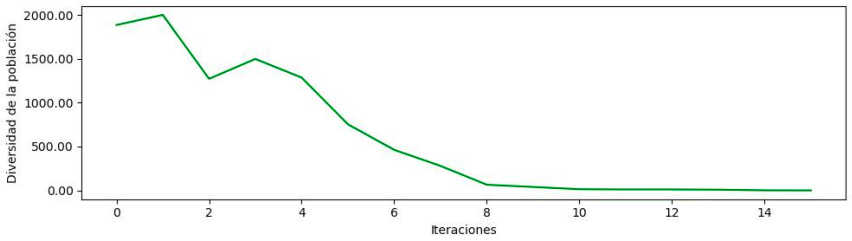
Se propusieron dos nuevos operadores de cruce en la Subsección 6.3.2: pbest1 y pbest2, ambos basados en la estrategia de cruce *current-to-pbest*. Con estas dos propuestas, se pretendía equilibrar explotación y exploración, guiando la búsqueda hacia soluciones prometedoras sin perder diversidad en la población.

Hasta ahora, excepto al utilizar la estrategia de adaptación de parámetros SHADE, siempre se generaba el parámetro de cruce F aleatoriamente en cada iteración dentro del rango (0.5, 0.8), ya que este rango de valores es el considerado como óptimo según los autores del algoritmo S-MDEClust. Sin embargo, con los dos nuevos operadores de cruce se han realizado pruebas tanto escogiendo el parámetro F en el rango mencionado, como en el rango (0.25, 0.4), es decir, reduciendo F a la mitad. Esto se debe a la forma en la que se genera la nueva solución con el cruce pbest1 y pbest2: como se observa en la Ecuación 6.8, se suma al individuo actual dos diferencias distintas, ambas escaladas por F. Esto implica un desplazamiento potencialmente mayor respecto al cruce original (Ecuación 5.3), en el que solo se suma una diferencia. Para compensar esto, es por lo que se reduce F a la mitad.

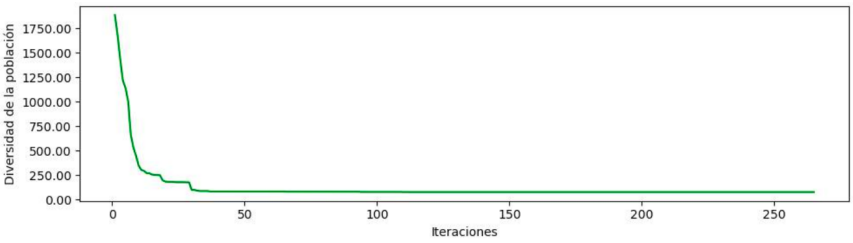
Por otro lado, como vimos anteriormente, el algoritmo S-MDEClust tiene tres criterios de parada, que vienen configurados por tres parámetros:

1. `Nmax`: número máximo de evaluaciones consecutivas sin mejora.
2. `max_iter`: número máximo de iteraciones del algoritmo.
3. `tol_pop`: mínimo de diversidad de la población. Esta diversidad, como vimos en el algoritmo 1, se mide como la suma de las diferencias del valor de la función objetivo para cada par de soluciones de la población. Cuando se alcanza este umbral se detiene el algoritmo.

De estos tres criterios de parada, el único que realmente ha sido utilizado hasta ahora es el tercero. El primero no se activa porque, por defecto, el valor de `max_iter` está fijado en infinito, y el segundo (`Nmax`) también es demasiado alto: está establecido en 5000 evaluaciones, lo que para una población de 20 individuos, equivale a 250 iteraciones completas sin mejora. En todas las ejecuciones que se han realizado hasta el momento, el algoritmo siempre se detiene porque la población alcanza el umbral mínimo de diversidad. Sin embargo, al ejecutar el algoritmo con los dos nuevos cruces, especialmente las versiones en las que se reduce F a la mitad, la diversidad decrece más despacio y cuanto más cercana a 0 es, más se ralentiza su decrecimiento, como podemos ver en la Figura 8.1. Esto hace que no se alcance el umbral de diversidad mínima y que el algoritmo llegue a las 250 iteraciones completas



(a) Algoritmo S-MDEClust



(b) Algoritmo PBEST1-F/2

Figura 8.1: Evolución de la diversidad

sin mejora antes de detenerse, lo que aumenta muchísimo los tiempos de ejecución. Sin embargo, hemos observado que después de 5 iteraciones sin mejora, es poco frecuente que la mejor solución vuelva a mejorar. Por lo cual, fijaremos el parámetro `Nmax` a $5 \cdot 20 = 100$. Este valor se mantendrá para todas las ejecuciones en las que se use el cruce pbest1 o pbest2, a no ser que se indique lo contrario.

El motivo de este ralentizamiento de la disminución de la diversidad puede ser en parte porque, al estar guiados por los mejores individuos de la población, tienden a explorar de forma más localizada en torno a las regiones prometedoras. Esto evita que toda la población converja rápidamente a un mismo punto, estabilizando la diversidad en un valor bajo pero no nulo, suficiente para impedir que se active el criterio de parada basado en diversidad.

El la Tabla 8.7 se presentan los resultados del algoritmo con el cruce pbest1 y pbest2, con el parámetro F en el rango (0.5, 0.8).

Como podemos ver, aunque con el cruce pbest2 se mejore el *Score* en dos conjuntos de datos más y empeore en uno menos, el promedio de las diferencias del *Score* es mayor que cero. Al analizar los casos en los que mejora el resultado con el pbest2, se observa que dichas mejoras suelen ser de menor magnitud en comparación con las obtenidas con el cruce pbest1. Además, en el único conjunto de datos en el que empeora el *Score* con pbest2 (*ECG500*)

	PBEST1-F		PBEST2-F	
Dataset	Score	Time (s)	Score	Time (s)
Iris	+0.000	+0.2	+0.000	+0.0
Wine	+0.000	+0.2	+0.000	+0.0
Connectionist	+0.000	+0.2	+0.000	+0.0
Seeds	+0.000	+0.1	+0.000	+0.2
Heart	-0.004	+0.0	-0.004	-0.6
Vertebral	+0.000	+0.2	+0.000	+0.1
Computers	-0.154	-1.1	-0.094	-2.3
Gene	+0.000	-6.2	+0.000	-20.0
Movement Libras	-0.303	-21.3	-0.027	-31.6
Toxicity	+0.000	+0.3	+0.000	+0.1
ECG5000	+0.020	-8.1	+0.241	-9.5
Ecoli	+0.003	-4.7	-0.004	-13.0
Glass	+0.000	-1.9	-0.009	-2.8
Accent	-0.025	-0.1	-0.050	-3.0
Promedio	-0.033	-3.0	+0.004	-5.9
Veces mejor	4	7	6	8
Veces igual	8	1	7	3
Veces peor	2	6	1	3

Tabla 8.7: Comparativa PBEST1-F y PBEST2-F

podemos ver que este empeoramiento es considerablemente mayor que el observado con pbest1. Por esto, podemos decir que con el cruce pbest1 se obtienen mejores resultados que con el cruce pbest2.

Además, en este caso sí que se puede apreciar que con el algoritmo PBEST1-F se consigue mejorar con respecto al algoritmo original, tanto en el *Score* como ya hemos comentado, pero también en el tiempo. Este se reduce en promedio en 3 segundos, siendo la mejora especialmente notable (de hasta 21 segundos) en los conjuntos de datos en los que el algoritmo original presentaba mayores tiempos de ejecución, como *Movement Libras* y *Gene*.

Por otro lado, en la Tabla 8.8 se muestran los resultados correspondientes al uso de los cruces pbest1 y pbest2, pero generando el parámetro F en el rango (0.25, 0.4).

Podemos observar que PBEST2-F/2 proporciona mejores resultados que PBEST1-F/2 tanto en términos de *Score*, como en tiempo de ejecución. Sin embargo, parece que PBEST1-F sigue siendo superior a PBEST2-F/2: aunque el *Score* con el primer algoritmo mejore en un conjunto de datos menos que con el segundo, en promedio, PBEST1-F consigue encontrar soluciones con mejor *Score*. Además, con PBEST2-F/2 la reducción del tiempo pro-

	PBEST1-F/2		PBEST2-F/2	
Dataset	Score	Time (s)	Score	Time (s)
Iris	+0.000	+0.4	+0.000	+0.2
Wine	+0.000	+0.4	+0.000	+0.4
Connectionist	+0.000	+0.5	+0.000	+0.5
Seeds	+0.000	+0.3	+0.000	+0.4
Heart	-0.004	+0.0	-0.004	-0.3
Vertebral	+0.000	+0.3	+0.000	+0.3
Computers	-0.203	-0.3	-0.203	-0.2
Gene	+0.000	+13.6	+0.000	+8.5
Movement Libras	-0.177	-8.7	-0.024	-9.8
Toxicity	+0.000	+0.4	+0.000	+0.3
ECG5000	+0.164	-3.6	-0.085	-1.4
Ecoli	-0.002	-2.2	+0.001	-5.9
Glass	+0.034	-1.6	+0.007	-0.2
Accent	-0.057	+4.4	-0.070	+0.9
Promedio	-0.018	+0.3	-0.027	-0.4
Veces mejor	5	5	5	6
Veces igual	7	1	7	0
Veces peor	2	8	2	8

Tabla 8.8: Comparativa PBEST1-F/2 y PBEST2-F/2

medio de ejecución es muy limitada, de apenas 0.4 segundos, y en el peor caso el tiempo empeora en más de 8 segundos. En cambio, con PBEST1-F la reducción promedio es de 3 segundos, y el mayor empeoramiento observado no supera las 0.3 décimas de segundo. Esta diferencia en tiempos se debe a que al utilizar un valor de F menor en *PBEST2-F/2*, las modificaciones introducidas al generar nuevas soluciones mediante el cruce son más pequeñas, por lo que el algoritmo tarda más en converger.

8.6. Búsqueda Local

8.6.1. Uso del algoritmo Solis Wets

El algoritmo Solis Wets, como comentamos en la Subsección 6.2.1, se aplica a un único individuo en cada iteración, justo antes de la fase de búsqueda local, con el objetivo de refinar la solución más prometedora antes de dicha fase. Recordamos que este algoritmo emplea dos parámetros que se adaptan dinámicamente: *bias*, que estima la dirección hasta el óptimo local y se inicializa a cero, pues *a priori* no disponemos de información acerca de dónde se encuentra el óptimo; y el parámetro ρ, que regula la

magnitud de los desplazamientos aleatorios que se aplican a la solución en cada paso. Un valor comúnmente utilizado para inicializar ρ es 0.1, pero como en nuestro caso los datos no están normalizados, este valor fijo puede resultar inadecuado. Por ello, el valor de ρ incial se calculará como el 10 % del rango promedio de los atributos del conjunto de datos. Esta forma de inicialización permite ajustar ρ a la escala del problema.

En la Tabla 8.9 se muestran los resultados del algoritmo SW-V1-WO-PEN junto con los resultados de la mejor propuesta hasta el momento: PBEST1-F.

	PBEST1-F		SW-V1-WO-PEN	
Dataset	Score	Time (s)	Score	Time (s)
Iris	+0.000	+0.2	+0.000	-0.1
Wine	+0.000	+0.2	+0.000	+0.0
Connectionist	+0.000	+0.2	+0.000	+0.4
Seeds	+0.000	+0.1	+0.000	+0.3
Heart	-0.004	+0.0	-0.004	+0.9
Vertebral	+0.000	+0.2	+0.000	+0.0
Computers	-0.154	-1.1	-0.154	+1.0
Gene	+0.000	-6.2	+0.000	+26.7
Movement Libras	-0.303	-21.3	+0.041	+2.7
Toxicity	+0.000	+0.3	+0.000	+0.1
ECG5000	+0.020	-8.1	+0.689	+2.0
Ecoli	+0.003	-4.7	+0.001	-0.1
Glass	+0.000	-1.9	-0.005	+2.2
Accent	-0.025	-0.1	-0.069	+4.1
Promedio	-0.033	-3.0	+0.036	+2.9
Veces mejor	4	7	4	2
Veces igual	8	1	7	2
Veces peor	2	6	3	10

Tabla 8.9: Comparativa PBEST1-F y SW-V1-WO-PEN

Por otro lado, en la Tabla 8.10 se presentan los resultados del algoritmo SW-WO-PEN comparados también con los de PBEST1-F.

Como podemos ver, tanto en el algoritmo SW-V1-WO-PEN como en el SW-WO-PEN, la diferencia se *Score* promedio es mayor que cero. Sin embargo, esta diferencia es considerablemente mayor en el segundo caso, debido principalmente al empeoramiento del *Score* obtenido para el conjunto de datos *Wine* con el algoritmo SW-WO-PEN. Cabe señalar que el valor absoluto del *Score* en este conjunto de datos es muy alto (más de tres millones), por lo que el impacto relativo de dicho empeoramiento es en realidad menor de lo que sugiere el valor bruto. Además, en otros conjuntos de da-

	PBEST1-F		SW-WO-PEN	
Dataset	Score	Time (s)	Score	Time (s)
Iris	+0.000	+0.2	+0.000	-0.1
Wine	+0.000	+0.2	+39.893	+0.0
Connectionist	+0.000	+0.2	+0.000	+0.6
Seeds	+0.000	+0.1	+0.000	+0.2
Heart	-0.004	+0.0	-0.004	+0.7
Vertebral	+0.000	+0.2	+0.000	+0.0
Computers	-0.154	-1.1	-0.203	+2.0
Gene	+0.000	-6.2	+0.000	+14.3
Movement Libras	-0.303	-21.3	-0.003	+2.9
Toxicity	+0.000	+0.3	+0.000	+0.1
ECG5000	+0.020	-8.1	-0.060	+2.9
Ecoli	+0.003	-4.7	+0.001	-1.0
Glass	+0.000	-1.9	+0.036	+2.2
Accent	-0.025	-0.1	+0.308	+4.9
Promedio	-0.033	-3.0	+2.855	+2.1
Veces mejor	4	7	4	2
Veces igual	8	1	6	2
Veces peor	2	6	4	10

Tabla 8.10: Comparativa PBEST1-F y SW-WO-PEN

tos como *Computers* la mejora es más grande con SW-WO-PEN que con SW-V1-WO-PEN, por lo que el rendimiento de ambos algoritmos en cuanto al *Score* realmente está más igualado de lo que puede parecer en un primer momento mirando las tablas.

Donde sí se aprecia una diferencia más clara es en el tiempo de ejecución: SW-WO-PEN presenta un tiempo promedio inferior, y el empeoramiento en el peor caso se limita a 14 segundos, frente a casi 27 segundos en el caso de SW-V1-WO-PEN.

De cualquier caso, queda claro que el rendimiento de estos dos algoritmos no supera al de la mejor propuesta hasta ahora: PBEST1-F.

En la tabla Tabla 8.11 se muestran los resultados del algoritmo SW-W-PEN, que es equivalente a SW-WO-PEN pero utilizando la función objetivo con penalización. Una vez más, los resultados se comparan con los obtenidos por la mejor propuesta hasta el momento: PBEST1-F.

En este caso, se ve claramente que la modificación no ha tenido un impacto positivo. Aunque el tiempo, en promedio, se reduce levemente, los resultados en cuanto al *Score* empeoran notablemente: solo mejora en dos de los 14 conjuntos de datos y empeora en seis de ellos.

	PBEST1-F		SW-W-PEN	
Dataset	Score	Time (s)	Score	Time (s)
Iris	+0.000	+0.2	+0.000	+0.1
Wine	+0.000	+0.2	+39.893	+0.0
Connectionist	+0.000	+0.2	+0.000	+0.1
Seeds	+0.000	+0.1	+0.000	+0.3
Heart	-0.004	+0.0	-0.004	+0.6
Vertebral	+0.000	+0.2	+0.000	+0.1
Computers	-0.154	-1.1	-0.203	-1.0
Gene	+0.000	-6.2	+0.000	+12.1
Movement Libras	-0.303	-21.3	+0.078	-9.7
Toxicity	+0.000	+0.3	+0.000	+0.3
ECG5000	+0.020	-8.1	+0.847	-5.4
Ecoli	+0.003	-4.7	+0.003	-9.1
Glass	+0.000	-1.9	+0.059	-0.9
Accent	-0.025	-0.1	+0.308	+5.3
Promedio	-0.033	-3.0	+2.927	-0.5
Veces mejor	4	7	2	5
Veces igual	8	1	6	1
Veces peor	2	6	6	8

Tabla 8.11: Comparativa PBEST1-F y SW-W-PEN

8.6.2. Combinando el uso de varias propuestas

Ahora vamos a estudiar qué sucede al combinar dos de las propuestas vistas: SW-WO-PEN y AGR-RAND, comparándolo con el resultado de SW-WO-PEN. Los resultados se muestran en la Tabla 8.12

Como podemos ver, al añadir la asignación *greedy* aleatorizada se obtienen mejores resultados en términos de *Score*. Podemos observar que en el conjunto de datos *Wine* ya no empeora el *Score* como sucedía con el algoritmo SW-WO-PEN, y en otros conjuntos de datos como el *Movement Libras* se consigue una mejora significativa. También es cierto que el tiempo de ejecución aumenta y hay en algunos casos en los que el *Score* empeora con respecto al algoritmo SW-WO-PEN, aunque en promedio sí que se consigue una mejora del *Score*.

	SW-WO-PEN		SW-WO-PEN-AGR-RAND	
Dataset	Score	Time (s)	Score	Time (s)
Iris	+0.000	-0.1	+0.000	+0.3
Wine	+39.893	+0.0	+0.000	+0.6
Connectionist	+0.000	+0.6	+0.000	+1.0
Seeds	+0.000	+0.2	+0.000	+0.8
Heart	-0.004	+0.7	-0.004	+1.3
Vertebral	+0.000	+0.0	+0.000	+0.5
Computers	-0.203	+2.0	-0.154	+4.2
Gene	+0.000	+14.3	+0.000	+29.2
Movement Libras	-0.003	+2.9	-0.328	+15.5
Toxicity	+0.000	+0.1	+0.000	+0.4
ECG5000	-0.060	+2.9	+0.058	+5.6
Ecoli	+0.001	-1.0	+0.003	+4.9
Glass	+0.036	+2.2	-0.031	+4.0
Accent	+0.308	+4.9	+0.064	+4.9
Promedio	+3.074	+1.2	-0.030	+3.4
Veces mejor	4	2	4	0
Veces igual	5	2	7	0
Veces peor	4	10	3	10

Tabla 8.12: Comparativa SW-WO-PEN y SW-WO-PEN-AGR-RAND

A continuación, en la Tabla 8.13, se muestran los resultados de la mejor propuesta hasta ahora: PBEST1-F junto con los resultados obtenidos combinando este operador de cruce y la búsqueda local Solis Wets (PBEST1-F-SW-WO-PEN).

Como se puede observar, al incorporar el uso del algoritmo de Solis Wets a la versión con el operador de cruce pbest1, el rendimiento general se ve ligeramente reducido en comparación con la versión que no lo incluye. La reducción en el tiempo promedio de ejecución es menor. Además, la mejora del *Score* promedio también disminuye, y es que aunque en dos conjuntos de datos (*ECG5000* y *Ecoli*) se mejora el *Score* con respecto al algoritmo PBEST1-F, hay tres casos en los que dicho valor empeora (*Glass*, *Accent* y *Movement Libras*).

Ahora vamos a analizar los resultados obtenidos al combinar el nuevo operador de cruce pbest1 con la asignación *greedy* aleatorizada, con y sin el uso del algoritmo Solis Wets. Los resultados se muestran en la Tabla 8.14.

Podemos observar que las soluciones obtenidas por el algoritmo PBEST1-F-AGR-RAND son ligeramente inferiores, tanto en *Score* como en tiempo de ejecución, en comparación con la versión original sin asignación *greedy* aleatorizada (PBEST1-F), que hasta el momento se considera la mejor pro-

	PBEST1-F		PBEST1-F-SW-WO-PEN	
Dataset	Score	Time (s)	Score	Time (s)
Iris	+0.000	+0.2	+0.000	+0.2
Wine	+0.000	+0.2	+0.000	+0.5
Connectionist	+0.000	+0.2	+0.000	+0.4
Seeds	+0.000	+0.1	+0.000	+0.5
Heart	-0.004	+0.0	-0.004	-0.3
Vertebral	+0.000	+0.2	+0.000	+0.2
Computers	-0.154	-1.1	-0.154	+0.8
Gene	+0.000	-6.2	+0.000	+15.4
Movement Libras	-0.303	-21.3	-0.110	-15.7
Toxicity	+0.000	+0.3	+0.000	+0.2
ECG5000	+0.020	-8.1	-0.055	-2.7
Ecoli	+0.003	-4.7	+0.001	-6.6
Glass	+0.000	-1.9	+0.010	+0.5
Accent	-0.025	-0.1	-0.017	-1.5
Promedio	-0.033	-3.0	-0.024	-0.6
Veces mejor	4	7	5	5
Veces igual	8	1	7	0
Veces peor	2	6	2	9

Tabla 8.13: Comparativa PBEST1-F y PBEST1-F-SW-WO-PEN

puesta. No obstante, al incorporar también el algoritmo de Solis Wets, los resultados mejoran notablemente, superando incluso a PBEST1-F. Aunque el *Score* promedio es muy similar, destaca el hecho de que no hay ningún conjunto de datos en el que se obtenga un *Score* peor que con el algoritmo original. Además, aunque el tiempo promedio de ejecución es superior al obtenido con PBEST1-F, sigue siendo competitivo, mejorando en promedio al algoritmo de referencia.

	PBEST1-F-AGR-RAND		PBEST1-F-AGR-RAND-SW	
Dataset	Score	Time (s)	Score	Time (s)
Iris	+0.000	+0.3	+0.000	+0.4
Wine	+0.000	+0.4	+0.000	+0.5
Connectionist	+0.000	+0.6	+0.000	+0.7
Seeds	+0.000	+0.5	+0.000	+0.6
Heart	-0.004	-0.5	-0.004	+0.2
Vertebral	+0.000	+0.4	+0.000	+0.4
Computers	-0.154	+0.4	-0.144	+1.8
Gene	+0.000	-7.9	+0.000	+10.9
Movement Libras	-0.103	-17.0	-0.153	-13.2
Toxicity	+0.000	+0.3	+0.000	+0.3
ECG5000	+0.077	-2.4	-0.087	-0.7
Ecoli	+0.001	-5.9	-0.001	-3.9
Glass	+0.018	-1.0	-0.016	+0.7
Accent	-0.033	-2.8	-0.070	-1.4
Promedio	-0.014	-2.5	-0.034	-0.2
Veces mejor	4	7	7	4
Veces igual	7	0	7	0
Veces peor	3	7	0	10

Tabla 8.14: Comparativa PBEST1-F-AGR-RAND y PBEST1-F-AGR-RAND-SW

8.6.3. Seleccionar a qué individuos aplicar la búsqueda local

En la Subsección 6.2.2 se propuso, en lugar de aplicar la búsqueda local a todas las soluciones de la población, seleccionar ciertas soluciones y aplicar la búsqueda local únicamente sobre ellas. En concreto, se selecciona el 10 % de las mejores soluciones, con el objetivo de refinar aquellas soluciones más prometedoras, y un 10 % adicional elegido de forma aleatoria entre el resto de la población, con la intención de favorecer la diversidad de la población.

En la Tabla 8.15 se muestran los resultados obtenidos siguiendo esta estrategia de selección, comparándola con la mejor propuesta hasta ahora.

Podemos ver que claramente la introducción de esta estrategia de selección ha tenido un impacto negativo en los resultados, tanto en *Score* como en el tiempo de ejecución.

Como ya se comentó en la Subsección 6.2.2, la búsqueda local es uno de los componentes más costosos computacionalmente del algoritmo, ya que implica resolver un problema de optimización para asignar las instancias del conjunto de datos a los centroides de la solución. Al no aplicar la búsqueda local sobre todas las soluciones, disminuye el tiempo de ejecución prome-

	PBEST1-F-AGR-RAND-SW		SEL-BL	
Dataset	Score	Time (s)	Score	Time (s)
Iris	+0.000	+0.4	+0.000	-0.2
Wine	+0.000	+0.5	+0.000	+0.3
Connectionist	+0.000	+0.7	+0.000	+0.3
Seeds	+0.000	+0.6	+0.000	-0.1
Heart	-0.004	+0.2	-0.001	-0.7
Vertebral	+0.000	+0.4	+0.000	+0.0
Computers	-0.144	+1.8	+0.669	-1.6
Gene	+0.000	+10.9	+0.000	-17.8
Movement Libras	-0.153	-13.2	+0.212	+41.3
Toxicity	+0.000	+0.3	+0.000	+0.3
ECG5000	-0.087	-0.7	+0.021	+7.5
Ecoli	-0.001	-3.9	+0.003	+6.6
Glass	-0.016	+0.7	+0.112	+5.4
Accent	-0.070	-1.4	+0.433	+4.9
Promedio	-0.034	-0.2	+0.103	+3.3
Veces mejor	7	4	1	5
Veces igual	7	0	7	1
Veces peor	0	10	6	8

Tabla 8.15: Comparativa PBEST1-F-AGR-RAND-SW y SEL-BL

dio por iteración. Sin embargo, esto provoca un aumento en el número de iteraciones necesarias para que el algoritmo converja, y lo hace hacia soluciones de menor calidad. Como consecuencia, se incrementa el tiempo total de ejecución y se obtiene un *Score* promedio peor.

8.7. Introducción de la estrategia de reinicio de población

En la Sección 6.4 comentamos que existen varios factores que determinan cúando y cómo se llevan a cabo los reinicios. En un primer momento, se establece un máximo de 4 reinicios, los cuales se desencadenan tras detectar un total de 2 iteraciones completas sin mejora en la solución. Es decir, para el parámetro `--Nmax` se usará el valor $20 \cdot 2 = 40$ y para `--restart` el valor 4. Además, se mantendrá el tamaño de la población constante en todo momento. Los resultados se muestran en la Tabla 8.16, junto con los resultados de la mejor propuesta hasta el momento: PBEST1-F-AGR-RAND-SW.

Observamos que, en general, las mejoras del *Score* son más pronunciadas que con el algoritmo PBEST1-F-AGR-RAND-SW, siendo la excepción

	PBEST1-F-AGR-RAND-SW		R-2IT-4	
Dataset	Score	Time (s)	Score	Time (s)
Iris	+0.000	+0.4	+0.000	+7.7
Wine	+0.000	+0.5	+0.000	+9.3
Connectionist	+0.000	+0.7	+0.000	+10.4
Seeds	+0.000	+0.6	+0.000	+10.4
Heart	-0.004	+0.2	-0.004	+15.9
Vertebral	+0.000	+0.4	+0.000	+6.5
Computers	-0.144	+1.8	-0.198	+39.6
Gene	+0.000	+10.9	+0.000	+607.2
Movement Libras	-0.153	-13.2	-0.194	+149.6
Toxicity	+0.000	+0.3	-4.983e+13	+4.5
ECG5000	-0.087	-0.7	-0.085	+96.8
Ecoli	-0.001	-3.9	-0.004	+86.0
Glass	-0.016	+0.7	-0.033	+32.8
Accent	-0.070	-1.4	+0.036	+64.4
Promedio	-0.034	-0.2	-3.559e+12	+81.5
Veces mejor	7	4	7	0
Veces igual	7	0	6	0
Veces peor	0	10	1	14

Tabla 8.16: Comparativa PBEST1-F-AGR-RAND-SW y R-2IT-4

el conjunto de datos *Accent*. Destaca especialmente la mejora obtenida en el conjunto de datos *Toxicity*. Sin embargo, esta mejora en calidad viene acompañada de un aumento drástico en el tiempo de ejecución, como era de esperar, que prácticamente se multiplica por cinco en todos los casos al introducir los cuatro reinicios.

Seguidamente, vamos a comprobar si aún se consigue una mejora en los resultados reduciendo el número de reinicios a 2 y aumentando a 3 el número de iteraciones sin mejora necesarias para activar el reinicio. De esta forma, se espera reducir el tiempo de ejecución promedio respecto a la versión R-2IT-4, manteniendo al mismo tiempo una buena calidad en las soluciones. En la Tabla 8.17 se muestran los resultados obtenidos con esta configuración, con y sin el uso del algoritmo de Solis Wets.

Vemos que con el algoritmo R-3IT-2 se consigue reducir prácticamente a la mitad el incremento promedio de tiempo de ejecución respecto a la variante con 4 reinicios R-2IT-4, aunque se obtienen peores resultados de media. Al introducir también la búsqueda de Solis Wets, observamos que aunque el tiempo promedio empeore levemente, los resultados en términos de *Score* mejoran notablemente. En concreto, no se observa ningún empeoramiento respecto al algoritmo original, y las mejoras obtenidas superan en la mayoría

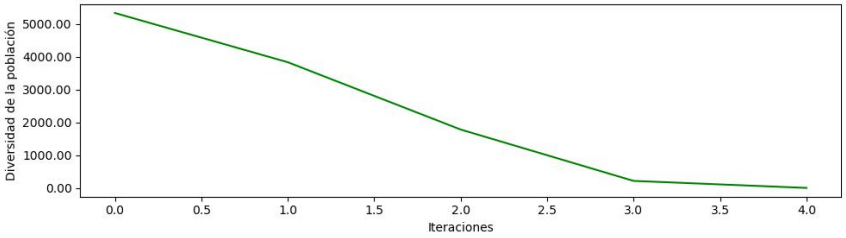
	R-3IT-2		R-3IT-2-SW	
Dataset	Score	Time (s)	Score	Time (s)
Iris	+0.000	+4.1	+0.000	+4.3
Wine	+0.000	+4.8	+0.000	+4.6
Connectionist	+0.000	+5.2	+0.000	+5.6
Seeds	+0.000	+5.6	+0.000	+5.7
Heart	-0.004	+8.8	-0.004	+9.2
Vertebral	+0.000	+3.4	+0.000	+3.7
Computers	-0.198	+19.7	-0.203	+20.1
Gene	+0.000	+353.1	+0.000	+405.7
Movement Libras	-0.081	+67.4	-0.164	+73.9
Toxicity	-4.983e+13	+2.4	-4.983e+13	+2.5
ECG5000	+0.002	+46.9	-0.069	+49.0
Ecoli	-0.003	+40.3	-0.004	+39.7
Glass	-0.034	+15.4	-0.038	+17.8
Accent	+0.254	+35.7	-0.063	+41.5
Promedio	-3.559e+12	+43.8	-3.559e+12	+48.8
Veces mejor	6	0	8	0
Veces igual	6	0	6	0
Veces peor	2	14	0	14

Tabla 8.17: Comparativa R-3IT-2 y R-3IT-2-SW

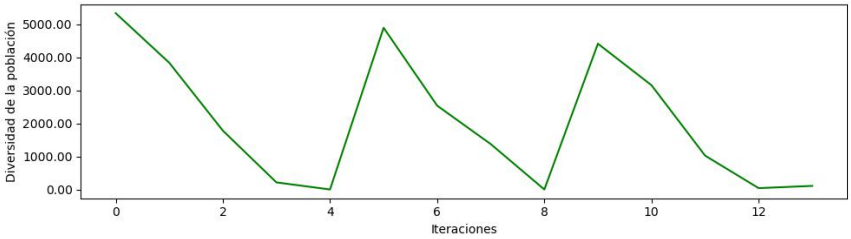
de los casos a las logradas por la propuesta PBEST1-AGR-RAND-SW.

Uno de los motivos principales por los que se consideró la estrategia de reinicio es mantener la diversidad de la población y, con ello, evitar una convergencia prematura hacia óptimos locales. La pérdida de diversidad puede provocar que las soluciones generadas en las últimas iteraciones sean muy similares entre sí, lo que limita la exploración del espacio de búsqueda y reduce la probabilidad de encontrar mejores soluciones. En la Figura 8.2 podemos ver una comparativa de la evolución de la diversidad de la población a lo largo de las iteraciones en la versión original del algoritmo y en la variante con dos reinicios R-3IT-2. Por otro lado, en la Figura 8.3 podemos ver la evolución del *Score* de la mejor solución, el *Score* medio y el peor *Score* de la población a lo largo de las iteraciones para ambos algoritmos.

Se aprecia como en el caso de la versión original del algoritmo, la diversidad decrece rápidamente en unas pocas iteraciones, mientras que al introducir los reinicios, la diversidad aumenta de nuevo hasta un valor cercano al inicial, ya que todas las soluciones de la población, excepto la mejor, se vuelven a generar de forma aleatoria. Además, si observamos la Figura 8.3, podemos ver que, tras el segundo reinicio, se produce una mejora en el *Score* de la mejor solución. Esto muestra la utilidad de los reinicios para escapar

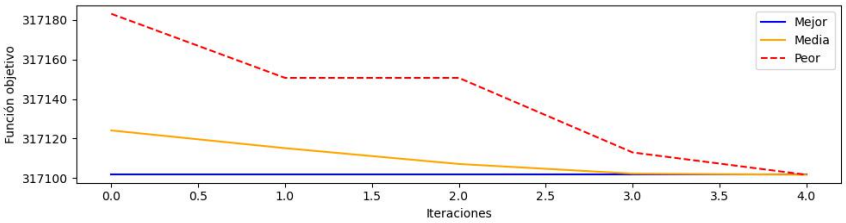


(a) Algoritmo S-MDEClust

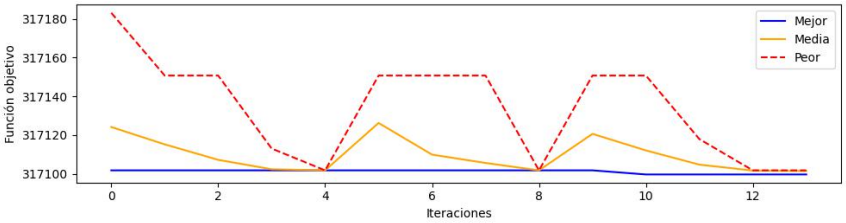


(b) Algoritmo R-3IT-2

Figura 8.2: Comparativa evolución de la diversidad con y sin reinicios



(a) Algoritmo S-MDEClust



(b) Algoritmo R-3IT-2

Figura 8.3: Comparativa evolución del *Score* con y sin reinicios

de óptimos locales y encontrar soluciones de mayor calidad, especialmente cuando el algoritmo se había estancado previamente sin registrar mejoras durante varias iteraciones.

8.7.1. Disminución del tamaño de la población

A continuación, en la Tabla 8.18, se muestran los resultados de las variantes R-3IT-2-DIS y R-3IT-2-DIS-SW, análogas a las dos últimas propuestas analizadas, pero disminuyendo el tamaño de la población después de cada reinicio. Con ello se pretende favorecer una mayor diversidad en las primeras etapas del algoritmo y acelerar la convergencia en las etapas finales, en las que la población es más reducida, centrando el esfuerzo computacional en refinar las mejores soluciones ya encontradas.

En concreto, el tamaño de la población antes del primer reinicio se establece como el doble del valor habitual. Dado que en todos los experimentos se trabaja con una población de 20 individuos, esto implica que el tamaño inicial será de 40, tras el primer reinicio se reduce a 20 y por último, tras el segundo reinicio será de 10.

	R-3IT-2-DIS		R-3IT-2-DIS-SW	
Dataset	Score	Time (s)	Score	Time (s)
Iris	+0.000	+4.7	+0.000	+5.1
Wine	+0.000	+5.6	+0.000	+6.0
Connectionist	+0.000	+6.9	+0.000	+7.2
Seeds	+0.000	+6.7	+0.000	+7.1
Heart	-0.004	+11.5	-0.004	+12.4
Vertebral	+0.000	+4.1	+0.000	+4.3
Computers	-0.203	+24.0	-0.154	+26.9
Gene	+0.000	+461.8	+0.000	+472.4
Movement Libras	-0.191	+126.9	-0.156	+127.9
Toxicity	-4.983e+13	+3.1	-4.983e+13	+3.3
ECG5000	-0.088	+60.3	-0.038	+63.4
Ecoli	-0.004	+58.2	-0.004	+63.1
Glass	-0.043	+23.7	-0.041	+25.5
Accent	+0.023	+49.6	-0.035	+50.4
Promedio	-3.559e+12	+60.5	-3.559e+12	+62.5
Veces mejor	7	0	8	0
Veces igual	6	0	6	0
Veces peor	1	14	0	14

Tabla 8.18: Comparativa R-3IT-2-DIS y R-3IT-2-DIS-SW

Podemos ver que en la variante sin el algoritmo de Solis Wets (R-3IT-

2-DIS), el tiempo promedio de ejecución es ligeramente menor. Además, aunque hay un conjunto de datos en el que el *Score* empeora, la mejora en el resto de los conjuntos de datos es, en general, mayor que la conseguida con R-3IT-2-DIS-SW. No obstante, el tiempo promedio ha aumentado considerablemente, en casi 20 segundos, con respecto a las variantes en las que no se disminuía el tamaño de la población. Esto se debe a que, al ser el tamaño de la población más grande al inicio, se requiere de un mayor número de evaluaciones de la función objetivo en cada iteración.

8.7.2. Combinando diferentes propuestas

En la Tabla 8.19 se presentan los resultados de utilizar el cruce pbest1 al mismo tiempo que la estrategia de reinicio. De nuevo, tenemos dos variantes: una haciendo uso del algoritmo Solis Wets y otra sin él.

	R-3IT-2-PBEST1-F		R-3IT-2-PBEST1-F-SW	
Dataset	Score	Time (s)	Score	Time (s)
Iris	+0.000	+4.0	+0.000	+4.0
Wine	+0.000	+4.9	+0.000	+4.7
Connectionist	+0.000	+5.3	+0.000	+5.6
Seeds	+0.000	+5.5	+0.000	+5.7
Heart	-0.004	+9.1	-0.004	+8.8
Vertebral	+0.000	+3.6	+0.000	+3.4
Computers	-0.154	+19.7	-0.154	+20.3
Gene	+0.000	+350.6	+0.000	+370.9
Movement Libras	-0.354	+56.8	-0.153	+53.8
Toxicity	-4.983e+13	+2.3	-4.983e+13	+2.3
ECG5000	-0.090	+41.3	-0.090	+44.2
Ecoli	-0.003	+39.9	-0.004	+34.9
Glass	-0.028	+15.2	-0.028	+15.6
Accent	-0.070	+35.3	-0.063	+30.7
Promedio	-3.559e+12	+42.4	-3.559e+12	+43.2
Veces mejor	8	0	8	0
Veces igual	6	0	6	0
Veces peor	0	14	0	14

Tabla 8.19: Comparativa R-3IT-2-PBEST1-F y R-3IT-2-PBEST1-F-SW

En este caso, ambas variantes presentan un rendimiento igualado. Sin embargo, la versión sin la búsqueda Solis Wets parece presentar un desempeño ligeramente superior, tanto en *Score* como en tiempo. Como podemos ver, hay dos conjuntos de datos para los que R-3IT-2-PBEST1-F encuentra mejores resultados que R-3IT-2-PBEST1-F-SW: *Movement Libras* y *Accent*, siendo esta mejora significativamente mayor en el primero. Además, el tiem-

po de ejecución promedio es menor en la versión sin Solis Wets, llegando incluso a superar levemente a la variante R-3IT-2 con el operador de cruce original, presentada previamente en la Tabla 8.17.

Seguidamente, en la Tabla 8.20 se muestran los resultados de añadir a las dos variantes que acabamos de analizar la técnica de disminución de la población.

	R-3IT-2-DIS-PBEST1-F		R-3IT-2-DIS-PBEST1-F-SW	
Dataset	Score	Time (s)	Score	Time (s)
Iris	+0.000	+5.3	+0.000	+5.1
Wine	+0.000	+6.0	+0.000	+6.1
Connectionist	+0.000	+6.8	+0.000	+7.1
Seeds	+0.000	+6.8	+0.000	+7.2
Heart	-0.004	+11.5	-0.004	+12.5
Vertebral	+0.000	+3.9	+0.000	+4.3
Computers	-0.154	+23.5	-0.154	+25.0
Gene	+0.000	+454.7	+0.000	+479.7
Movement Libras	-0.377	+92.3	-0.434	+97.8
Toxicity	-4.983e+13	+3.2	-4.983e+13	+3.2
ECG5000	-0.090	+61.0	-0.073	+60.3
Ecoli	+0.000	+47.6	-0.004	+55.5
Glass	-0.047	+21.4	-0.029	+20.9
Accent	-0.070	+44.1	-0.070	+45.9
Promedio	-3.559e+12	+56.3	-3.559e+12	+59.3
Veces mejor	7	0	8	0
Veces igual	7	0	6	0
Veces peor	0	14	0	14

Tabla 8.20: Comparativa R-3IT-2-DIS-PBEST1-F y R-3IT-2-DIS-PBEST1-F-SW

Podemos observar que el rendimiento de ambas propuestas, de nuevo, es bastante similar. En algunos casos, se obtiene un mejor *Score* con R-3IT-2-DIS-PBEST1-F, mientras que en otros es R-3IT-2-DIS-PBEST1-F-SW la que ofrece mejores resultados. Cabe destacar especialmente la mejora obtenida por esta última en el conjunto de datos *Movement Libras*, siendo la más significativa lograda hasta el momento en dicho *dataset*. En cuanto al tiempo de ejecución, R-3IT-2-DIS-PBEST1-F-SW presenta un promedio ligeramente superior, una tendencia que ya se ha observado en otras variantes que incorporan la búsqueda de Solis Wets. Sin embargo, mejora en cuanto al *Score* en un conjunto de datos más que R-3IT-2-DIS-PBEST1-F.

Por último, analizaremos los resultados obtenidos al incorporar la asignación *greedy* aleatorizada a una de las propuestas que mejor desempeño ha

mostrado: R-3IT-2-PBEST1-F. Los resultados se muestran en la Tabla 8.21

	R-3IT-2-PBEST1-F		R-3IT-2-PBEST1-F-AGR-RAND	
Dataset	Score	Time (s)	Score	Time (s)
Iris	+0.000	+4.0	+0.000	+4.4
Wine	+0.000	+4.9	+0.000	+5.7
Connectionist	+0.000	+5.3	+0.000	+6.4
Seeds	+0.000	+5.5	+0.000	+6.5
Heart	-0.004	+9.1	-0.004	+10.4
Vertebral	+0.000	+3.6	+0.000	+4.0
Computers	-0.154	+19.7	-0.154	+24.1
Gene	+0.000	+350.6	+0.000	+348.9
Movement Libras	-0.354	+56.8	-0.340	+66.1
Toxicity	-4.983e+13	+2.3	-4.983e+13	+2.5
ECG5000	-0.090	+41.3	-0.090	+50.4
Ecoli	-0.003	+39.9	-0.004	+40.6
Glass	-0.028	+15.2	-0.047	+17.6
Accent	-0.070	+35.3	-0.070	+34.7
Promedio	-3.559e+12	+42.4	-3.559e+12	+44.5
Veces mejor	8	-	8	-
Veces igual	6	-	6	-
Veces peor	0	-	0	-

Tabla 8.21: Comparativa R-3IT-2-PBEST1-F y R-3IT-2-PBEST1-F-AGR-RAND

Aunque en ambas variantes observamos que el *Score* de las soluciones encontradas mejora con respecto a la versión original del algoritmo para todos los conjuntos de datos, esta mejora parece ligeramente mayor en la versión que hace uso de la asignación *greedy* aleatorizada: hay en dos casos en los que la mejora en *Score* es mayor que la conseguida con la variante R-3IT-2-PBEST1-F, aunque hay un caso (*Movement Libras*) en el que la mejora es más pequeña. Además, el tiempo de ejecución promedio es levemente mayor en dicha variante.

8.8. Resumen de los resultados experimentales

En esta sección haremos un breve repaso por las propuestas desarrolladas, valorando si han supuesto una mejora significativa con respecto al algoritmo original.

- **GRASP**. Esta propuesta ha resultado claramente negativa, ya que

las soluciones obtenidas presentan un *Score* muy deficiente en comparación con el algoritmo original-.

- **Variaciones de la asignación *greedy***. Se probaron dos variantes: la asignación *greedy* aleatorizada y la asignación *greedy* aleatorizada con penalización. De estas, la asignación *greedy* aleatorizada parece ser la más prometedora, aunque su impacto por sí sola no queda completamente claro respecto al algoritmo original. No obstante, al combinarla con otras propuestas, se observa una mejora más consistente en los resultados.
- **SHADE**. La incorporación de SHADE ha mejorado el *Score* en varios conjuntos de datos y reducido el tiempo de ejecución promedio. Sin embargo, en algunos casos ha disminuido la calidad de los resultados, lo que sugiere que su efectividad depende del contexto y las características específicas del problema abordado.
- **Cambios en el operador de cruce**. Estas modificaciones han tenido un impacto positivo en el rendimiento. En particular, la variante pbest1 con el parámetro F en el rango (0.5, 0.8) ha mejorado tanto el *Score* como el tiempo de ejecución promedio, convirtiéndose en una de las propuestas más destacadas.
- **Búsqueda local Solis Wets**. Los resultados obtenidos con esta técnica son ambiguos. La versión que utiliza la función objetivo sin penalización parece ser mejor que la que la incluye. Aunque en algunos casos mejora el *Score*, en otros no, y además implica un incremento leve en el tiempo de ejecución, por lo que su aplicación debe evaluarse caso por caso.
- **Selección de los individuos a los que se aplica la búsqueda local**. Esta estrategia ha permitido reducir el tiempo de ejecución promedio por iteración, aunque ha aumentado el número total de iteraciones necesarias para converger, lo que ha provocado una disminución en la calidad final de las soluciones y, en consecuencia, un empeoramiento del *Score* promedio y del tiempo total.
- **Estrategia de reinicio de población**. Incorporar reinicios ha sido útil para mantener la diversidad y evitar la convergencia prematura a óptimos locales, permitiendo mejorar el *Score* de las soluciones obtenidas especialmente cuando se combina con otras técnicas como el operador pbest1 y la disminución del tamaño de la población en cada reinicio. No obstante, presenta y gran inconveniente que es un aumento notable en el tiempo de ejecución promedio.

En la Figura 8.4 se muestra una comparativa global del *Score* de todas las propuestas desarrolladas, detallando para cada variante, el número de

casos en los que el resultado mejoró, se mantuvo igual o empeoró respecto a la versión original del algoritmo. En la Tabla 8.22 se muestran los mismos resultados, pero en formato de tabla.

Podemos observar que, en general, el rendimiento de las propuestas ha ido mejorando progresivamente a medida que se han incorporado nuevas ideas y ajustes. Especialmente en las variantes finales, se aprecia que el número de casos en los que el *Score* empeora respecto a la versión original se reduce a cero, lo que indica una mayor robustez y efectividad de las últimas propuestas evaluadas.

ID	**Algoritmo**	**Veces Mejor**	**Veces Igual**	**Veces Peor**
1	GRASP	1	0	13
2	AGR-RAND	4	8	2
3	AGR-RAND-P	5	6	3
4	SHADE	3	6	5
5	PBEST1-F	4	8	2
6	PBEST2-F	6	7	1
7	PBEST1-F/2	5	7	2
8	PBEST2-F/2	5	7	2
9	SW-V1-WO-PEN	4	7	3
10	SW-WO-PEN	4	6	4
11	SW-W-PEN	2	6	6
12	SW-WO-PEN-AGR-RAND	4	7	3
13	PBEST1-F-SW-WO-PEN	5	7	2
14	PBEST1-F-AGR-RAND	4	7	3
15	PBEST1-F-AGR-RAND-SW	7	7	0
16	SEL-BL	1	7	6
17	R-2IT-4	7	6	1
18	R-3IT-2	6	6	2
19	R-3IT-2-SW	8	6	0
20	R-3IT-2-DIS	7	6	1
21	R-3IT-2-DIS-SW	8	6	0
22	R-3IT-2-PBEST1-F	8	6	0
23	R-3IT-2-PBEST1-F-SW	8	6	0
24	R-3IT-2-DIS-PBEST1-F	7	7	0
25	R-3IT-2-DIS-PBEST1-F-SW	8	6	0
26	R-3IT-2-PBEST1-F-AGR-RAND	8	6	0

Tabla 8.22: Resultados comparativos del *Score* de las propuestas

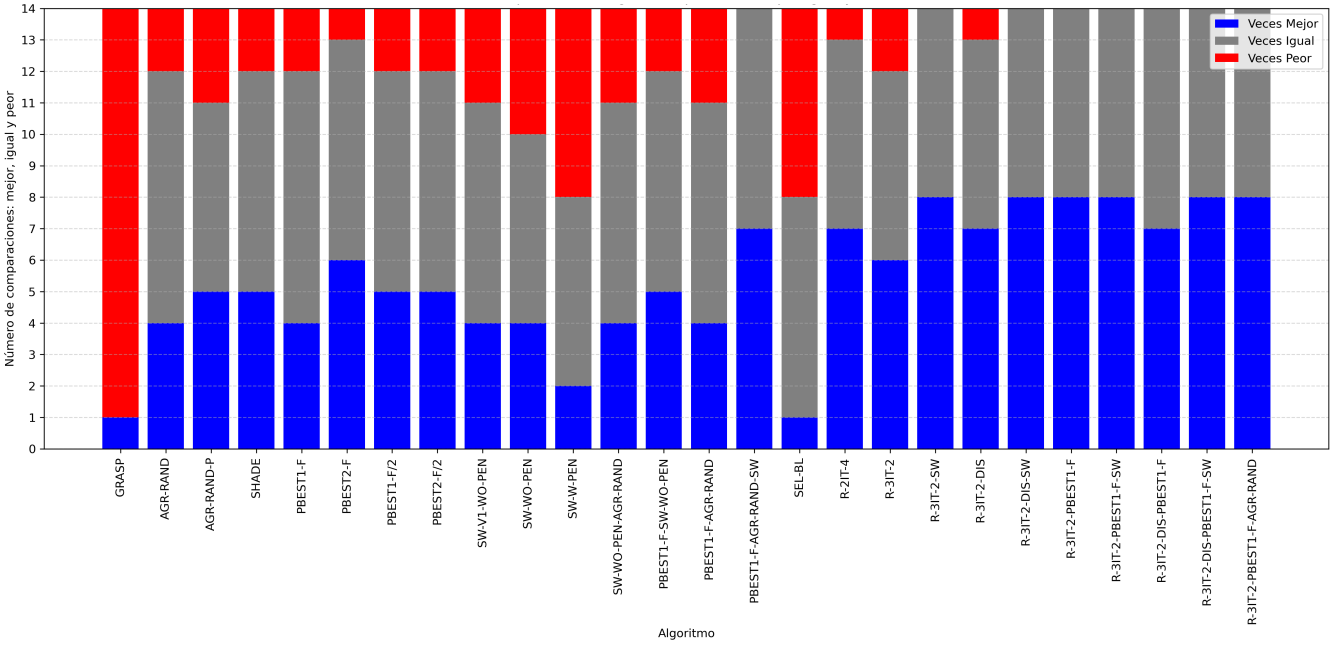


Figura 8.4: Comparativa del *Score* de todas las propuestas: veces que ha sido mejor, igual o peor que el algoritmo S-MDEClust original.

Capítulo 9

Conclusiones y trabajo futuro

En este trabajo, nos hemos enfocado en el estudio del *clustering* semisupervisado con restricciones. Esta técnica combina métodos de *clustering* no supervisado con un conocimiento parcial del problema, que se expresa a través de restricciones entre pares de datos. Este enfoque es especialmente útil en situaciones donde conseguir datos etiquetados resulta complicado o costoso, algo que sucede con frecuencia en muchos campos prácticos.

El objetivo principal de este trabajo ha sido mejorar un algoritmo ya competente de *clustering* semisupervisado: S-MDEClust, mediante la propuesta de múltiples variantes y estrategias de mejora. Podemos decir que hemos alcanzado con creces este objetivo: hemos desarrollado una amplia batería de propuestas, muchas de las cuales han demostrado tener un impacto positivo en la calidad de las soluciones, evaluadas a través de la función objetivo.

El desarrollo de este trabajo ha abarcado todas las etapas de una experimentación completa. Comenzamos con una revisión bibliográfica detallada que nos ayudó a contextualizar el problema y a elegir las técnicas más relevantes. Luego, realizamos un estudio teórico para entender a fondo tanto el problema como los algoritmos de referencia que utilizamos. A ello le han seguido la implementación de propuestas, su evaluación experimental y un análisis exhaustivo de los resultados.

A nivel experimental, se ha comprobado que las propuestas que mejor resultado han dado han sido aquellas basadas en modificaciones sobre el operador de cruce, especialmente el uso del cruce pbest1, así como la introducción de estrategias de reinicio, especialmente cuando estas se combinan con técnicas adicionales como la disminución progresiva del tamaño de la población o el propio cruce pbest1. Además, se ha observado que el uso del

cruce pbest1 no solo mejora la calidad de las soluciones, sino que también reduce el tiempo de ejecución en promedio.

Sin embargo, las propuestas que incorporan estrategias de reinicios, a pesar de ser las que logran las mayores mejoras en términos de la función objetivo, también presentan un incremento considerable en el tiempo de ejecución. Por lo tanto, su uso deberá valorarse según el contexto, teniendo en cuenta los recursos computacionales disponibles y las exigencias de tiempo.

Por último, veremos algunas posibles líneas de extensión del proyecto.

- **Mejora del tiempo de ejecución.** Dado que algunas de las propuestas más efectivas en cuanto a la mejora del *Score*, como la estrategia de reinicio, implican un aumento significativo del tiempo de ejecución, una posible línea de trabajo relevante sería optimizar la implementación mediante técnicas de paralelización. Una alternativa interesante es el modelo de islas, en el que varias subpoblaciones evolucionan de manera paralela y periódicamente intercambian sus mejores soluciones con las subpoblaciones vecinas mediante migración. Este enfoque no solo puede mejorar el rendimiento computacional aprovechando múltiples núcleos o máquinas, sino también favorecer la diversidad genética y evitar la convergencia prematura.

- **Considerar una estrategia de reinicio adaptativo.** En lugar de utilizar un número fijo de reinicios y una reducción predeterminada del tamaño poblacional, se podría investigar una estrategia adaptativa que tome decisiones basadas en el comportamiento dinámico del algoritmo. Por ejemplo, se puede monitorizar la tasa de mejora de la mejor solución o la diversidad de la población, y activar el reinicio solo cuando estas caigan por debajo de ciertos umbrales. También pueden usarse mecanismos probabilísticos, donde la probabilidad de reinicio aumenta con el tiempo sin mejoras, o aplicar reinicios parciales que reemplacen solo a los individuos más similares.

- **Explorar otros enfoques.** El trabajo se ha centrado principalmente en mejorar el algoritmo memético S-MDEClust, proponiendo modificaciones sobre este. Aunque también se ha propuesto un algoritmo GRASP, no se ha profundizado mucho en él ni se han propuesto modificaciones para mejorar a la versión inicial propuesta. Sería interesante dedicar esfuerzos a explorar variantes más sofisticadas de GRASP, además de otros enfoques completamente distintos.

Bibliografía

[1] Kiri Wagstaff and Claire Cardie. Clustering with instance-level constraints. *AAAI/IAAI*, 1097(577-584):197, 2000.

[2] S. Basu, Ian Davidson, and K.L. Wagstaff. *Constrained clustering: Advances in algorithms, theory, and applications*. 01 2008.

[3] Kiri Wagstaff, Claire Cardie, Seth Rogers, Stefan Schrödl, et al. Constrained k-means clustering with background knowledge. In *Icml*, volume 1, pages 577–584, 2001.

[4] Germán González-Almagro, Daniel Peralta, Eli De Poorter, José-Ramón Cano, and Salvador García. Semi-supervised constrained clustering: an in-depth overview, ranked taxonomy and future research directions. *Artificial Intelligence Review*, 58(5):157, March 2025. doi:10.1007/s10462-024-11103-8.

[5] Daniel Aloise, Amit Deshpande, Pierre Hansen, and Preyas Popat. Np-hardness of euclidean sum-of-squares clustering. *Machine Learning*, 75(2):245–248, May 2009. doi:10.1007/s10994-009-5103-0.

[6] Kashif Hussain, Mohd Najib Mohd Salleh, Shi Cheng, and Yuhui Shi. Metaheuristic research: a comprehensive survey. *Artificial Intelligence Review*, 52(4):2191–2233, 2019. doi:10.1007/s10462-017-9605-z.

[7] Pierluigi Mansueto and Fabio Schoen. Memetic differential evolution methods for semi-supervised clustering. *arXiv preprint arXiv:2403.04322*, 2024.

[8] Veronica Piccialli, Anna Russo Russo, and Antonio M. Sudoso. An exact algorithm for semi-supervised minimum sum-of-squares clustering. *Computers Operations Research*, 147:105958, 2022. URL: https://www.sciencedirect.com/science/article/pii/S0305054822002076, doi:10.1016/j.cor.2022.105958.

[9] James MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium*

on Mathematical Statistics and Probability, Volume 1: Statistics, volume 5, pages 281–298. University of California press, 1967.

[10] WEI TAN, YAN YANG, and TIANRUI LI. *AN IMPROVED COP-KMEANS ALGORITHM FOR SOLVING CONSTRAINT VIOLATION*, pages 690–696. URL: https://www.worldscientific.com/doi/abs/10.1142/9789814324700_0104, arXiv:https://www.worldscientific.com/doi/pdf/10.1142/9789814324700_0104, doi:10.1142/9789814324700_0104.

[11] Tonny Rutayisire, Yan Yang, Chao Lin, and Jinyuan Zhang. A modified cop-kmeans algorithm based on sequenced cannot-link set. In *Rough Sets and Knowledge Technology: 6th International Conference, RSKT 2011, Banff, Canada, October 9-12, 2011. Proceedings 6*, pages 217–225. Springer, 2011.

[12] Philipp Baumann. A binary linear programming-based k-means algorithm for clustering with must-link and cannot-link constraints. In *2020 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, pages 324–328, 2020. doi:10.1109/IEEM45057.2020.9309775.

[13] Sugato Basu, Arindam Banerjee, and Raymond J. Mooney. *Active Semi-Supervision for Pairwise Constrained Clustering*, pages 333–344. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611972740.31, arXiv:https://epubs.siam.org/doi/pdf/10.1137/1.9781611972740.31, doi:10.1137/1.9781611972740.31.

[14] Yu Xia. A global optimization method for semi-supervised clustering. *Data mining and knowledge discovery*, 18:214–256, 2009.

[15] Jiming Peng and Yu Xia. *A Cutting Algorithm for the Minimum Sum-of-Squared Error Clustering*, pages 150–160. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611972757.14, arXiv:https://epubs.siam.org/doi/pdf/10.1137/1.9781611972757.14, doi:10.1137/1.9781611972757.14.

[16] Daniel Aloise and Pierre Hansen. A branch-and-cut sdp-based algorithm for minimum sum-of-squares clustering. *Pesquisa Operacional*, 29:503–516, 2009.

[17] Behrouz Babaki, Tias Guns, and Siegfried Nijssen. Constrained clustering using column generation. In Helmut Simonis, editor, *Integration of AI and OR Techniques in Constraint Programming*, pages 438–454, Cham, 2014. Springer International Publishing.

[18] Tias Guns, Thi-Bich-Hanh Dao, Christel Vrain, and Khanh-Chuong Duong. Repetitive branch-and-bound using constraint programming for constrained minimum sum-of-squares clustering. In *ECAI 2016*, pages 462–470. IOS Press, 2016.

[19] Viet-Vu Vu, Nicolas Labroche, and Bernadette Bouchon-Meunier. Leader ant clustering with constraints. In *2009 IEEE-RIVF International Conference on Computing and Communication Technologies*, pages 1–8, 2009. doi:10.1109/RIVF.2009.5174648.

[20] Xiaohua Xu, Lin Lu, Ping He, Zhoujin Pan, and Ling Chen. Improving constrained clustering via swarm intelligence. *Neurocomputing*, 116:317–325, 2013. Advanced Theory and Methodology in Intelligent Computing. URL: https://www.sciencedirect.com/science/article/pii/S0925231212007278, doi:10.1016/j.neucom.2012.03.031.

[21] Daniel Gribel, Michel Gendreau, and Thibaut Vidal. Semi-supervised clustering with inaccurate pairwise annotations. *Information Sciences*, 607:441–457, 2022. URL: https://www.sciencedirect.com/science/article/pii/S0020025522004558, doi:10.1016/j.ins.2022.05.035.

[22] Daniel Gribel and Thibaut Vidal. Hg-means: A scalable hybrid genetic algorithm for minimum sum-of-squares clustering. *Pattern Recognition*, 88:569–583, 2019. URL: https://www.sciencedirect.com/science/article/pii/S0031320318304436, doi:10.1016/j.patcog.2018.12.022.

[23] Germán González-Almagro, Julián Luengo, José-Ramón Cano, and Salvador García. Dils: Constrained clustering through dual iterative local search. *Computers Operations Research*, 121:104979, 2020. URL: https://www.sciencedirect.com/science/article/pii/S0305054820300964, doi:10.1016/j.cor.2020.104979.

[24] Ian Davidson and S. S. Ravi. Intractability and clustering with constraints. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, page 201–208, New York, NY, USA, 2007. Association for Computing Machinery. doi:10.1145/1273496.1273522.

[25] Ian Davidson and S. S. Ravi. *Clustering With Constraints: Feasibility Issues and the ¡italic¿k¡/italic¿-Means Algorithm*, pages 138–149. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611972757.13, arXiv:https://epubs.siam.org/doi/pdf/10.1137/1.9781611972757.13, doi:10.1137/1.9781611972757.13.

[26] Pierluigi Mansueto and Fabio Schoen. Memetic differential evolution methods for clustering problems. *Pattern Recognition*, 114:107849, 2021. URL: https://www.sciencedirect.com/science/article/pii/S0031320321000364, doi:10.1016/j.patcog.2021.107849.

[27] David F. Crouse. On implementing 2d rectangular assignment algorithms. *IEEE Transactions on Aerospace and Electronic Systems*, 52(4):1679–1696, 2016. doi:10.1109/TAES.2016.140952.

[28] Jiming Peng and Yu Wei. Approximating k-means-type clustering via semidefinite programming. *SIAM Journal on Optimization*, 18(1):186–205, 2007. arXiv:https://doi.org/10.1137/050641983, doi:10.1137/050641983.

[29] Veronica Piccialli, Antonio M Sudoso, and Angelika Wiegele. Sos-sdp: an exact solver for minimum sum-of-squares clustering. *INFORMS Journal on Computing*, 34(4):2144–2162, 2022.

[30] Francisco J Solis and Roger J-B Wets. Minimization by random search techniques. *Mathematics of operations research*, 6(1):19–30, 1981.

[31] Ryoji Tanabe and Alex Fukunaga. Success-history based parameter adaptation for differential evolution. In *2013 IEEE Congress on Evolutionary Computation*, pages 71–78, 2013. doi:10.1109/CEC.2013.6557555.

[32] Jingqiao Zhang and Arthur C. Sanderson. Jade: Adaptive differential evolution with optional external archive. *IEEE Transactions on Evolutionary Computation*, 13(5):945–958, 2009. doi:10.1109/TEVC.2009.2014613.

[33] Dheeru Dua and Casey Graff. Uci machine learning repository, 2017. URL: http://archive.ics.uci.edu/ml.

[34] Yanping Chen, Eamonn Keogh, Bing Hu, Nurjahan Begum, Anthony Bagnall, Abdullah Mueen, and Gustavo Batista. The ucr time series classification archive, July 2015. www.cs.ucr.edu/~eamonn/time_series_data/.

[35] Defeng Sun, Kim-Chuan Toh, Yancheng Yuan, and Xin-Yuan Zhao. Sdpnal+: A matlab software for semidefinite programming with bound constraints (version 1.0). *Optimization Methods and Software*, 35(1):87–115, 2020.

[36] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2024. URL: https://www.gurobi.com.

[37] Nizar Grira, Michel Crucianu, and Nozha Boujemaa. Unsupervised and semi-supervised clustering: a brief survey. *A review of machine learning techniques for processing multimedia content*, 1(2004):9–16, 2004.

[38] O. Chapelle, B. Scholkopf, and A. Zien, Eds. Semi-supervised learning (chapelle, o. et al., eds.; 2006) [book reviews]. *IEEE Transactions on Neural Networks*, 20(3):542–542, 2009. doi:10.1109/TNN.2009.2015974.

[39] Jianghui Cai, Jing Hao, Haifeng Yang, Xujun Zhao, and Yuqing Yang. A review on semi-supervised clustering. *Information Sciences*, 632:164–200, 2023. URL: https://www.sciencedirect.com/science/article/pii/S0020025523002840, doi:10.1016/j.ins.2023.02.088.

Apéndice A

Parámetros de ejecución del algoritmo PC-SOS-SDP

Parámetro	**Valor**	**Descripción**
BRANCH_AND_BOUND_TOL	$1 \cdot 10^{-4}$	Tolerancia de optimalidad del algoritmo.
BRANCH_AND_BOUND_PARALLEL	16	Número de hilos para ejecución paralela.
BRANCH_AND_BOUND_MAX_NODES	200	Número máximo de nodos en el árbol de búsqueda.
BRANCH_AND_BOUND_VISITING_STRATEGY	0	Estrategia de recorrido: 0 (best first), 1 (depth first), 2 (breadth first).
SDP_SOLVER_SESSION_THREADS_ROOT	16	Hilos para la sesión MATLAB en la raíz.
SDP_SOLVER_SESSION_THREADS	1	Hilos para nodos ML y CL.
SDP_SOLVER_FOLDER	-	Ruta completa de SDPNAL+.
SDP_SOLVER_TOL	$1 \cdot 10^{-5}$	Exactitud de SDPNAL+.
SDP_SOLVER_VERBOSE	0	Mostrar log: 0 (no), 1 (sí).
SDP_SOLVER_MAX_CP_ITER_ROOT	80	Iteraciones máximas. de plano de corte en la raíz.
SDP_SOLVER_MAX_CP_ITER	40	Iteraciones máximas. del plano de corte en nodos ML y CL.
SDP_SOLVER_CP_TOL	$1 \cdot 10^{-6}$	Tolerancia del plano de corte entre dos iteraciones seguidas.
SDP_SOLVER_MAX_INEQ	100000	Número máximo de desigualdades consideradas.
SDP_SOLVER_INHERIT_PERC	1.0	Fracción de desigualdades heredadas.
SDP_SOLVER_EPS_INEQ	$1 \cdot 10^{-4}$	Tolerancia para detectar las desigualdades incumplidas.
SDP_SOLVER_EPS_ACTIVE	$1 \cdot 10^{-6}$	Tolerancia para desigualdades activas.
SDP_SOLVER_MAX_PAIR_INEQ	100000	Número máximo de desigualdades de pares.
SDP_SOLVER_PAIR_PERC	0.05	Fracción de las desigualdades de pares incumplidas a añadir.
SDP_SOLVER_MAX_TRIANGLE_INEQ	100000	Número máximo de desigualdades triangulares.
SDP_SOLVER_TRIANGLE_PERC	0.05	Fracción de las desigualdades triangulares incumplidas a añadir.

Tabla A.1: Parámetros de configuración del algoritmo exacto PC-SOS-SDP