TP 2: Heurísticas para el Problema de Ruteo de Vehículos con Capacidad (CVRP)

Integrantes:
Camus Sol
Luchetti Olivia
Martinez Kiara

8 de julio de 2025

Índice

1. Introducción

El Problema de Ruteo de Vehículos con Capacidad (CVRP, por sus siglas en inglés) es un problema clásico de optimización combinatoria con aplicaciones directas en logística, distribución y transporte. El objetivo principal es diseñar rutas para una flota de vehículos idénticos, cada uno con una capacidad limitada, que deben atender a un conjunto de clientes dispersos geográficamente, partiendo y regresando a un depósito central. Cada cliente tiene una demanda conocida y debe ser atendido exactamente una vez, sin exceder la capacidad de los vehículos.

El CVRP es un problema NP-hard, lo que implica que no existen algoritmos exactos eficientes para resolver instancias de gran tamaño en tiempo razonable. Por esta razón, se recurre a heurísticas y metaheurísticas que permiten obtener soluciones de buena calidad en tiempos computacionales aceptables. En este trabajo, se exploran e implementan diferentes enfoques heurísticos para abordar el CVRP, analizando su desempeño y las características de las soluciones obtenidas.

1.1. Definición Formal

Dado un grafo $G = (V, E)$, donde $V = \{0, ..., n\}$ es el conjunto de nodos (0 es el depósito) y E es el conjunto de aristas. Cada cliente $i \in V \setminus \{0\}$ tiene una demanda q_i, y la distancia entre nodos i y j es d_{ij}. El objetivo es encontrar un conjunto de rutas de mínimo costo total para una flota de vehículos de capacidad Q, tal que cada cliente sea visitado exactamente una vez.

2. Heurísticas Constructivas Implementadas

2.1. Algoritmo de Ahorros de Clarke & Wright

Es una de las heurísticas más conocidas. Parte de una solución inicial donde cada cliente es servido individualmente y luego fusiona rutas de forma iterativa si esto genera un .ªhorro.en la distancia total.

2.1.1. Concepto de Ahorros

El ahorro s_{ij} al conectar los clientes i y j se calcula como: $s_{ij} = d_{0i} + d_{0j} - d_{ij}$. Representa la distancia que se ahorra al no tener que volver al depósito entre las visitas a i y j.

2.1.2. Análisis de Complejidad

- **Cálculo de ahorros:** Se calculan para cada par de nodos, resultando en una complejidad de $O(n^2)$.
- **Ordenamiento de ahorros:** Ordenar la lista de ahorros tiene una complejidad de $O(n^2 \log n)$.
- **Construcción de rutas:** El bucle principal itera sobre la lista de ahorros. Con estructuras de datos optimizadas (como un mapa para localizar nodos), las operaciones internas son rápidas. La complejidad total está dominada por el ordenamiento: $O(n^2 \log n)$.

2.2. Algoritmo del Vecino Más Cercano (Nearest Neighbor)

Es una heurística voraz que construye rutas secuencialmente. Desde el último nodo añadido a una ruta, se selecciona el cliente no visitado más cercano que no viole la restricción de capacidad.

2.2.1. Variante Implementada

Nuestra implementación evalúa una lista restringida de los 5 candidatos más cercanos para añadir a la ruta, en lugar de solo el más cercano, para diversificar la construcción.

2.2.2. Análisis de Complejidad

En cada paso de la construcción de una ruta, se busca el vecino más cercano entre los nodos no visitados. En el peor caso, esto se repite para cada uno de los n nodos, resultando en una complejidad de $O(n^2)$.

3. Operadores de Búsqueda Local

3.1. Descripción de los Operadores

Con el objetivo de mejorar las soluciones iniciales generadas por las heurísticas constructivas, implementamos operadores de búsqueda local clásicos, específicamente los operadores de reubicación (relocate) e intercambio (swap).

El operador de **relocate** consiste en seleccionar un cliente de una ruta y moverlo a otra posición, ya sea dentro de la misma ruta o en una ruta diferente, siempre que se respete la restricción de capacidad de los vehículos. En nuestra implementación, se exploran todas las posibles reubicaciones y se aplica la primera que produce una mejora en el costo total de la solución. Este proceso se repite de manera iterativa hasta que no se encuentran más movimientos beneficiosos.

Por otro lado, el operador de **swap** selecciona dos clientes, que pueden pertenecer a rutas distintas o a la misma ruta, y los intercambia de lugar. Antes de realizar el intercambio, se verifica que las demandas resultantes en ambas rutas no superen la capacidad máxima permitida. Si el intercambio resulta en una reducción del costo total, se realiza el cambio y se actualizan las rutas afectadas.

Ambos operadores se aplican de forma alternada y repetida, permitiendo así refinar progresivamente la solución inicial. Esta estrategia de búsqueda local nos permitió obtener soluciones de mejor calidad, acercándonos más a los óptimos de referencia y superando las limitaciones de las heurísticas constructivas puras. La implementación se diseñó para ser eficiente, priorizando la rapidez en la evaluación de movimientos y la actualización de las rutas tras cada modificación.

Impacto en los resultados: La incorporación de estos operadores de búsqueda local tuvo un efecto significativo en la calidad de las soluciones obtenidas. En la mayoría de los casos, se logró reducir el costo total de las rutas y, en ocasiones, disminuir la cantidad de vehículos necesarios. Esto demuestra la importancia de complementar las heurísticas constructivas con técnicas de mejora local para abordar problemas complejos como el CVRP.

3.1.1. Análisis de Complejidad

El costo computacional de los operadores de búsqueda local depende principalmente del número de rutas (R) y del número máximo de clientes por ruta (N).

- **Relocate:** La complejidad por iteración es $O(R^2N^2)$, ya que para cada par de rutas se consideran todas las posiciones posibles para mover un cliente de una ruta a otra.
- **Swap:** La complejidad por iteración también es $O(R^2N^2)$, dado que se evalúan todos los pares posibles de clientes entre todas las rutas.

En la práctica, si las rutas están balanceadas, el número de clientes por ruta suele ser aproximadamente $N \approx n/R$, donde n es el número total de clientes. Sin embargo, la cota teórica general sigue siendo $O(R^2N^2)$. Además, como la implementación detiene la búsqueda al encontrar la primera mejora, el tiempo promedio de ejecución suele ser menor que el peor caso, aunque para instancias grandes el costo computacional puede seguir siendo significativo si se buscan mejoras exhaustivas en cada iteración.

4. Métodos Híbridos

Para la generación de soluciones iniciales, se implementó un método que combina una heurística constructiva clásica con una fase de mejora local básica. En particular, utilizamos el algoritmo de **Clarke & Wright** para construir una primera solución factible, aprovechando su eficiencia y capacidad para generar rutas de buena calidad en poco tiempo.

Una vez obtenida la solución inicial, se aplica de manera iterativa el operador de **relocate** como técnica de búsqueda local. Este proceso consiste en intentar mover clientes entre rutas para reducir el costo total, repitiendo la operación hasta que no se logran más mejoras o se alcanza un número máximo de iteraciones. De esta forma, se refina la solución constructiva y se incrementa la calidad de los resultados obtenidos.

Decidimos esta estrategia híbrida porque nos permite partir de una solución razonable y mejorarla rápidamente, combinando la rapidez de las heurísticas constructivas con la capacidad de ajuste fino de la búsqueda local. En la práctica, observamos que este enfoque logra soluciones competitivas en tiempos computacionales reducidos.

4.0.1. Análisis de Complejidad

- **Clarke & Wright:** La complejidad es $O(n^2 \log n)$, ya que se calcula la matriz de ahorros y se ordenan los ahorros.
- **Relocate:** La complejidad es $O(R^2N^2)$, ya que se evalúan todas las posiciones posibles para mover un cliente de una ruta a otra.

Complejidad total: $O(n^2 \log n + R^2N^2)$ (Clarke & Wright + Relocate)

5. Consideraciones de Implementación y Dificultades Encontradas

El proyecto fue desarrollado en C++11, priorizando la eficiencia en el manejo de memoria y el rendimiento computacional.

Una de las principales dificultades encontradas fue el procesamiento y parsing de las instancias en formato VRPLIB, que requirió el desarrollo de una clase específica para la lectura robusta de los datos. Además, la validación de las restricciones de capacidad y la correcta construcción de las rutas demandó una cuidadosa gestión de las estructuras de datos, especialmente al fusionar rutas o al seleccionar el siguiente cliente en las heurísticas.

Otra dificultad relevante fue la depuración y validación de los resultados obtenidos, ya que fue necesario comparar las soluciones generadas con resultados de referencia y analizar casos particulares donde las heurísticas no lograban cumplir todas las restricciones. Finalmente, se optimizaron ciertas operaciones críticas, como la búsqueda de vecinos más cercanos y la actualización de rutas, para reducir el tiempo de ejecución en instancias de mayor tamaño.

6. Comandos para compilar y ejecutar el proyecto

6.1. Usando make

```
# Compilar el proyecto
make

# Limpiar archivos compilados
make clean
```

6.2. Ejecutando el proyecto

```
./bin/cvrp_solver.exe <instancia>
```

Luego de la ejecución, se dispondrá un menú con las siguientes opciones:

- **1:** Ejecutar el algoritmo de Clarke & Wright.
- **2:** Ejecutar el algoritmo de Nearest Neighbor.
- **3:** Ejecutar el algoritmo de GRASP.

Si se selecciona alguna de las primeras dos opciones, se dispondrá nuevamente otro menú con las siguientes opciones:

- **0:** No aplicar búsqueda local
- **1:** Aplicar operador Relocate
- **2:** Aplicar operador Swap
- **3:** Aplicar ambos operadores (Relocate + Swap)

6.2.1. Ejemplos de ejecución

```
./bin/cvrp_solver.exe instances/E045-04f.vrp
```

6.3. Ejecutando los tests

```
./bin/test_local_search.exe  (o ./bin/test_solution_reader.exe)
```

6.4. Clases Principales

A continuación se describen las clases principales desarrolladas en el proyecto:

Clase	**Descripción**
VRPLIBReader	Responsable de leer y parsear las instancias desde archivos de formato VRPLIB, almacenando los datos relevantes como nodos, demandas, capacidades y la matriz de distancias.
Solution y Ruta	Modelan una solución al problema y las rutas individuales, permitiendo almacenar y manipular la secuencia de clientes, demandas, costos y otras métricas asociadas.
ClarkeWrightSolver	Implementa la heurística constructiva de ahorros de Clarke & Wright, generando soluciones iniciales eficientes a partir de los datos de la instancia.
LocalSearch	Contiene la implementación de los operadores de búsqueda local, como reubicación (relocate) e intercambio (swap), para mejorar soluciones existentes.
GRASP	Implementa el metaheurístico GRASP (Greedy Randomized Adaptive Search Procedure), que construye soluciones de manera aleatorizada y aplica búsqueda local intensiva para refinar los resultados.
initial_methods	Función que combina la heurística de Clarke & Wright con búsqueda local, utilizada para generar soluciones iniciales de buena calidad de forma eficiente.

Cuadro 1: Clases principales desarrolladas en el proyecto y su función.

6.5. Casos de Test

Utilizamos ChatGPT para generar los tests unitarios para la clase **Solution** y **LocalSearch**. La idea principal de los tests para Solution es utilizarla para facilitar la comparación entre soluciones generadas y las soluciones dadas. Además, dispusimos de la IA para que, durante la experimentación, se registren los tiempos, costos y métodos de ejecución de cada instancia en un archivo .txt para luego facilitar el análisis de los resultados.
Link a la conversación con la IA

7. Experimentación y Resultados

7.1. Plan de Comparación de Métodos

La experimentación comparará el rendimiento de (al menos) los siguientes enfoques:

1. Cada heurística constructiva de manera independiente (realizado).
2. Heurística constructiva + un operador de búsqueda local (pendiente).
3. Heurística constructiva + combinación de operadores de búsqueda local (pendiente).

7.2. Resultados Preliminares (Heurísticas Constructivas)

- Para la instancia **E045-04f**:
  - **Clarke & Wright:** Costo = 834.34, Rutas = 5.
  - **Nearest Neighbor:** Costo = 1075.33, Rutas = 4.
  - **Referencia:** Costo = 723.54, Rutas = 4.
- **Observación:** Clarke & Wright obtiene un mejor costo, mientras que Nearest Neighbor utiliza un número de vehículos más cercano al óptimo en este caso.

7.3. Análisis de Resultados Finales (Pendiente)

En esta sección se presentará un análisis exhaustivo de los resultados de todos los métodos propuestos, utilizando tablas y gráficos para comparar su rendimiento en el conjunto de instancias.

8. Conclusiones

8.1. Hallazgos Preliminares

- Las heurísticas constructivas proveen una base sólida, pero sus resultados pueden ser mejorados significativamente.
- Existe un trade-off entre la calidad de la solución (costo) y el número de vehículos utilizados, donde diferentes heurísticas favorecen uno u otro.

8.2. Posibles Mejoras y Trabajo Futuro

- **Búsqueda Local:** La implementación de operadores de búsqueda local es el siguiente paso crucial.
- **Randomización y Metaheurísticas:** Para obtener la nota máxima, se podría implementar una estrategia de randomización en las heurísticas constructivas o una metaheurística como Simulated Annealing o Búsqueda Tabú para guiar la búsqueda local.
- **Paralelización:** Para instancias de gran tamaño, el cálculo de la matriz de ahorros podría ser paralelizado.