



Universidad Politécnica
de Madrid

**Escuela Técnica Superior de
Ingenieros Informáticos**



Grado en Ingeniería Informática

Trabajo Fin de Grado

**Partición estática del espacio para
asignación de tareas**

Autor: Andrés Alexander Sánchez Hidalgo

Tutor(a): Juan Antonio Fernández del Pozo de Salamanca

Madrid, Enero 2026

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Grado en Ingeniería Informática

Título: Partición estática del espacio para asignación de tareas
Enero 2026

Autor: Andrés Alexander Sánchez Hidalgo
Tutor: Juan Antonio Fernández del Pozo de Salamanca
Departamento de Inteligencia Artificial
Escuela Técnica Superior de Ingenieros Informáticos
Universidad Politécnica de Madrid

Resumen

Este Trabajo Fin de Grado (TFG) aborda el problema de la partición estática del espacio para la asignación de tareas, tomando como referencia un escenario inspirado en la sectorización del espacio aéreo: un conjunto de entidades distribuidas en un plano, cada una con una carga asociada, que debe repartirse en sectores de forma equilibrada y razonable desde el punto de vista geométrico. La dificultad principal no está solo en “dividir el mapa”, sino en conseguir que la partición resultante mantenga un buen reparto de carga, evite sectores degenerados y genere fronteras estables, incluso cuando la distribución espacial es irregular y aparecen zonas densas junto con estructuras alargadas tipo flujo.

Para estudiar el problema con control y reproducibilidad, se desarrolla un generador de datos sintéticos que combina *hubs* (nubes gaussianas) y *flujos* (corredores lineales con ruido), incorporando además pesos heterogéneos. Sobre estos datos se definen métricas que permiten evaluar soluciones de forma comparable: equilibrio de carga mediante el coeficiente de variación, penalización de capacidad respecto a un rango aceptable, compacticidad normalizada y medidas de robustez ligadas a márgenes o fronteras. Estas métricas se integran en funciones objetivo escalares, lo que permite comparar métodos distintos bajo un mismo marco y analizar los compromisos entre criterios del problema multiobjetivo.

En cuanto a los métodos de partición, se parte de enfoques base como la rejilla ortogonal y k -means, y se exploran variantes que introducen restricciones de capacidad. Sin embargo, el núcleo del trabajo se apoya en técnicas de optimización basadas en EDA (en particular, UMDAc), empleadas para ajustar parámetros y buscar configuraciones que mejoren simultáneamente el equilibrio y la calidad geométrica. Además, se incluye un análisis tipo Pareto para visualizar la relación entre criterios como equilibrio y compacticidad, y así entender mejor qué soluciones son dominantes y cuáles representan compromisos razonables.

Como resultado, se obtiene un procedimiento reproducible que genera particiones coherentes en distintos tamaños de problema, incluyendo escenarios con 20, 50 y 300 entidades. Los experimentos muestran que los métodos simples sirven como referencia, pero se quedan cortos cuando la carga y la geometría entran en conflicto. En cambio, el uso de EDA permite encontrar soluciones más equilibradas y robustas, justificando su aplicación en un problema donde no existe una única respuesta óptima, sino un conjunto de alternativas que deben analizarse con criterios claros. Finalmente, se deja trazabilidad completa

mediante ficheros CSV con la asignación de cada entidad a su sector y métricas agregadas, junto con figuras generadas automáticamente, de forma que el análisis pueda repetirse y ampliarse fácilmente.

Abstract

This Final Degree Project addresses the problem of static space partitioning for task assignment, using an airspace sectorization-inspired setting: a set of weighted entities distributed on a 2D plane that must be divided into sectors in a balanced and geometrically meaningful way. The difficulty is not only “splitting the map”, but obtaining a partition that keeps the workload evenly distributed, avoids degenerate sectors, and produces stable boundaries, even when the spatial distribution is irregular and dense areas coexist with elongated flow-like structures.

To study the problem under controlled and reproducible conditions, a synthetic data generator is developed that combines hubs (Gaussian clusters) and flows (noisy linear corridors), also introducing heterogeneous weights. On top of this dataset, several metrics are defined to evaluate solutions consistently: workload balance through the coefficient of variation, a capacity penalty with respect to an acceptable range, normalized compactness, and robustness-related measures linked to margins and internal borders. These metrics are then combined into scalar objective functions, which makes it possible to compare different methods within the same evaluation framework and to analyze trade-offs between criteria within the multiobjective problem.

Regarding partitioning methods, the work starts from baseline approaches such as an orthogonal grid and k-means, and explores variants that introduce soft capacity constraints. The core of the project, however, relies on EDA-based optimization techniques (specifically UMDAc), used to tune parameters and search for configurations that improve balance and geometric quality at the same time. In addition, a Pareto-style analysis is included to visualize the relationship between criteria such as balance and compactness, helping to identify non-dominated solutions and understand which results represent reasonable compromises.

As an outcome, the project delivers a reproducible pipeline that generates coherent partitions across different problem sizes, including scenarios with 20, 50, and 300 entities. The experiments show that simple methods are useful as references but fall short when workload and geometry conflict, while EDA helps to find more balanced and robust solutions, which justifies its use in a setting where there is not a single universal optimum but a set of alternatives that must be assessed with clear criteria. Finally, full traceability is provided through CSV outputs containing the final assignment of each entity to a sector and aggregated metrics, together with automatically generated figures, so the

Abstract

analysis can be repeated and extended easily.

Tabla de contenidos

Resumen	i
Abstract	iii
1. Introducción	1
1.1. Contexto y motivación	1
1.2. Objetivos	2
1.3. Metodología de trabajo	3
1.4. Alcance y limitaciones	3
1.5. Estructura de la memoria	3
2. Estado del arte	5
2.1. Partición del espacio: dos familias principales	5
2.2. Particiones rectangulares: rejilla y Mondrian	6
2.3. Particiones por centros: Voronoi y k -means	7
2.4. Clustering con restricciones de capacidad	8
2.5. Optimización basada en EDA	8
2.6. Evaluación multi-criterio y análisis de Pareto	8
2.7. Resumen del capítulo	9
3. Generador de datos sintéticos	11
3.1. Estructura de los datos generados	11
3.2. Métricas y funciones objetivo	13
3.2.1. Formulación escalarizada	14
3.3. Parámetros principales del generador y control de escenarios	15
3.3.1. Tamaño del dataset	15
3.3.2. Hubs: centros y dispersión	15
3.3.3. Flujos: longitud, dirección y ruido perpendicular	15
3.3.4. Pesos: rango y heterogeneidad	16
3.3.5. Reproducibilidad mediante semilla	16
4. Propuesta de solución y algoritmos	17
4.1. Definición del problema	17
4.2. Métodos base	17
4.2.1. Rejilla rectangular	17
4.2.2. k -means	18
4.2.3. <i>Balanced k-means</i>	18

TABLA DE CONTENIDOS

4.3. Métricas de evaluación	18
4.4. Optimización mediante EDA (UMDAc)	18
4.4.1. Validación inicial del entorno EDA	19
4.4.2. Función objetivo agregada	20
4.5. Propuesta final: Mondrian recursivo + EDA sobre hiperparámetros	20
4.5.1. Lógica del Mondrian recursivo	21
4.5.2. Ejemplo simple de funcionamiento	21
4.5.3. Hiperparámetros optimizados por EDA	21
4.5.4. Función objetivo usada para evaluar hiperparámetros	22
4.5.5. Rol del EDA en la propuesta final (implementación)	23
4.6. Salida de resultados: imagen y CSV de solución	23
5. Pruebas y análisis de rendimiento	25
5.1. Condiciones experimentales y trazabilidad de resultados	25
5.2. Línea base: partición en rejilla	26
5.3. Comparación entre k -means y EDA en partición por centros (función objetivo v1)	26
5.3.1. Lectura de las métricas agregadas	29
5.4. Resultados con <i>balanced k-means</i>	29
5.5. Exploración de compromisos mediante análisis de Pareto (función objetivo v2)	31
5.6. Escenarios probados	33
5.6.1. Guía de interpretación de las columnas del CSV	34
5.6.2. Escenario $n = 20$	35
5.6.3. Escenario $n = 50$	37
5.6.4. Escenario $n = 300$	39
5.7. Discusión global	42
6. Conclusiones y trabajo futuro	43
6.1. Evaluación de los objetivos del trabajo	43
6.2. Limitaciones	44
6.3. Trabajo futuro	44
Bibliografía	47
Anexos	51
A. Fragmentos de código relevantes	51
A.1. Métricas usadas en la evaluación	51
A.2. Particiones base: rejilla y k -means	52
A.3. Mondrian recursivo optimizado	53
A.3.1. Cuantil ponderado para el corte	53
A.3.2. División de una celda (sin huecos)	54
A.3.3. Regla de parada y K automático	55
A.3.4. Función objetivo y penalización suave sobre K	56
A.3.5. Optimización de hiperparámetros con UMDAc	58

B. Listas de figuras, tablas y código fuente	61
Lista de figuras	61
Lista de tablas	62
Lista de código fuente	63
C. Informe de originalidad	65

Capítulo 1

Introducción

1.1. Contexto y motivación

La asignación de carga en un espacio geográfico es un problema muy habitual en sistemas donde existen entidades distribuidas (puntos) con una carga asociada (peso): tráfico aéreo, reparto urbano, vigilancia, logística o, en general, cualquier sistema donde se necesite *dividir el territorio* en regiones para repartir trabajo.

Como referencia visual del contexto operativo, la Figura 1.1 muestra una captura de tráfico aéreo en tiempo real. En situaciones de alta densidad, la carga no se reparte de forma uniforme y cambia con rapidez, lo que justifica la necesidad de particiones que se adapten a la distribución espacial y al “peso” asociado a cada entidad. [1]

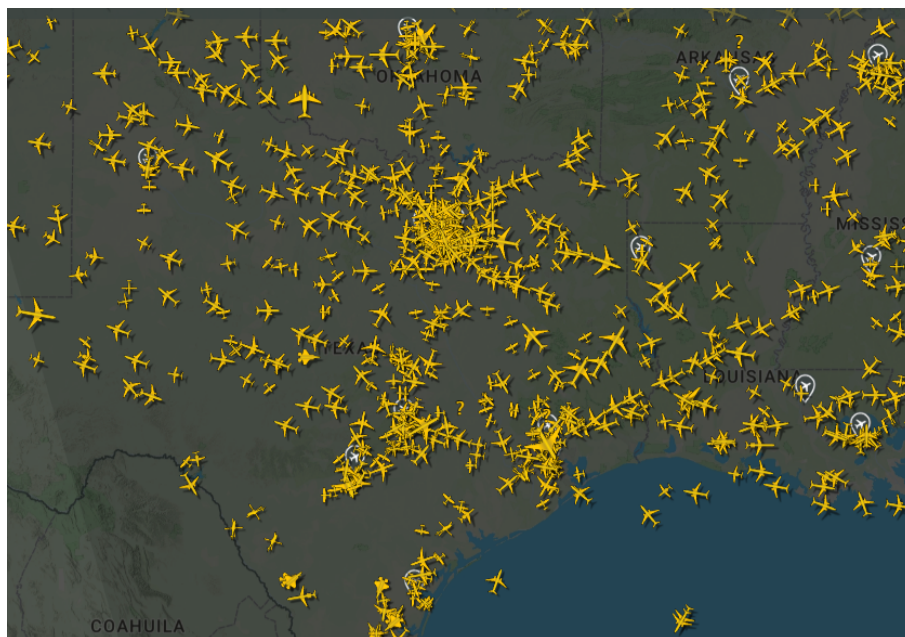


Figura 1.1: Ejemplo de tráfico aéreo en tiempo real.

Capítulo 1. Introducción

En el contexto del control del tráfico aéreo, la idea es especialmente clara: el espacio se divide en sectores y a cada sector se le asigna un conjunto de aeronaves o rutas. Si una zona concentra demasiada carga, el sector se vuelve difícil de gestionar; si se fuerza demasiado la compacidad, se pueden crear sectores muy poco equilibrados; y si se hacen cortes estéticos geoméricamente pero sin tener en cuenta el peso, la solución puede ser poco útil.

En la práctica, estas decisiones suelen implicar compromisos: una partición puede ser muy compacta pero desbalanceada, o muy equilibrada pero con sectores alargados y poco naturales. Además, cuando la distribución espacial no es uniforme (por ejemplo, hubs con mucha concentración y flujos alargados), los métodos simples tienden a fallar de forma evidente. Por eso, antes de hablar de un “mejor” método, conviene dejar claro qué se mide y cómo se compara.

Este Trabajo Fin de Grado aborda ese problema desde una perspectiva práctica: se estudian métodos de partición sencillos (rejilla), métodos basados en centros (k-means y variantes), y finalmente un enfoque más flexible basado en particiones rectangulares tipo *Mondrian*, combinado con optimización basada en EDA (*Estimation of Distribution Algorithms*). El objetivo no es únicamente “dibujar sectores”, sino justificar con métricas y resultados qué aporta cada paso del enfoque.

1.2. Objetivos

El trabajo se organiza alrededor de los siguientes objetivos:

- Definir un problema de partición espacial con pesos y criterios de calidad medibles.
- Generar un conjunto de datos sintético realista, inspirado en patrones típicos de tráfico (hubs y flujos), que permita controlar el tamaño del problema y repetir experimentos con garantías.
- Implementar y comparar enfoques de partición:
 - Rejilla rectangular (línea base geométrica).
 - Partición por centros (k-means) y variante *balanced k-means*.
 - Optimización mediante EDA (UMDAc) para ajustar particiones según una función objetivo.
 - Partición rectangular recursiva tipo *Mondrian*, con reglas de parada y parámetros optimizados.
- Analizar resultados en distintos escenarios de tamaño (20, 50 y 300 entidades) y justificar decisiones con métricas, apoyándose en tablas, figuras y ficheros de salida.
- Mantener trazabilidad y reproducibilidad: que cada figura y tabla incluida pueda asociarse a un script y a un CSV generado.

1.3. Metodología de trabajo

La metodología seguida en el proyecto se apoya en tres ideas:

- **Construcción incremental:** empezar con baselines simples (rejilla, k-means) para tener referencias claras, e ir añadiendo complejidad solo cuando se ve que aporta valor.
- **Evaluación por métricas:** medir equilibrio de carga, penalización de capacidad, compacticidad y criterios geométricos adicionales cuando aplica.
- **Reproducibilidad:** fijar semillas y guardar resultados intermedios (CSV y figuras) para poder repetir comparaciones y documentar el proceso.

Este enfoque permite un análisis más completo.

1.4. Alcance y limitaciones

El trabajo se centra en particiones 2D sobre un espacio continuo y con cortes ortogonales (rectángulos) o asignación a centros (Voronoi aproximado). El dataset es sintético, lo cual permite reproducibilidad y control, pero también implica que no se cubren particularidades de datos operacionales reales (restricciones de espacio aéreo, meteorología, rutas reales, etc.).

En cuanto a la optimización, se trabaja con funciones objetivo agregadas (multi-criterio convertido en escalar) y con análisis de compromisos mediante Pareto. No se persigue “la solución perfecta” universal, sino un método que sea razonable, explicable y que produzca soluciones útiles bajo distintos pesos y escenarios.

También se asume un modelo simplificado de sectorización: no se incorporan restricciones operativas complejas (por ejemplo, conectividad de rutas, límites administrativos o exclusión de áreas). Aun así, el marco propuesto es útil como punto de partida, porque permite introducir esas restricciones más adelante como términos adicionales en la función objetivo o como reglas de validación.

1.5. Estructura de la memoria

La memoria se organiza de la siguiente forma:

- Capítulo 2: revisión del estado del arte, métodos de partición y optimización relacionados.
- Capítulo 3: generación del dataset sintético y formato de los datos.
- Capítulo 4: propuesta de solución y algoritmos (métricas, funciones objetivo y EDA).
- Capítulo 5: resultados experimentales y análisis comparativo, incluyendo escenarios de 20, 50 y 300 entidades.
- Capítulo 6: conclusiones y líneas de trabajo futuro.

Capítulo 2

Estado del arte

Este capítulo resume conceptos y enfoques relacionados con la partición espacial y la asignación de carga. El objetivo no es hacer una revisión exhaustiva, sino dejar claros los bloques que aparecen luego en la propuesta: particiones rectangulares, particiones por centros, restricciones de capacidad y optimización basada en EDA.

2.1. Partición del espacio: dos familias principales

En términos prácticos, la mayoría de enfoques usados en problemas de sectorización pueden agruparse en dos familias:

- **Particiones por cortes (rectangulares u ortogonales):** el espacio se divide mediante cortes verticales/horizontales. Su ventaja principal es que producen regiones fáciles de describir y representar (por ejemplo, rectángulos que encajan sin huecos).
- **Particiones por centros (tipo Voronoi):** se eligen centros y cada punto se asigna al centro más cercano. Esto produce sectores más “orgánicos” y normalmente compactos, pero no garantiza equilibrio de carga ni límites rectangulares.

Como referencia visual e inspiración estética, la Figura 2.1 muestra una obra de Mondrian. No se trata de usar arte como algoritmo, pero sí es una buena metáfora: cortes simples, regiones que encajan y un resultado final que se entiende de un vistazo.

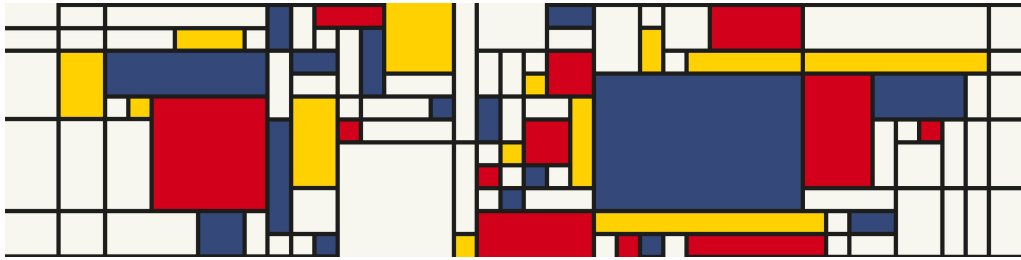


Figura 2.1: Referencia visual: composición de Mondrian como inspiración conceptual para particiones rectangulares.

2.2. Particiones rectangulares: rejilla y Mondrian

La partición más simple es la rejilla (*grid partition*): se fija un número de celdas $n_x \times n_y$ y se asigna cada punto a su celda por coordenadas. La ventaja es su simplicidad y reproducibilidad; el problema es que ignora por completo el peso y la distribución real de los datos. En datasets no uniformes suele dar sectores muy desequilibrados, y ese desequilibrio suele mantenerse incluso aunque se cambie el tamaño de la rejilla.

La Figura 2.2 ilustra este tipo de partición de manera conceptual.



Figura 2.2: Ejemplo conceptual de partición por rejilla: cortes regulares sin tener en cuenta la distribución de carga.

Frente a una rejilla fija, los enfoques tipo **Mondrian** aplican cortes recursivos para refinar zonas donde hay más puntos o más carga. Con reglas de parada adecuadas, el número de sectores puede ajustarse automáticamente. Esto es interesante porque permite crear más detalle donde se necesita sin obligar a un tamaño fijo para todo el dominio.

Una ventaja práctica de las particiones rectangulares es que generan límites muy fáciles de almacenar y explicar: basta con guardar intervalos $[x_{\min}, x_{\max}]$ y $[y_{\min}, y_{\max}]$ por celda. Esto es útil cuando, además del resultado, interesa tener una representación “limpia” para visualizar, depurar o justificar ante terceros.

2.3. Particiones por centros: Voronoi y k -means

En el otro extremo están las particiones por centros. Si se fijan K centros, cada punto se asigna al centro más cercano, lo que equivale a una aproximación de un diagrama de Voronoi.

La Figura 2.3 muestra esta idea: regiones determinadas por distancia a centros.

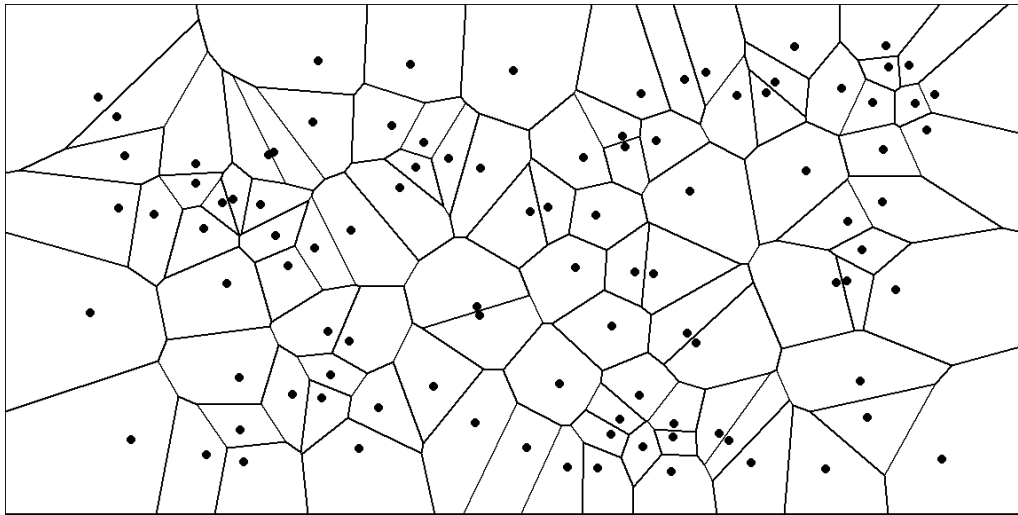


Figura 2.3: Ejemplo conceptual de partición tipo Voronoi: asignación al centro más cercano.

El algoritmo más conocido para ajustar esos centros es k -means. Su fortaleza principal es que tiende a generar clusters compactos. Su debilidad, en problemas con carga, es clara: no impone equilibrio en los pesos por sector. Si hay hubs densos o flujos concentrados, puede aparecer un sector con demasiada carga aunque la partición sea geométricamente simétrica.

Además, k -means asume implícitamente que las regiones resultantes se separan bien por distancia euclídea a centros. En problemas donde interesa imponer fronteras ortogonales, o donde hay restricciones de capacidad fuertes, es habitual que se necesiten ajustes extra (penalizaciones, reasignaciones, variantes balanceadas, etc.).

Dentro del estado del arte, los diagramas de Voronoi y sus variantes se usan a menudo para dividir el plano en regiones a partir de centros, y son una referencia clásica en partición geométrica [2], [3]. En particiones por centros, el caso más conocido es k -means, cuya formulación y convergencia se apoya en resultados clásicos de cuantización y en mosaicos Voronoi centroidales [4], [5].

Para problemas donde interesa controlar el peor caso (máxima distancia dentro del sector), aparecen formulaciones tipo *k-center* [6]. Finalmente, cuando se modela el problema como un grafo, existen heurísticas y enfoques bien conocidos para partición y corte, desde procedimientos clásicos hasta aproximaciones espectrales [7], [8], [9].

2.4. Clustering con restricciones de capacidad

Para tratar el problema del desequilibrio existen variantes que incorporan restricciones, ya sea como penalización o como regla explícita. En este trabajo se utiliza una variante sencilla tipo *balanced k-means*, donde se intenta limitar la carga por sector (capacidad) durante la asignación de puntos. Esto suele mejorar el balance sin destruir del todo la compactidad.

Aun así, estas variantes suelen ser **heurísticas**: pueden mejorar mucho en algunos casos, pero no garantizan un óptimo y pueden depender bastante de la inicialización o del orden de asignación. Por eso, cuando se quieren optimizar varios criterios a la vez (equilibrio, compactidad, margen a fronteras, etc.), suele interesar un método de optimización más general.

2.5. Optimización basada en EDA

Los *Estimation of Distribution Algorithms* (EDA) son una familia de métodos de optimización estocásticos que sustituyen operadores clásicos (cruce/mutación) por un enfoque probabilístico: se muestrean soluciones, se seleccionan las mejores, se ajusta una distribución a esas élites y se vuelve a muestrear. El proceso repite hasta convergencia.

En este TFG se utiliza UMDAc, que modela cada variable de forma independiente mediante distribuciones continuas (univariantes). Su ventaja práctica es que permite optimizar parámetros reales (como centros o hiperparámetros de reglas de corte) sin diseñar operadores específicos para el problema. A nivel conceptual, el método encaja bien cuando el espacio de búsqueda es continuo y las soluciones se evalúan con una función objetivo construida a partir de métricas.

2.6. Evaluación multi-criterio y análisis de Pareto

En problemas reales no suele haber un único criterio. Por ejemplo:

- equilibrio de carga (CV bajo),
- compactidad (distancias intra-sector bajas),
- restricciones de capacidad,
- regularidad geométrica,
- márgenes respecto a fronteras.

Una forma práctica de manejarlo es definir una función objetivo agregada (suma ponderada). Sin embargo, incluso con una función agregada, suele existir más de una solución razonable dependiendo de qué se priorice. Por eso, además de mirar “la mejor” solución, es útil estudiar la nube de soluciones y su frente de Pareto, que recoge soluciones no dominadas cuando se minimizan dos criterios (por ejemplo, CV y compacticidad). Esta idea se utiliza después en los experimentos para justificar selecciones y entender el comportamiento del modelo.

2.7. Resumen del capítulo

En resumen, las particiones por rejilla sirven como referencia simple, las particiones por centros suelen dar buenos resultados en compacticidad, y las particiones rectangulares recursivas aportan una estructura clara y controlable. Cuando se quiere equilibrar varios criterios a la vez, la optimización basada en EDA permite explorar soluciones de forma flexible, compararlas con métricas y justificar mejor por qué una configuración final tiene sentido.

Capítulo 3

Generador de datos sintéticos

En este capítulo se describe el generador de datos sintéticos utilizado a lo largo del trabajo. Su objetivo es proporcionar conjuntos de datos controlados, reproducibles y suficientemente expresivos para evaluar los distintos métodos de partición espacial que se presentan en capítulos posteriores.

El generador se ha diseñado para emular configuraciones típicas en problemas de sectorización espacial: por un lado, zonas con alta densidad de entidades (que representan áreas de concentración de carga) y, por otro, estructuras alargadas que simulan flujos o corredores preferentes. Esta combinación permite comprobar si los algoritmos son capaces de adaptarse a irregularidades geométricas y a pesos heterogéneos sin “romper” el equilibrio global.

3.1. Estructura de los datos generados

El conjunto de datos generado está formado por un conjunto de entidades

$$\mathcal{E} = \{e_i\}_{i=1}^N,$$

donde cada entidad e_i se define por su posición espacial $(x_i, y_i) \in \mathbb{R}^2$, un peso $w_i > 0$ que representa su importancia relativa (o carga), y una etiqueta que identifica el origen del punto dentro del generador (por ejemplo, `hub_0` o `flow_2`).

En los experimentos de este trabajo se parte de un dataset base de tamaño aproximado $N \approx 600$, a partir del cual se extraen subconjuntos para construir escenarios de distinto tamaño. En concreto, en capítulos posteriores se emplean escenarios con $n \in \{20, 50, 300\}$ entidades para estudiar comportamiento y escalabilidad.

El espacio se considera bidimensional y se normaliza para que las coordenadas queden aproximadamente en el rango $[-1, 1] \times [-1, 1]$. Esta decisión simplifica comparaciones entre ejecuciones y evita que las métricas dependan de la escala absoluta del problema.

El generador combina dos tipos de estructuras:

Capítulo 3. Generador de datos sintéticos

- **Hubs:** nubes de puntos distribuidas alrededor de centros bien definidos. Representan zonas de concentración, típicas de nodos relevantes o áreas con actividad elevada.
- **Flujos:** puntos distribuidos a lo largo de segmentos rectilíneos con ruido perpendicular. Simulan corredores o trayectorias preferentes que atraviesan el espacio, introduciendo direccionalidad.

Además, cada entidad recibe un peso entero dentro de un rango acotado, lo que permite modelar escenarios en los que no todas las entidades contribuyen igual a la carga de un sector. Esta heterogeneidad es clave: obliga a los algoritmos a gestionar compromisos reales entre equilibrio (reparto de carga), compacidad (cohesión espacial) y forma (geometría del particionado).

La Figura 3.1 muestra un ejemplo representativo del tipo de datos generados, donde se distinguen claramente las nubes correspondientes a los hubs y las estructuras alargadas asociadas a los flujos.

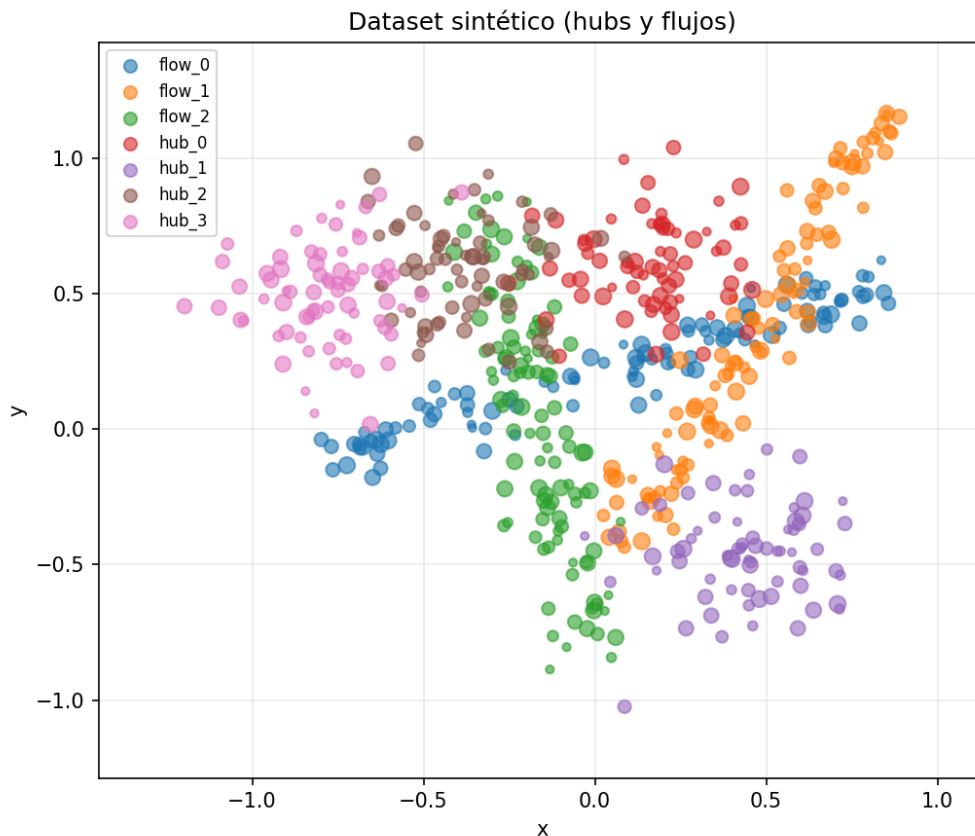


Figura 3.1: Ejemplo de dataset sintético generado, combinando hubs (4 nubes de puntos) y flujos (3 estructuras alargadas).

Para concretar el formato y el tipo de valores utilizados, la Tabla 3.1 muestra un ejemplo de las diez primeras filas del CSV generado. Se incluyen el identificador

3.2. Métricas y funciones objetivo

id, las coordenadas (x, y) , el peso w y la etiqueta.

id	x	y	w	etiqueta
0	0.1116	0.5239	5	hub_0
1	0.2883	0.6998	9	hub_0
2	0.2191	0.4681	7	hub_0
3	0.1949	0.7607	3	hub_0
4	-0.0418	0.5532	8	hub_0
5	-0.1421	0.4034	8	hub_0
6	-0.1314	0.5932	7	hub_0
7	-0.0280	0.6844	7	hub_0
8	0.2284	0.6019	3	hub_0
9	-0.2529	0.5386	7	hub_0

Tabla 3.1: Ejemplo de las primeras filas del dataset sintético generado.

3.2. Métricas y funciones objetivo

El problema de partición espacial tratado en este trabajo es multi-criterio: no existe una única métrica que capture de forma completa la calidad de una partición. Por este motivo se definen varias métricas complementarias, que se utilizan tanto para analizar resultados como para construir funciones objetivo escalariizadas dentro de los algoritmos de optimización.

Sea S el número de sectores de una partición, y sea $\mathcal{E}_s \subset \mathcal{E}$ el conjunto de entidades asignadas al sector s .

Peso total por sector. El peso total asociado a un sector se define como

$$W_s = \sum_{e_i \in \mathcal{E}_s} w_i.$$

Esta magnitud representa la carga efectiva del sector y es el punto de partida para evaluar equilibrio y capacidad.

Equilibrio (coeficiente de variación). Para medir la uniformidad en la distribución de carga entre sectores se utiliza el coeficiente de variación:

$$CV = \frac{\sigma_W}{\mu_W}, \quad \mu_W = \frac{1}{S} \sum_{s=1}^S W_s,$$

donde σ_W es la desviación típica de los pesos sectoriales. Valores bajos de CV indican una partición más equilibrada. Esta métrica es especialmente útil porque es adimensional y permite comparar resultados entre escenarios con distinto tamaño total.

Capítulo 3. Generador de datos sintéticos

Penalización de capacidad. Además de equilibrar, en muchos casos interesa que cada sector esté dentro de un rango de capacidad aceptable. Se define un intervalo relativo a la media:

$$\underline{W} = \alpha \mu_W, \quad \overline{W} = \beta \mu_W,$$

con $\alpha < 1 < \beta$.

La penalización por capacidad se calcula como

$$\text{CapPen} = \sum_{s=1}^S \max(0, \underline{W} - W_s) + \max(0, W_s - \overline{W}),$$

y se utiliza también una versión normalizada dividiéndola por el peso total del sistema, de forma que sea comparable entre datasets de distinto tamaño.

Compacticidad. La compacticidad mide la cohesión espacial de los sectores. En particiones por centros, se define como la distancia media ponderada de las entidades al centro del sector al que están asignadas, y se normaliza por la diagonal del *bounding box* del conjunto de datos:

$$\text{Compact} = \frac{\sum_i w_i \|(x_i, y_i) - c_{l(i)}\|}{\sum_i w_i} / \Delta,$$

donde $c_{l(i)}$ es el centro del sector asignado a la entidad i y Δ es la diagonal del dominio espacial. Valores más bajos indican sectores más compactos.

Margen a frontera. De forma complementaria, se considera el margen mínimo de las entidades respecto a las fronteras de su sector, como medida de robustez geométrica. Valores bajos indican entidades muy próximas a los límites, lo que puede ser indeseable desde un punto de vista operativo (pequeñas variaciones podrían provocar reasignaciones o fronteras poco estables).

3.2.1. Formulación escalarizada

Aunque el problema es multiobjetivo, en la práctica se trabaja con funciones de coste escalarizadas que combinan las métricas anteriores mediante pesos ajustables. De forma general, se utiliza una función del tipo:

$$J = w_{cv} CV + w_{cap} \text{CapPen}_{\text{norm}} + w_{comp} \text{Compact} + w_m (1 - \text{margin}_{\text{mín}}^{\text{norm}}),$$

donde los coeficientes w . permiten explorar distintos compromisos entre equilibrio, capacidad, compacidad y robustez geométrica.

Esta formulación proporciona un marco flexible para evaluar y comparar soluciones. Sobre esta base se construyen los algoritmos de optimización y los experimentos presentados en los capítulos siguientes.

3.3. Parámetros principales del generador y control de escenarios

Aunque el generador produce un dataset “realista” a nivel geométrico, su principal ventaja es que permite controlar de forma explícita el tipo de patrón espacial y su dificultad. Los parámetros más relevantes se pueden agrupar en: tamaño, forma (hubs y flujos) y heterogeneidad de pesos.

3.3.1. Tamaño del dataset

El número total de entidades N viene determinado por el número de hubs y flujos, y por cuántos puntos aporta cada uno:

$$N = n_{\text{hubs}} \cdot n_{\text{hub}} + n_{\text{flows}} \cdot n_{\text{flow}}.$$

Este valor fija el “tamaño base” del dataset, en este caso se ha generado uno de aproximadamente 600 datos, sobre el que se construyen escenarios más pequeños mediante muestreo. En este trabajo, esos escenarios se resumen en $n \in \{20, 50, 300\}$, lo que permite estudiar desde casos fáciles de inspeccionar visualmente hasta casos donde la estabilidad del método es más importante.

3.3.2. Hubs: centros y dispersión

Cada hub se genera alrededor de un centro (μ_x, μ_y) elegido dentro del dominio. A partir de ese centro, se muestrean puntos con una gaussiana bidimensional. De forma simplificada:

$$(x, y) \sim \mathcal{N}((\mu_x, \mu_y), \sigma_{\text{hub}}^2 I),$$

donde σ_{hub} controla la dispersión de la nube. Valores bajos producen hubs muy compactos; valores altos hacen que los hubs se solapen y que la partición sea menos evidente. Este efecto se aprecia en la Figura 3.1, donde las nubes aparecen como agrupaciones claras.

3.3.3. Flujos: longitud, dirección y ruido perpendicular

Los flujos se construyen a partir de un segmento definido por un centro, una dirección y una longitud. Luego se muestrean puntos a lo largo del segmento y se añade un ruido perpendicular para evitar una recta perfecta. Conceptualmente:

$$p(t) = p_0 + t(p_1 - p_0) + \varepsilon n_{\perp}, \quad t \sim U(0, 1),$$

donde $n_{\perp}$ es el vector perpendicular a la dirección del flujo, y ε es un ruido (normal) cuya desviación típica controla el “grosor” del corredor. Cuanto mayor es esa dispersión, menos “lineal” es el flujo y más se parece a una banda ancha de puntos.

3.3.4. Pesos: rango y heterogeneidad

El peso w de cada entidad se genera como un valor discreto dentro de un intervalo acotado. Este detalle es importante: aunque la geometría pueda parecer sencilla, una distribución de pesos heterogénea puede provocar sectores con cargas muy distintas si el algoritmo no lo tiene en cuenta.

En la práctica, la Tabla 3.1 sirve como referencia rápida del formato: además de (x, y) , aparecen pesos con variación (por ejemplo, $w = 3$ frente a $w = 9$) y una etiqueta que permite distinguir el origen del punto.

3.3.5. Reproducibilidad mediante semilla

Para garantizar reproducibilidad, el generador trabaja con una semilla fija. Esto implica que, para un mismo conjunto de parámetros, el CSV resultante es siempre el mismo. Gracias a esto, las figuras y métricas que aparecen en los capítulos experimentales se pueden regenerar de forma determinista a partir de los scripts, manteniendo trazabilidad entre resultados y configuración.

Capítulo 4

Propuesta de solución y algoritmos

Este capítulo describe los algoritmos y métricas utilizados para construir particiones y evaluarlas. La idea principal es convertir el problema en una optimización donde, dadas unas decisiones (centros, cortes o hiperparámetros), se obtiene una partición y se mide su calidad. Las implementaciones clave de las métricas y algoritmos se incluyen en el Anexo A.

4.1. Definición del problema

Dado un conjunto de N entidades con coordenadas (x_i, y_i) y peso w_i , se busca una partición del espacio en K sectores tal que:

- el peso total por sector sea lo más equilibrado posible,
- los sectores sean razonablemente compactos,
- se respeten límites de capacidad (cuando se activan),
- y, en la propuesta final, se eviten sectores degenerados y puntos demasiado pegados a fronteras.

En este trabajo se consideran dos formas de definir la partición:

- **Por centros:** cada punto se asigna al centro más cercano.
- **Por cortes rectangulares:** el espacio se divide en rectángulos mediante cortes verticales y horizontales.

4.2. Métodos base

4.2.1. Rejilla rectangular

La rejilla se usa como línea base porque es simple y muy estable. Dado un $n_x \times n_y$, cada punto cae en una celda según sus coordenadas. Su principal limitación es

que no reacciona a hubs o flujos: si el dataset está muy concentrado en una zona, esa celda se “carga” demasiado.

4.2.2. *k*-means

El *k*-means genera centros que minimizan la varianza en el cluster. En este trabajo se usa como referencia para comparar con métodos que sí incorporan balance de carga. Su punto fuerte es la compacidad; su punto débil es que no garantiza equilibrio.

4.2.3. *Balanced k*-means

Como paso intermedio se utiliza una variante tipo *balanced k*-means donde se limita de forma suave la carga por sector durante la asignación. La idea práctica es sencilla: se intenta asignar primero los puntos con mayor peso sin reventar la capacidad del sector. Esta técnica ayuda en escenarios con pesos muy heterogéneos, aunque no “optimiza todo”, porque no incluye márgenes ni regularización geométrica más allá de lo que induce la asignación. Los fragmentos de implementación de las particiones base (rejilla y *k*-means) se recogen en el Anexo A.2.

4.3. Métricas de evaluación

Para comparar soluciones se utilizan varias métricas:

- **CV (coeficiente de variación)** del peso por sector. Se busca minimizarlo.
- **Penalización de capacidad** (y versión normalizada): mide cuánto se viola un rango deseado de carga.
- **Compactidad** (normalizada): media ponderada de distancias al centroide/centro.
- **Margen mínimo normalizado**: mide si hay puntos demasiado cerca de fronteras (mejor alto).
- **Frontera interna normalizada**: regulariza soluciones con exceso de cortes (mejor bajo).

Estas métricas no sustituyen una evaluación “operacional”, pero permiten comparar soluciones de forma consistente, especialmente cuando el dataset cambia de tamaño.

4.4. Optimización mediante EDA (UMDAc)

Los algoritmos EDA se apoyan en la idea de sustituir operadores clásicos de recombinación por el aprendizaje explícito de una distribución de probabilidad sobre el espacio de búsqueda [10]. En la práctica, esto permite guiar la exploración de forma progresiva a partir de las mejores soluciones vistas, y existe

4.4. Optimización mediante EDA (UMDAc)

bibliografía de referencia que resume familias y variantes habituales [11]. Para la parte experimental, la implementación se ha apoyado en herramientas en Python que facilitan la ejecución de EDA continuos y el registro de resultados [12].

La optimización mediante EDA se plantea como un bucle: se generan candidatos, se evalúan con una función objetivo, se elige una élite y se actualiza la distribución de muestreo.

En el caso de partición por centros, la decisión principal es el vector de centros:

$$\theta = (c_{1x}, c_{1y}, \dots, c_{Kx}, c_{Ky})$$

y cada candidato define una partición asignando cada punto al centro más cercano.

4.4.1. Validación inicial del entorno EDA

Antes de aplicar EDA al problema real, se validó el entorno de optimización con dos pruebas sencillas. La Figura 4.1 muestra una prueba inicial donde se comprueba que el flujo básico del optimizador funciona (evaluación, selección, actualización y convergencia).

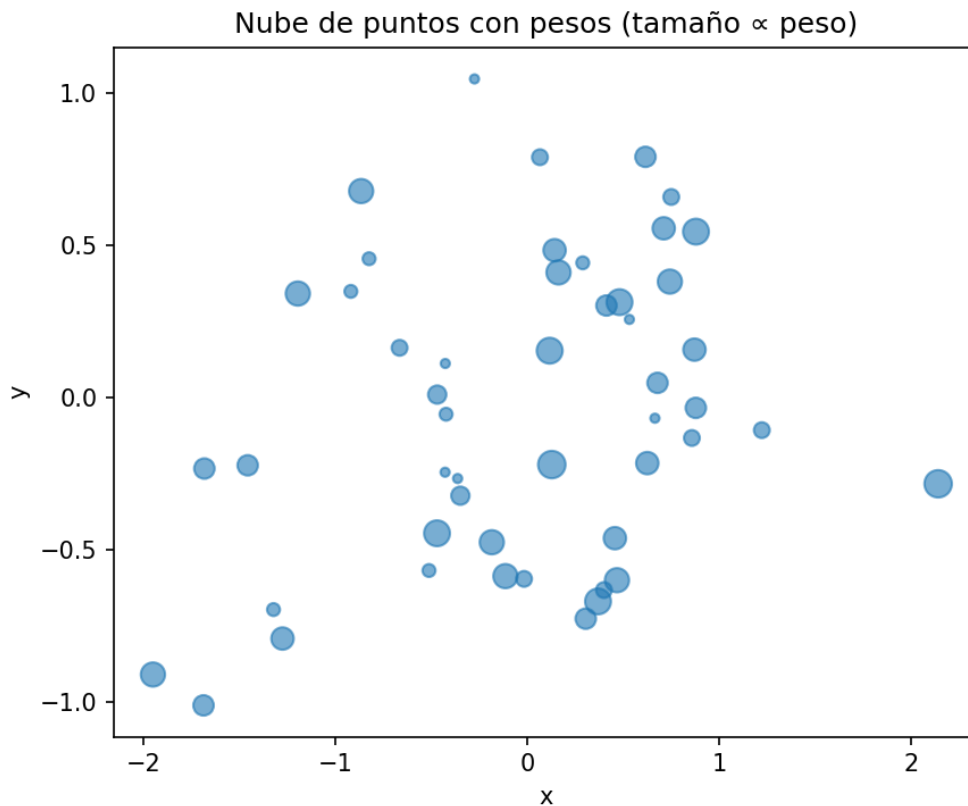


Figura 4.1: Validación inicial: *smoke test* del bucle de optimización.

Capítulo 4. Propuesta de solución y algoritmos

La Figura 4.2 muestra una prueba sobre una función continua de referencia, útil para detectar errores típicos (límites mal definidos, escalas, o problemas numéricos).

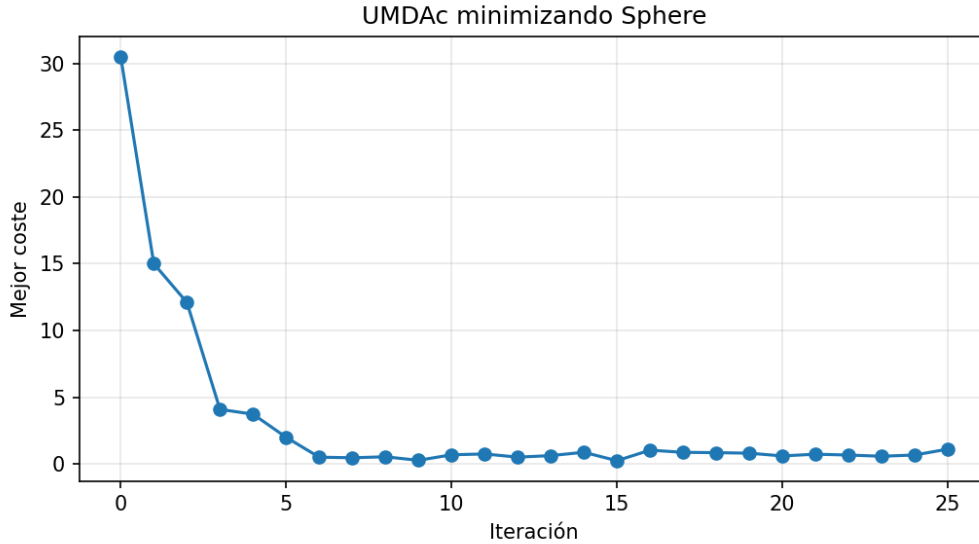


Figura 4.2: Validación inicial: ejecución de UMDAc sobre una función continua de referencia (Sphere).

4.4.2. Función objetivo agregada

Para evaluar cada candidato se utiliza una función objetivo escalar que combina varias métricas. La combinación exacta (pesos de cada término) se ajusta según el experimento. La idea es clara: minimizar desequilibrio y violaciones de capacidad, mantener compacidad razonable y, cuando aplica, favorecer márgenes y regularización.

Además, se usa un análisis de Pareto cuando interesa estudiar compromisos entre dos criterios principales (por ejemplo, CV y compactidad). Esto permite entender qué se gana y qué se pierde al mover pesos en la función objetivo.

4.5. Propuesta final: Mondrian recursivo + EDA sobre hiperparámetros

La propuesta final utiliza una partición rectangular recursiva tipo Mondrian: se parte del rectángulo que contiene a todas las entidades (el *bounding box*) y se va dividiendo en sub-rectángulos mediante cortes verticales u horizontales. La idea es mantener una geometría simple (sectores rectangulares sin huecos) pero adaptativa: las zonas densas o muy cargadas “fuerzan” más divisiones, mientras que las zonas poco pobladas quedan agrupadas. El detalle de implementación de la partición Mondrian optimizada y su configuración se resume en el Anexo A.3.

4.5. Propuesta final: Mondrian recursivo + EDA sobre hiperparámetros

4.5.1. Lógica del Mondrian recursivo

A nivel operativo, el Mondrian recursivo aplicado en este trabajo sigue siempre el mismo esquema:

1. **Celda inicial.** Se toma una celda raíz R que cubre el dominio completo.
2. **Medición local.** Para cada celda C se calculan estadísticas básicas: número de puntos n_C , peso total W_C , y (cuando aplica) métricas auxiliares como margen a fronteras y longitud de frontera interna inducida.
3. **Criterio de parada.** Si la celda ya es “aceptable” (por ejemplo, $n_C \leq \text{max_points}$ y/o su carga no rompe el objetivo de equilibrio), entonces se declara *hoja* y no se divide más.
4. **Selección del corte.** Si la celda es demasiado densa o cargada, se elige un eje (vertical u horizontal) y se propone una posición de corte en función de la distribución de puntos dentro de la celda. En este trabajo, la posición del corte se controla con un cuantil cut_q : un valor cercano a 0,5 tiende a cortes “centrales”, mientras que valores más extremos inducen cortes más sesgados. En particular, el cuantil ponderado usado para decidir el punto de corte se implementa como se muestra en el Anexo A.3.1.
5. **División y recursión.** Se parte C en dos subceldas ($C_{\text{izq}}/C_{\text{der}}$ o $C_{\text{sup}}/C_{\text{inf}}$), se reasignan los puntos según su coordenada y se repite recursivamente en cada subcelda.

4.5.2. Ejemplo simple de funcionamiento

Para fijar ideas, supongamos una celda inicial con $n = 8$ puntos y se fija $\text{max_points} = 4$. En la primera iteración, como $8 > 4$, se fuerza un corte (por ejemplo vertical) usando $\text{cut_q} \approx 0,5$, de manera que la celda se divide en dos rectángulos con ~ 4 puntos cada uno. A partir de ahí:

- si en una de las mitades cae un hub y quedan $n = 6$ puntos, se vuelve a dividir esa mitad;
- si la otra mitad queda con $n = 2$ puntos, se detiene ahí (ya cumple el criterio de parada).

El resultado final son más sectores donde hay densidad/carga y menos donde el espacio está “vacío”, manteniendo siempre sectores rectangulares.

4.5.3. Hiperparámetros optimizados por EDA

A diferencia de la rejilla o del k -means, aquí la partición depende de varios hiperparámetros que controlan cuándo y cómo se corta. En el código del proyecto estos hiperparámetros se agrupan en una estructura `params` (se definen y se consumen dentro del script de la propuesta final), y EDA busca automáticamente una configuración que minimice una función objetivo global. La optimización de hiperparámetros mediante UMDAc se muestra en el Anexo A.3.5.

Capítulo 4. Propuesta de solución y algoritmos

Los hiperparámetros optimizados son:

- **max_points**: máximo de puntos permitido en una celda hoja. Valores bajos generan más sectores; valores altos producen sectores más grandes.
- **cv_target**: objetivo de equilibrio (umbral) sobre la variabilidad de carga entre celdas/sectores; si no se alcanza, se incentivan más divisiones.
- **margin**: margen mínimo (en coordenadas normalizadas) para evitar fronteras degeneradas con puntos “pegados” a los límites.
- **cut_q**: cuantil usado para proponer la posición del corte dentro de una celda (controla si el corte es más central o más sesgado).
- **alpha_weight**: peso relativo del término de regularización geométrica asociado a frontera interna (penaliza particiones con demasiados cortes).

Ejemplo de individuo (candidato) en la población. Un individuo de EDA, en este contexto, es simplemente una tupla de hiperparámetros:

$$\theta_{\text{mondrian}} = (\text{max_points}, \text{cv_target}, \text{margin}, \text{alpha_weight}, \text{cut_q}).$$

Por ejemplo, una configuración típica evaluada en los experimentos finales tiene la forma:

$$(9, 0,38, 0,063, 0,092, 0,50),$$

que se interpreta como “permitir hasta 9 puntos por celda, buscar un equilibrio objetivo cercano a 0,38, exigir un margen $\approx 0,063$, regularizar frontera interna con peso $\approx 0,092$ y cortar con un cuantil cercano a la mediana”.

4.5.4. Función objetivo usada para evaluar hiperparámetros

Cada candidato θ_{mondrian} induce una partición completa: se ejecuta el Mondrian recursivo con esos hiperparámetros, se obtienen las celdas finales y se calculan métricas globales de la partición. La función objetivo J que minimiza EDA es una combinación agregada de los criterios principales del trabajo:

- **Equilibrio**: CV de carga entre celdas/sectores (minimizar).
- **Compacticidad**: versión normalizada (minimizar).
- **Regularización geométrica**: frontera interna normalizada (minimizar).
- **Robustez**: penalización asociada a márgenes pequeños (favorecer márgenes altos).

De esta forma, EDA no “decide” cortes directamente: decide hiperparámetros, y el Mondrian recursivo transforma esos hiperparámetros en una partición. Esto es precisamente lo que hace que el método sea manejable entre tamaños ($n = 20, 50, 300$): el patrón de generación cambia, pero el mecanismo de búsqueda sigue siendo el mismo.

4.5.5. Rol del EDA en la propuesta final (implementación)

El bucle de EDA (muestreo $\rightarrow$ evaluación $\rightarrow$ élite $\rightarrow$ actualización) es el mismo descrito en la Sección 4.4, pero aplicado al espacio de hiperparámetros del Mondrian. En esta parte se reutiliza el enfoque de UMDAc apoyándose en librerías de Python para la mecánica del EDA, mientras que el generador de particiones Mondrian (recursión, criterios de parada, cálculo de métricas y exportación de resultados) es implementación propia del proyecto.

Este diseño separa claramente responsabilidades: EDA explora el espacio de configuraciones, y el Mondrian recursivo actúa como “decodificador” que convierte una configuración en una partición evaluable.

4.6. Salida de resultados: imagen y CSV de solución

Para cada escenario de tamaño ($n = 20$, $n = 50$ y $n = 300$) se generan:

- una imagen `figs/FINAL_mondrian_{n}.png` con la partición,
- un CSV `data/FINAL_mondrian_{n}.csv` con la asignación y métricas por punto/celda.

El CSV final incluye dos bloques de información:

- **Información por punto:** coordenadas, peso y celda asignada.
- **Información de celda y métricas globales:** límites del rectángulo, estadísticas de la celda y métricas agregadas para toda la partición.

En el Capítulo 5 se analizan estas salidas con detalle, comparando los tres tamaños y relacionando métricas con lo que se observa en las figuras.

Capítulo 5

Pruebas y análisis de rendimiento

En este capítulo se presentan los resultados experimentales al aplicar los métodos descritos en el Capítulo 4 sobre el conjunto de datos sintéticos del Capítulo 3. El objetivo no es únicamente enseñar particiones finales, sino justificar con métricas y evidencias visuales qué aporta cada enfoque y en qué situaciones funciona mejor.

Todos los experimentos se han planteado para ser reproducibles: los *scripts* generan automáticamente figuras en `figs/` y resultados numéricos en `data/`. Además, el uso de semillas fijas permite repetir ejecuciones y comparar cambios de forma controlada.

5.1. Condiciones experimentales y trazabilidad de resultados

Los experimentos se han ejecutado en un entorno Python estándar, utilizando bibliotecas científicas habituales (NumPy, Pandas, Matplotlib, scikit-learn) y EDAspy para la parte de optimización EDA.

En este trabajo interesa que cada figura tenga un “rastros” claro hacia los datos que la respaldan. Por eso, cuando se produce una partición:

- la **figura** muestra la geometría o la asignación,
- el **CSV** conserva valores (métricas, asignaciones, resumen) para poder explicar el resultado.

Las métricas utilizadas (CV, penalización de capacidad, compacticidad, margen, frontera interna) son las definidas en el Capítulo 4. En la práctica, cada una mira un aspecto distinto, y por eso conviene interpretarlas juntas. La implementación concreta del cálculo agregado de estas métricas se muestra en el Anexo A.1.

5.2. Línea base: partición en rejilla

Como primer punto de comparación se evalúa la partición en rejilla ortogonal (3×3). Esta partición es útil como referencia porque no depende de ninguna optimización: divide el espacio por coordenadas y asigna cada punto a su celda.

La Tabla 5.1 resume el peso total W_s y el número de entidades N_s por sector. La lectura es directa: hay sectores que concentran una parte muy grande del peso total, mientras otros quedan muy ligeros. Esto confirma el principal problema de la rejilla: funciona si la distribución es uniforme, pero falla cuando aparecen hubs o corredores.

Sector	W_s	N_s
1	186.0	38
2	170.0	38
3	297.0	54
4	937.0	185
5	525.0	100
6	188.0	36
7	503.0	103
8	280.0	56
9	0.0	0

Tabla 5.1: Peso total y número de entidades por sector en la partición por rejilla.

En términos prácticos, este resultado ya justifica que hace falta un método que “se adapte” a la densidad real del dataset. La rejilla no puede decidir que una zona necesite más detalle, porque el particionado está fijado desde el principio. Además, al evaluarla con las métricas del Capítulo 4, se obtiene un CV alto ($\approx 0,762$) y una penalización de capacidad muy elevada ($\approx 803,8$), lo que confirma el desequilibrio observado.

5.3. Comparación entre k -means y EDA en partición por centros (función objetivo v1)

A continuación se comparan dos métodos con el mismo tipo de partición (por centros) y el mismo número de sectores:

- **k -means:** referencia clásica.
- **EDA (UMDAc):** optimización sobre los centros para minimizar una función objetivo con balance y capacidad.

La comparación se resume con métricas agregadas. En esta versión (v1), la función objetivo prioriza el equilibrio y la capacidad, permitiendo que la compactidad empeore ligeramente si eso elimina violaciones de carga. El resultado es coherente con lo que se espera: EDA tiende a mover centros para repartir mejor los pesos, aunque eso implique sectores algo menos compactos.

5.3. Comparación entre *k*-means y EDA en partición por centros (función objetivo v1)

La Figura 5.1 muestra una partición típica de *k*-means comparada con la mejor partición encontrada con EDA bajo la misma configuración. Aunque ambas se ven “razonables”, la diferencia aparece en los valores cuantitativos: *k*-means presenta violaciones de capacidad importantes, mientras que EDA encuentra una solución sin penalización de capacidad y con menor CV.

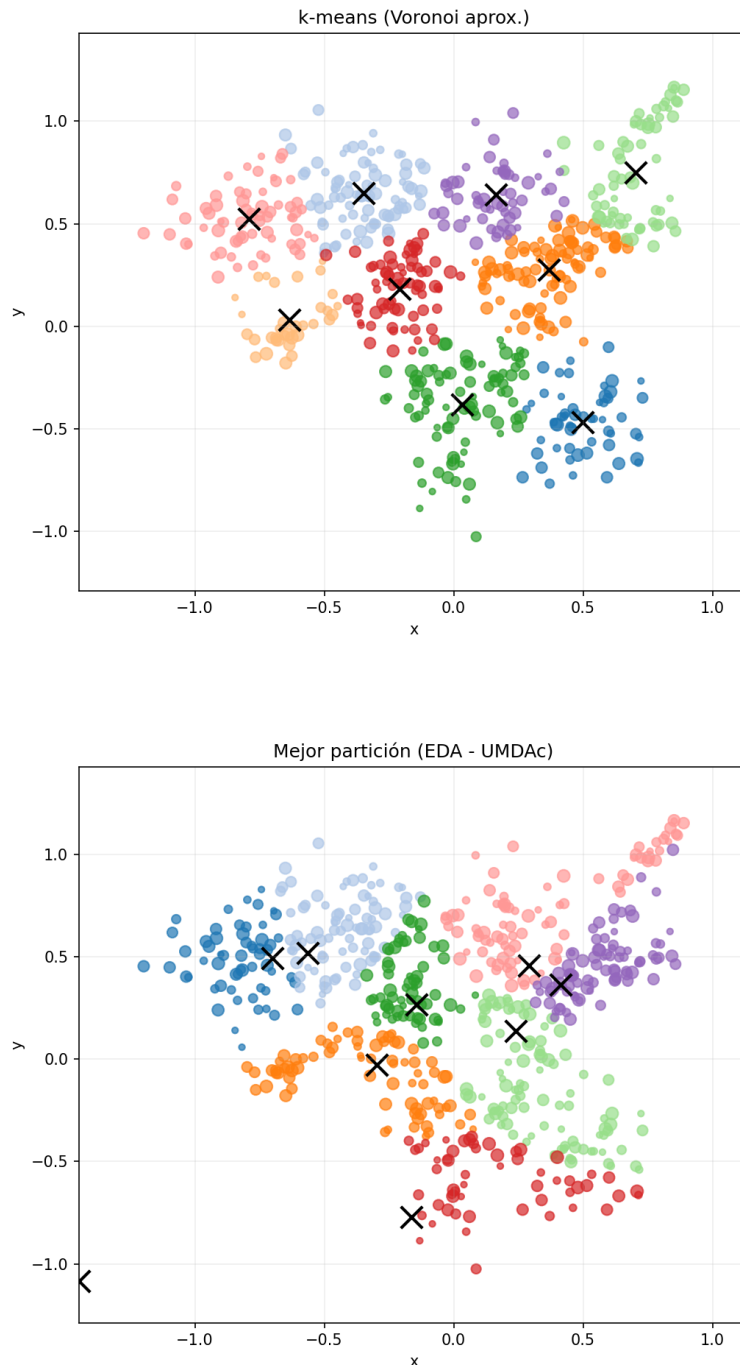


Figura 5.1: Partición por centros obtenida mediante *k*-means vs Mejor partición por centros obtenida mediante EDA (UMDAc) con función objetivo v1.

Capítulo 5. Pruebas y análisis de rendimiento

La Tabla 5.2 resume los valores obtenidos para ambas soluciones (extraídos del CSV comparativo `data/metricas_eda_vs_kmeans.csv`). Se observa que EDA reduce de forma clara el desequilibrio (CV) y elimina la penalización de capacidad, a costa de una compacticidad normalizada mayor (peor), que es el intercambio esperado dada la definición del objetivo $v1$.

Método	J (v1)	CV	CapPen	Compact
k -means	0.3070	0.2763	56.6889	0.0617
EDA (UMDAc)	0.1990	0.1800	0.0000	0.0951

Tabla 5.2: Comparación cuantitativa entre k -means y EDA (UMDAc) en partición por centros con función objetivo $v1$.

Además de la solución final, interesa ver si la optimización “se comporta bien” o si baja por casualidad. La Figura 5.2 muestra la evolución del coste J por iteración. La tendencia descendente indica que el algoritmo encuentra mejoras reales y estabiliza alrededor de una zona buena del espacio de búsqueda. Como referencia base, k -means obtiene $J = 0,3070$ (Tabla 5.2), mientras que EDA converge a $J \approx 0,1990$.

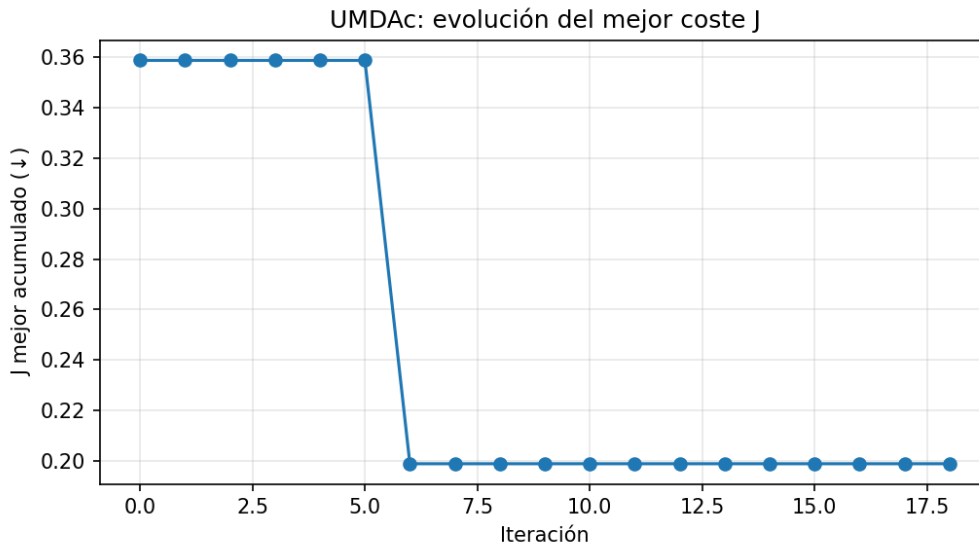


Figura 5.2: Evolución del coste J durante la optimización con EDA (función objetivo $v1$).

Como resultado “final” de esta fase, también se genera la Figura 5.3. Es equivalente en contenido a la anterior, pero con el formato final usado en la memoria.

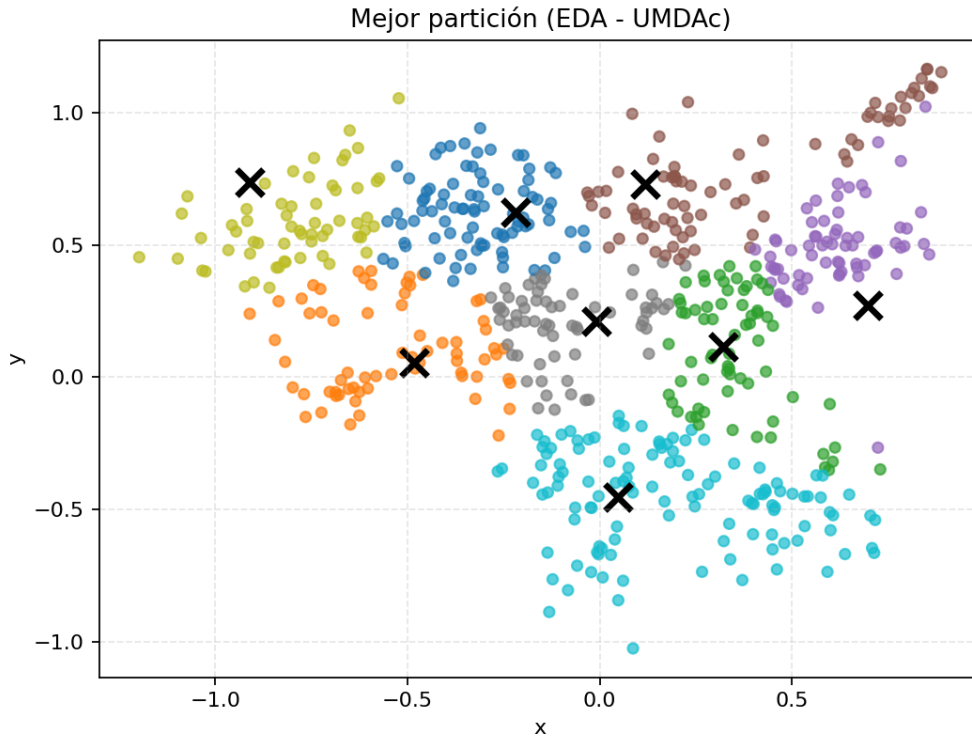


Figura 5.3: Mejor partición obtenida mediante EDA con función objetivo $v1$ (formato final). Valores: $J = 0,1990$, $CV = 0,1800$, $CapPen = 0,0000$, $Compact = 0,0951$.

5.3.1. Lectura de las métricas agregadas

En la práctica, la interpretación de esta comparación es:

- Si solo interesa compacidad geométrica, *k-means* suele ser competitivo.
- En cuanto se exige equilibrio de carga y límites, EDA produce mejoras claras porque el objetivo se lo “obliga”.
- La compacticidad puede empeorar un poco, pero es un intercambio controlado (y depende de los pesos del objetivo).

Este resultado sirve como justificación técnica: no se está usando EDA “por usar algo avanzado”, sino porque el problema real no tiene un único criterio, y los métodos base no están diseñados para optimizar varios a la vez.

5.4. Resultados con *balanced k-means*

La variante *balanced k-means* se estudia como enfoque intermedio. La idea de fondo es mejorar el reparto de carga sin meterse todavía en una optimización completa multi-criterio.

En muchos casos se observa que este método reduce violaciones fuertes de capacidad y mejora algo el CV respecto a *k-means*, manteniendo una geometría

similar. Sin embargo, sigue teniendo una limitación: no puede optimizar márgenes o regularización geométrica si no se modelan explícitamente. Por eso suele funcionar bien como “parche” rápido, pero no sustituye a una optimización completa cuando el objetivo es más exigente.

La Figura 5.4 muestra este resultado:

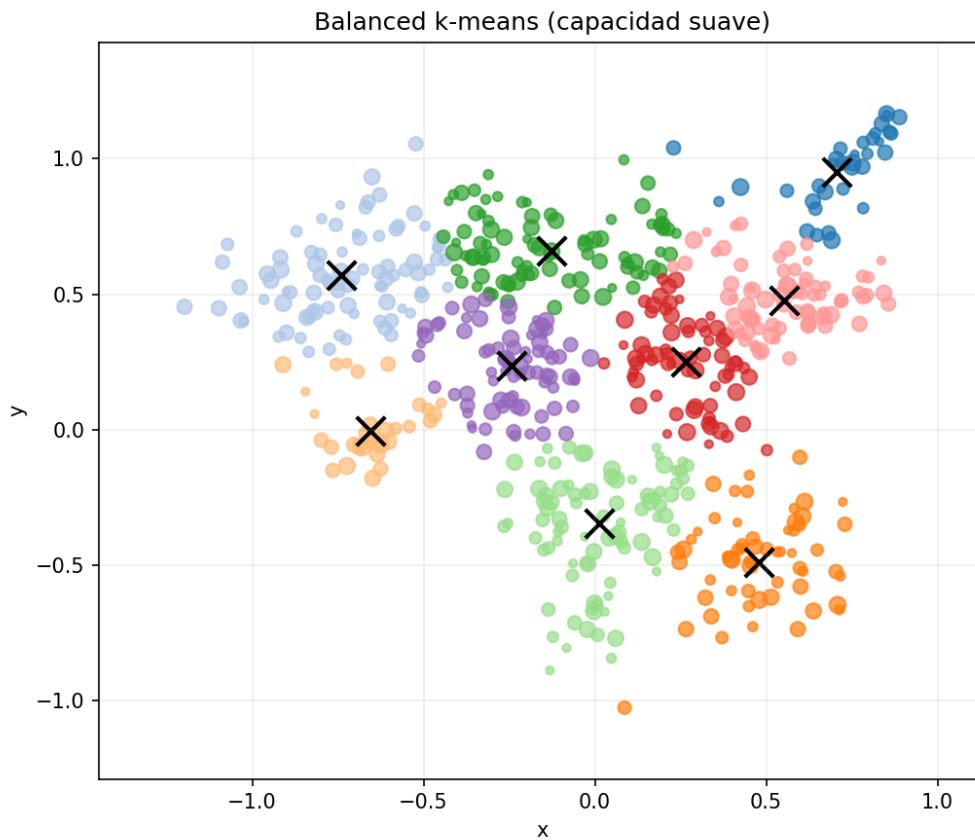


Figura 5.4: Partición por centros obtenida mediante *balanced k-means*

De forma cuantitativa, *balanced k-means* mejora parcialmente los indicadores de carga frente a *k-means*, aunque no alcanza la mejora de EDA cuando el objetivo penaliza explícitamente la capacidad. En concreto, para la misma configuración evaluada con la función *v1*: $J = 0,3072$, $CV = 0,2803$, $\text{CapPen} = 45,4667$ y $\text{Compact} = 0,0609$. Para facilitar la comparación, la Tabla 5.3 resume los tres métodos.

5.5. Exploración de compromisos mediante análisis de Pareto (función objetivo v2)

Método	J (v1)	CV	CapPen	Compact
k -means	0.3070	0.2763	56.6889	0.0617
<i>balanced k-means</i>	0.3072	0.2803	45.4667	0.0609
EDA (UMDAc)	0.1990	0.1800	0.0000	0.0951

Tabla 5.3: Métricas agregadas (función objetivo v1) para partición por centros: k -means, *balanced k-means* y EDA.

5.5. Exploración de compromisos mediante análisis de Pareto (función objetivo v2)

Para estudiar compromisos entre equilibrio y compacticidad, se utiliza un enfoque de *escalarización* (suma ponderada) sobre una función objetivo extendida (v2). Es decir, no se ejecuta un algoritmo multiobjetivo “puro”, sino que se lanzan múltiples ejecuciones del optimizador EDA cambiando los pesos de la función agregada y observando cómo varían las métricas resultantes. En ese sentido, el procedimiento es equivalente a un análisis de sensibilidad de los pesos, pero también permite aproximar un conjunto de soluciones no dominadas bajo el método de suma ponderada.

En concreto, se toma como configuración de referencia un vector de pesos $(c_v, cap, m) = (0.5, 0.25, 0.05)$, y se explora un entorno discreto alrededor mediante una rejilla:

$$c_v \in \{0, 0.3, 0.5, 0.7\}, \quad cap \in \{0, 0.15, 0.25, 0.35\}, \quad m \in \{0, 0.05, 0.1\}.$$

Aquí c_v pondera el término de desequilibrio (CV), cap el término de penalización de capacidad, y m el término asociado al margen (se favorecen soluciones con mayor margen mínimo). Los ejes de las figuras no representan pesos: representan los **valores de las métricas** obtenidas (CV y compacticidad), mientras que las etiquetas (c_v, cap, m) indican la configuración de pesos con la que se generó cada solución.

La Figura 5.5 muestra una primera visualización compacta de ese análisis, útil para ver la tendencia general. La Figura 5.6 muestra la nube completa con mayor calidad, y la Figura 5.7 destaca el frente de Pareto (soluciones no dominadas dentro del conjunto generado).

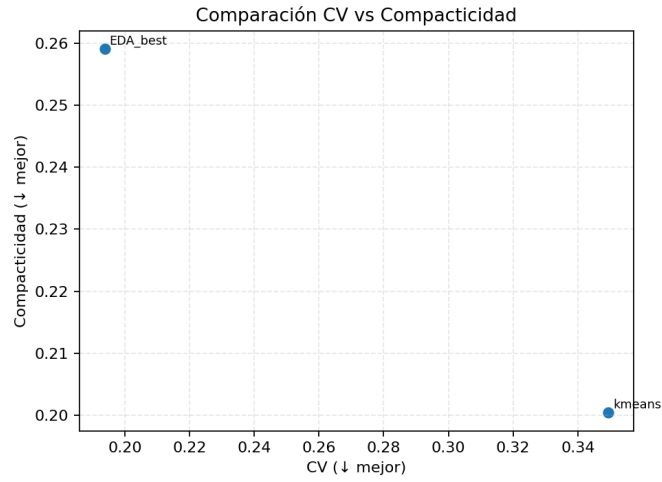


Figura 5.5: Compromiso CV vs compacticidad (valores de las métricas) al variar pesos en la función objetivo v2. Referencias en la misma figura: k -means (CV= 0,349, Compact= 0,200) y una solución EDA de referencia (CV= 0,194, Compact= 0,259).

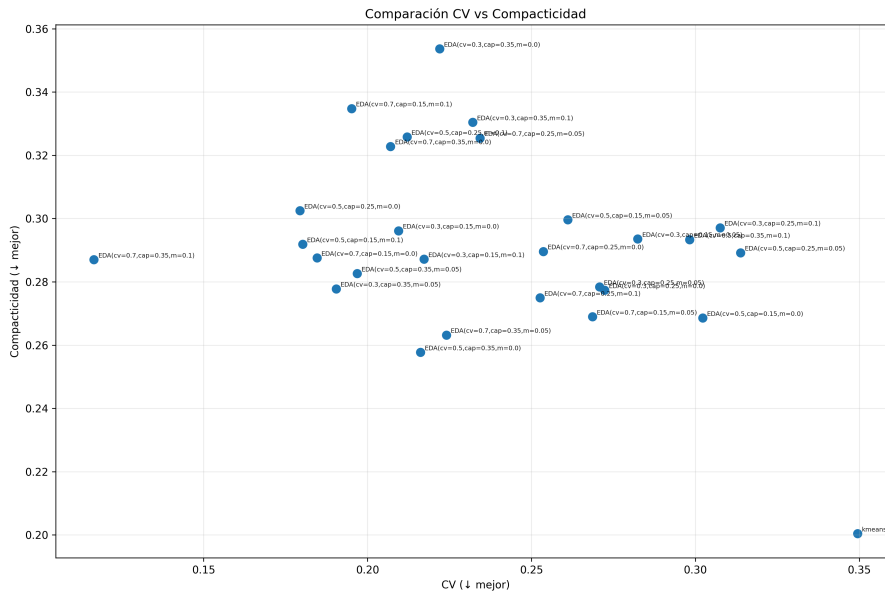


Figura 5.6: Nube de soluciones obtenidas al variar pesos en la función objetivo v2 (EDA). Cada punto representa una ejecución; los ejes muestran los valores de CV y compacticidad obtenidos por la solución. Las etiquetas: cv (desequilibrio), cap (capacidad) y m (margen), indican los pesos usados en la función agregada

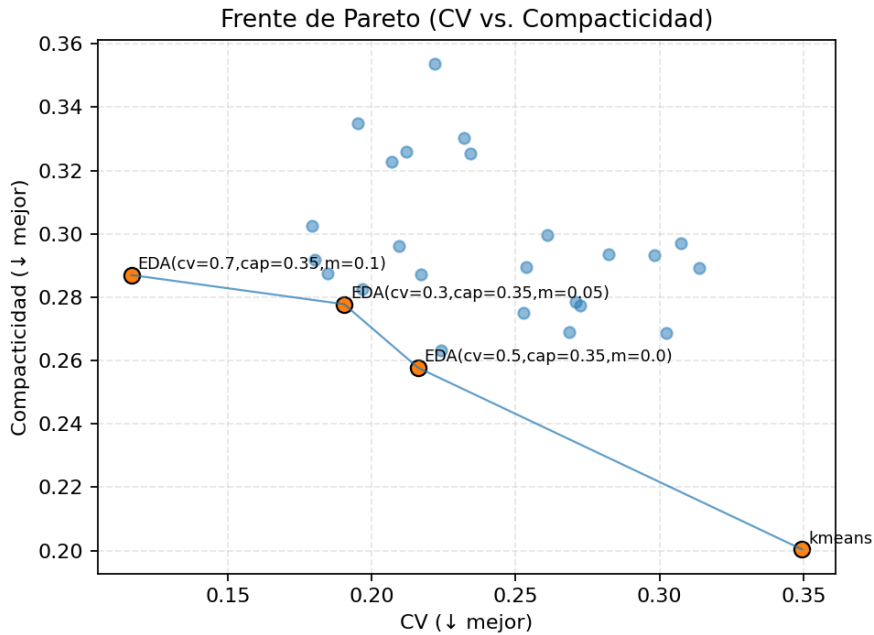


Figura 5.7: Frente de Pareto (soluciones no dominadas) extraído de la nube de la Figura 5.6. Cada punto del frente corresponde a una configuración de pesos distinta; no existe una única asignación de pesos para todo el frente.

La interpretación es importante: no existe una única solución perfecta. Si se baja mucho el CV, normalmente se sacrifica compacticidad, y si se fuerza compacticidad extrema, aparecen sectores con carga descompensada. En este trabajo, el análisis sirve para visualizar compromisos y para justificar una elección razonable de pesos (por ejemplo, la configuración de referencia $(0.5, 0.25, 0.05)$). Si se necesitase seleccionar automáticamente una solución representativa, habría que fijar un criterio de decisión adicional (p.ej., un punto ideal o preferencias del dominio).

Como el problema combina criterios que compiten entre sí, el análisis multiobjetivo ayuda a entender compromisos en lugar de forzar una única solución. Este enfoque está bien establecido en optimización evolutiva [13], y existen algoritmos clásicos para aproximar frentes de Pareto o para explorar soluciones por descomposición [14], [15].

5.6. Escenarios probados

En esta parte se analizan los tres escenarios finales sobre los que se aplica el algoritmo completo: un caso pequeño ($n = 20$), uno intermedio ($n = 50$) y uno grande ($n = 300$). En los tres casos se obtiene una partición rectangular mediante cortes recursivos tipo Mondrian, buscando un compromiso entre equilibrio de carga, compactidad geométrica y estabilidad de las fronteras.

Cada ejecución produce dos artefactos principales: (i) una figura con la partición

Capítulo 5. Pruebas y análisis de rendimiento

final, y (ii) un fichero CSV con el detalle de la solución. En este CSV cada fila representa una entidad del dataset y recoge tanto su asignación a celda como métricas agregadas de la celda y del escenario completo. Esto permite reconstruir el resultado y, a la vez, analizarlo sin depender de trazas externas.

Además, durante la optimización (EDA sobre hiperparámetros) se registra la traza del mejor valor de la función objetivo J por iteración, junto con la configuración de hiperparámetros asociada. Esta traza es útil para verificar convergencia, comparar ejecuciones y explorar el *landscape* de la función objetivo sobre el espacio de soluciones factibles.

La Tabla 5.4 resume las métricas globales más representativas de cada escenario a partir de las columnas con prefijo `global_`. En concreto: $CV \equiv \text{global_cv_w_cells}$, $Compact \equiv \text{global_compactness_norm}$, $Border \equiv \text{global_internal_border_norm}$ y $\overline{margin}_{\min} \equiv \text{global_margin_min_norm_mean}$. Son métricas globales calculadas a partir de las celdas (por ejemplo, el CV se calcula sobre los pesos por celda), y aparecen como constantes en todo el CSV (prefijo `global_*`). Se observa que el escenario intermedio ($n = 50$) consigue el equilibrio más homogéneo (menor CV). En cambio, el caso grande ($n = 300$) incrementa el número de celdas y el perímetro interno, reflejando una partición más fina (más cortes).

n	K	W_{total}	CV	Compact	Border	$\overline{margin}_{\min}$
20	7	89.0	0.177	0.0897	0.864	$2,80 \cdot 10^{-5}$
50	8	265.0	0.091	0.0760	0.973	$1,24 \cdot 10^{-2}$
300	26	1546.0	0.203	0.0426	2.027	$3,49 \cdot 10^{-3}$

Tabla 5.4: Resumen de métricas globales para los tres escenarios finales (valores obtenidos de `FINAL_mondrian_20.csv`, `FINAL_mondrian_50.csv` y `FINAL_mondrian_300.csv`).

5.6.1. Guía de interpretación de las columnas del CSV

Para evitar ambigüedades, a continuación se resumen las columnas más relevantes que aparecen en los ficheros `FINAL_mondrian_n.csv` de los escenarios. La misma estructura se mantiene en los tres tamaños, lo que facilita comparar escenarios.

Columna	Descripción
point_id	Identificador de la entidad dentro del escenario analizado.
x, y	Coordenadas normalizadas de la entidad en el plano.
w	Peso o carga asociada a la entidad.
cell_id	Identificador de la celda/sector asignado a la entidad.
cell_*	Métricas agregadas de la celda: número de puntos, peso total, área, densidad, compacidad, relación de aspecto y margen mínimo normalizado. Estas columnas se repiten en todas las filas que pertenecen a la misma celda.
global_*	Métricas globales del escenario: número de celdas K , peso total, CV entre celdas, perímetro interno normalizado, compacidad global y media del margen mínimo. Estas columnas son constantes en todo el fichero.

Tabla 5.5: Descripción de columnas para interpretar los CSV finales del Mondrian optimizado.

5.6.2. Escenario $n = 20$

Este escenario sirve como comprobación visual rápida: con pocas entidades se entiende bien cómo se realizan los cortes y cómo quedan las celdas resultantes. La partición final se muestra en la Figura 5.8.

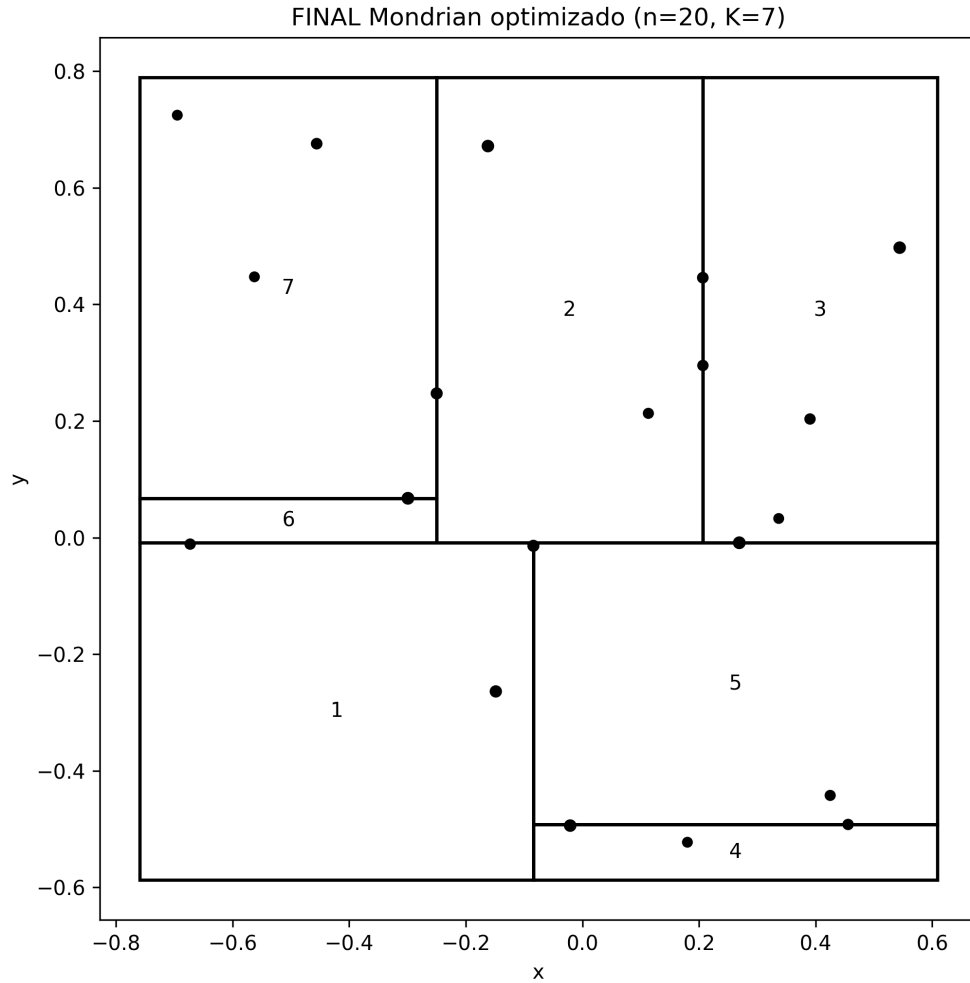


Figura 5.8: Partición final para el escenario de 20 entidades (Mondrian optimizado). Valores globales: $K = 7$, $W_{\text{total}} = 89,0$, $CV = 0,177$, $Compact = 0,0897$, $Border = 0,864$, $margin_{\text{mín}} = 2,80 \cdot 10^{-5}$, $J = 1,4370$.

En este caso se obtienen $K = 7$ celdas. El equilibrio de carga es razonable (columna CV de la Tabla 5.4, equivalente a $global_cv_w_cells \approx 0,177$), aunque todavía hay diferencias entre celdas: el peso total por celda (Tabla 5.7) oscila aproximadamente entre 9 y 21 (media $\approx 12,9$). También aparecen celdas con muy pocos puntos (incluso 1), lo que suele generar regiones pequeñas con densidad alta. En un escenario tan pequeño esto entra dentro de lo esperado, porque el algoritmo prioriza separar grupos sin perder el control de la carga.

Para ver el formato real de la solución, la Tabla 5.6 muestra las primeras 10 filas del CSV. Se aprecia que varias filas comparten el mismo `cell_id` y repiten los campos agregados (por ejemplo, `cell_w_sum` o `cell_n_points`) porque son propiedades de la celda, no del punto individual.

5.6. Escenarios probados

point_id	x	y	w	cell_id	cell_n_points	cell_w_sum	cell_margin_min_norm
1	-0.6220	0.3965	3.0	3	3	11.0	0.0000
2	-0.7415	0.5238	4.0	6	2	11.0	0.0374
3	-0.5585	0.4270	6.0	1	4	16.0	0.0000
4	-0.4156	0.2245	5.0	1	4	16.0	0.0000
5	0.0572	-0.3287	7.0	7	2	10.0	0.0000
6	-0.0631	-0.3300	3.0	7	2	10.0	0.0000
7	0.4728	-0.2808	4.0	4	4	21.0	0.0000
8	0.3624	0.1228	3.0	3	3	11.0	0.0000
9	0.3999	-0.3059	8.0	4	4	21.0	0.0000
10	0.2114	0.6301	3.0	2	2	9.0	0.0000

Tabla 5.6: Primeras 10 filas del fichero `FINAL_mondrian_20.csv` (asignación por celda y métricas asociadas).

Estas filas permiten una lectura rápida del CSV:

- Las filas 3 y 4 pertenecen a la misma celda (`cell_id = 1`) y comparten `cell_n_points = 4` y `cell_w_sum = 16`; el CSV está pensado para que la información agregada sea inmediata al inspeccionarlo.
- Cuando aparece `cell_margin_min_norm = 0.0000` (muy habitual en $n = 20$), significa que *algún* punto de esa celda queda prácticamente pegado a una frontera; con el formato `%.4f` muchos valores pequeños se redondean a 0.0000.

La Tabla 5.7 resume las métricas por celda (una fila por `cell_id`). Se observan áreas pequeñas y densidades altas en celdas con pocos puntos, lo que indica cortes más finos alrededor de puntos aislados. Además, aparece alguna celda claramente alargada (`cell_aspect_ratio > 7`), típico cuando se separa un grupo pequeño sin fragmentar en exceso el resto. Para aportar contexto estadístico, se incluye también la media y la desviación típica por columna: idealmente, una solución “muy homogénea” tendería a desviaciones pequeñas (cercanas a 0) en las columnas de carga.

cell_id	cell_n_points	cell_w_sum	cell_area	cell_density	cell_aspect_ratio
1	4	16.0	0.3431	46.6	1.06
2	2	9.0	0.2907	31.0	1.06
3	3	11.0	0.8430	13.0	1.52
4	4	21.0	0.1888	111.2	7.23
5	3	12.0	1.1299	10.6	1.19
6	2	11.0	0.1614	68.2	1.19
7	2	10.0	0.4465	22.4	2.32
Media	2.8571	12.8571	0.4862	43.2857	2.2243
Desv. típ.	0.8330	3.9071	0.3372	34.3262	2.0844

Tabla 5.7: Resumen por celda para el escenario $n = 20$ (una fila por `cell_id`).

5.6.3. Escenario $n = 50$

Este escenario intermedio ya representa un caso más estable: hay suficientes puntos para que los cortes se apoyen en la distribución espacial, pero todavía se

puede inspeccionar la partición con facilidad. La Figura 5.9 muestra la solución final.

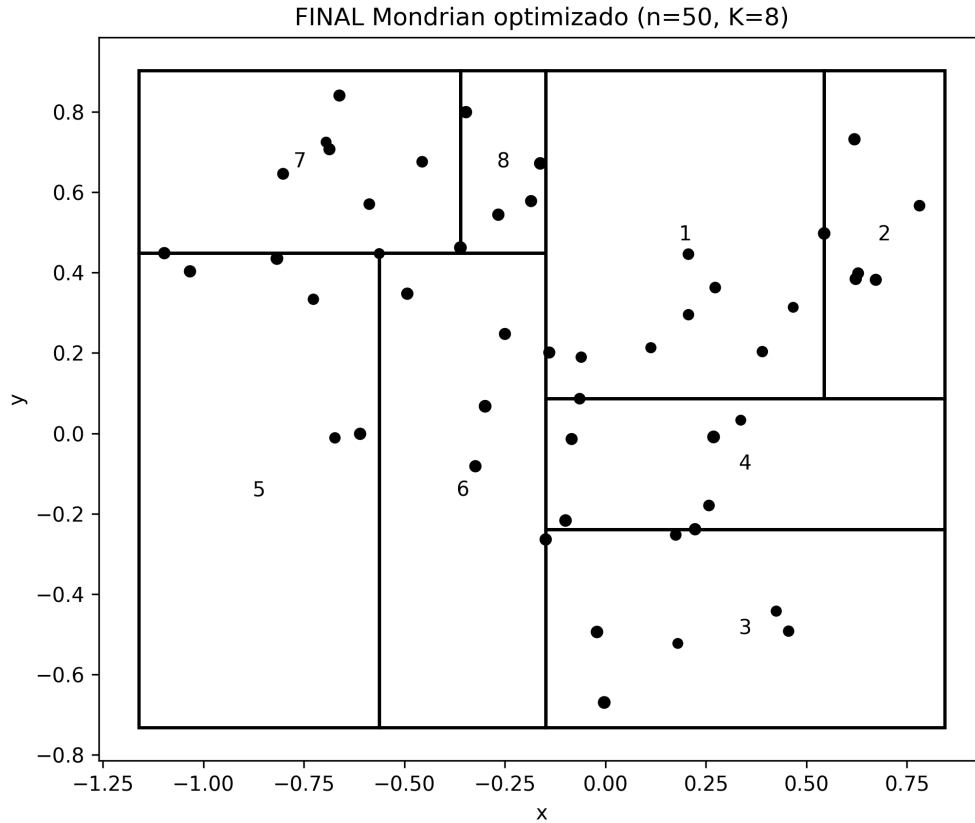


Figura 5.9: Partición final para el escenario de 50 entidades (Mondrian optimizado). Valores globales: $K = 8$, $W_{\text{total}} = 265,0$, $CV = 0,091$, $Compact = 0,0760$, $Border = 0,973$, $\overline{margin}_{\text{mín}} = 1,24 \cdot 10^{-2}$, $J = 1,5699$.

En este caso se obtienen $K = 8$ celdas y el equilibrio mejora de forma clara (columna CV de la Tabla 5.4, equivalente a $\text{global_cv_w_cells} \approx 0,091$). El peso total por celda (Tabla 5.9) queda concentrado en un rango estrecho (aprox. 28 a 37), que es justo lo que se busca cuando la carga es el factor principal a repartir. La compacidad global también mejora respecto a $n = 20$ (menor $\text{global_compactness_norm}$, columna $Compact$ de la Tabla 5.4), lo que indica celdas más “cerradas” en torno a sus puntos.

La Tabla 5.8 muestra el inicio del CSV. Como ejemplo, las dos primeras filas comparten $\text{cell_id} = 2$, con el mismo número de puntos y el mismo peso total por celda.

5.6. Escenarios probados

point_id	x	y	w	cell_id	cell_n_points	cell_w_sum	cell_margin_min_norm
1	-0.6220	0.3965	3.0	2	7	33.0	0.0000
2	-0.7415	0.5238	4.0	2	7	33.0	0.0000
3	-0.5585	0.4270	6.0	4	8	37.0	0.0000
4	-0.4156	0.2245	5.0	4	8	37.0	0.0000
5	0.0572	-0.3287	7.0	5	5	33.0	0.0000
6	-0.0631	-0.3300	3.0	5	5	33.0	0.0000
7	0.4728	-0.2808	4.0	5	5	33.0	0.0000
8	0.3624	0.1228	3.0	3	10	37.0	0.0000
9	0.3999	-0.3059	8.0	5	5	33.0	0.0000
10	0.2114	0.6301	3.0	7	6	32.0	0.0000

Tabla 5.8: Primeras 10 filas del fichero `FINAL_mondrian_50.csv` (asignación por celda y métricas asociadas).

En estas filas aparece un patrón útil para interpretar el resultado: hay celdas con pocos puntos pero con carga elevada (por ejemplo, `cell_id` = 5 con 5 puntos y peso 33), y otras con más puntos pero peso total similar (por ejemplo, `cell_id` = 3 con 10 puntos y peso 37). Esto confirma que el criterio principal es equilibrar *carga*, no igualar el número de entidades.

La Tabla 5.9 resume las celdas completas. El número de puntos por celda se mantiene entre 1 y 10, y la relación de aspecto es moderada en la mayoría (aprox. 1–4), lo que ayuda a que los sectores no queden “raros” geométricamente. Se incluyen media y desviación típica para aportar contexto: en particular, el CV bajo se refleja también en una desviación típica pequeña de `cell_w_sum`.

cell_id	cell_n_points	cell_w_sum	cell_area	cell_density	cell_aspect_ratio
1	7	34.0	0.2630	129.3	1.71
2	7	33.0	0.3718	88.7	1.15
3	10	37.0	0.3843	96.3	1.69
4	8	37.0	0.3467	106.7	1.53
5	5	33.0	0.2432	135.7	2.10
6	6	32.0	0.3105	103.0	1.30
7	6	32.0	0.3083	103.8	1.16
8	1	28.0	0.0965	290.2	4.29
<i>Media</i>	6.2500	33.2500	0.2905	131.7125	1.8663
<i>Desv. típ.</i>	2.4875	2.9475	0.0838	63.6121	1.0133

Tabla 5.9: Resumen por celda para el escenario $n = 50$ (una fila por `cell_id`).

5.6.4. Escenario $n = 300$

Este escenario es el más exigente. Con 300 entidades aparecen hubs y flujos más marcados, y el algoritmo tiende a realizar más cortes para adaptarse a la heterogeneidad. La partición resultante se muestra en la Figura 5.10.

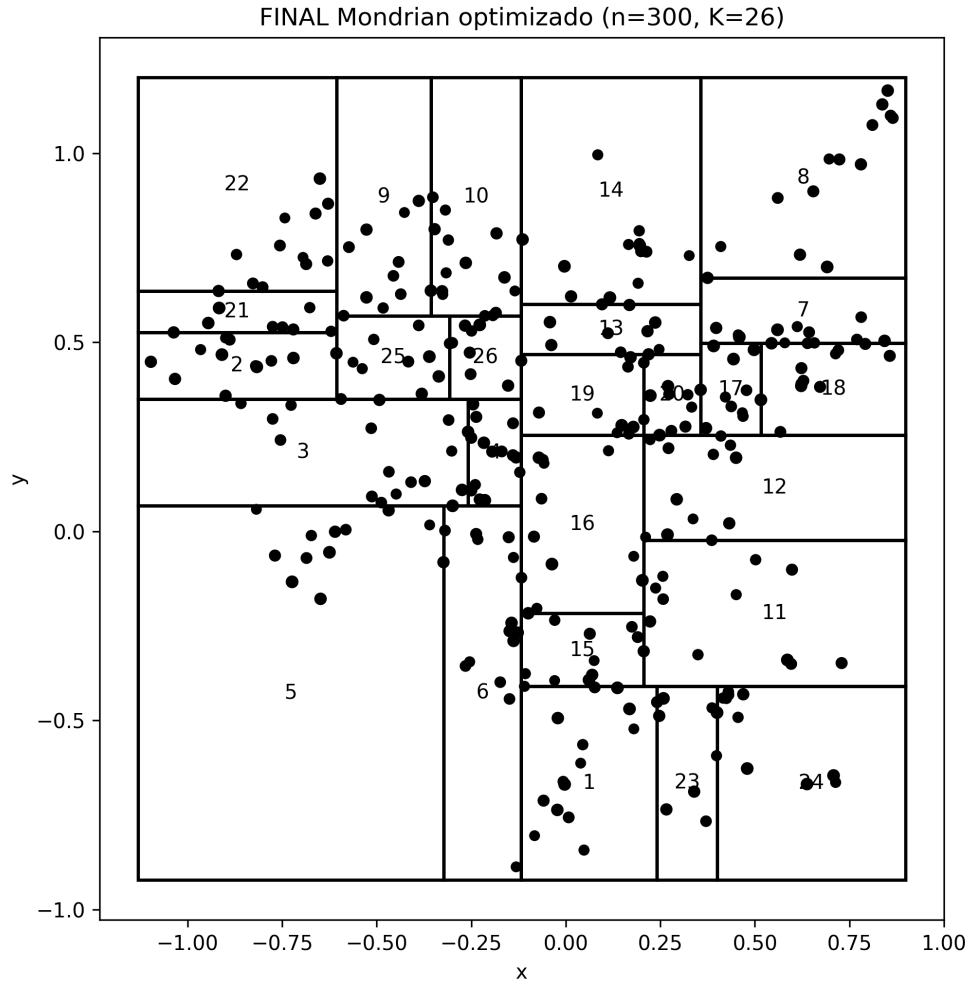


Figura 5.10: Partición final para el escenario de 300 entidades (Mondrian optimizado). Valores globales: $K = 26$, $W_{\text{total}} = 1546,0$, $CV = 0,203$, $Compact = 0,0426$, $Border = 2,027$, $margin_{\text{mín}} = 3,49 \cdot 10^{-3}$, $J = 2,0714$.

Aquí se obtienen $K = 26$ celdas. Esto se refleja en el perímetro interno normalizado (`global_internal_border_norm`, columna *Border* en la Tabla 5.4): al aumentar el número de cortes, aumenta la cantidad de frontera interna respecto a la frontera exterior del dominio. A nivel de equilibrio, el CV sube, lo que es esperable porque con más celdas hay más variación posible entre zonas densas y zonas dispersas. Por otro lado, la compacidad global mejora porque al haber más sectores, cada celda puede “ajustarse” mejor a un grupo local sin abarcar demasiado espacio.

La Tabla 5.10 muestra las primeras filas del CSV. Se aprecia, por ejemplo, que la celda 12 aparece repetida en varias filas y que agrupa 19 puntos con peso total 80. Este tipo de repetición permite localizar rápidamente qué celdas concentran más entidades y carga.

5.6. Escenarios probados

point_id	x	y	w	cell_id	cell_n_points	cell_w_sum	cell_margin_min_norm
1	-0.6220	0.3965	3.0	20	8	53.0	0.0000
2	-0.7415	0.5238	4.0	20	8	53.0	0.0000
3	-0.5585	0.4270	6.0	12	19	80.0	0.0000
4	-0.4156	0.2245	5.0	12	19	80.0	0.0000
5	0.0572	-0.3287	7.0	16	16	87.0	0.0000
6	-0.0631	-0.3300	3.0	16	16	87.0	0.0000
7	0.4728	-0.2808	4.0	11	18	87.0	0.0000
8	0.3624	0.1228	3.0	15	17	79.0	0.0000
9	0.3999	-0.3059	8.0	11	18	87.0	0.0000
10	0.2114	0.6301	3.0	12	19	80.0	0.0000

Tabla 5.10: Primeras 10 filas del fichero `FINAL_mondrian_300.csv` (asignación por celda y métricas asociadas).

En esta muestra aparece la última columna con valores 0.0000 porque se imprime con cuatro decimales: en $n = 300$ muchos márgenes mínimos normalizados por celda son muy pequeños (del orden de 10^{-4} o menor), y se redondean a 0.0000 en la tabla. Aun así, el valor medio global $\overline{margin}_{\min}$ (Tabla 5.4) es claramente no nulo ($\approx 3,49 \cdot 10^{-3}$), lo que confirma que no todas las celdas tienen margen prácticamente cero.

En este escenario se entiende mejor por qué el CSV incluye métricas por celda:

- El peso por celda varía de forma notable (aprox. 39 a 115), lo que explica el aumento del CV. Las celdas con peso alto suelen corresponder a hubs del dataset.
- La densidad llega a valores muy altos en algunas celdas (por encima de 900), lo que indica cortes muy finos alrededor de zonas muy concentradas. Esto suele ser útil si se quiere que un hub no “arrastre” demasiada carga a un único sector.
- También aparecen celdas alargadas (`cell_aspect_ratio` > 10), normalmente asociadas a flujos o corredores, donde es difícil encajar regiones compactas sin mezclar grupos.

La Tabla 5.11 recoge el resumen completo por celda. Con 26 celdas, esta tabla permite localizar sectores extremos (muy densos, muy cargados o muy alargados) y relacionarlos con la geometría observada en la Figura 5.10. Al final se incluyen media y desviación típica por columna para contextualizar la variabilidad: en particular, se observa que la densidad y la relación de aspecto presentan desviaciones altas, coherentes con un dataset heterogéneo con hubs y corredores.

cell_id	cell_n_points	cell_w_sum	cell_area	cell_density	cell_aspect_ratio
1	12	60.0	0.1178	509.3	1.72
2	7	45.0	0.0834	539.5	1.72
3	14	72.0	0.0728	989.6	1.79
4	7	40.0	0.2190	182.6	1.01
5	8	58.0	0.1317	440.4	1.01
6	6	44.0	0.1337	329.2	1.31
7	13	70.0	0.1254	558.6	1.31
8	4	39.0	0.0656	594.8	1.78
9	13	69.0	0.0794	869.0	1.78
10	11	51.0	0.0645	790.2	2.13
11	18	87.0	0.1564	556.2	2.13
12	19	80.0	0.1944	411.6	1.18
13	9	51.0	0.0699	729.1	1.18
14	9	44.0	0.1684	261.3	1.53
15	17	79.0	0.2326	339.6	1.53
16	16	87.0	0.0711	1223.9	3.31
17	10	55.0	0.1012	543.4	3.31
18	8	52.0	0.0915	568.0	3.92
19	10	51.0	0.1014	502.9	3.92
20	8	53.0	0.2074	255.5	1.07
21	11	54.0	0.2118	254.9	1.07
22	10	50.0	0.2286	218.7	1.23
23	14	74.0	0.3260	226.9	1.23
24	10	44.0	0.1359	323.8	13.21
25	5	46.0	0.1005	457.6	13.21
26	8	115.0	0.1148	1001.4	1.37
<i>Media</i>	10.6538	60.3846	0.1387	526.0769	2.6908
<i>Desv. típ.</i>	3.8625	17.8801	0.0655	264.8996	3.1505

Tabla 5.11: Resumen por celda para el escenario $n = 300$ (una fila por `cell_id`).

5.7. Discusión global

Los resultados permiten sacar conclusiones claras:

- La rejilla es útil como referencia, pero no es adecuada cuando la distribución es heterogénea.
- *k*-means produce sectores compactos, pero no controla carga; por eso falla cuando el peso importa. Evaluado con la función objetivo *v1* sirve también como referencia cuantitativa: $J = 0,3070$ (Tabla 5.2).
- *balanced k-means* mejora el reparto, aunque sigue limitado si se exigen más criterios.
- EDA permite optimizar directamente lo que interesa (balance, capacidad, margen), de forma controlada mediante una función objetivo.
- El análisis de Pareto es clave para justificar que no existe una única solución óptima: hay compromisos.
- La propuesta final (Mondrian recursivo + EDA) es la que mejor encaja con el objetivo del TFG: sectores rectangulares, explicables, con K automático y métricas controladas.

Capítulo 6

Conclusiones y trabajo futuro

Este capítulo resume las conclusiones principales del trabajo y plantea mejoras razonables para continuar la línea propuesta.

6.1. Evaluación de los objetivos del trabajo

A lo largo del trabajo se ha estudiado un problema de partición espacial con carga asociada, inspirado en escenarios de sectorización. La evaluación se ha planteado de forma reproducible con un dataset sintético controlado y con métricas que permiten comparar soluciones más allá de “cómo se ven”.

De los resultados presentados en el Capítulo 5 se extraen conclusiones claras:

- Las particiones geométricas simples, como la rejilla, sirven como referencia, pero no se adaptan a distribuciones no uniformes. En presencia de hubs o flujos concentrados, generan sectores con carga descompensada.
- Las particiones por centros (*k*-means) tienden a ser compactas, pero no garantizan equilibrio de carga ni cumplimiento de capacidades. Cuando el peso importa, no basta con minimizar distancia.
- Variantes como *balanced k-means* ayudan a reducir desbalances fuertes de carga, pero siguen siendo un enfoque limitado si se exigen criterios adicionales (margen, regularización geométrica, etc.).
- Los métodos basados en EDA (UMDAc) permiten optimizar explícitamente lo que se mide. Esto aporta una mejora práctica: el algoritmo no “espera” que el balance salga solo, sino que lo busca porque forma parte del objetivo.
- El análisis de Pareto refuerza una idea importante: no existe una única solución óptima universal. Dependiendo de si se prioriza balance o compactidad, aparecen compromisos inevitables.
- La propuesta final (Mondrian recursivo + EDA para hiperparámetros) encaja bien con el objetivo del TFG: sectores rectangulares encajados, número de sectores automático y métricas globales controladas, con un formato de salida (CSV + figura) fácil de auditar.

6.2. Limitaciones

El trabajo utiliza datos sintéticos en 2D, lo que simplifica la geometría y permite explicar bien los resultados. Sin embargo, en un escenario real podrían aparecer restricciones adicionales: zonas prohibidas, rutas preferentes, límites no rectangulares o dinámicas temporales.

También se ha usado una función objetivo agregada, que es práctica y explicable, pero depende de pesos. Aunque el análisis de Pareto ayuda a entender compromisos, la selección final de pesos siempre tiene un componente de criterio del decisor.

6.3. Trabajo futuro

A partir de los resultados obtenidos, una línea clara de mejora es reforzar la parte de visualización y de expresividad geométrica de las particiones. En la versión actual, el enfoque Mondrian genera sectores rectangulares encajados, lo que simplifica la construcción y el análisis, pero limita la forma final de los límites entre sectores. En un escenario operativo real, es habitual querer fronteras que se ajusten mejor a la distribución espacial y, además, disponer de zonas que no se puedan atravesar o asignar (por ejemplo, áreas restringidas o regiones sin interés operativo). Asimismo, se requiere el perfeccionamiento del algoritmo encargado de garantizar una distancia mínima de seguridad entre los objetos y las fronteras de los sectores; si bien esta restricción ya ha sido considerada en el algoritmo, su implementación actual debe ser optimizada para asegurar una mayor precisión y robustez.

En primer lugar, se propone mejorar la visualización de la solución final: asignación de colores por sector, leyenda y métricas resumidas directamente en la figura. Esto facilitaría comparar particiones a simple vista, detectar sectores con carga excesiva y comunicar resultados sin necesidad de consultar tablas adicionales.

En segundo lugar, se plantea permitir sectores no rectangulares y la inclusión de regiones “prohibidas” entre sectores. Una alternativa natural sería representar los sectores mediante particiones basadas en regiones (por ejemplo, aproximaciones tipo Voronoi) y después imponer restricciones geométricas adicionales: suavizado de fronteras, penalización de sectores demasiado irregulares y exclusión explícita de zonas vetadas. La Figura 6.1 ilustra la idea: sectores coloreados, límites más flexibles y huecos que representan áreas no asignables.

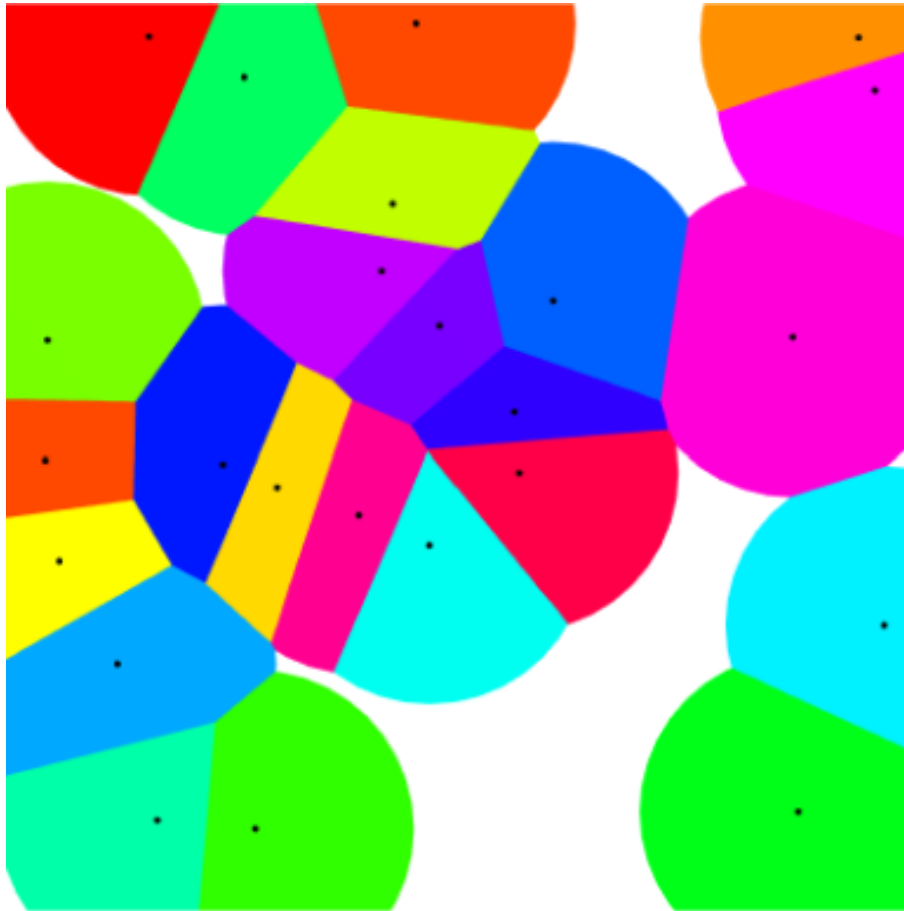


Figura 6.1: Idea de mejora futura: sectores coloreados, fronteras no rectangulares y zonas prohibidas entre sectores.

Por último, otra extensión relevante sería introducir estabilidad temporal: en lugar de recalculer una partición desde cero, incorporar un término que penalice cambios bruscos entre soluciones consecutivas. Esto permitiría particiones más “suaves” en el tiempo, evitando reconfiguraciones innecesarias cuando la distribución de carga cambia ligeramente.

Bibliografía

- [1] AirNav Radar. «AirNav Radar: Live Flight Tracker», visitado 3 de ene. de 2026. dirección: <https://www.airnavradar.com/>.
- [2] A. Okabe, B. Boots, K. Sugihara y S. N. Chiu, *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*, 2.^a ed. Wiley, 2000.
- [3] F. Aurenhammer, «Voronoi Diagrams—A Survey of a Fundamental Geometric Data Structure», *ACM Computing Surveys*, vol. 23, n.º 3, págs. 345-405, 1991.
- [4] S. P. Lloyd, «Least Squares Quantization in PCM», *IEEE Transactions on Information Theory*, vol. 28, n.º 2, págs. 129-137, 1982.
- [5] Q. Du, V. Faber y M. Gunzburger, «Centroidal Voronoi Tessellations: Applications and Algorithms», *SIAM Review*, vol. 41, n.º 4, págs. 637-676, 1999.
- [6] T. F. Gonzalez, «Clustering to Minimize the Maximum Intercluster Distance», *Theoretical Computer Science*, vol. 38, págs. 293-306, 1985.
- [7] B. W. Kernighan y S. Lin, «An Efficient Heuristic Procedure for Partitioning Graphs», *The Bell System Technical Journal*, vol. 49, n.º 2, págs. 291-307, 1970.
- [8] J. Shi y J. Malik, «Normalized Cuts and Image Segmentation», *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, n.º 8, págs. 888-905, 2000.
- [9] U. von Luxburg, «A Tutorial on Spectral Clustering», *Statistics and Computing*, vol. 17, n.º 4, págs. 395-416, 2007.
- [10] H. Mühlenbein y G. Paaß, «From Recombination of Genes to the Estimation of Distributions I. Binary Parameters», en *Parallel Problem Solving from Nature (PPSN)*, Springer, 1996, págs. 178-187.
- [11] P. Larrañaga y J. A. Lozano, «A Review of Estimation of Distribution Algorithms», *IEEE Transactions on Evolutionary Computation*, vol. 6, n.º 1, págs. 45-65, 2002.
- [12] «EDAspy: Estimation of Distribution Algorithms in Python». Repositorio y documentación del proyecto. dirección: <https://github.com/EDAspy/EDAspy>.
- [13] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*. Wiley, 2001.

BIBLIOGRAFÍA

- [14] K. Deb, A. Pratap, S. Agarwal y T. Meyarivan, «A Fast and Elitist Multi-objective Genetic Algorithm: NSGA-II», *IEEE Transactions on Evolutionary Computation*, vol. 6, n.º 2, págs. 182-197, 2002.
- [15] Q. Zhang y H. Li, «MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition», *IEEE Transactions on Evolutionary Computation*, vol. 11, n.º 6, págs. 712-731, 2007.

Anexos

Apéndice A

Fragmentos de código relevantes

Este anexo recoge fragmentos representativos del código desarrollado. Se incluyen únicamente las partes más relevantes para entender la lógica, dejando fuera detalles auxiliares.

A.1. Métricas usadas en la evaluación

Las métricas se basan en sumar pesos por sector y medir equilibrio (CV), penalización por capacidad, compacticidad y margen mínimo normalizado.

```
# metrics.py
def compute_all_metrics(
    X: np.ndarray,
    w: np.ndarray,
    labels: np.ndarray,
    centers: np.ndarray,
    alpha: float = 0.6,
    beta: float = 1.4,
    bbox_diag: float | None = None,
) -> dict:
    X = np.asarray(X, float)
    w = np.asarray(w, float)
    labels = np.asarray(labels, int)
    centers = np.asarray(centers, float)

    k = centers.shape[0]
    W = np.array([w[labels == s].sum() for s in range(k)], dtype=
        ⇨ float)

    cv = balance_cv(W)

    mu = float(W.mean())
    lower = alpha * mu
    upper = beta * mu
```

```
cap_pen = capacity_penalty(W, lower=lower, upper=upper)
cap_pen_norm = float(cap_pen / (float(W.sum()) + 1e-12))

d = np.linalg.norm(X - centers[labels], axis=1)
comp = float(np.average(d, weights=w))

if bbox_diag is None:
    xmin, ymin = X.min(axis=0)
    xmax, ymax = X.max(axis=0)
    bbox_diag = float(np.hypot(xmax - xmin, ymax - ymin) + 1e
        ↪ -12)
comp_norm = float(comp / bbox_diag)

m = margin_stats(X, centers, labels) ["margin_min_norm"]

return {
    "CV": cv,
    "CapPen": cap_pen,
    "CapPen_norm": cap_pen_norm,
    "Compact": comp_norm,
    "Margin_min_norm": float(m),
}
```

Listing A.1: Cálculo agregado de métricas.

A.2. Particiones base: rejilla y *k*-means

La rejilla asigna sectores por coordenadas, y *k*-means asigna según cercanía a centros.

```
# grid.py
def grid_partition(df: pd.DataFrame, nx: int, ny: int, bbox:
    ↪ Tuple[float, float, float, float]):
    xmin, xmax, ymin, ymax = bbox
    dx = (xmax - xmin) / nx
    dy = (ymax - ymin) / ny

    ix = np.floor((df["x"] - xmin) / dx).astype(int)
    iy = np.floor((df["y"] - ymin) / dy).astype(int)
    ix = np.clip(ix, 0, nx-1)
    iy = np.clip(iy, 0, ny-1)

    sector_id = iy * nx + ix

    out = df.copy()
    out["ix"] = ix
    out["iy"] = iy
```

A.3. Mondrian recursivo optimizado

```
out["sector"] = sector_id
return out
```

Listing A.2: Partición en rejilla.

```
# kmeans.py
def kmeans_partition(df: pd.DataFrame, k: int, random_state: int
    ↪ = 0):
    X = df[["x", "y"]].to_numpy(float)
    km = KMeans(n_clusters=k, n_init=10, random_state=
        ↪ random_state)
    labels = km.fit_predict(X)
    out = df.copy()
    out["sector"] = labels
    return out, km.cluster_centers_
```

Listing A.3: Partición *k*-means.

A.3. Mondrian recursivo optimizado

La solución final implementa una partición tipo *Mondrian* recursiva. La idea es dividir recursivamente el dominio mediante cortes ortogonales (verticales u horizontales), ajustando el corte para que no aparezcan huecos y deteniendo el proceso cuando se cumplen criterios de calidad (balance de pesos) y tamaño (máximo número de puntos por celda). Además, los hiperparámetros del proceso se optimizan mediante un EDA continuo (UMDAc) para minimizar una función objetivo multi-criterio.

A.3.1. Cuantil ponderado para el corte

Para que el corte sea sensible al peso de las entidades, se utiliza un cuantil ponderado. Con ello, los cortes tienden a separar mejor carga, no solo número de puntos.

```
# FINAL_mondrian_recursive.py
def weighted_quantile(x: np.ndarray, w: np.ndarray, q: float) ->
    ↪ float:
    """Cuantil ponderado q in [0,1]."""
    q = float(np.clip(q, 0.0, 1.0))
    order = np.argsort(x)
    xs = x[order]
    ws = w[order]
    c = np.cumsum(ws)
    t = q * float(c[-1])
    j = int(np.searchsorted(c, t))
    return float(xs[min(j, len(xs) - 1)])
```

Listing A.4: Cálculo de cuantil ponderado.

A.3.2. División de una celda (sin huecos)

Cada celda se divide en dos subceldas. Si el corte falla (por ejemplo, se quedaría un lado vacío), se aplica una división aleatoria para garantizar progreso. El detalle importante es el ajuste del corte: se coloca en el máximo del grupo izquierdo para que el encaje sea perfecto.

```
# FINAL_mondrian_recursive.py
def split_cell(points: np.ndarray, w: np.ndarray, cell: Cell,
    ↪ axis: int, seed: int, cut_q: float) -> tuple[Cell, Cell]:
    """
    Divide una celda en vertical (axis=0) u horizontal (axis=1).
    Corte en cuantil ponderado cut_q (optimizable).
    Si falla (lado vacío), split aleatorio.
    Encaje sin huecos: el corte se ajusta al máximo del grupo
    ↪ izquierdo.
    """
    rng = np.random.default_rng(seed)
    idx = cell.idx
    pts = points[idx]
    ww = w[idx]

    coord = pts[:, axis]
    if np.allclose(coord, coord[0]):
        shuffled = rng.permutation(idx)
        mid = len(shuffled) // 2
        left_idx = shuffled[:mid]
        right_idx = shuffled[mid:]
    else:
        cut = weighted_quantile(coord, ww, cut_q)
        left_mask = coord <= cut
        right_mask = ~left_mask

        if (not left_mask.any()) or (not right_mask.any()):
            shuffled = rng.permutation(idx)
            mid = len(shuffled) // 2
            left_idx = shuffled[:mid]
            right_idx = shuffled[mid:]
        else:
            left_idx = idx[left_mask]
            right_idx = idx[right_mask]

    if axis == 0:
        cut = float(points[left_idx, 0].max())
        c1 = Cell(cell.x_min, cut, cell.y_min, cell.y_max, np.
            ↪ array(left_idx, int))
        c2 = Cell(cut, cell.x_max, cell.y_min, cell.y_max, np.
            ↪ array(right_idx, int))
```

```

else:
    cut = float(points[left_idx, 1].max())
    c1 = Cell(cell.x_min, cell.x_max, cell.y_min, cut, np.
        ↪ array(left_idx, int))
    c2 = Cell(cell.x_min, cell.x_max, cut, cell.y_max, np.
        ↪ array(right_idx, int))

return c1, c2

```

Listing A.5: Split Mondrian con ajuste de corte.

A.3.3. Regla de parada y K automático

El número de sectores K no se fija manualmente. El proceso va dividiendo la celda “más problemática” hasta cumplir simultáneamente: (i) máximo de puntos por celda (`max_points`), y (ii) un umbral de equilibrio medido como CV de pesos por celda (`cv_target`).

```

# FINAL_mondrian_recursive.py
def mondrian_auto(
    points: np.ndarray,
    w: np.ndarray,
    seed: int = 7,
    margin: float = 0.05,
    max_points: int = 3,
    cv_target: float = 0.30,
    alpha_weight: float = 0.20,
    cut_q: float = 0.50,
    k_max: int | None = None,
) -> list[Cell]:
    """
    Mondrian recursivo con K automático.
    Para cuando:
        (max puntos por celda <= max_points) Y (CV pesos por celda
        ↪ <= cv_target)
    """
    n = len(points)
    (x_min, y_min) = points.min(axis=0) - margin
    (x_max, y_max) = points.max(axis=0) + margin

    cells: list[Cell] = [Cell(float(x_min), float(x_max), float(
        ↪ y_min), float(y_max), np.arange(n, dtype=int))]
    if k_max is None:
        k_max = n

    it = 0
    while True:
        it += 1

```

```
weights_per_cell = np.array([w[c.idx].sum() for c in cells
    ↪ ], float)
max_pts = max(c.idx.size for c in cells)
cur_cv = cv(weights_per_cell) if len(cells) > 1 else 0.0

if (max_pts <= max_points) and (cur_cv <= cv_target):
    break
if len(cells) >= k_max:
    break

pts_counts = np.array([c.idx.size for c in cells], int)
score = (pts_counts / (pts_counts.max() + 1e-12)) + float(
    ↪ alpha_weight) * (
    weights_per_cell / (weights_per_cell.max() + 1e-12)
)
j = int(np.argmax(score))
target = cells[j]

pts = points[target.idx]
var_x = float(np.var(pts[:, 0]))
var_y = float(np.var(pts[:, 1]))
axis = 0 if var_x >= var_y else 1

c1, c2 = split_cell(points, w, target, axis=axis, seed=
    ↪ seed + it, cut_q=cut_q)

cells.pop(j)
cells.extend([c1, c2])

return cells
```

Listing A.6: Mondrian recursivo con K automático.

A.3.4. Función objetivo y penalización suave sobre K

El problema es multiobjetivo. Para poder optimizarlo con UMDAc, se construye un coste escalar J que combina equilibrio, compacticidad, regularización geométrica y margen. También se añade una penalización suave para evitar soluciones degeneradas con K demasiado pequeño o excesivo.

```
# FINAL_mondrian_recursive.py
def objective_for_params(df: pd.DataFrame, params: dict, seed:
    ↪ int) -> tuple[float, dict, list[Cell]]:
    """
    Devuelve J (menor mejor) + métricas + cells.
    J combina criterios (multiobjetivo en escalar) +
    ↪ penalizaciones anti-degeneración.
    """
```

```

X = df[["x", "y"]].to_numpy(float)
ww = df["w"].to_numpy(float)

cells = mondrian_auto(
    X, ww,
    seed=seed,
    margin=params["margin"],
    max_points=int(params["max_points"]),
    cv_target=params["cv_target"],
    alpha_weight=params["alpha_weight"],
    cut_q=params["cut_q"],
    k_max=params["k_max"],
)
M = compute_metrics(df, cells)
n = len(df)
K = M["K"]

K_min, K_max_soft, K_target = _k_soft_bounds(n)

if K < K_min:
    k_pen = (K_min - K) / (K_min + 1e-12)
    k_pen = 2.0 * (k_pen ** 2) + 1.0
elif K > K_max_soft:
    k_pen = (K - K_max_soft) / (K_max_soft + 1e-12)
    k_pen = 1.0 * (k_pen ** 2)
else:
    k_pen = abs(K - K_target) / (K_target + 1e-12)

J = (
    1.35 * M["cv_w_cells"] +
    0.80 * M["compactness_norm"] +
    0.55 * M["internal_border_norm"] +
    0.65 * (1.0 - M["margin_min_norm_mean"]) +
    0.85 * k_pen
)

if K <= 1:
    J += 5.0
if K >= n:
    J += 1.5

return float(J), M, cells

```

Listing A.7: Función objetivo escalarizada.

A.3.5. Optimización de hiperparámetros con UMDAc

La optimización con EDA se aplica sobre hiperparámetros del Mondrian: `max_points`, `cv_target`, `margin`, `alpha_weight` y `cut_q`. Se usa una evaluación robusta promediando varias semillas internas para evitar soluciones frágiles.

```
# FINAL_mondrian_recursive.py
def eda_optimize_params(df: pd.DataFrame, seed: int = 7) -> dict
    ↪ :
    """
    EDA continuo sobre hiperparámetros (UMDAc).
    Evalúa robustamente (media sobre varias semillas internas)
        ↪ para evitar
    soluciones "frágiles" que colapsan K por azar.
    """
    rng = np.random.default_rng(seed)
    n = len(df)

    bounds = {
        "max_points": (2.0, float(max(10, min(30, n)))),
        "cv_target": (0.15, 0.65),
        "margin": (0.02, 0.12),
        "alpha_weight": (0.05, 0.70),
        "cut_q": (0.35, 0.65),
    }

    mean = np.array([
        float(max(2, min(12, round(math.sqrt(n) / 2)))),
        0.35,
        0.05,
        0.20,
        0.50,
    ], float)
    std = np.array([2.5, 0.10, 0.02, 0.15, 0.06], float)

    pop = 50 if n <= 60 else 60
    elite = max(10, pop // 5)
    iters = 12 if n <= 60 else 14

    robust_seeds = [seed + 101, seed + 202, seed + 303]

    best_J = float("inf")
    best_params = None

    for t in range(iters):
        samples = rng.normal(loc=mean, scale=std, size=(pop, len(
            ↪ mean)))
```

```

for j, k in enumerate(bounds.keys()):
    lo, hi = bounds[k]
    samples[:, j] = np.clip(samples[:, j], lo, hi)

scored = []
for s in samples:
    cand = dict(zip(bounds.keys(), s.tolist()))
    cand["max_points"] = int(np.clip(int(round(cand["
        ↪ max_points"])), 2, n))
    cand["k_max"] = n

    Js = []
    for rs in robust_seeds:
        J, _, _ = objective_for_params(df, cand, seed=rs +
            ↪ 1000 * t)
        Js.append(J)
    Jm = float(np.mean(Js))
    scored.append((Jm, cand))

scored.sort(key=lambda x: x[0])
J0, p0 = scored[0]
if J0 < best_J:
    best_J = J0
    best_params = p0

elites = np.array(
    [[d["max_points"], d["cv_target"], d["margin"], d["
        ↪ alpha_weight"], d["cut_q"]] for _, d in scored[:
        ↪ elite]],
    float,
)
mean = elites.mean(axis=0)
std = elites.std(axis=0) + 1e-6

std_floor = np.array([1.0, 0.04, 0.008, 0.05, 0.02], float
    ↪ )
std = np.maximum(std, std_floor)

assert best_params is not None
return best_params

```

Listing A.8: EDA continuo (UMDAc) para optimizar hiperparámetros.

Apéndice B

Listas de figuras, tablas y código fuente

Lista de figuras

1.1. Ejemplo de tráfico aéreo en tiempo real.	1
2.1. Referencia visual: composición de Mondrian como inspiración conceptual para particiones rectangulares.	6
2.2. Ejemplo conceptual de partición por rejilla: cortes regulares sin tener en cuenta la distribución de carga.	6
2.3. Ejemplo conceptual de partición tipo Voronoi: asignación al centro más cercano.	7
3.1. Ejemplo de dataset sintético generado, combinando hubs (4 nubes de puntos) y flujos (3 estructuras alargadas).	12
4.1. Validación inicial: <i>smoke test</i> del bucle de optimización.	19
4.2. Validación inicial: ejecución de UMDAc sobre una función continua de referencia (Sphere).	20
5.1. Partición por centros obtenida mediante <i>k</i> -means vs Mejor partición por centros obtenida mediante EDA (UMDAc) con función objetivo v1.	27
5.2. Evolución del coste J durante la optimización con EDA (función objetivo v1).	28
5.3. Mejor partición obtenida mediante EDA con función objetivo v1 (formato final). Valores: $J = 0,1990$, $CV = 0,1800$, $CapPen = 0,0000$, $Compact = 0,0951$	29
5.4. Partición por centros obtenida mediante <i>balanced k-means</i>	30
5.5. Compromiso CV vs compacticidad (valores de las métricas) al variar pesos en la función objetivo v2. Referencias en la misma figura: <i>k</i> -means ($CV = 0,349$, $Compact = 0,200$) y una solución EDA de referencia ($CV = 0,194$, $Compact = 0,259$).	32

5.6. Nube de soluciones obtenidas al variar pesos en la función objetivo v2 (EDA). Cada punto representa una ejecución; los ejes muestran los valores de CV y compacticidad obtenidos por la solución. Las etiquetas: <i>cv</i> (desequilibrio), <i>cap</i> (capacidad) y <i>m</i> (margen), indican los pesos usados en la función agregada	32
5.7. Frente de Pareto (soluciones no dominadas) extraído de la nube de la Figura 5.6. Cada punto del frente corresponde a una configuración de pesos distinta; no existe una única asignación de pesos para todo el frente.	33
5.8. Partición final para el escenario de 20 entidades (Mondrian optimizado). Valores globales: $K = 7$, $W_{\text{total}} = 89,0$, $CV = 0,177$, $Compact = 0,0897$, $Border = 0,864$, $\overline{margin}_{\text{mín}} = 2,80 \cdot 10^{-5}$, $J = 1,4370$	36
5.9. Partición final para el escenario de 50 entidades (Mondrian optimizado). Valores globales: $K = 8$, $W_{\text{total}} = 265,0$, $CV = 0,091$, $Compact = 0,0760$, $Border = 0,973$, $\overline{margin}_{\text{mín}} = 1,24 \cdot 10^{-2}$, $J = 1,5699$	38
5.10. Partición final para el escenario de 300 entidades (Mondrian optimizado). Valores globales: $K = 26$, $W_{\text{total}} = 1546,0$, $CV = 0,203$, $Compact = 0,0426$, $Border = 2,027$, $\overline{margin}_{\text{mín}} = 3,49 \cdot 10^{-3}$, $J = 2,0714$	40
6.1. Idea de mejora futura: sectores coloreados, fronteras no rectangulares y zonas prohibidas entre sectores.	45

Lista de tablas

3.1. Ejemplo de las primeras filas del dataset sintético generado.	13
5.1. Peso total y número de entidades por sector en la partición por rejilla.	26
5.2. Comparación cuantitativa entre <i>k</i> -means y EDA (UMDAc) en partición por centros con función objetivo v1.	28
5.3. Métricas agregadas (función objetivo v1) para partición por centros: <i>k</i> -means, <i>balanced k-means</i> y EDA.	31
5.4. Resumen de métricas globales para los tres escenarios finales (valores obtenidos de <code>FINAL_mondrian_20.csv</code> , <code>FINAL_mondrian_50.csv</code> y <code>FINAL_mondrian_300.csv</code>).	34
5.5. Descripción de columnas para interpretar los CSV finales del Mondrian optimizado.	35
5.6. Primeras 10 filas del fichero <code>FINAL_mondrian_20.csv</code> (asignación por celda y métricas asociadas).	37
5.7. Resumen por celda para el escenario $n = 20$ (una fila por <code>cell_id</code>).	37
5.8. Primeras 10 filas del fichero <code>FINAL_mondrian_50.csv</code> (asignación por celda y métricas asociadas).	39
5.9. Resumen por celda para el escenario $n = 50$ (una fila por <code>cell_id</code>).	39
5.10. Primeras 10 filas del fichero <code>FINAL_mondrian_300.csv</code> (asignación por celda y métricas asociadas).	41
5.11. Resumen por celda para el escenario $n = 300$ (una fila por <code>cell_id</code>).	42

Lista de código fuente

A.1. Cálculo agregado de métricas.	51
A.2. Partición en rejilla.	52
A.3. Partición k -means.	53
A.4. Cálculo de cuantil ponderado.	53
A.5. Split Mondrian con ajuste de corte.	54
A.6. Mondrian recursivo con K automático.	55
A.7. Función objetivo escalarizada.	56
A.8. EDA continuo (UMDAc) para optimizar hiperparámetros.	58

Apéndice C

Informe de originalidad

ANDRES ALEXANDER SANCHEZ HIDALGO

TFG___Andres_Alexander_Sanchez_Hidalgo.pdf

-  Turnitin Memoria Final
-  TFG ETSIINF (Moodle PP)
-  Universidad Politécnica de Madrid

Detalles del documento

Identificador de la entrega

trn:oid:::1:3456772384

Fecha de entrega

13 ene 2026, 2:55 p.m. GMT+1

Fecha de descarga

13 ene 2026, 3:01 p.m. GMT+1

Nombre del archivo

8677_ANDRES_ALEXANDER_SANCHEZ_HIDALGO_TFG___Andres_Alexander_Sanchez_Hidalgo_53046....pdf

Tamaño del archivo

3.2 MB

75 páginas

15.462 palabras

80.238 caracteres

Capítulo C. Informe de originalidad



Página 2 de 80 - Descripción general de integridad

Identificador de la entrega trn:oid::1:3456772384

5% Similitud general

El total combinado de todas las coincidencias, incluidas las fuentes superpuestas, para ca...

Fuentes principales

- 4% Fuentes de Internet
- 3% Publicaciones
- 0% Trabajos entregados (trabajos del estudiante)



Recibo digital

Este recibo confirma que su trabajo ha sido recibido por Turnitin. A continuación podrá ver la información del recibo con respecto a su entrega.

La primera página de tus entregas se muestra abajo.

Autor de la entrega: ANDRES ALEXANDER SANCHEZ HIDALGO
Título del ejercicio: Turnitin Memoria Final
Título de la entrega: TFG__Andres_Alexander_Sanchez_Hidalgo.pdf
Nombre del archivo: 8677_ANDRES_ALEXANDER_SANCHEZ_HIDALGO_TFG__Andre...
Tamaño del archivo: 3.21M
Total páginas: 75
Total de palabras: 15,462
Total de caracteres: 80,238
Fecha de entrega: 13-ene-2026 02:55p. m. (UTC+0100)
Identificador de la entrega: 2856159240



Derechos de autor 2026 Turnitin. Todos los derechos reservados.

Este documento esta firmado por



Firmante	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
Fecha/Hora	Wed Jan 14 14:58:25 CET 2026
Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
Numero de Serie	561
Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)