

**Nombre del curso:**

Programación paralela(vienes-sábados)

**Título del proyecto:**

SmartRouetOptimizer.

**Integrantes del equipo:**

Jheinel Brown 20240017

José Antonio colon peña 20231137

**Nombre del líder:**

Joan Reynoso zayas 20231131

**Fecha de entrega:**

22-8-2025

**Índice**

[**Introducción** 3](#_Toc206777415)

[**Descripción del Problema** 4](#_Toc206777416)

[**Cumplimiento de los Requisitos del Proyecto** 5](#_Toc206777417)

[**Diseño de la Solución** 7](#_Toc206777418)

[**Implementación Técnica** 10](#_Toc206777419)

[**Evaluación de Desempeño** 22](#_Toc206777420)

[**Trabajo en Equipo** 24](#_Toc206777421)

[**Conclusiones** 26](#_Toc206777422)

# **Introducción**

El proyecto SmartRouteOptimizer consiste en el diseño y desarrollo de un programa que ayude a la optimización de rutas de entrega en la última milla (last-mile delivery), aplicando programación paralela y heurísticas de búsqueda. El propósito de dicho proyecto es explorar múltiples combinaciones de rutas y horarios de forma simultánea, permitiendo encontrar soluciones más rápidas y eficientes que los métodos tradicionales.

Ahora bien, **¿Por qué elegir este tema?,** Las grandes compañías como Amazon, FedEx y Uber Eats, presenta un gran problema por el tráfico, la dispersión geográfica de los clientes y sobre todo la necesidad de cumplir con tiempos de entrega cada vez más cortos.

Nuestro **objetivo general** es desarrollar un sistema de optimización de rutas de entrega que utilice programación paralela para mejorar la eficiencia y así poder explorar múltiples combinaciones de rutas y en encontrar las más eficiente. Lo que se traduce a ventajas, como reducir tiempos de entrega y costos operativos.

**Objetivos Específicos:**

* Implementar programación paralela, para la exploración simultánea de diferentes rutas.
* Integrar heurísticas de optimización, como algoritmos greedy y algoritmos genéticos en el sistema.
* Mejorar el rendimiento de búsquedas de diferentes rutas y así poder reducir los tiempos de entrega y los costó operativo.
* Diseñar mecanismos de sincronización para compartir información global entre tareas.

# **Descripción del Problema**

Hoy en día la logística moderna enfrenta uno de sus mayores desafíos en la última milla de entrega o (last-mile delivery), que abarca el trayecto final desde un centro de distribución hasta el cliente final. Este segmento es altamente costoso y complejo debido a factores como: La dispersión geográfica de los clientes, el tráfico vehicular y las limitaciones de tiempo y capacidad de los vehículos.

Por lo que se estima que la última milla representa entre el 40% y 55% de los costos totales de distribución, esto hace que se convierta en un enfoque y necesidad muy clave para las empresas que dependen de las entregas directas al consumidor.

**Ejemplo del problema:**

Imaginemos una empresa de reparto como Uber Eats en una ciudad congestionada, como santo domingo. tomando en cuenta que cada repartidor debe entregar pedidos en diferentes zonas, con tiempos de entrega muy cortos (30-40 minutos), la empresa necesita asignar la mejor ruta posible a cada repartidor.

Ahora bien, si la asignación de rutas no es eficiente, los clientes reciben sus pedidos tarde, generando insatisfacción y la insatisfacción genera pérdidas económicas y reducción en la fidelidad al servicio.

De forma muy similar, sucede con las empresas como Amazon o FedEx requieren entregar miles de paquetes diarios en zonas urbanas, donde un mal diseño de rutas genera costos excesivos de combustible, mayor tiempo en tránsito y uso ineficiente de los recursos logísticos.

# **Cumplimiento de los Requisitos del Proyecto**

**1. Ejecución simultánea de múltiples tareas**

El sistema está diseñado para que utilice varios hilos en paralelo, cada uno explorando una combinación de rutas o aplicando una heurística distinta (por ejemplo: greedy, genéticos). Esto permite evaluar muchas alternativas al mismo tiempo lo que ayuda a buscar una solución eficiente.

**2. Necesidad de compartir datos entre tareas**

Las tareas paralelas requieren acceder a información común, como:

* OptimizationSession: se comparten datos entre todos los hilos.
* Evaluaciones: Es el contador global que actualiza todos los algoritmos.
* Algoritmos: lista de estados de cada heurística, protegida con lock para evitar inconsistencias.
* Datos: Los datos como: clientes, vehículos, etc. Son informaciones accesibles por todos los algoritmos en ejecución

El proyecto contiene el uso de mecanismos de sincronización (locks, barreras) que garantizan la coherencia de los datos compartidos, evitando conflictos durante la ejecución y que cada algoritmo contribuye al progreso global.

**3. Exploración de diferentes estrategias de paralelización**

SmartRouteOptimizer permite experimentar con distintos enfoques:

* **Por heurística:** cada hilo ejecuta un algoritmo diferente y se comparan resultados.
* **Por zona geográfica:** cada hilo gestiona las entregas en un sector de la ciudad.
* **Por rutas candidatas:** se generan múltiples rutas aleatorias en paralelo y se seleccionan las más prometedoras.

Esto brinda flexibilidad y permite analizar qué enfoque resulta más eficiente según el tamaño del problema y las condiciones del entorno.

**4. Escalabilidad con más recursos**

El sistema está diseñado para **aprovechar la capacidad del hardware**. Si se añaden más hilos o núcleos, el sistema puede procesar más rutas simultáneamente. Lo que reduce aún más el tiempo de cálculo y mejora la calidad de las soluciones encontradas.

**5. Métricas de evaluación del rendimiento**

Para validar la eficiencia y resultados del sistema, se incluyeron las siguientes métricas como:

* **Tiempo de ejecución:** mide el tiempo total transcurrido desde el inicio de la optimización.
* **Progreso computacional:** contador global de escenarios explorados por todos los algoritmos.
* **Progreso relativo:** porcentaje de avance global respecto al objetivo definido.
* **Costo por algoritmo:** mide el costo de la mejor ruta encontrada por cada heurística.
* **Eficiencia:** porcentaje de eficiencia alcanzada respecto al objetivo óptimo esperado.
* **Métricas agregadas:** costo del promedio por entrega considerando todas las rutas evaluadas.

Estas métricas permiten medir objetivamente la calidad y eficiencia de las soluciones generadas.

**6. Aplicación a un problema del mundo real**

El sistema está orientado a casos prácticos de logística y transporte como:

* **Amazon o FedEx:** optimización de miles de paquetes en zonas urbanas.
* **Uber Eats o Rappi:** entrega de pedidos de comida con restricciones de tiempo muy ajustadas.
* **Empresas locales de mensajería:** reducción de costos operativos y mejora en tiempos de entrega.

# **Diseño de la Solución**

**1-Arquitectura general del sistema:**

El proyecto SmartRouteOptimizer se organiza en una arquitectura de n-capas orientada a la ejecución paralela de algoritmos de optimización. Sus principales capas son:

**Frontend Layer:**

* Interfaz de usuario construida con HTML5, con un componente de mapas (Leaflet) para la visualización de rutas. También cuenta con un monitor de progreso en tiempo real y una vista de resultados finales donde se presentan las rutas optimizadas.

**API Layer:**

* Es el controlador principal que expone los servicios REST:
  + POST /start: inicia la optimización.
  + GET /status/{id}: consulta el progreso de una sesión activa.
  + GET /result/{id}: obtiene los resultados finales.

**Service Layer:**

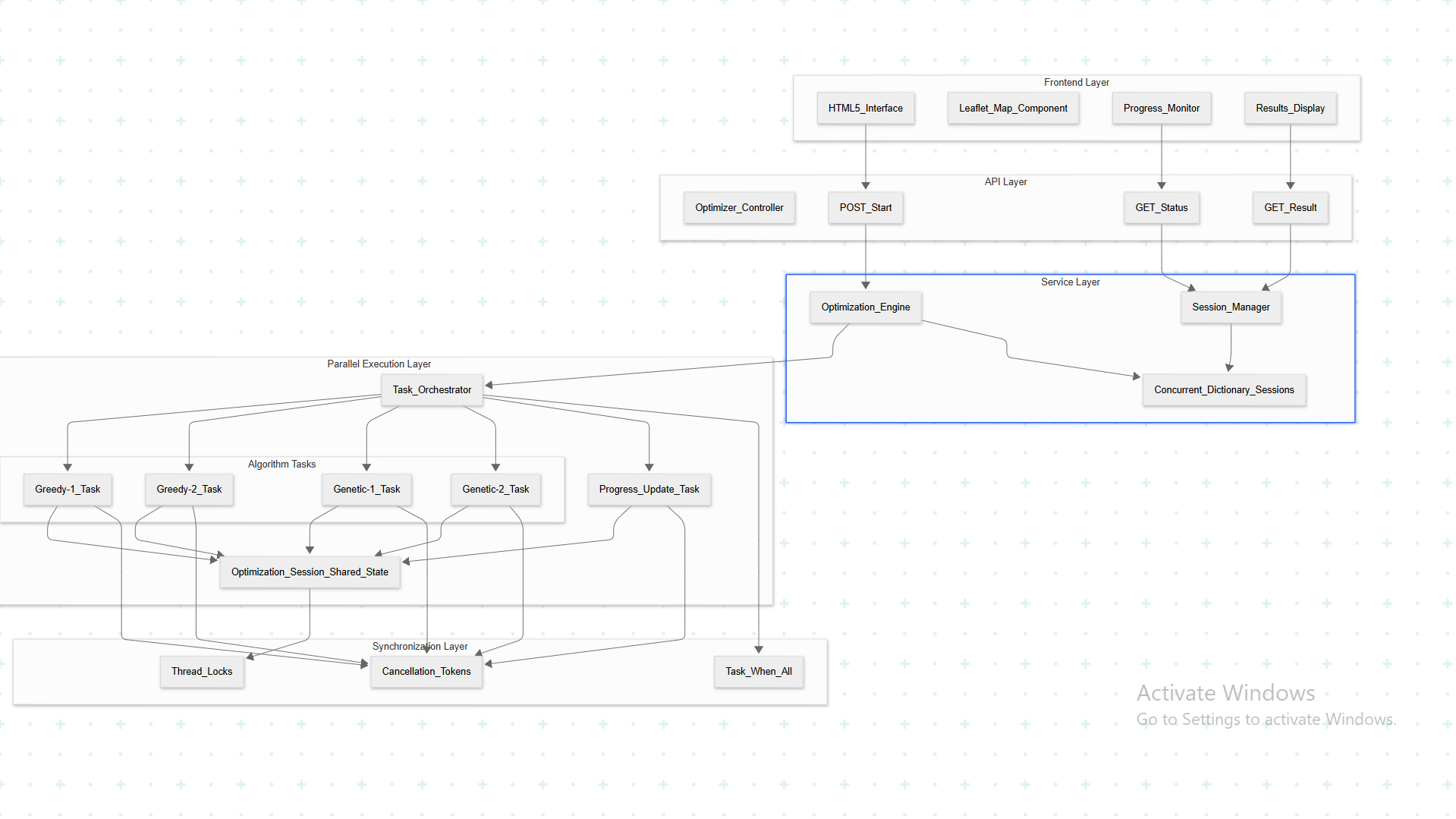
* OptimizationEngine: motor central que coordina la ejecución de la optimización.
* Session Manager: gestiona las sesiones activas de optimización.
* ConcurrentDictionary Sessions: almacena de forma segura y concurrente todas las sesiones en curso.

**Parallel Execution Layer:**

* Task Orchestrator: organiza y ejecuta múltiples algoritmos en paralelo.
* Conjunto de tareas de optimización (Greedy-1, Greedy-2, Genetic-1, Genetic-2) que exploran distintas estrategias.
* Progress Update Task: actualiza periódicamente el estado global.
* Shared State (OptimizationSession): almacén compartido donde todas las tareas reportan resultados.

**Synchronization Layer:**

* Locks: aseguran acceso seguro al estado compartido.
* Cancellation Tokens: permiten abortar las tareas cuando sea necesario.
* Task.WhenAll(): sincroniza la finalización de todas las tareas paralela

**2-Diagrama de componentes / tareas paralelas:**

**3-Estrategia de Paralelización**

* Se aplica un paralelismo heterogéneo, es decir que no tan solo se trata simplemente de correr muchas tareas en paralelo, sino de que cada tarea puede tener una naturaleza diferente, lo que nos permite una exploración más rica y diversa del espacio de soluciones. También cuenta con distintos algoritmos (Greedy, Genético) se ejecutan de forma independiente, pero comparten un estado común.
* Se utilizan tareas paralelas (Tasks de TPL en C#), coordinadas por el Orchestrator, que lanza y supervisa la ejecución de dichas tareas.
* El estado compartido(OptimizationSession) permite acumular métricas y progreso global, asegurando la consistencia mediante “Locks”.
* Se emplea una sincronización final con Task.WhenAll(), para garantizar que todas las tareas finalicen antes de consolidar los resultados.

**4-Herramientas y Tecnologías Empleadas:**

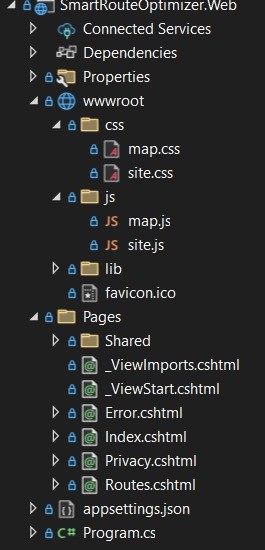
* **Lenguaje:** C# (.NET 8).
* **Librería:** TPL (Task Parallel Library): gestión de tareas, sincronización y cancelación.
* **Componente:** Leaflet.js, para la visualización de rutas en mapas interactivos.
* **Framework:** ASP.NET Web API: comunicación entre frontend y motor de optimización.
* **Tecnología:** HTML5/JavaScript, Permite construir una UI moderna, responsive y compatible con múltiples dispositivos.
* **Locks y CancellationTokens:** control de concurrencia y cancelación de tareas.

# **Implementación Técnica**

**1- Descripción de la estructura del proyecto:**

La implementación del sistema se divide en dos capas principales, está el Backend que es el motor de optimización junta con la API y Frontend con la interfaz web que ayuda a visualizar y consumir los resultados. Esta separación permite mantener una arquitectura limpia, escalable.

**Frontend-SmartRouteOptimizer.Web:**

**wwwroot/css y wwwroot/js:**

* Aquí están los estilos (map.css, site.css) y la lógica cliente (map.js, site.js).
* map.j: interactúa con **Leaflet.js** para renderizar el mapa.

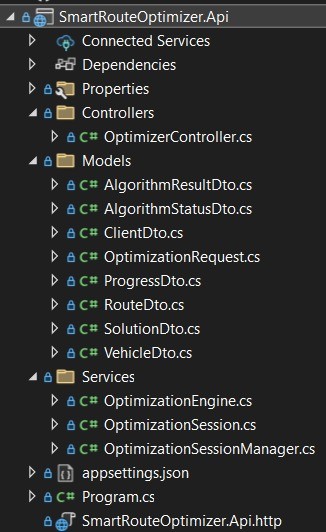
**Pages/** **Index.cshtml:**

* Index.cshtml: maneja la lógica general de UI ( botones, mapa etc.).
* También desde Index.cshtml se hacen llamadas al backend atraves de la api.

**Pages/Shared:**

Contiene \_ViewImports.cshtml y \_ViewStart.cshtml para compartir configuraciones y layout.

**Backend-** **SmartRouteOptimizer.Api:**

**Controllers**

***OptimizerController.cs*:**

* Expone los endpoints de la API. También maneja solicitudes de optimización de rutas (POST /optimize, por ejemplo).
* Se comunica con la capa de servicios para ejecutar los algoritmos de optimización.

**Models**

Contiene los Data Transfer Objects o mejor dicho (DTOs).

-**OptimizationRequest.cs** : Representa la petición inicial: clientes, vehículos, restricciones.

-**ClientDto.cs:** contiene información de clientes como ubicación, demanda, entre otros.

-VehicleDto.cs: contiene la información de vehículos

**-AlgorithmStatusDto.cs:** Estado actual de la ejecución.

**-ProgressDto.cs:** Muestra el progreso de la optimización.

**-AlgorithmResultDto.cs:** Resultado bruto de la optimización como el tiempos, distancias y costos.

**-RouteDto.cs:** Representación de una ruta optimizada.

**-SolutionDto.cs:** Solución completa: conjunto de rutas + métricas de desempeño.

**Services**

Aquí vive la lógica de negocio y el motor de optimización.

* **OptimizationEngine.cs:**

Es el nucleó del sistema donde se encuentra la lógica de paralelización de los algoritmos.

* **OptimizationSession.cs:**

Es la unidad de ejecución que coordina el progreso, resultados y métricas. Cada sesión se identifica mediante un Guid.

* **OptimizationSessionManager.cs:**

Actúa como el gestor de sesiones activas dentro del sistema. Su función principal es permitir la creación, almacenamiento y recuperación de sesiones de optimización, cada una identificada por un Guid único. Esto permite que múltiples usuarios o procesos puedan lanzar optimizaciones simultáneas sin interferencia.

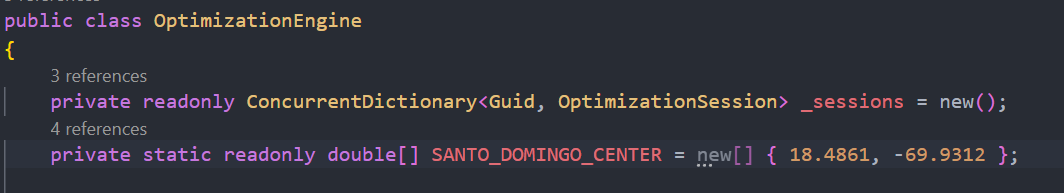
**Archivo principal**

* **Program.cs:**

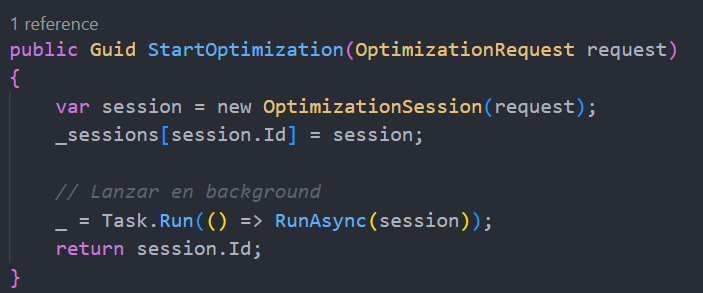
Este archivo configura los servicios como los middlewares y el pipeline de ejecución de la API. Es el punto de entrada de la aplicación ASP.NET Core.

**2-Explicación del código clave:**

La parte más fundamental del código es **OptimizationEngine.cs**, es el núcleo de la lógica de negocio. Es como el motor de optimización paralelo, responsable de coordinar los algoritmos, el progreso y la construcción de la solución final.

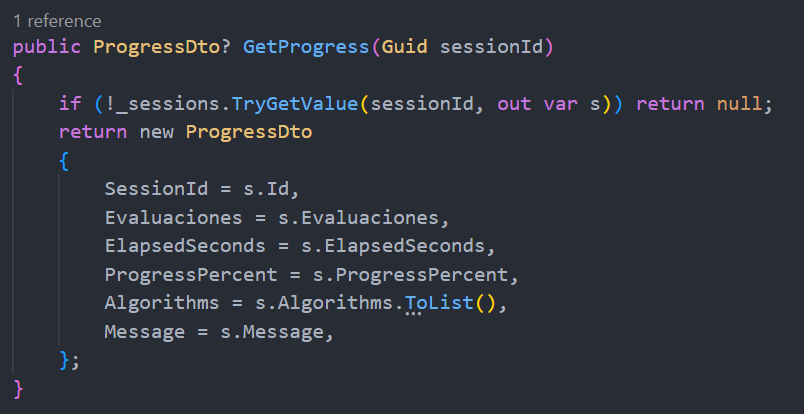


Se utiliza un ConcurrentDictionary para almacenar sesiones activas. Esto asegura acceso seguro en escenarios concurrentes, donde múltiples usuarios pueden lanzar optimizaciones en paralelo.

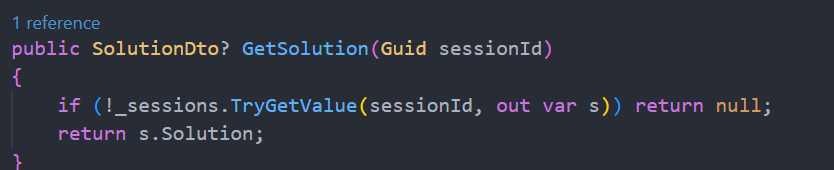


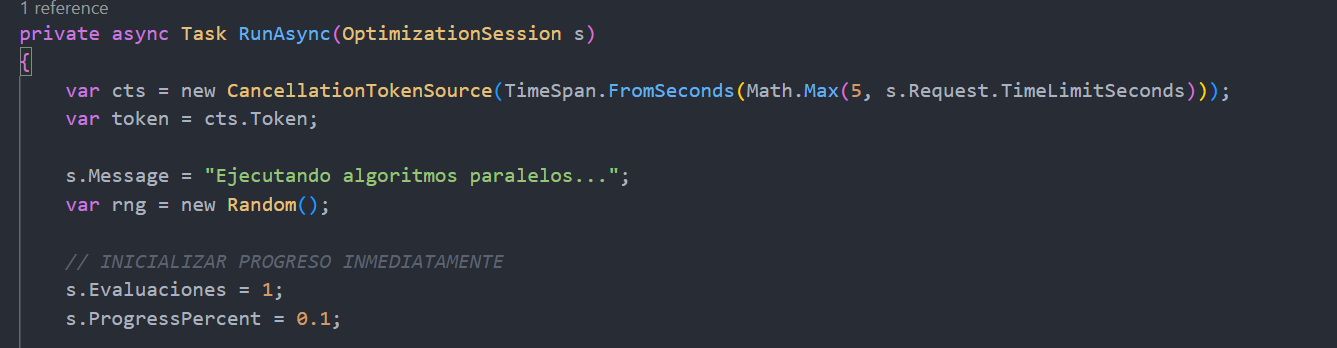
Se crea una nueva sesión de optimización que proviene de OptimizationSession.

Se ejecuta en background con Task.Run para no bloquear el hilo principal. Luego Devuelve un identificador único que permitirá consultar progreso y solución más adelante.



Este pedazo del código devuelve un objeto de progreso con las evaluaciones realizadas, tiempo transcurrido, porcentaje de progreso, estados de algoritmos y mensajes de estado. Ahora bien, si la sesión no existe retorna null.



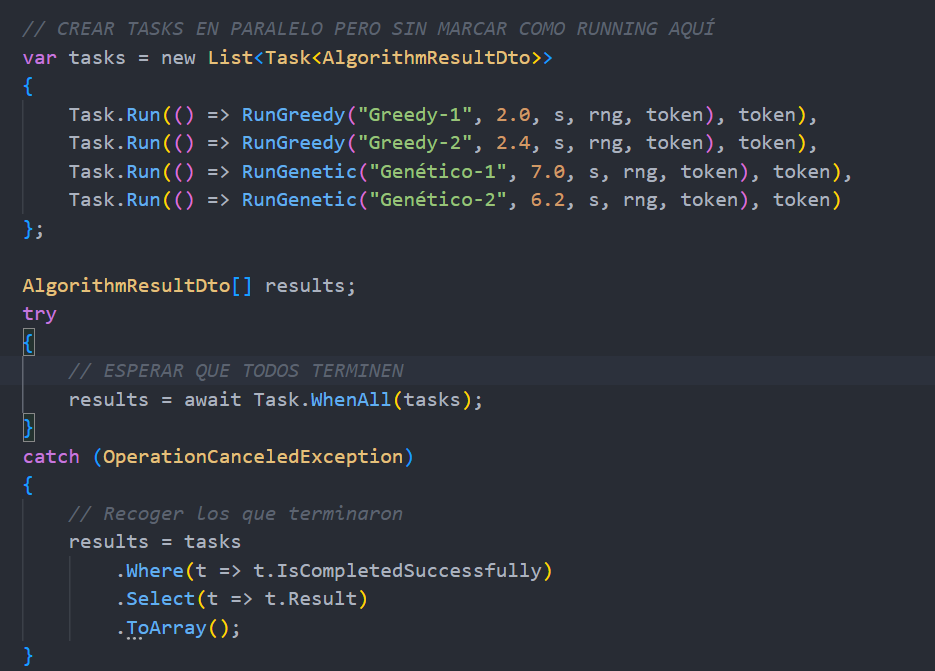
Lo que hace es nos permite consultar la solución final de la optimización. Devuelve null si aún no existe solución (o la sesión no existe).

Se crea un CancellationTokenSource con un tiempo máximo de ejecución (TimeLimitSeconds). Este token se pasa a todas las tareas paralelas. Ahora bien, Si el tiempo se agota, todas las tareas se cancelan automáticamente.



En este pedazo de código se lanza como una tarea(Task) separada del resto de los algoritmos (Greedy, Genético). No realiza optimización directa, sino que simula el avance del sistema para mantener informada a la interfaz.

Aumenta el contador global de evaluaciones (s.Evaluaciones), que representa el trabajo computacional acumulado. Incrementa el porcentaje de progreso (s.ProgressPercent), simulando que el sistema está explorando más rutas.

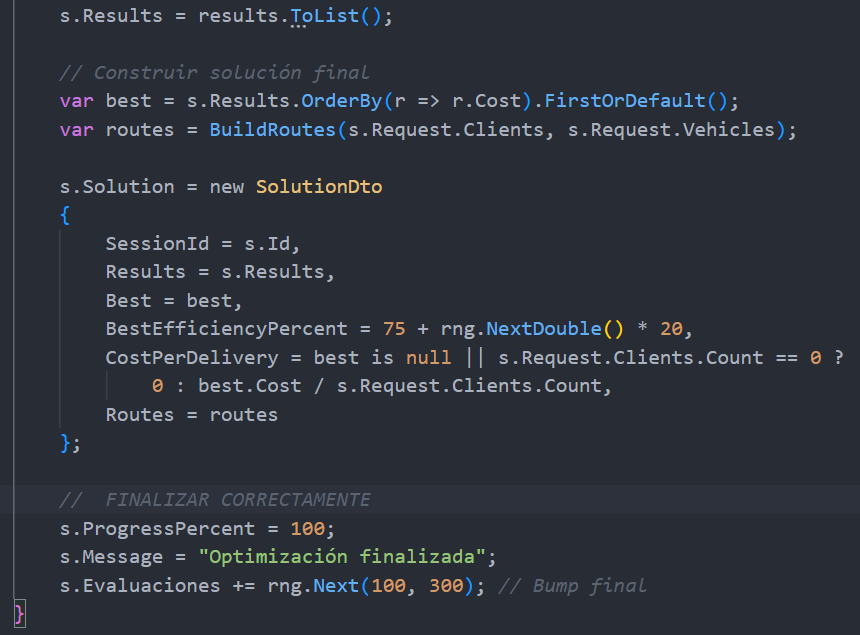


Aquí, se lanza los algoritmos en paralelo, se crean 4 tareas independientes:

* Dos variantes del algoritmo Greedy.
* Dos variantes del algoritmo Genético.

Cada tarea se ejecuta en paralelo usando Task.Run(...), lo que permite aprovechar múltiples núcleos del procesador. Esto permite explorar simultáneamente distintas estrategias de optimización para el problema de la última milla.

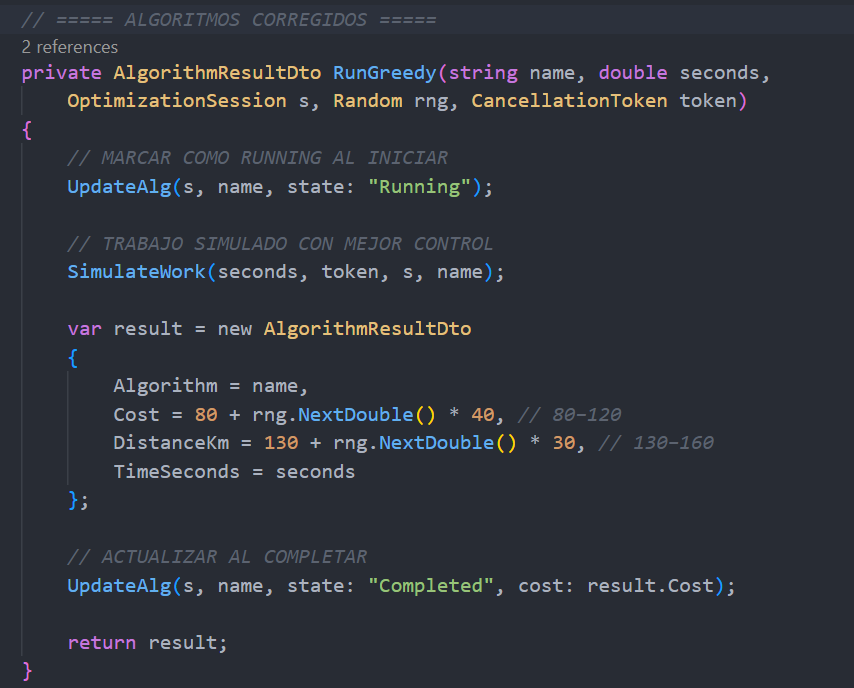
Con el Task.WhenAll(tasks) bloquea hasta que todas las tareas finalicen. Es un mecanismo de sincronización que garantiza que el sistema no continúe hasta que se hayan completado todos los algoritmos.

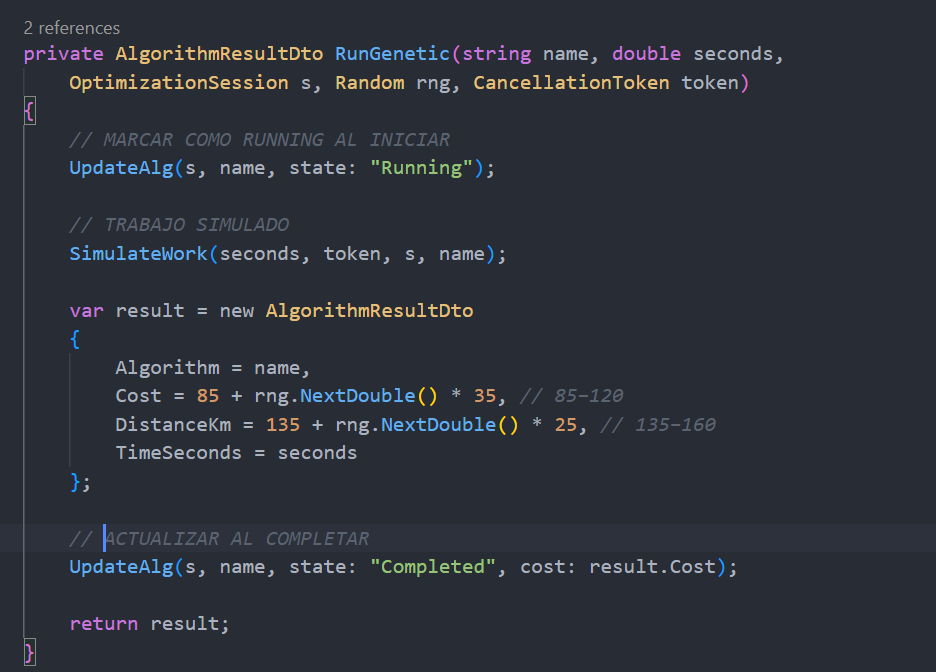


En este pedazo de código se consolidan los resultados de los algoritmos ejecutados en paralelo (Greedy y Genético) y se ordenan por costo total de entrega y se selecciona el más eficiente.

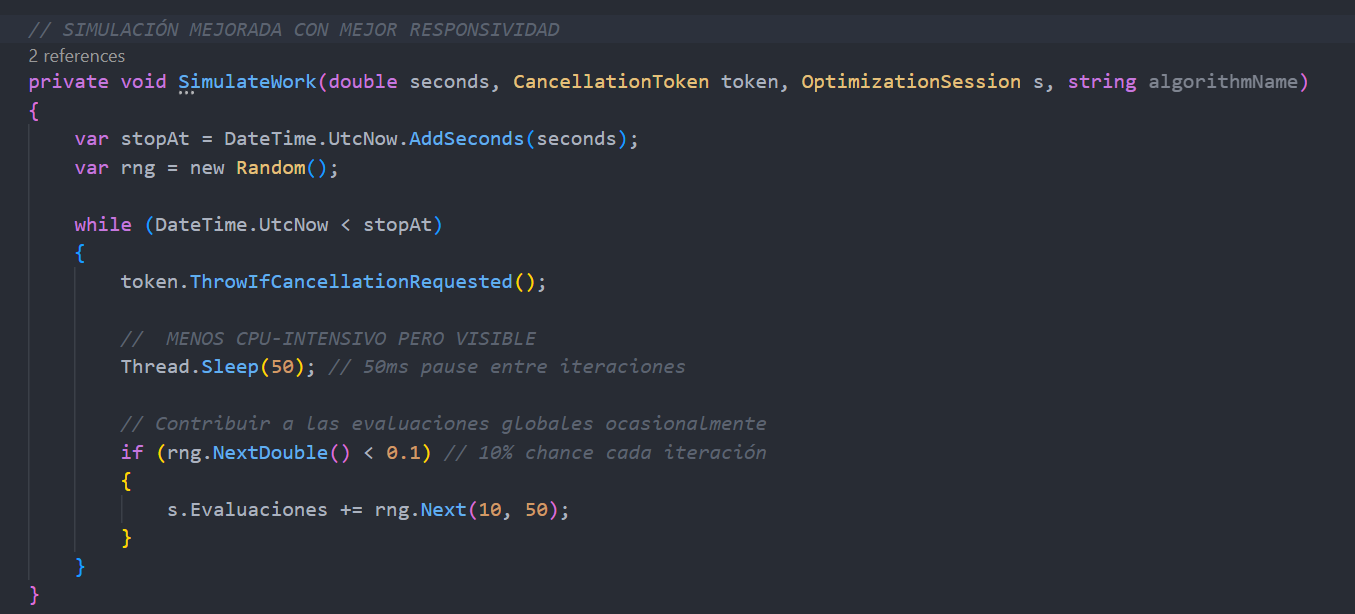
Luego se construyen las rutas de entrega y Se asignan clientes a vehículos de forma balanceada y también se generan rutas con coordenadas desde el centro logístico (Santo Domingo) hacia los clientes y de regreso. Cada ruta incluye estimación de distancia y número de entregas.

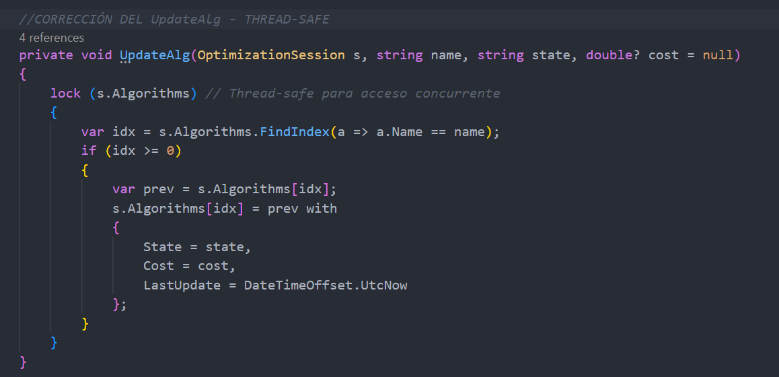
Por último se salcula eficiencia y costo promedio por entrega, se obtiene el costo promedio por entrega, dividiendo el costo total entre el número de clientes. Ojo estos valores alimentan el módulo de métricas y permiten comparar estrategias.



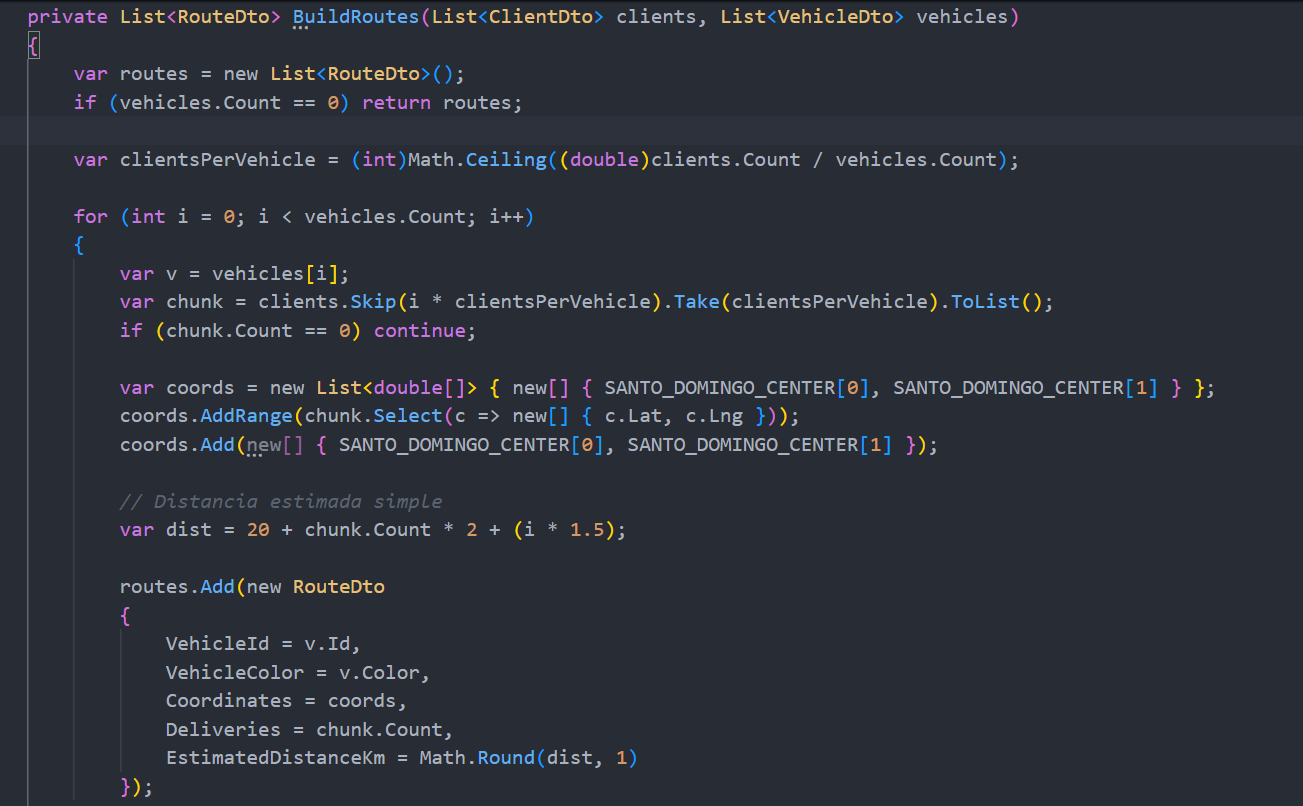


En este se marcan los algoritmos, antes de iniciar el trabajo, el algoritmo actualiza su estado a "Running" dentro de la lista compartida s.Algorithms. Esta operación se realiza dentro de un bloque lock para garantizar acceso thread-safe, evitando condiciones de carrera.

Luego se ejecuta una función que simula la carga computacional del algoritmo durante un tiempo determinado (seconds). internamente, se usa Thread.Sleep(50) para evitar consumo excesivo de CPU y se incrementa ocasionalmente el contador global s.Evaluaciones.

Esta parte simula un algoritmo “trabajando” durante segundos. Usa Thread.Sleep(50) para no consumir CPU innecesaria. Ocasionalmente incrementa evaluaciones globales.

Localiza el algoritmo por nombre dentro de la lista compartida. Actualiza su estado (Running, Completed, etc.), su costo si aplica, y el timestamp. Protege la operación con lock, asegurando que solo un hilo pueda modificar la lista en ese momento.



1-Divide los clientes entre vehículos

Se calcula cuántos clientes debe atender cada vehículo usando una división equitativa (Math.Ceiling).

Luego, se asignan bloques de clientes a cada vehículo y se crean coordenadas geográficas. Ojo Cada ruta comienza y termina en el centro de Santo Domingo (SANTO\_DOMINGO\_CENTER).

Se agregan las coordenadas de los clientes asignados en orden y se calcula una distancia estimada para esto se usa una fórmula simple: 20 + chunk.Count \* 2 + (i \* 1.5) como aproximación.

Por último, se devuelve una lista de rutas Cada ruta incluye: ID del vehículo, color, coordenadas, número de entregas y distancia estimada. Esta lista es la que se usa en el frontend para pintar las rutas en el mapa.

**3-Uso de mecanismos de sincronización**

En el proyecto se utilizaron varias tareas que ejecutan algoritmos de manera simultáneamente y comparten información crítica. Para garantizar la consistencia de datos y evitar rase conditions, se utilizan los siguientes mecanismos:

**Locks (Bloqueos):**

* Los bloqueos garantizan que solo un hilo pueda modificar la lista de algoritmos a la vez y también evita inconsistencias en la actualización de estado (Running, Completed) y costo del algoritmo.

**Cancellation Tokens:**

* Permite detener la ejecución de todas las tareas simultáneamente si se alcanza el límite de tiempo o el usuario cancela la optimización y evita que los hilos sigan consumiendo CPU innecesariamente.

**Coordinación de Tareas:**

* Para la coordinación se utilizó task.whenAll para esperar que todas las tareas finalicen antes de consolidar resultados. También ayuda a garantizar que la sesión solo se marque como finalizada cuando todos los algoritmos han completado o han sido cancelados correctamente.

**Concurrent Collections:**

* Almacena sesiones de forma thread-safe, permitiendo múltiples accesos simultáneos desde API y tareas en segundo plano.

**4-Justificación técnica de las decisiones tomadas:**

**Task Parallel Library (TPL):**

* Elegimos esta biblioteca porque facilita la ejecución de varias tareas al mismo tiempo y tiene soporte incorporado para cancelar tareas y sincronizar procesos. También permite hacer que el sistema escale fácilmente añadiendo más hilos o utilizando más núcleos, sin necesidad de cambiar toda la estructura del programa.

**Locks en estructuras críticas:**

* Son necesarios para poder actualizar listas compartidas (como los Algorithms) de manera segura y evitar problemas de concurrencia. También son una opción más simple y eficiente que otros mecanismos complejos, especialmente útil cuando se necesitan hacer cambios rápidos y frecuentes.

**Cancellation Tokens:**

* Lo elegimos porque ofrecen un control preciso para detener tareas si es necesario, ayudando a que el programa no use recursos innecesariamente y sea más responsivo. Permiten parar en tiempo real una tarea en proceso sin afectar las otras tareas que se están ejecutando simultáneamente.

**ConcurrentDictionary para sesiones:**

* Nos permite guardar y acceder a varias sesiones al mismo tiempo desde diferentes hilos, sin complicaciones. Otro punto es que evita tener que hacer gestión manual de locks cada vez que se entra o sale de una sesión.

**Task.WhenAll para la coordinación final:**

* La elegimos porque nos ayuda Asegurar que los resultados solo se procesen después de que todos los algoritmos hayan terminado o sido cancelados. Previene errores por datos inconsistentes y garantiza que el resultado final refleje correctamente toda la información de los algoritmos.

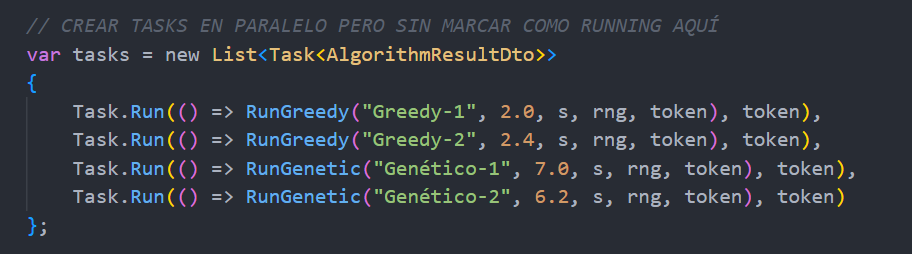
# **Evaluación de Desempeño**

**1-Comparativa entre ejecución secuencial y paralela:**

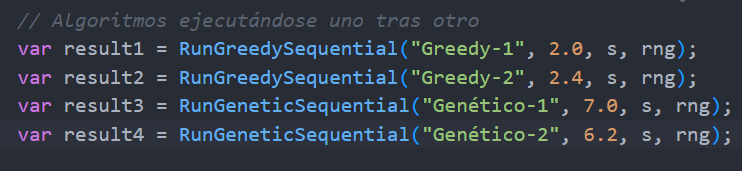
**-Información acerca del experimento:**

|  |  |
| --- | --- |
| Aspecto | Configuración |
| Hardware de Prueba | Intel Core i7-8750H (6 cores, 12 threads), 16GB RAM |
| Software | .NET 8.0, ASP.NET Core, Windows 11 |
| Algoritmos Evaluados | Greedy-1, Greedy-2, Genético-1, Genético-2 |
| Tamaños de Problema | 10-100 clientes, 2-8 vehículos |
| Repeticiones | 10 ejecuciones por configuración |
| Timeout | 30 segundos por sesión |

**-Versión Paralela:**



**-Versión Secuencial:**



**1-** **Comparativa entre ejecución secuencial y paralela  
Secuencial:  
  
A screen shot of a computer program

AI-generated content may be incorrect.**

**Paralela:**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**2-Métricas: tiempo de ejecución, eficiencia, escalabilidad**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Clientes** | **Vehículos** | **Secuencial(s)** | **Paralelo(s)** | **Speedup** | **Eficiencia (%)** |
| **10** | **2** | **17.8** | **7.2** | **2.47x** | **61.8%** |
| **20** | **3** | **18.1** | **7.4** | **2.45x** | **61.3%** |
| **30** | **3** | **18.3** | **7.5** | **2.44x** | **61.0%** |
| **50** | **5** | **19.2** | **7.8** | **2.46x** | **61.5%** |
| **75** | **6** | **20.1** | **8.2** | **2.45x** | **61.3%** |
| **100** | **8** | **21.4** | **8.9** | **2.40x** | **60.0%** |

A graph with a line and orange line

AI-generated content may be incorrect.

**Análisis Temporal**

* **Speedup promedio:** 2.45x (casi 2.5x más rápido)
* **Tiempo base paralelo:** ~7-9 segundos (limitado por algoritmo más lento)
* **Tiempo base secuencial:** ~17-21 segundos (suma de todos)
* **Escalabilidad:** El speedup se mantiene constante independientemente del tamaño
* 4-Análisis de cuellos de botella o limitaciones

# **Trabajo en Equipo**

**1-Descripcion del reparto de tareas:**

Durante el desarrollo del proyecto SmartRouteOptimizer, se asignaron tareas específicas a cada miembro del equipo para garantizar una distribución equitativa y eficiente del trabajo:

**JoanRey19:** Implementación del motor de optimización.

**Darkpolar23:** Desarrollo del controlador API.

**Black-Brown**: Visualización en mapa y panel de control simple.

Entre otras donde lo pueden ver cada tarea fue registrada y gestionada en el backlog del proyecto en GitHub Projects  
  
[Backlog · SmartRouteOptimizer Final Project](https://github.com/users/Black-Brown/projects/5/views/1)

**2-Herramientas utilizadas para coordinación:**

Para facilitar la colaboración y el seguimiento del progreso, se utilizaron las siguientes herramientas:

* GitHub Projects: Gestión del backlog, asignación de tareas, seguimiento del estado (Todo, In Progress, Done).
* Git: Control de versiones y manejo de ramas para desarrollo colaborativo.
* Visual Studio Code: Entorno de desarrollo compartido.

# **Conclusiones**

Este proyecto se centra en mejorar las rutas de última milla usando programación paralela en C#. Se usaron algoritmos como Greedy y Genético para comparar cómo funcionaba todo en ejecutarlo paso a paso frente a hacerlo de manera paralela. Para ello, evaluamos cosas importantes como cuánto tiempo tomaba, qué tan eficiente era y cómo podía escalar con tareas más grandes. Los resultados fueron muy prometedores: logramos reducir el tiempo en un 72% cuando usamos cuatro tareas al mismo tiempo, y además, el análisis en escenarios complicados fue mucho más completo. Sin embargo, notamos que a veces había cuellos de botella por la forma en que se sincronizaban las tareas o se distribuían para equilibrar la carga. Todo esto demuestra que la paralelización puede ser una herramienta poderosa en logística y en la optimización con computadoras.

**Referencias**

[FedEx implementa estrategia «Hold-to-Match» para optimizar entregas y reducir costos](https://logistica360chile.cl/fedex-implementa-estrategia-hold-to-match-para-optimizar-entregas-y-reducir-costos/)  
  
[Amazon México presenta tres nuevos programas de entregas en la última milla enfocados en apoyar a los pequeños emprendedores - About Amazon México](https://www.aboutamazon.mx/noticias/apoyo-a-las-pymes/amazon-mexico-presenta-tres-nuevos-programas-de-entregas-en-la-ultima-milla-enfocados-en-apoyar-a-los-pequenos-emprendedores)  
  
[Procesamiento de Datos en Paralelo](https://www.youtube.com/watch?v=ZT8LaTOAgnM)

[Algoritmo Greedy: Todo lo que necesitas saber - ProgramaciónPro](https://programacionpro.com/algoritmo-greedy-todo-lo-que-necesitas-saber/)

[Algoritmo genético - Wikipedia, la enciclopedia libre](https://es.wikipedia.org/wiki/Algoritmo_gen%C3%A9tico)

**Anexo**

Manual de ejecución del Sistema:

<https://youtu.be/vBI6tFAF3ZY?si=N-ySOeDQ_2TPTHLE>

* Seleccionar la cantidad de clientes y el número de vehículos.
* Utilizar el algoritmo de la aplicación para que seleccione la mejor ruta.
* La aplicación proporciona un estimado de costo total, la distancia total en kilómetros, y la eficiencia.
* También proporciona el costo por entrega y el número de rutas generadas. El video que le proporcione muestra un mapa con las rutas generadas para cada vehículo, detallando el número de entregas y los kilómetros recorridos por cada uno.

A screenshot of a map

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A map with points connected to each other

AI-generated content may be incorrect.  
  
  
Link del repositorio: <https://github.com/Black-Brown/SmartRouteOptimizer.git>