

Sistema de Optimización Multi Algorítmica para Procesos de Distribución Inteligente

Diseño e Implementación del Prototipo Funcional

Fase 2

Iker Acevedo Vargas

Universidad Sergio Arboleda

Escuela de Ciencias Exactas e Ingeniería

Javier Ochoa

SIST 0175 – Análisis de Algoritmos

Bogotá D.C, Noviembre 2025

1. Introducción al diseño del sistema

La fase de diseño permite transformar los requerimientos analizados en la primera etapa en una estructura técnica clara y funcional. En esta fase se definen los componentes del sistema, su interacción y la forma en que los algoritmos seleccionados se integran para resolver los diferentes subprocesos logísticos de manera eficiente.

El diseño busca garantizar la modularidad, escalabilidad y facilidad de mantenimiento, mediante la implementación de tres submódulos principales: ordenamiento de pedidos, cálculo de rutas óptimas y asignación de recursos. Cada uno de estos submódulos se diseña bajo dos principios algorítmicos distintos, permitiendo la comparación de su rendimiento y logrando seleccionar entre las dos opciones algorítmicas.

2. Arquitectura general del sistema

El sistema de optimización para procesos de distribución inteligente se estructura bajo una arquitectura modular que permite integrar de forma independiente, los distintos algoritmos utilizados en cada etapa del proceso logístico. Esta organización favorece la escalabilidad del proyecto, el mantenimiento del código, la separación de módulos y la evaluación comparativa de los métodos implementados.

La arquitectura está compuesta por tres módulos dentro de una carpeta principal que se comunican entre sí a través de un archivo central de control main.py, el cual coordina la ejecución secuencial de cada componente:

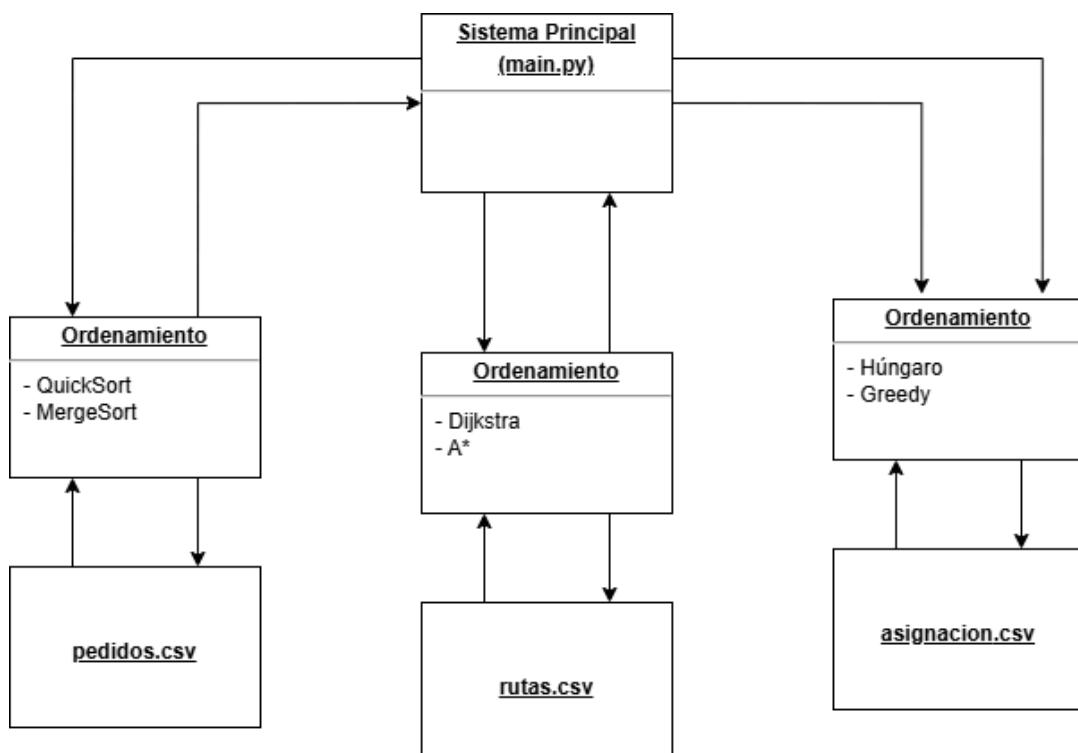
Módulo de Ordenamiento: responsable de organizar los pedidos según distintos criterios prioridad, distancia o peso mediante la selección de algoritmos de ordenamiento eficientes como QuickSort y MergeSort.

Módulo de Rutas: encargado de calcular las rutas más cortas entre puntos de entrega utilizando algoritmos de grafos como Dijkstra y A*. Este módulo trabaja sobre una estructura de red representada a partir de un conjunto de datos de conexiones y distancias.

Módulo de Asignación: implementa el Algoritmo Húngaro y una alternativa Greedy para optimizar la distribución de recursos, permitiendo asignar vehículos o repartidores a rutas específicas minimizando el costo total.

Cada módulo opera de manera independiente sobre sus respectivos conjuntos de datos en formato CSV, y los resultados son integrados en el módulo principal para su análisis y visualización.

El sistema utiliza un conjunto de bibliotecas especializadas de Python, como Pandas, NetworkX, NumPy y Matplotlib, que facilitan la manipulación de datos, la representación de grafos y la generación de visualizaciones estadísticas.



3. Descripción de módulos

3.1 Módulo de ordenamiento

Este módulo se encarga de organizar los pedidos con base en criterios establecidos como la prioridad, el peso o la distancia de entrega. Su función principal es facilitar la gestión de los pedidos antes de proceder a la planificación de rutas.

Se implementaron dos algoritmos de ordenamiento pertenecientes al paradigma de divide y vencerás:

QuickSort: ofrece un rendimiento promedio de $O(n \log n)$, ideal para conjuntos grandes de datos aleatorios.

MergeSort: garantiza una complejidad constante de $O(n \log n)$ incluso en el peor de los casos y permite comparar su estabilidad frente al QuickSort.

Entradas: archivo pedidos.csv con columnas que representan el ID del pedido, destino, prioridad y peso.

Salidas: lista o archivo de pedidos ordenados según el criterio seleccionado.

3.2 Módulo de cálculo de rutas

Este módulo tiene como objetivo encontrar las rutas óptimas de entrega entre los distintos puntos definidos en el mapa logístico. Se basa en el uso de estructuras de grafos para representar las conexiones entre los nodos de origen y destino.

Los algoritmos implementados para el cálculo de las rutas son:

Dijkstra: garantiza la obtención de la ruta más corta determinística entre dos nodos del grafo, con complejidad $O(V^2)$ o $O(E \log V)$ si se usa una cola de prioridad.

A* (A estrella): emplea una heurística basada en la distancia estimada entre nodos para acelerar el proceso de búsqueda, reduciendo el tiempo en escenarios grandes.

Entradas: archivo rutas.csv con datos de origen, destino y distancia entre nodos.

Salidas: ruta más corta encontrada y distancia total recorrida.

3.3 Módulo de asignación de recursos

El módulo de asignación de recursos se utiliza para distribuir eficientemente los vehículos o repartidores en las rutas calculadas, con el propósito de minimizar los costos operativos y balancear la carga de trabajo.

Se implementan los siguientes algoritmos:

Algoritmo Húngaro: técnica de optimización combinatoria con complejidad $O(n^3)$, que garantiza la asignación óptima entre recursos y tareas.

Algoritmo Greedy: enfoque heurístico que selecciona las opciones más económicas de manera iterativa, permitiendo una comparación en tiempo y precisión frente al método óptimo.

Entradas: archivo asignación.csv que contiene las relaciones entre vehículos, rutas y costos asociados.

Salidas: asignaciones óptimas con su costo total.

3.4 Módulo principal (controlador del sistema)

El módulo principal (main.py) actúa como el núcleo del sistema, encargado de coordinar la ejecución de los tres submódulos en orden lógico. Se comunica con cada componente, procesa los resultados y genera la salida final, que incluye reportes comparativos y gráficas de rendimiento.

Funciones principales:

- Cargar los archivos de datos iniciales (pedidos, rutas y asignaciones).
- Ejecutar los algoritmos seleccionados y registrar su rendimiento.
- Consolidar resultados en tablas comparativas y visualizaciones gráficas.

4. Diseño de datos

Esta información se gestiona a través de archivos en formato CSV, los cuales permiten la lectura, procesamiento y almacenamiento de datos de manera eficiente y compatible con las librerías empleadas en Python, como Pandas y NumPy.

4.1 Archivo de pedidos – pedidos.csv

Contiene la información base de los pedidos que deben ser priorizados antes de planificar las rutas de entrega.

El archivo es utilizado por el módulo de ordenamiento.

Campo	Tipo de dato	Descripción
id	Entero	Identificador único del pedido.
destino	Cadena	Ciudad o punto de entrega del pedido.
prioridad	Entero	Nivel de prioridad (1 = alta, 3 = baja).
peso	Decimal	Peso total del pedido en kilogramos.

Ejemplo:

A	B	C	D
1	id	destino	prioridad
2	1	Bogotá	2
3	2	Medellín	1
4	3	Cali	3
5	4	Barranquilla	2
-			11

Uso:

El sistema lee este archivo, lo convierte en una lista o DataFrame, y aplica los algoritmos QuickSort o MergeSort para obtener la lista de pedidos ordenada según el criterio seleccionado.

4.2 Archivo de rutas – rutas.csv

Cada fila indica un tramo entre dos nodos junto con su distancia o peso asociado. Es utilizado por el módulo de cálculo de rutas.

Campo	Tipo de dato	Descripción
origen	Cadena	Nodo de inicio del trayecto.
destino	Cadena	Nodo de llegada.
distancia	Decimal	Distancia entre ambos nodos.

Ejemplo:

	A	B	C	
1	origen	destino	distancia	
2	A	B		5
3	B	C		4
4	A	C		8
5	C	D		3
6	B	E		7
7	D	E		2

Uso:

El módulo convierte esta información en un grafo con la librería NetworkX.

Los algoritmos Dijkstra y A* utilizan estos datos para calcular la ruta más corta o eficiente entre dos puntos.

4.3 Archivo de asignación – asignación.csv

Es utilizado por el módulo de asignación de recursos.

Campo	Tipo de dato	Descripción
vehículo	Cadena	Identificador o nombre del vehículo.
ruta	Cadena	Identificador de la ruta o tarea asignada.
costo	Decimal	Costo asociado a la asignación.

	A	B	C
1	vehiculo	ruta	costo
2	V1	R1	4
3	V1	R2	6
4	V2	R1	3
5	V2	R2	5

Uso:

El sistema transforma la información en una matriz de costos. Posteriormente, aplica el Algoritmo Húngaro para encontrar la asignación óptima y una versión Greedy para comparar la eficiencia y los tiempos de ejecución.

4.4 Estructura de datos interna

Durante la ejecución, los datos se transforman internamente en estructuras adecuadas para cada algoritmo:

Módulo	Estructura utilizada	Descripción
Ordenamiento	Lista de diccionarios o DataFrame	Contiene los pedidos y sus atributos.
Rutas	Grafo no dirigido.	Representa nodos y conexiones con pesos.
Asignación	Matriz NumPy o DataFrame.	Representa los costos entre recursos y tareas.

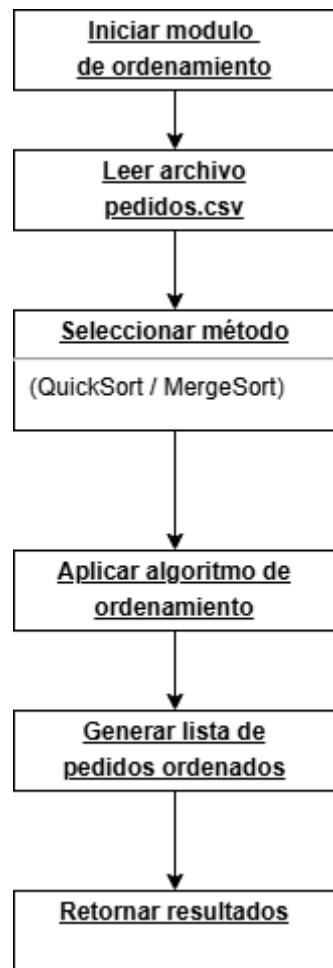
5. Diagramas

A continuación, se presentan los principales diagramas elaborados durante la fase de diseño.

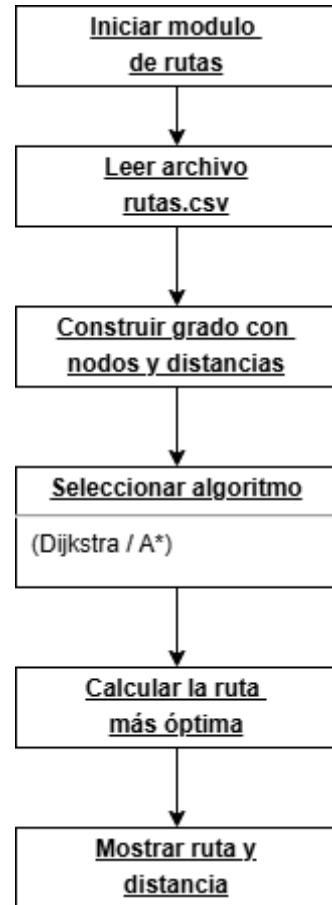
5.1 Diagrama de flujo general del sistema



5.2 Diagrama de flujo del módulo de ordenamiento



5.3 Diagrama de flujo del módulo de cálculo de rutas



5.4 Diagrama de flujo del módulo de asignación de recursos



6. Pseudocódigos de los algoritmos seleccionados

6.1 Algoritmos de ordenamiento

QuickSort

```
PROCEDIMIENTO QuickSort(lista)
    SI longitud(lista) ≤ 1 ENTONCES
        RETORNAR lista
    FIN SI

    pivote ← seleccionarElemento(lista)
    menores ← [x ∈ lista | x < pivote]
    iguales ← [x ∈ lista | x = pivote]
    mayores ← [x ∈ lista | x > pivote]

    RETORNAR QuickSort(menores) + iguales + QuickSort(mayores)
FIN PROCEDIMIENTO
```

MergeSort

```
PROCEDIMIENTO MergeSort(lista)
    SI longitud(lista) ≤ 1 ENTONCES
        RETORNAR lista
    FIN SI

    mitad ← longitud(lista) / 2
    izquierda ← MergeSort(lista[0:mitad])
    derecha ← MergeSort(lista[mitad:])

    RETORNAR Merge(izquierda, derecha)
FIN PROCEDIMIENTO

PROCEDIMIENTO Merge(izquierda, derecha)
    resultado ← []
    MIENTRAS izquierda ≠ Ø Y derecha ≠ Ø HACER
        SI izquierda[0] ≤ derecha[0] ENTONCES
            resultado.agregar(izquierda.eliminarPrimero())
        SINON
            resultado.agregar(derecha.eliminarPrimero())
        FIN SI
    FIN MIENTRAS

    RETORNAR resultado + izquierda + derecha
FIN PROCEDIMIENTO
```

6.2 Algoritmos de cálculo de rutas

Dijkstra

```
PROCEDIMIENTO Dijkstra(Grafo, origen)
    PARA cada nodo EN Grafo HACER
        distancia[nodo] ← ∞
        previo[nodo] ← NULO
    FIN PARA
    distancia[origen] ← 0

    conjuntoSinVisitar ← todos los nodos de Grafo

    MIENTRAS conjuntoSinVisitar ≠ Ø HACER
        nodoActual ← nodo con menor distancia en conjuntoSinVisitar
        eliminar nodoActual de conjuntoSinVisitar

        PARA cada vecino DE nodoActual HACER
            peso ← Grafo.peso(nodoActual, vecino)
            nuevaDistancia ← distancia[nodoActual] + peso

            SI nuevaDistancia < distancia[vecino] ENTONCES
                distancia[vecino] ← nuevaDistancia
                previo[vecino] ← nodoActual
            FIN SI
        FIN PARA
    FIN MIENTRAS

    RETORNAR distancia, previo
FIN PROCEDIMIENTO
```

A* (A estrella)

```

PROCEDIMIENTO A_Estrella(Grafo, inicio, meta, heuristica)
    abiertos ← {inicio}
    cerrados ← Ø
    gScore[inicio] ← 0
    fScore[inicio] ← heuristica(inicio, meta)

    MIENTRAS abiertos ≠ Ø HACER
        actual ← nodo en abiertos con menor fScore
        SI actual = meta ENTONCES
            RETORNAR reconstruirRuta(actual)
        FIN SI

        eliminar actual de abiertos
        agregar actual a cerrados

        PARA cada vecino DE actual HACER
            SI vecino EN cerrados CONTINUAR
            tentativo_g ← gScore[actual] + peso(actual, vecino)

            SI vecino NO EN abiertos O tentativo_g < gScore[vecino] ENTONCES
                previo[vecino] ← actual
                gScore[vecino] ← tentativo_g
                fScore[vecino] ← gScore[vecino] + heuristica(vecino, meta)
                agregar vecino a abiertos
            FIN SI
        FIN PARA
    FIN MIENTRAS
FIN PROCEDIMIENTO

```

6.3 Algoritmos de asignación de recursos

Algoritmo Húngaro

```

PROCEDIMIENTO Hungaro(matrizCostos)
    Reducir cada fila restando su mínimo
    Reducir cada columna restando su mínimo
    Cubrir todos los ceros con el menor número posible de líneas
    MIENTRAS número de líneas < tamaño de la matriz HACER
        Encontrar el menor valor no cubierto
        Restar este valor de todos los elementos no cubiertos
        Sumarlo a los elementos en la intersección de las líneas
        Volver a cubrir ceros
    FIN MIENTRAS
    Asignar ceros únicos en filas y columnas
    RETORNAR asignaciones óptimas
FIN PROCEDIMIENTO

```

Algoritmo Greedy para asignación

```

PROCEDIMIENTO Asignacion_Greedy(matrizCostos)
    asignaciones ← Ø
    MIENTRAS existan tareas sin asignar HACER
        (vehiculo, ruta, costo) ← menorCostoDisponible(matrizCostos)
        agregar (vehiculo, ruta) a asignaciones
        eliminar vehiculo y ruta de la matriz
    FIN MIENTRAS
    RETORNAR asignaciones
FIN PROCEDIMIENTO

```

7. Herramientas utilizadas

A continuación, se describen las principales herramientas utilizadas durante la fase de diseño e implementación.

7.1 Lenguaje de programación

- Python 3.13

7.2 Bibliotecas y módulos empleados

Biblioteca	Descripción	Uso principal
Pandas	Manipulación y análisis de datos en estructuras tipo <i>DataFrame</i> .	Lectura de archivos CSV, filtrado y ordenamiento de información.
NumPy	Operaciones matemáticas y manejo de matrices.	Implementación de algoritmos de asignación y estructuras de costo.
NetworkX	Modelado y análisis de grafos.	Representación de rutas y cálculo de caminos óptimos (Dijkstra, A*).
Matplotlib / Seaborn	Modelado y análisis de grafos.	Creación de tablas comparativas y gráficos de rendimiento.
Time / Timeit	Medición del rendimiento temporal.	Análisis empírico de ejecución de algoritmos.
OS / CSV	Gestión de archivos y rutas del sistema.	Análisis empírico de ejecución de algoritmos.

7.3 Entorno de desarrollo

- Visual Studio Code.
- Git y GitHub

7.4 Control de datos y archivos

El proyecto emplea una estructura organizada de carpetas que facilita el mantenimiento y la trazabilidad del sistema:

```
Proyecto/
|
|   main.py
|   modulos/
|       |   ordenamiento.py
|       |   rutas.py
|       \   asignacion.py
|
|   informacion/
|       |   pedidos.csv
|       |   rutas.csv
|       \   asignacion.csv
|
\   resultados/
    |   comparativas.csv
    \   graficos/
```

8. Conclusión de la fase de diseño

En esta fase se logró definir cómo se estructurará el sistema de optimización multialgorítmica y la forma en que los diferentes módulos trabajarán juntos para resolver el problema logístico planteado. Se diseñó una arquitectura modular que facilita la comprensión, el mantenimiento y la futura implementación del proyecto, asegurando que cada parte del sistema cumpla una función específica y clara.

Además, se establecieron las estructuras de datos, diagramas y pseudocódigos necesarios para guiar el desarrollo del prototipo funcional. Gracias a este diseño, el sistema contará con una base técnica sólida que permitirá comparar el rendimiento de los algoritmos seleccionados y avanzar hacia la siguiente etapa de implementación con una visión más clara del funcionamiento general del proyecto.