UMAG Universidad de Magallanes | Facultad de Ingeniería

Implementación de un sistema de repartos

Miguel Loaiza[†], **Duvan Figueroa**[†] **and Diego Sanhueza**[†]

[†]*Universidad de Magallanes*

This manuscript was compile on May 17, 2025

Abstract

En este informe se presentará la implementación del sistema de repartos en lenguaje C.

Keywords: *Algoritmos,Lenguaje C, Programación*

Contents

1. Introducción

El presente informe tiene como objetivo exponer, comparar y analizar los resultados obtenidos durante el desarrollo e implementación de un sistema de reparto, enfocado en la optimización de la asignación de recursos, mediante algoritmos voraces (Greedy). En particular, se trabajó con estrategias como *Earliest Deadline First (EDF)*, *Priority-Based Scheduling (PB)* y *Nearest Neighbor (NN)*, las cuales fueron optimizadas en función de su desempeño y adaptabilidad.

Los tres algoritmos seleccionados —*Earliest Deadline First (EDF)*, *Priority-Based Scheduling (PB)* y *Nearest Neighbor (NN)*— han sido analizados en este informe. Para cada uno, se evaluó su comportamiento en diferentes escenarios, considerando el caso promedio, el mejor y el peor caso, con el fin de entender su eficiencia y escalabilidad. Este análisis incluye el estudio de la complejidad de cada estrategia, expresada mediante notación Big-O (*Big O Notation*), lo cual permite cuantificar su rendimiento teórico frente a diferentes volúmenes de datos y restricciones del sistema.

A lo largo del documento se detallan las decisiones de diseño adoptadas, las dificultades enfrentadas durante la implementación, así como los mecanismos utilizados para abordar casos no favorables, tales como entregas inviables por limitaciones temporales o de capacidad.

1.1. Objetivo general

Desarrollar e implementar un sistema de reparto eficiente que optimice la asignación de recursos mediante el uso de algoritmos voraces, tales como *Earliest Deadline First (EDF)*, *Priority-Based Scheduling (PB)* y *Nearest Neighbor (NN)*, evaluando su desempeño, adaptabilidad y robustez frente a errores.

1.2. Objetivos específicos

- Diseñar e implementar los algoritmos EDF, PB y NN para la asignación de entregas a vehículos disponibles.
- Analizar el rendimiento de cada estrategia en términos de distancia recorrida, cumplimiento de plazos y tiempo de ejecución.
- Incorporar mecanismos de optimización como el balanceo de carga y detección de conflictos entre vehículos para mejorar la eficiencia global del sistema.
- Gestionar adecuadamente los casos en los que las entregas no pueden ser asignadas, proponiendo una estrategia para su reprogramación y priorización futura.
- Generar reportes detallados en formato CSV con las métricas obtenidas, permitiendo evaluar el impacto de cada algoritmo sobre los recursos utilizados.
- Generar una carta gantt para poder visualizar de forma grafica cómo se realizan las entregas por parte de los vehículos.

2. Algoritmos

2.1. Earliest Deadline First (EDF)

En este caso, se utilizó la estrategia **Earliest Deadline First (EDF)** para asignar las entregas a los vehículos disponibles, priorizando aquellas entregas con fecha límite más próxima. Esta técnica es útil en contextos donde se desea minimizar el riesgo de incumplir los plazos de entrega.

La función principal `schedule_edf` comienza ordenando todas las entregas mediante la función `custom_qsort`, según el criterio de menor fecha límite (`EARLIEST_DEADLINE_FIRST`).

Posteriormente, se recorre la lista ordenada de entregas. Para cada entrega, se evalúan todos los vehículos disponibles y se selecciona el más adecuado que cumpla con las siguientes condiciones:

- Tipo de vehículo compatible con la entrega.
- Capacidad suficiente de volumen y peso.
- Disponibilidad horaria dentro del rango de flexibilidad definido.

Entre los vehículos que cumplen estas condiciones, se elige aquel que minimice una **función de urgencia** definida como:

$$\text{urgency\_score} = \text{deadline} + \text{distancia al origen}$$

Esta fórmula da preferencia a vehículos que estén más cerca del punto de origen y que puedan cumplir entregas más urgentes.
Una vez asignada una entrega a un vehículo:

- Se calcula la distancia total recorrida por el vehículo (incluyendo el trayecto hacia el origen y luego al destino).
- Se calcula el consumo de combustible estimado y su costo.
- Se actualiza la posición actual del vehículo al destino de la entrega.
- Se reduce su capacidad disponible de volumen y peso.
- Se acumulan métricas como tiempo de espera, distancia total y entregas realizadas.

Al finalizar la ejecución del algoritmo:

- Se imprime un resumen con las entregas realizadas por cada vehículo.
- Se indican las entregas no asignadas, si las hay.
- Se generan estadísticas globales como distancia total, litros de combustible utilizados, costo total y tiempo de ejecución.
- Se exporta un informe en formato CSV con el detalle de cada entrega.

2.2. Priority-Based Scheduling

La estrategia **Greedy Priority-Based Scheduling** implementada en este algoritmo tiene como objetivo asignar entregas a vehículos disponibles de manera eficiente, priorizando aquellas entregas que requieren mayor atención. En este enfoque, las entregas son ordenadas previamente según su nivel de prioridad, lo que permite que las más urgentes se procesen antes. Esta decisión resulta fundamental en contextos donde algunas entregas tienen una mayor criticidad o menor tolerancia al retraso.

La función principal, schedule_pb, se encarga de recorrer todas las entregas ya ordenadas y asignarlas al vehículo más conveniente que cumpla con una serie de requisitos. Para que un vehículo sea considerado válido, debe cumplir con lo siguiente:

- Tener un tipo igual o superior al requerido por la entrega.
- Contar con suficiente capacidad de carga (en volumen y peso).
- Estar disponible dentro del intervalo horario permitido.

Durante el proceso, el algoritmo evalúa todos los vehículos disponibles para cada entrega y selecciona aquel que minimice la distancia total del recorrido, definida como la suma de la distancia desde la posición actual del vehículo hasta el origen de la entrega, más la distancia desde el origen hasta el destino.

$$\text{real\_distance} = \text{distancia al origen} + \text{distancia de entrega}$$

Esta métrica llamada **real_distance** se utiliza como criterio de optimización, con el objetivo de reducir el consumo de combustible y aprovechar al máximo los recursos disponibles.

Una vez asignada una entrega, se actualizan diversos parámetros del vehículo como su ubicación (ahora en el destino de la entrega), su capacidad restante y el número de entregas realizadas. También se calcula el consumo de combustible en función de la distancia recorrida y el tipo de vehículo, y se registran estadísticas como el tiempo de espera, la distancia acumulada y el costo operativo asociado. Además, se asigna un nivel de satisfacción al usuario que depende del tiempo de espera en relación con la prioridad de la entrega.

En caso de que una entrega no pueda ser asignada a ningún vehículo debido a restricciones de tipo, capacidad o disponibilidad horaria, esta se deja sin asignar y se reporta al final del proceso. Al concluir, el algoritmo genera un archivo CSV con la información detallada de las entregas asignadas (informe_entregas_pb.csv) y presenta un resumen de métricas globales, incluyendo el número de entregas completadas, la distancia total recorrida, el combustible utilizado, el costo total y el tiempo de ejecución del algoritmo.

2.3. Nearest Neighbor

En nuestro caso, se uso **Nearest Neighbor Scheduling** para asignar cada entrega al vehículo más conveniente disponible, evaluando criterios como la distancia al origen, el tiempo de espera, la carga actual del vehículo y su disponibilidad temporal. Esto permite encontrar una solución eficiente sin necesidad de evaluar todas las combinaciones posibles.

Al igual que el algoritmo anterior se asigna un nivel de satisfacción al usuario.

La función principal `schedule_nn` invoca a la función auxiliar `assign_nearest_neighbor`, donde ocurre la lógica del algoritmo. Se recorren todas las entregas y, para cada una, se evalúan todos los vehículos disponibles para encontrar el que minimice una función de costo definida como:

$$\text{score} = \text{distancia al origen} + 0.5 \cdot \text{tiempo de espera} + 5 \cdot \text{entregas asignadas} + 10 \cdot \text{uso de capacidad}$$

Donde:

- **Distancia al origen**: Distancia entre la posición actual del vehículo y el punto de origen de la entrega.
- **Tiempo de espera**: Diferencia entre la hora disponible del vehículo y la hora de inicio de la entrega.
- **Entregas asignadas**: Número de entregas que el vehículo ya ha realizado.
- **Uso de capacidad**: Promedio entre la proporción de volumen y peso utilizado.

El vehículo seleccionado debe cumplir con los siguientes requisitos:

- Tipo de vehículo compatible con el requerido por la entrega.
- Suficiente capacidad de volumen y peso.
- Disponibilidad temporal dentro del rango de flexibilidad permitido.

Una vez asignada una entrega a un vehículo:

- Se actualiza la posición del vehículo al destino de la entrega.
- Se reduce su capacidad disponible.
- Se incrementa el contador de entregas asignadas.
- Se acumulan estadísticas como la distancia recorrida y el tiempo de espera.

Finalmente, se genera un informe CSV que resume los datos de cada entrega realizada, incluyendo información de la asignación, distancia total y capacidades restantes de cada vehículo.

3. Optimizaciones

3.1. Balanceo de carga

Una de las optimizaciones aplicadas fue el **balanceo de carga** entre los vehículos. El objetivo de esta mejora fue distribuir equitativamente las entregas entre los conductores disponibles, evitando la sobreasignación a unos pocos y la inutilización del resto.

Esta optimización se implementó mediante la incorporación de un nuevo atributo en la estructura **Vehicle**:

```
typedef struct {
    char id[MAX_ID_LENGTH];
    .
    .
    int deliveries_assigned; // NUEVO: Cantidad de entregas asignadas
} Vehicle;
```

El campo `deliveries_assigned` permite contabilizar cuántas entregas ha realizado cada vehículo, y su valor se utiliza dentro de la función de puntuación del algoritmo **Nearest Neighbor**:

$$\text{score} = \text{distancia al origen} + 0.5 \cdot \text{tiempo de espera} + 5 \cdot \text{entregas asignadas} + 10 \cdot \text{uso de capacidad}$$

La penalización por entregas asignadas incentiva al algoritmo a preferir vehículos que han sido menos utilizados, logrando una distribución más uniforme del trabajo. Esto ayuda a:

- Reducir los tiempos muertos de vehículos poco usados.
- Maximizar la utilización de los vehiculos.
- Evitar la saturación de vehículos individuales.

3.2. Detección de conflictos

La optimización implementada en esta etapa se enfoca en evitar solapamientos temporales al asignar múltiples entregas a un mismo vehículo. Esta estrategia busca asegurar que un vehículo no sea programado para realizar dos entregas en intervalos de tiempo que se intersequen, ya que esto podría provocar inconsistencias en la ejecución del plan logístico y, en última instancia, fallos en el cumplimiento de las entregas.

El objetivo principal de esta optimización es garantizar la coherencia temporal en la planificación de entregas, respetando tanto la disponibilidad del vehículo como las ventanas de tiempo requeridas por cada cliente. Al prevenir asignaciones simultáneas, se evita la sobrecarga de recursos y se asegura una distribución realista y ejecutable.

La detección de conflictos temporales se implementó mediante la función `has_time_conflict`, que se invoca dentro del algoritmo de asignación (`schedule_pb`).

```
bool has_time_conflict(Delivery *deliveries, int n_deliveries, Delivery *current)
{
int current_start = time_to_minutes(current->start);
int current_end = current_start + current->duration;
for (int i = 0; i < n_deliveries; i++)
    if (strcmp(deliveries[i].vehicle_assigned, "") != 0 &&
        strcmp(deliveries[i].vehicle_assigned, current->vehicle_assigned) == 0)
    {
        int assigned_start = time_to_minutes(deliveries[i].start);
        int assigned_end = assigned_start + deliveries[i].duration;
        if ((current_start < assigned_end && current_end > assigned_start))
            return true; // Hay solapamiento
    }
return false;
}
```

Esta función compara la entrega actualmente evaluada (`current`) con todas las entregas previamente asignadas al mismo vehículo. Para cada comparación, convierte los horarios de inicio a minutos y calcula el intervalo de duración de ambas entregas. Luego, se evalúa si existe una intersección entre los intervalos temporales. Si se detecta un solapamiento, la función retorna `true`, indicando un conflicto de tiempo. Este enfoque permite:

- Asegura que ninguna entrega se asigne a un vehículo que ya tenga otra entrega en el mismo intervalo de tiempo.
- Al considerar el tiempo de inicio y duración de cada entrega, se mantiene la integridad del cronograma de rutas.
- Se evita la asignación conflictiva, lo que disminuye la necesidad de reprogramación y reduce la probabilidad de incumplimientos.
- La verificación de conflictos es una condición necesaria para mantener la viabilidad operativa del plan logístico generado.

4. Captura de errores y robustez del sistema

4.1. Casos no manejados

Un caso que actualmente no es manejado por los algoritmos implementados corresponde a las entregas que no pueden ser asignadas a ningún vehículo. Esto ocurre cuando, por restricciones de capacidad, tipo de vehículo o disponibilidad horaria, nigún vehículo de transporte cumple con los requisitos para realizar la entrega solicitada.

Estas entregas quedan sin completar y son reportadas explícitamente en el informe generado por cada estrategia. Sin embargo, no existe un mecanismo posterior que gestione estas entregas pendientes.

En un contexto real de una empresa de envíos, este tipo de situaciones no puede ser ignorado. Cuando una entrega no logra ser concretada durante el día planificado, esta debe ser reprogramada para el día siguiente. Además, debería recibir una mayor prioridad, dado que su incumplimiento impacta directamente en la percepción del servicio por parte del cliente.

Por lo tanto, una mejora al sistema sería implementar una cola de entregas pendientes que se arrastran al siguiente día, aumentando su prioridad para asegurar que sean consideradas con mayor urgencia. Esto permitiría simular de mejor manera la operación real de una empresa de envíos, donde las entregas fallidas no se descartan, sino que se reintentan hasta ser completadas.

4.2. Errores de sistema

Con el objetivo de mejorar la robustez y mantenibilidad del sistema, se implementó un archivo exclusivo (centralizado) para el manejo de errores, denominado `error.c`, ubicado dentro de la estructura de directorios del proyecto. Este archivo contiene una serie de funciones diseñadas para gestionar y reportar adecuadamente los distintos tipos de errores que pueden surgir durante la ejecución del sistema.

Existe una función llamada `fatal_error` que recibe como argumento un mensaje descriptivo del problema y, opcionalmente, información contextual que permite identificar con precisión el origen del fallo. Al ser ejecutada, esta función:

1. Muestra un mensaje de error claro y específico en la terminal.
2. Señala la ubicación más probable del origen del error dentro del flujo del programa.
3. Finaliza inmediatamente la ejecución del sistema para evitar consecuencias indeseadas o resultados inconsistentes.

Este enfoque presenta múltiples ventajas:

- Permite un control uniforme de todos los errores críticos desde un solo punto del sistema.
- Al emitir mensajes detallados, el programador puede identificar rápidamente la causa y el contexto del error.
- La separación de la lógica de errores del resto del programa contribuye a un código más limpio y modular.
- Detener la ejecución ante errores graves evita efectos colaterales o corrupción de datos.

5. Explicación de la entrada y salida

5.1. Entrada del sistema

El sistema permite realizar operaciones previas a la asignación, como la generación de una base de datos de vehículos y repartos, así como la ejecución de los algoritmos de asignación.

```
--- Ayuda: Opciones de Algoritmos ---
Uso: ./build/program.out [opciones]

Opciones disponibles:
  --h, --help                              Muestra este menú de ayuda.
  --db, --database <entregas> <autos>   Crea las bases de datos de manera aleatoria.
  --edf                                 Ejecuta el algoritmo Earliest Deadline First (EDF).
  --pb, --priority                       Ejecuta el algoritmo Priority-Based Scheduling.
  --nn                                  Ejecuta el algoritmo Nearest Neighbor.

Ejemplo:
  ./build/program.out --edf
  ./build/program.out --db 20 10
```

Al crear la base de datos de manera aleatoria, esto genera un CSV con vehículos y entregas aleatorias. Una vez la base de datos es generada, se puede ejecutar uno de los algoritmos, al hacer esto, se leen la base de datos generada y se asignan los datos a las estructuras. Una vez completado esto, el sistema ya tiene los datos cargados, y puede comenazar a realizar las operaciones de comparación, calculo de distancias, asignación de vehiculos, etc.

5.2. Salida del sistema

El sistema cuenta con tres tipos principales de salidas (outputs), diseñadas para que el usuario pueda visualizar, analizar y comprender los resultados de la ejecución de manera efectiva. Estas salidas abarcan desde la terminal, hasta archivos estructurados y representaciones gráficas.

La primera salida se produce directamente en la terminal al finalizar la ejecución del sistema. Esta proporciona al usuario una visión general del proceso realizado, mostrando tanto el estado de los recursos utilizados (vehículos y entregas) como un resumen detallado de los resultados obtenidos.

```
Entregas (15) y Vehiculos (15) generados/cargados con exito.

--- Estrategia: (Estrategia escogida) ---

--- Vehiculos ---
Vehiculo V001: Entregas asignadas: 3
Capacidad restante del vehiculo V001: Volumen = 3.36 m3, Peso = 18.40 kg
...

--- Entregas ---
Entrega D004: Vehiculo Agignado: V001
Requisitos de la Entrega D004: Volumen = 2.58 m3, Peso = 6.29 kg
...

--- Metricas Totales de la Simulación ---
Numero de entregas completadas: 15/15 entregas
Tiempo total de espera: 21.00 hrs
Distancia total recorrida: 436.79 km
Satisfacción del cliente: 3.20/5
Utilización de recursos: 9/10 vehiculos
Tiempo de ejecucion del algoritmo: 0.000050 seg
Litros totales de combustible usados: 72.53 L
Costo total del combustible: $97921 CLP
```

La segunda forma de salida consiste en la generación de archivos CSV, los cuales contienen información estructurada y detallada sobre cada entrega realizada (o no realizada). Estos archivos permiten una revisión más minuciosa y la posibilidad de realizar análisis posteriores o integrar los datos con otras herramientas.

Table 1. informe_entregas_pb.csv

ID Entrega	ID Vehiculo	Litros de Combustible Usados	Distancia Recorrida (KM)	Inicio del Viaje	Fin del Viaje
D006	V001	3.86	21.45	09:00	12:00
D012	V002	4.34	24.10	10:00	12:00
D001	V002	4.68	25.99	10:30	13:00
D008	V004	4.18	27.86	09:00	13:00
D005	V006	6.87	38.19	09:00	13:30
D011	V007	3.61	24.04	08:30	13:30
D002	V007	4.97	33.15	08:00	13:30
D014	V001	6.14	34.12	08:30	13:30
D009	V001	5.58	31.02	09:00	14:00
D004	V001	4.11	22.83	11:00	14:00
D003	V005	6.00	33.35	11:00	14:00
D010	V010	5.61	31.14	11:30	14:30
D007	V005	5.60	31.12	08:00	15:00
D015	V003	3.26	27.19	11:30	15:30
D013	V008	4.58	30.51	11:00	16:00

La tercera y última forma de salida es una visualización gráfica del sistema, desarrollada con Python a partir de los archivos CSV mencionados anteriormente. Esta visualización adopta el formato de una carta Gantt, la cual permite observar de forma clara y ordenada la asignación de entregas a lo largo del tiempo para cada vehículo.

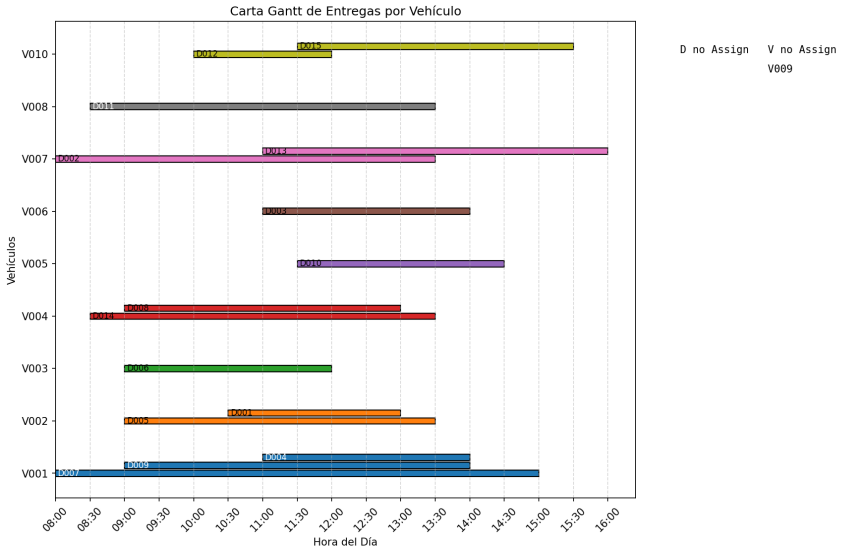

Figure 1. Gráfico del sistema con la estrategia Priority Based

5.3. ¿Cómo funciona el sistema?

El sistema ha sido diseñado para ejecutarse desde la terminal mediante una serie de comandos que permiten tanto la generación de los datos como la aplicación de diferentes estrategias de asignación de entregas. El flujo de ejecución se compone de dos fases principales: generación de datos y ejecución del algoritmo de planificación.

Antes de utilizar cualquier estrategia, es necesario crear una base de datos simulada de entregas y vehículos. Esto se logra ejecutando el siguiente comando en la terminal:

```
./build/program.out --db <número_de_entregas> <número_de_vehículos>
```

Este comando genera de forma aleatoria un conjunto de entregas y vehículos, y los almacena en archivos CSV que luego serán utilizados por el sistema. Estos archivos contienen información como coordenadas de origen y destino, ventanas de tiempo, prioridades, tipo de vehículo requerido, capacidades de los vehículos, entre otros parámetros.

Una vez generados los datos, se puede ejecutar el sistema utilizando una estrategia específica de planificación, por ejemplo:

```
./build/program.out --<algoritmo de planificación>
```

El sistema está diseñado para aceptar distintos modificadores según la estrategia que se desee aplicar (por ejemplo, por prioridad).

Durante la ejecución, el programa realiza las siguientes operaciones:

1. Los archivos CSV generados anteriormente son leídos y sus datos se cargan en estructuras de datos internas del programa: `Vehicle` y `Delivery`.
2. Según el comando ejecutado, el sistema llama a la función encargada de ejecutar la estrategia seleccionada. Cada estrategia aplica distintos criterios de asignación de entregas a vehículos.
3. Se analiza cada entrega en función de diversos criterios (como prioridad, distancia, capacidad, horarios y tipo de vehículo requerido) y se determina el vehículo más adecuado para cada una. Esta asignación se realiza respetando todas las restricciones del sistema (peso, volumen, disponibilidad horaria, etc.).
4. Se toma el tiempo total de ejecución del algoritmo y se calculan diversas métricas, como el número de entregas completadas, distancia total recorrida, combustible consumido, costo operativo, tiempo de espera promedio y nivel de satisfacción del usuario.
5. Finalizada la ejecución, el sistema exporta un archivo CSV con los resultados obtenidos y las entregas asignadas. Este archivo sirve como base para la visualización posterior de los resultados.
6. Utilizando un script en Python, se procesan los datos exportados para crear una representación visual (por ejemplo, una carta Gantt) que permite analizar de forma intuitiva la planificación y distribución de entregas entre los vehículos.

Este flujo de trabajo modular permite no solo una fácil modificación o ampliación del sistema (agregando nuevas estrategias, métricas u optimizaciones), sino también una clara separación entre las etapas de generación de datos, ejecución del algoritmo y análisis de resultados.

6. Análisis

6.1. Earliest Deadline First

El algoritmo `schedule_edf` implementa la estrategia de planificación *Earliest Deadline First (EDF)*, que asigna prioridades a las entregas en función de sus fechas límite: a menor fecha de vencimiento, mayor prioridad. A continuación, se presenta un análisis detallado de su complejidad temporal.

Parámetros

- n: número total de entregas a programar.
- m: número de vehículos disponibles.

1. Ordenamiento de entregas

El primer paso consiste en ordenar las entregas por su fecha de vencimiento. Se utiliza una función de ordenamiento personalizada (`custom_qsort`), cuya complejidad es:

$$T_{\text{orden}}(n) = \begin{cases} O(n^2) & \text{(Peor caso)} \\ O(n \log n) & \text{(Caso promedio y mejor caso)} \end{cases}$$

2. Bucle de asignación

Tras el ordenamiento, el algoritmo itera sobre cada entrega (n) y, para cada una, evalúa todos los vehículos disponibles (m) para determinar el más adecuado. Como las operaciones internas son de tiempo constante, la complejidad de esta sección es:

$$T_{\text{asignación}}(n, m) = O(n \cdot m)$$

3. Operaciones constantes y finales

La inicialización de variables y la captura de tiempos de ejecución son operaciones de tiempo constante: $O(1)$. Las operaciones finales, como imprimir resultados o guardar archivos, dependen linealmente de n y m:

$$T_{\text{final}}(n, m) = O(n + m)$$

Complejidad total

Sumando todas las contribuciones:

$$T(n, m) = T_{\text{orden}}(n) + T_{\text{asignación}}(n, m) + T_{\text{final}}(n, m)$$

- **Peor caso:** $O(n^2 + n \cdot m)$
- **Mejor y caso promedio:** $O(n \log n + n \cdot m)$

6.2. Priority-Based Scheduling

Sea n el número de entregas y m el número de vehículos.

La función `schedule_pb` primero ordena las entregas por prioridad y luego, para cada entrega, busca el mejor vehículo disponible. El costo total del algoritmo es:

$$T(n, m) = T_{\text{sort}}(n) + T_{\text{asignación}}(n, m)$$

donde:

$$T_{\text{sort}}(n) = O(n \log n) \quad \text{(ordenamiento de entregas)}$$

$$T_{\text{asignación}}(n, m) = \sum_{i=1}^{n} O(m) = O(n \cdot m) \quad \text{(búsqueda de vehículo para cada entrega)}$$

Por lo tanto, la complejidad temporal total es:

$$T(n, m) = O(n \log n + nm)$$

Análisis de casos:

- **Mejor caso:**
$$T_{\text{mejor}}(n, m) = O(n \log n + n)$$
Esto ocurre si, para cada entrega, el primer vehículo disponible cumple con todos los requisitos, es decir, la búsqueda de vehículo es $O(1)$ por entrega.
- **Peor caso:**
$$T_{\text{peor}}(n, m) = O(n \log n + nm)$$
Esto ocurre si, para cada entrega, es necesario revisar todos los vehículos para encontrar uno adecuado (o ninguno cumple).
- **Caso promedio:**
$$T_{\text{promedio}}(n, m) = O(n \log n + nk)$$
donde k es el número promedio de vehículos revisados por entrega, con $1 \leq k \leq m$. Si no hay información adicional sobre la distribución de los datos, se asume $k = m/2$, por lo que el caso promedio es también $O(n \log n + nm)$.

¿Opera el algoritmo en el peor caso?

En la práctica, si para la mayoría de las entregas es necesario revisar muchos vehículos para encontrar uno adecuado, el algoritmo opera en el peor caso ($O(n \log n + nm)$). Si normalmente se encuentra un vehículo adecuado rápidamente, el tiempo real será mejor que el peor caso.

6.3. Nearest Neighbor

El algoritmo implementa una estrategia voraz para asignar entregas al vehículo más cercano según distancia y disponibilidad. A continuación se presenta el análisis de su complejidad temporal:

- Sea n el número total de entregas (`n_deliveries`).
- Sea m el número total de vehículos (`n_vehicles`).

6.3.1. Peor Caso

El mayor peso del algoritmo consiste en un bucle externo que itera sobre todas las entregas:

- Dentro de este bucle externo, existe un bucle interno que recorre todos los vehículos para encontrar el vehículo óptimo para cada entrega.
- Para cada combinación de entrega y vehículo, se realizan varias operaciones con complejidad constante $\mathcal{O}(1)$, tales como: verificación de condiciones, cálculo de distancias, evaluación de tiempos y puntuaciones.

Por lo tanto, la complejidad del algoritmo en el peor caso es:

$$\mathcal{O}(n \cdot m)$$

Este comportamiento se debe a que, en el peor caso, para cada una de las n entregas se evalúan los m vehículos disponibles.

6.3.2. Mejor Caso

En el mejor de los casos, cada entrega encuentra rápidamente un vehículo adecuado. Sin embargo, el algoritmo no rompe el bucle interno una vez asignado un vehículo, sino que sigue evaluando todos los vehículos para asegurar que se seleccione el de menor puntuación. Por esta razón, incluso en el mejor caso, el algoritmo sigue evaluando todos los vehículos por cada entrega.

$$\text{Complejidad en el mejor caso: } \mathcal{O}(n \cdot m)$$

donde n es el número de entregas y m el número de vehículos.

6.3.3. Caso Promedio

En promedio, el algoritmo debe evaluar un número considerable de vehículos, el costo promedio es comparable al mejor y peor caso:

$$\text{Complejidad promedio: } \mathcal{O}(n \cdot m)$$

Operaciones posteriores:

Luego del bucle principal, existen otros bucles adicionales que iteran sobre los vehículos y entregas para mostrar resultados y generar reportes. Cada uno de estos tiene complejidad lineal:

$$\mathcal{O}(m) + \mathcal{O}(n)$$

Estas operaciones no afectan la complejidad, por lo que se ve que la complejidad total sigue siendo:

$$\mathcal{O}(n \cdot m)$$

7. Comparación de los algoritmos

7.1. Bencina como métrica de rendimiento

Dado que los algoritmos tienen enfoques diferentes, se decidió utilizar el consumo de combustible (litros de bencina) como métrica para la comparación. A continuación, se resumen los resultados más relevantes de cada estrategia, según la siguiente entrada:

Table 2. Entregas

ID	Origin X	Origin Y	Dest X	Dest Y	Start	End	Duración	Prioridad	Tipo Vehículo	Volumen	Peso
D001	5.00	11.97	25.11	19.17	07:00	16:30	31	5	1	2.56	17.46
D002	5.35	10.80	20.59	23.42	11:00	14:45	36	5	3	3.54	12.90
D003	5.92	19.81	18.32	24.02	09:30	14:45	40	4	2	3.27	19.87
D004	8.65	13.71	29.65	24.45	08:15	12:00	25	5	2	3.30	12.17
D005	7.38	14.12	17.19	13.33	11:00	12:45	26	2	1	1.96	19.80
D006	9.94	13.99	11.81	28.96	10:30	13:45	29	1	2	3.82	5.75
D007	12.62	21.55	26.56	12.51	11:15	14:45	24	5	1	3.67	8.50
D008	8.06	15.27	20.27	21.82	08:00	14:30	47	2	1	1.86	7.67
D009	6.54	18.57	26.05	10.66	11:00	16:45	29	1	1	3.53	7.40
D010	7.13	20.72	12.61	11.82	09:30	15:45	33	4	2	1.54	9.76

Table 3. Vehículos

ID	Tipo	Cap. Volumen	Cap. Peso	Start	End	Pos X	Pos Y	Especialidad
V001	2	9.56	30.26	07:45	17:45	1.48	5.87	1
V002	3	11.69	44.47	08:00	16:00	5.65	2.52	0
V003	3	11.73	29.41	10:30	19:15	6.51	6.22	0
V004	1	8.34	38.63	09:30	20:00	1.42	9.47	0
V005	3	11.20	28.23	09:00	16:30	1.35	7.83	1
V006	2	8.17	53.31	10:45	16:00	9.09	2.50	0
V007	3	8.54	46.01	10:15	20:30	6.33	4.39	0
V008	1	9.92	59.55	07:00	16:45	5.14	1.03	1
V009	1	11.14	40.40	09:30	17:00	7.07	7.42	1
V010	2	8.67	41.22	07:45	20:30	6.82	2.00	1

- **Nearest Neighbor (NN):**
  - Litros totales de combustible usados: 47.49 L
  - Distancia total recorrida: 275.89 km
  - Tiempo total de espera: 2.25 hrs
  - Costo total del combustible: $64,112 CLP
- **Priority-Based (PB):**
  - Litros totales de combustible usados: 44.41 L
  - Distancia total recorrida: 261.43 km
  - Tiempo total de espera: 3.50 hrs
  - Costo total del combustible: $59,960 CLP
- **Earliest Deadline First (EDF):**
  - Litros totales de combustible usados: 42.02 L
  - Distancia total recorrida: 263.51 km
  - Tiempo total de espera: 6.50 hrs
  - Costo total del combustible: $56,724 CLP

Análisis

Como se observa, el algoritmo EDF logra el menor consumo de combustible (42.02 L), seguido por PB (44.41 L) y finalmente NN (47.49 L). Esto muestra que, desde un punto de vista de eficiencia energética y economica, **EDF es el algoritmo más eficiente en términos de bencina**.

Sin embargo, esta eficiencia tiene un costo: el **tiempo total de espera** con EDF fue de 6.50 horas, significativamente mayor que en las otras estrategias. Esto indica que EDF prioriza entregas con plazos más cortos, pero sin necesariamente minimizar el recorrido total.

En contraste, NN tiene el menor tiempo de espera (2.25 hrs) pero es el más ineficiente en consumo de combustible y distancia recorrida, lo cual es esperable ya que su enfoque está centrado únicamente en la cercanía geográfica y no en la optimización del uso de recursos.

Por lo tanto, la elección del algoritmo dependerá del objetivo de la empresa: si se prioriza el cumplimiento rápido de entregas, NN puede ser adecuado; si se prioriza el ahorro de combustible y costos, EDF resulta más conveniente; y PB ofrece un equilibrio intermedio entre ambas elecciones.

8. Dificultades enfrentadas

8.1. Generación de gráficos

Desde las etapas iniciales del desarrollo del sistema, se contempló la posibilidad de generar gráficos tipo carta Gantt para visualizar de manera clara y estructurada la asignación de entregas según el algoritmo de planificación utilizado. La intención era implementar esta funcionalidad directamente en C utilizando la biblioteca Gnuplot, conocida por su capacidad para generar gráficos a partir de datos. Sin embargo, a medida que se avanzó en la implementación, surgieron varias dificultades técnicas.

Gnuplot está principalmente orientado a la creación de gráficos de tipos convencionales que utilizan un sistema de coordenadas con eje X e Y. Si bien es una herramienta poderosa para representar series temporales o datos de rendimiento, crear una carta Gantt no resultó tan directo. La manipulación de los estilos de línea, las etiquetas, y la alineación precisa de los bloques temporales resultó ser compleja y poco intuitiva dentro del flujo de programación en C.

Frente a estas limitaciones, se optó por una solución alternativa: procesar los datos en formato CSV desde C y posteriormente utilizar Python para generar los gráficos. Python, con bibliotecas como Matplotlib y Plotly, permite crear gráficos tipo Gantt de manera mucho más sencilla, flexible y visualmente clara. Estas herramientas ofrecen funciones específicas para representar tareas en intervalos de tiempo, categorizarlas, ajustar colores y etiquetas con gran facilidad, lo que reduce significativamente la complejidad del desarrollo y mejora la calidad visual del resultado.

8.2. Comparación entre algoritmos

Comparar de forma equitativa los algoritmos implementados fue un desafío, ya que, si bien todos buscan resolver el mismo problema, lo hacen mediante enfoques metodológicos distintos. En un inicio, no fue evidente cómo establecer un criterio común que permitiera una evaluación justa en el desempeño.

Tras analizar distintos escenarios y métricas, se concluyó que una de las formas más efectivas de comparación es a través del consumo de combustible. Como los vehículos deben completar entregas, el consumo de bencina se convierte en un recurso medible del esfuerzo de cada algoritmo. Evaluar cuánta bencina utilizan para completar las tareas permite inferir no solo la eficiencia, sino también el costo asociado en términos de dinero.

Es importante destacar que existen otras métricas válidas para la comparación, tales como el tiempo total de ejecución, el número de paquetes entregados, el tiempo "perdido" de los vehículos, entre otros. En este trabajo, el foco se ha puesto principalmente en el consumo de combustible, lo cual también puede interpretarse como una medida indirecta del costo económico.

8.3. Satisfacción del cliente

Se planteó la necesidad de incorporar una métrica que permitiera estimar la satisfacción del cliente en función del tiempo de espera asociado a la entrega, considerando la prioridad asignada a cada pedido. Sin embargo, en las etapas iniciales del desarrollo, esta tarea representó un desafío, ya que las soluciones disponibles eran excesivamente complejas para el alcance y los tiempos acotados del proyecto.

Como alternativa práctica y eficiente, se optó por implementar un modelo de satisfacción basado en rangos de tiempo de espera máximos, diferenciados según la prioridad de cada entrega. Es decir, a mayor prioridad (por ejemplo, prioridad 1), más exigente es el umbral de tiempo para mantener una alta satisfacción.

El sistema funciona de la siguiente manera: si una entrega de prioridad 1 es realizada en menos de 20 minutos desde la disponibilidad del vehículo, se le asigna una satisfacción máxima (5 puntos). Si el tiempo de espera aumenta, el puntaje disminuye progresivamente según tramos predefinidos, hasta llegar a un mínimo de 1 punto si se excede el límite aceptable. Este esquema se replica para cada nivel de prioridad, con valores ajustados a sus respectivas urgencias.

Finalmente, se calcula el promedio de satisfacción de todas las entregas completadas, obteniendo así un indicador global, expresado como un valor flotante entre 1 y 5, que permite evaluar el desempeño del sistema desde la perspectiva del usuario final.

9. Conclusiones

El presente informe ha demostrado la viabilidad y flexibilidad de un sistema de asignación de entregas en C, apoyado en tres algoritmos voraces (Earliest Deadline First, Priority-Based y Nearest Neighbor), para optimizar distintos criterios de operación en una empresa de logística. A partir de los objetivos planteados, implementar, comparar rendimiento, integrar mecanismos de balanceo de carga y detección de conflictos, y generar salidas tanto textuales (CSV) como gráficas (Gantt), se logró:

- Implementación robusta de los tres métodos, cada uno adecuado para priorizar cumplimiento de plazos (EDF), prioridad (PB) o reducción de tiempo de espera (NN).
- Módulos de optimización que equilibran la carga de trabajo y evitan solapamientos temporales, garantizando rutas factibles para cada vehículo.
- Generación de reportes claros y comparables, como: métricas de consumo, distancia recorrida, tiempos de espera y satisfacción del cliente, así como diagramas Gantt que facilitan la visualización temporal de las rutas.

Los resultados muestran que no existe un “mejor” algoritmo en sentido absoluto, sino que la elección debe alinearse con la necesidad que se requiera cubrir. El sistema propuesto, además de cumplir los objetivos, ofrece un equilibrio eficiente tanto en recursos de cómputo como en rendimiento operativo. En conjunto, este desarrollo sienta las bases para una solución de reparto ágil, configurable y cuantificable, capaz de adaptarse a distintos escenarios logísticos.