Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Informe de la Práctica 5 de DAA
Curso 2024-2025

Vehicle Routing Problem with Transshipments for Solid Waste Collection with Transfer Stations (VRPT-SWTS)

Pablo Hernández Jiménez

La Laguna, 22 de abril de 2025

Agradecimientos

Agradezco a los profesores de la asignatura Diseño y Análisis de Algoritmos por su guía y apoyo durante el desarrollo de esta práctica. También quiero agradecer a mis compañeros por las discusiones y sugerencias que han contribuido a mejorar este trabajo.

Licencia

Resumen

En este trabajo se aborda el Problema de Rutas de Vehículos con Trasbordos para la recogida de residuos sólidos con estaciones de transferencia (VRPT-SWTS). Se han implementado y evaluado diversos algoritmos para resolver este problema, incluyendo un algoritmo voraz constructivo, GRASP y GVNS. El objetivo principal es minimizar el número de vehículos necesarios para la recolección y transporte de residuos, considerando las restricciones de capacidad, tiempo y sincronización entre vehículos de recolección (CV) y vehículos de transporte (TV). Los resultados experimentales muestran que los algoritmos metaheurísticos proporcionan soluciones de mejor calidad que el algoritmo voraz, con GVNS obteniendo los mejores resultados en términos de número de vehículos utilizados y tiempo de ejecución.

Palabras clave: Waste collection, VRPT-SWTS, GRASP, LS, VNS.

Índice general

1. Introducción **1**
1.1. Contexto 1
1.1.1. Objetivos 1
1.2. Motivación 2

2. Vehicle Routing Problem with Transshipments for Solid Waste Collection with Transfer Stations (VRPT-SWTS) **3**
2.1. Descripción 3
2.1.1. Representación de las soluciones 4
2.1.2. Uso de KD-Tree para optimización espacial 5

3. Algoritmos **8**
3.1. Un algoritmo voraz para construir las rutas de los vehículos de recolección (Algorithm 1) 8
3.2. Un algoritmo constructivo para generar las rutas de los vehículos de transporte (Algorithm 2) 10
3.3. Algoritmo para seleccionar el mejor vehículo para una tarea (Algorithm 3) . 11
3.4. Algoritmo MultiStart-RVND para mejorar las rutas de vehículos de recolección 12

4. Experimentos y resultados computacionales **14**
4.1. Constructivo voraz 14
4.2. GRASP 15
4.3. Análisis comparativo de resultados 17
4.3.1. Comparación de algoritmos constructivos 17
4.3.2. Análisis de eficiencia 18
4.3.3. Análisis de calidad de solución 18
4.3.4. Conclusiones del análisis comparativo 19

5. Conclusiones y trabajo futuro **21**
5.1. Conclusiones 21
5.2. Trabajo Futuro 22

Índice de Figuras

2.1. Representación gráfica del esqiema de gestión de los residuo sólidos 4
2.2. Representación gráfica de un KD-Tree en 2 dimensiones 6

4.1. Comparativa de tiempo de ejecución entre algoritmos 18
4.2. Comparativa de número de vehículos entre algoritmos 18

Índice de Tablas

4.1. VRPT Solver (MultiStart-RVND + GreedyTVScheduler) + No neighborhoods. Tabla de resultados 14
4.2. VRPT Solver (MultiStart-RVND + GreedyTVScheduler) + No Neighborhoods. Tabla de resultados 15
4.3. VRPT Solver (MultiStart-RVND + GreedyTVScheduler). + No neighborhoods. Tabla de resultados 16
4.4. VRPT Solver (MultiStart-RVND (GRASP) + GreedyTVScheduler). + All neighborhoods. Tabla de resultados 16
4.5. Comparativa de rendimiento entre algoritmos constructivos 17

Capítulo 1

Introducción

1.1. Contexto

La gestión eficiente de residuos sólidos urbanos es un desafío creciente para las ciudades modernas. Con el aumento de la población urbana y las regulaciones ambientales más estrictas, los vertederos se ubican cada vez más lejos de los centros urbanos. Esto ha llevado a la implementación de sistemas de recogida de residuos que utilizan estaciones de transferencia intermedias (SWTS), donde los residuos son transferidos desde vehículos de recolección pequeños (CV) a vehículos de transporte de mayor capacidad (TV).

En este contexto, el problema de optimización de rutas de vehículos con trasbordos para la recogida de residuos sólidos (VRPT-SWTS) busca determinar las rutas óptimas tanto para los vehículos de recolección como para los de transporte, considerando las restricciones de capacidad, tiempo y la necesaria sincronización entre ambos tipos de vehículos en las estaciones de transferencia.

1.1.1. Objetivos

Durante el desarrollo de esta práctica, se han cumplido los objetivos listados a continuación:

- Implementar un algoritmo voraz constructivo para generar rutas de vehículos de recolección (CV) y de transporte (TV).
- Desarrollar la fase constructiva de un algoritmo GRASP para mejorar la calidad de las soluciones iniciales.
- Implementar diferentes estructuras de vecindad para búsqueda local, incluyendo reinserción y intercambio de tareas dentro y entre rutas, así como 2-opt.
- Desarrollar un algoritmo GVNS (General Variable Neighborhood Search) que utilice múltiples estructuras de vecindad para mejorar las soluciones.
- Realizar un análisis comparativo de los diferentes algoritmos implementados en términos de calidad de la solución y tiempo de ejecución.

1.2. Motivación

La motivación principal para abordar este problema radica en su relevancia práctica y su complejidad algorítmica. Desde una perspectiva práctica, la optimización de sistemas de recogida de residuos tiene un impacto significativo en:

- **Reducción de costos operativos:** Minimizar el número de vehículos necesarios y optimizar sus rutas reduce significativamente los costos de combustible, mantenimiento y personal.
- **Impacto ambiental:** Rutas más eficientes implican menor consumo de combustible y, por tanto, menor emisión de gases contaminantes.
- **Eficiencia del servicio:** Una planificación óptima permite cumplir con los horarios de recogida y mejorar la calidad del servicio.

Desde una perspectiva algorítmica, el VRPT-SWTS representa un desafío interesante por ser una variante compleja del problema de rutas de vehículos (VRP), que es NP-difícil. La necesidad de sincronización entre diferentes tipos de vehículos añade una dimensión adicional de complejidad que requiere el desarrollo de algoritmos heurísticos y metaheurísticos eficientes.

Capítulo 2

Vehicle Routing Problem with Transshipments for Solid Waste Collection with Transfer Stations (VRPT-SWTS)

2.1. Descripción

En esta práctica estudiaremos un Problema de Rutas de Vehículos con Trasbordos para la recogida de residuos sólidos con estaciones de transferencia (VRPT-SWTS) Ghiani et al., 2021, que trata sobre la optimización de la recolección de residuos sólidos urbanos considerando estaciones de transferencia intermedias (*Solid Waste Transfer Stations*, SWTS). Debido a regulaciones ambientales más estrictas y al aumento de la distancia de los vertederos con respecto a los centros urbanos, se ha incentivado el uso de estas estaciones, donde los residuos son transferidos de vehículos de recolección pequeños (*Collection Vehicles*, CV) a vehículos de transporte de mayor capacidad (*Transportation Vehicles*, TV).

El problema se modela como un *Vehicle Routing Problem with Transshipment* (VRPT), en el que se deben determinar las rutas de los vehículos de recolección y transporte, así como su sincronización en las estaciones de transferencia. Se descompone en dos fases (ver Figura 2.1):

- **Fase de recolección:** Se deben diseñar las rutas de los vehículos de recolección pequeños (CV), asegurando que visiten los puntos de recolección asignados y transfieran su carga en una estación SWTS antes de regresar al depósito. Cada ruta puede constar de múltiples trayectos, dependiendo del tiempo disponible y la capacidad del vehículo.
- **Fase de transporte:** Se determinan las rutas de los vehículos de transporte de mayor capacidad (TV), los cuales recogen los residuos de las estaciones SWTS y los transportan al vertedero. Estas rutas deben sincronizarse con la llegada de los vehículos de recolección.

El problema introduce desafíos adicionales con respecto al *Vehicle Routing Problem* (VRP) clásico, debido a la necesidad de coordinación entre los diferentes tipos de vehículos

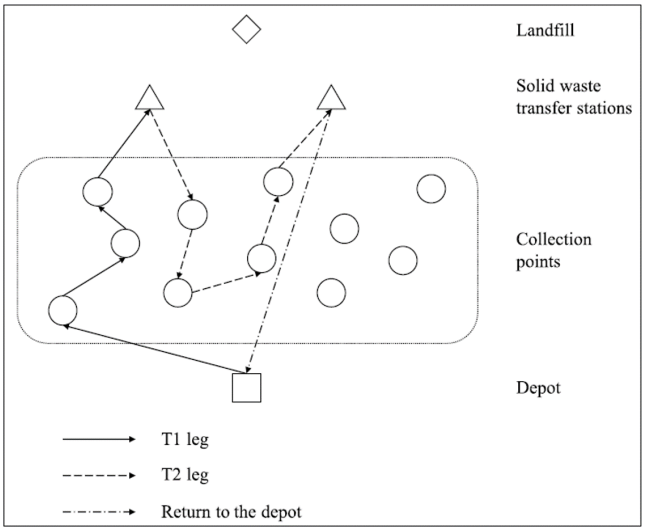

Figura 2.1: Representación gráfica del esqiema de gestión de los residuo sólidos

y la gestión eficiente del transbordo de carga en las estaciones de transferencia. El problema consiste en diseñar las rutas para los vehículos de recolección de residuos y para los vehículos de transporte requeridos para realizar las tareas de recogida de residuos sólidos, con el **objetivo de minimizar el número de vehículos a utilizar**.

Para resolverlo, haremos uso de algoritmos heurísticos y metaheurísticos, como son los algoritmos voraces, GRASP, búsqueda locales y GVNS.

2.1.1. Representación de las soluciones

Para representar las soluciones del problema VRPT-SWTS, se ha diseñado una estructura de datos que permite modelar de manera eficiente tanto las rutas de los vehículos de recolección (CV) como las de los vehículos de transporte (TV), así como la sincronización entre ambos tipos de vehículos.

Estructura general de la solución

La solución se representa mediante una clase `VRPTSolution` que contiene:

- **Rutas de vehículos de recolección (CV):** Un vector de objetos `CVRoute`, donde cada ruta contiene una secuencia de ubicaciones (depósito, zonas de recolección y estaciones SWTS).
- **Rutas de vehículos de transporte (TV):** Un vector de objetos `TVRoute`, donde cada ruta contiene una secuencia de visitas a estaciones SWTS y al vertedero.
- **Tareas de entrega:** Generadas a partir de las rutas CV, cada tarea representa una entrega de residuos en una estación SWTS en un momento determinado.

Representación de rutas CV

Cada ruta CV se representa mediante un objeto `CVRoute` que contiene:

- **Identificador del vehículo**

- **Secuencia de ubicaciones:** Vector de identificadores de ubicaciones (depósito, zonas, SWTS)
- **Capacidad residual:** Capacidad restante del vehículo en cada punto de la ruta
- **Tiempo acumulado:** Tiempo transcurrido en cada punto de la ruta
- **Duración total:** Duración total de la ruta

Representación de rutas TV

Cada ruta TV se representa mediante un objeto `TVRoute` que contiene:

- **Identificador del vehículo**
- **Secuencia de ubicaciones:** Vector de identificadores de ubicaciones (vertedero, SWTS)
- **Tiempos de llegada:** Tiempo de llegada a cada ubicación
- **Tiempos de espera:** Tiempo de espera en cada SWTS para sincronizarse con los CV
- **Cantidades recogidas:** Cantidad de residuos recogida en cada SWTS
- **Capacidad residual:** Capacidad restante del vehículo en cada punto de la ruta

Representación de tareas de entrega

Las tareas de entrega, que sirven como enlace entre las rutas CV y TV, se representan mediante objetos `DeliveryTask` que contienen:

- **Identificador de la estación SWTS**
- **Tiempo de llegada del CV**
- **Cantidad de residuos entregada**
- **Identificador del CV que realiza la entrega**

Esta estructura de datos permite una representación completa y eficiente de las soluciones, facilitando tanto la construcción de rutas como la evaluación de su factibilidad y calidad.

2.1.2. Uso de KD-Tree para optimización espacial

Para mejorar la eficiencia en la búsqueda de ubicaciones cercanas (como estaciones SWTS próximas a zonas de recolección), se implementa una estructura de datos KD-Tree (K-Dimensional Tree). Esta estructura permite almacenar puntos en un espacio k-dimensional y realizar búsquedas de vecinos más cercanos de manera eficiente.

Fundamentos del KD-Tree

Un KD-Tree es una estructura de datos de particionamiento espacial que organiza los puntos en un espacio k-dimensional. Para nuestro problema VRPT-SWTS, utilizamos un 2D-Tree donde:

- Cada nodo representa una ubicación (depósito, zona de recolección, estación SWTS o vertedero)
- Las coordenadas x e y corresponden a la posición geográfica de cada ubicación
- La estructura divide recursivamente el espacio en regiones rectangulares, alternando entre divisiones verticales y horizontales

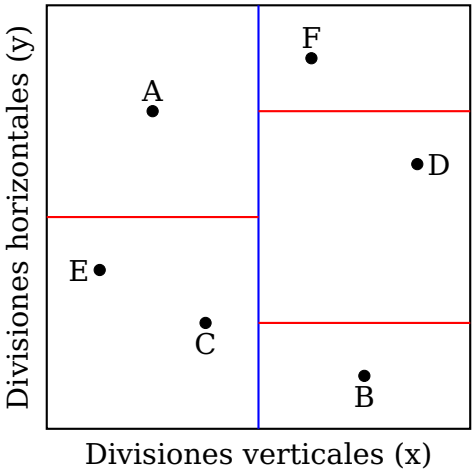

Figura 2.2: Representación gráfica de un KD-Tree en 2 dimensiones

Construcción del KD-Tree

La construcción del KD-Tree para nuestro problema sigue estos pasos:

1. Se recopilan todas las ubicaciones relevantes (depósito, zonas de recolección, estaciones SWTS y vertedero)
2. Se selecciona la dimensión a dividir (alternando entre x e y)
3. Se encuentra la mediana de los puntos en esa dimensión
4. Se crea un nodo con el punto mediano y se divide el espacio en dos mitades
5. Se construye recursivamente el subárbol izquierdo/inferior y el derecho/superior

La complejidad temporal de la construcción es $O(n \log n)$, donde n es el número de ubicaciones.

Búsqueda de vecinos más cercanos

Una de las principales ventajas del KD-Tree es su capacidad para encontrar rápidamente los vecinos más cercanos a un punto dado. Aunque la búsqueda del vecino más cercano tiene una complejidad teórica de $O(\log n)$ en el caso promedio, en la práctica, para problemas de rutas con distribuciones de puntos relativamente uniformes como el nuestro, se comporta de manera muy eficiente, aproximándose a $O(1)$ para búsquedas de vecinos inmediatos en muchos casos de uso.

El algoritmo de búsqueda realiza los siguientes pasos:

1. Recorre el árbol hasta encontrar la región que contendría el punto de consulta
2. Guarda el punto encontrado como el "mejor hasta ahora"
3. Retrocede a través del árbol, comprobando si podría haber puntos más cercanos en otras regiones
4. Si una región podría contener un punto más cercano, se explora también
5. Se actualiza el "mejor hasta ahora"si se encuentra un punto más cercano

Aplicación en el VRPT-SWTS

En nuestro problema, el KD-Tree se utiliza principalmente para:

- Encontrar rápidamente la estación SWTS más cercana a una zona de recolección cuando un vehículo CV necesita descargar
- Determinar qué zonas de recolección están más próximas entre sí para construir rutas eficientes
- Optimizar la asignación de vehículos TV a estaciones SWTS basándose en proximidad geográfica

Esta estructura permite reducir significativamente el tiempo computacional en comparación con la búsqueda lineal de vecinos más cercanos, especialmente en instancias grandes del problema con numerosas ubicaciones. Mientras que una búsqueda lineal tendría una complejidad de $O(n)$, el KD-Tree proporciona una alternativa mucho más eficiente que se aproxima a tiempo constante para muchos casos prácticos en este tipo de problemas de rutas.

Capítulo 3

Algoritmos

3.1. Un algoritmo voraz para construir las rutas de los vehículos de recolección (Algorithm 1)

En esta sección se muestra un algoritmo constructivo voraz que permite generar las rutas de recolección. Cuando el tamaño del problema se acerca a un caso real, la **complejidad del modelo matemático de optimización** hace que sea difícil obtener una solución óptima en un tiempo razonable. Para abordar esta limitación, se usa una **heurística constructiva voraz rápida**, cuyo pseudocódigo se presenta en Algorithm 1.

La idea principal del algoritmo es **construir las rutas de los vehículos agregando una zona de recolección a la vez**.

1. **Inicio de una nueva ruta**
    - Se crea una nueva ruta desde el **depósito**, asignándole la **máxima capacidad disponible** y el **tiempo máximo permitido** para operar.
2. **Asignación de zonas de recolección**
    - Mientras haya **zonas de recolección sin asignar**, se selecciona la zona más cercana a la última ubicación de la ruta.
    - Si agregar esta zona no **excede la capacidad del vehículo** ni el **tiempo máximo de ruta**, se **añade a la ruta**, se elimina del conjunto de zonas pendientes y se actualizan los valores de capacidad y tiempo restantes.
    - Si no es posible agregarla porque **el vehículo se llenó**, se dirige a la **estación de transferencia de residuos (SWTS) más cercana**, se vacía la carga y se actualizan los valores de capacidad y tiempo.
    - Si la zona no puede añadirse porque **no hay suficiente tiempo restante**, se **finaliza la ruta**.
3. **Finalización de la ruta**
    - Si la última parada del vehículo no fue una estación de transferencia, se añade la **más cercana** antes de regresar al depósito.
    - Se guarda la ruta y se asigna un **nuevo vehículo** para continuar con las zonas restantes.

- Este proceso **se repite hasta que todas las zonas hayan sido atendidas**.

El **objetivo de esta heurística** es construir rutas eficientes sin necesidad de resolver el problema de forma exacta, permitiendo encontrar soluciones en **menos tiempo**.

Algoritmo 1 A Constructive Heuristic for Collection Vehicle Routes

Require: The set of collection zones C, the waste quantity d_c for each $c \in C$, vehicle capacity Q, maximum route duration L, the set of SWTS S.
Ensure: A set K of collection vehicles.

$K \leftarrow \emptyset$
$k \leftarrow 1$
while $C \neq \emptyset$ **do**
    $R_k \leftarrow \{\text{depot}\}$ ▷ Initialize the route for vehicle k
    $q_k \leftarrow Q$ ▷ Residual vehicle capacity
    $T_k \leftarrow L$ ▷ Residual route time
    while true **do**
        $c' \leftarrow$ Closest collection zone to last(R_k) in C
        $t' \leftarrow$ time to visit c', a SWTS, and the depot
        if $d_{c'} \leq q_k$ and $t' \leq T_k$ **then**
            $R_k \leftarrow R_k \cup \{c'\}$
            $q_k \leftarrow q_k - d_{c'}$
            Update T_k
            $C \leftarrow C \setminus \{c'\}$
        else
            if $t' \leq T_k$ **then**
                $s' \leftarrow$ Closest SWTS to c'
                $R_k \leftarrow R_k \cup \{s'\}$
                $q_k \leftarrow Q$
                Update T_k
            else
                break
            end if
        end if
    end while
    if last(R_k) $\notin S$ **then**
        $s' \leftarrow$ Closest SWTS to last(R_k)
        $R_k \leftarrow R_k \cup \{s'\} \cup \{\text{depot}\}$
    else
        $R_k \leftarrow R_k \cup \{\text{depot}\}$
    end if
    $K \leftarrow K \cup \{R_k\}$
    $k \leftarrow k + 1$
end while
return K

3.2. Un algoritmo constructivo para generar las rutas de los vehículos de transporte (Algorithm 2)

Los vehículos de transporte inician su ruta en el vertedero, visitan una o más SWTS para recoger residuos de los vehículos de recolección y regresan al vertedero para vaciar su carga. Las rutas de los vehículos de recolección determinan un conjunto de tareas H para los vehículos de transporte, donde cada tarea $h \in H$ se representa como una tripleta:

$$h = (D_h, s_h, \tau_h) \tag{3.1}$$

Donde:

- D_h: Cantidad de residuos transportados por el vehículo de recolección en la SWTS s_h.
- s_h: SWTS visitada por el vehículo de recolección.
- τ_h: Tiempo en el que el vehículo de recolección llega a la SWTS.

Una tarea h solo puede ser asignada a un vehículo de transporte $e \in E$ si se cumplen las siguientes condiciones:

(a) Si la ruta R_e ya tiene tareas asignadas y h' es la última tarea asignada a e, entonces el tiempo de viaje desde $s_{h'}$ hasta s_h debe ser menor o igual a la diferencia $\tau_h - \tau_{h'}$.

(b) La capacidad restante del vehículo q_e debe ser mayor o igual a la cantidad de residuos D_h.

(c) La duración total de la ruta R_e, considerando el regreso al vertedero, no debe exceder un umbral L'.

El algoritmo 2 sigue los siguientes pasos:

1. Ordenar las tareas H en orden creciente según τ_h.
2. Inicializar un conjunto vacío de vehículos de transporte E.
3. Para cada tarea $h \in H$:
   a) Buscar el vehículo disponible que minimice el costo de inserción cumpliendo las restricciones (a)-(c).
   b) Si no hay un vehículo disponible, crear un nuevo vehículo.
   c) Agregar la tarea a la ruta del vehículo seleccionado.
   d) Si la capacidad restante del vehículo cae por debajo de un umbral mínimo, programar un viaje al vertedero antes de asignarle nuevas tareas.
4. Asegurar que todas las rutas terminan en el vertedero.

Este método garantiza que los vehículos de transporte operen de manera eficiente y estén sincronizados con los vehículos de recolección, optimizando la logística del sistema de gestión de residuos.

Algoritmo 2 A Constructive Heuristic for Transportation Vehicle Routes

Require: The set of tasks H, transportation vehicle capacity Q', maximum route duration L'.
Ensure: A set E of transportation vehicles.

Sort H in ascending order of arrival time τ_h.
$E \leftarrow \emptyset$
$q_{\text{mín}} \leftarrow \text{mín}\{D_h : h \in H\}$
while $H \neq \emptyset$ **do**
    $h \leftarrow$ first element in H
    $H \leftarrow H \setminus \{h\}$
    $e \leftarrow$ ChooseVehicle(E, h)
    if $e =$ null **then**
        $e \leftarrow |E| + 1$
        $R_e \leftarrow \{\text{landfill}, s_h\}$
        $q_e \leftarrow Q' - D_h$
        Update T_e ($T_e \leftarrow L' -$ {time to go from the landfill to s_h})
        $E \leftarrow E \cup \{e\}$
    else
        $R_e \leftarrow R_e \cup \{s_h\}$
        $q_e \leftarrow q_e - D_h$
        Update T_e
        if $q_e < q_{\text{mín}}$ **then**
            $R_e \leftarrow R_e \cup \{\text{landfill}\}$
            $q_e \leftarrow Q'$
        end if
    end if
end while
for $e \in E$ **do**
    if last$(R_e) \neq$ landfill **then**
        $R_e \leftarrow R_e \cup \{\text{landfill}\}$
    end if
end for
return E

3.3. Algoritmo para seleccionar el mejor vehículo para una tarea (Algorithm 3)

Esta sección muestra en el Algoritmo 3 el procedimiento utilizado para determinar el vehículo de transporte al que se asignará una tarea.

Algoritmo 3 Selecting the Best Vehicle for a Task

Require: Set of vehicles E, task h.
Ensure: Vehicle e to which the task is assigned.

$e \leftarrow$ null
$bestInsertionCost \leftarrow +\infty$
for $e' \in E$ **do**
    $cost \leftarrow$ compute best insertion satisfying conditions (a)-(c)
    if $cost < bestInsertionCost$ **then**
        $e \leftarrow e'$
        $bestInsertionCost \leftarrow cost$
    end if
end for
return e

3.4. Algoritmo MultiStart-RVND para mejorar las rutas de vehículos de recolección

Para mejorar la calidad de las soluciones obtenidas mediante los algoritmos constructivos, se implementa un método metaheurístico híbrido denominado MultiStart-RVND (Random Variable Neighborhood Descent). Este algoritmo, cuyo pseudocódigo se presenta en el Algoritmo 4, combina la exploración mediante múltiples soluciones iniciales con un proceso de búsqueda intensiva utilizando diferentes estructuras de busqueda local.

Algoritmo 4 MultiStart con RVND para optimización de rutas VRPT

Require: Problema VRPT, número de inicios $numStarts$, generador de soluciones iniciales, conjunto de vecindarios $N = \{N_1, N_2, ..., N_k\}$.
Ensure: La mejor solución encontrada.

$bestSolution \leftarrow null$
Paralelizar en $numThreads$ hilos:
for $i = 1$ to $numStarts$ **do**
    $currentSolution \leftarrow$ GenerarSoluciónInicial() ▷ Utilizando GRASP u otro generador
    $improved \leftarrow true$
    while $improved$ **do**
        $improved \leftarrow false$
        $availableNeighborhoods \leftarrow \{1, 2, ..., k\}$ ▷ Inicializar bitmap de vecindarios
        while $availableNeighborhoods \neq \emptyset$ and not $improved$ **do**
            $j \leftarrow$ SeleccionarAleatorio($availableNeighborhoods$)
            $candidateSolution \leftarrow$ AplicarVecindario(N_j, $currentSolution$)
            if MejorQue($candidateSolution$, $currentSolution$) **then**
                $currentSolution \leftarrow candidateSolution$
                $improved \leftarrow true$
            else
                $availableNeighborhoods \leftarrow availableNeighborhoods \setminus \{j\}$
            end if
        end while
    end while
    Actualizar $bestSolution$ si $currentSolution$ es mejor (con bloqueo)
end for
return $bestSolution$

Las principales características del algoritmo MultiStart-RVND son:

1. **Paralelización**: El algoritmo utiliza múltiples hilos de ejecución para explorar diferentes soluciones iniciales simultáneamente, aprovechando al máximo los recursos computacionales disponibles.

2. **Generación de soluciones iniciales**: Se utilizan algoritmos constructivos como GRASP (Greedy Randomized Adaptive Search Procedure) para generar soluciones iniciales diversas con un componente aleatorio controlado.

3. **Descenso de Vecindario Variable Aleatorio (RVND)**: A diferencia del VND clásico que explora los vecindarios en un orden predeterminado, RVND selecciona aleatoriamente el vecindario a explorar en cada iteración, lo que ayuda a evitar sesgos y explorar mejor el espacio de soluciones.

4. **Estructuras de vecindario implementadas**: Se utilizan cinco estructuras de vecindario complementarias para mejorar las rutas:

- **2-Opt**: Invierte segmentos de rutas rompiendo dos arcos y reconectándolos de manera diferente para reducir la distancia total.
- **Reinserción dentro de rutas**: Reubica una zona de recolección en una posición diferente dentro de la misma ruta.
- **Reinserción entre rutas**: Mueve una zona de recolección de una ruta a otra.
- **Intercambio dentro de rutas**: Intercambia la posición de dos zonas dentro de una misma ruta.
- **Intercambio entre rutas**: Intercambia zonas entre diferentes rutas.

5. **Criterio de aceptación**: Una solución candidata se acepta si mejora la solución actual según varios criterios: primero se minimiza el número de vehículos de recolección, luego se maximiza el número de zonas visitadas y finalmente se minimiza la duración total de las rutas.

6. **Actualización thread-safe**: Como el algoritmo está paralelizado, se utilizan mecanismos de sincronización para actualizar la mejor solución global de manera segura entre hilos.

El proceso de RVND continúa hasta que no se encuentren más mejoras en ninguno de los vecindarios disponibles. Al finalizar todos los inicios paralelos, el algoritmo retorna la mejor solución encontrada entre todas las ejecuciones.

Esta implementación aprovecha eficientemente los sistemas multicore modernos y permite explorar un espacio de soluciones mucho más amplio en comparación con algoritmos secuenciales tradicionales, lo que resulta en soluciones de mayor calidad para el problema VRPT.

Capítulo 4

Experimentos y resultados computacionales

4.1. Constructivo voraz

VRPT Solver (MultiStart-RVND + GreedyTVScheduler)

$Instance$	$\#Zonas$	$\|LRC\|$	$Ejecución$	$\#CV$	$\#TV$	CPU_Time
$instance1$	20	1	1	5	3	4.84
$instance1$	20	2	2	5	3	4.41
$instance1$	20	3	3	5	3	4.08
$instance10$	20	1	1	5	3	5.09
$instance10$	20	2	2	5	3	4.31
$instance10$	20	3	3	5	3	4.10
$instance11$	20	1	1	5	3	4.02
$instance11$	20	2	2	5	3	3.55
$instance11$	20	3	3	5	3	3.34
$instance12$	20	1	1	6	2	3.50
$instance12$	20	2	2	6	2	4.20
$instance12$	20	3	3	6	2	3.86
$instance13$	20	1	1	5	4	4.70
$instance13$	20	2	2	5	4	3.18
$instance13$	20	3	3	5	4	3.20
...	...	...	...	...	...	...
$average$				5.15	3.55	4.82

Cuadro 4.1: VRPT Solver (MultiStart-RVND + GreedyTVScheduler) + No neighborhoods. Tabla de resultados

VRPT Solver (MultiStart-RVND (Greedy) + GreedyTVScheduler)

Instance	*#Zonas*	*|LRC|*	*Ejecución*	*#CV*	*#TV*	*CPU_Time*
*instance*1	20	1	1	5	3	289.42
*instance*1	20	2	2	5	3	324.68
*instance*1	20	3	3	5	3	302.79
*instance*10	20	1	1	5	2	566.90
*instance*10	20	2	2	5	2	531.81
*instance*10	20	3	3	5	2	591.42
*instance*11	20	1	1	5	2	470.47
*instance*11	20	2	2	5	2	458.19
*instance*11	20	3	3	5	2	481.27
*instance*12	20	1	1	6	2	715.39
*instance*12	20	2	2	6	2	536.04
*instance*12	20	3	3	6	2	536.87
*instance*13	20	1	1	5	4	142.97
*instance*13	20	2	2	5	4	141.60
*instance*13	20	3	3	5	4	202.23
...	...	...	...	...	...	...
average				5.15	3.05	370.61

Cuadro 4.2: VRPT Solver (MultiStart-RVND + GreedyTVScheduler) + No Neighborhoods. Tabla de resultados

4.2. GRASP

Comparativa probando diferentes búsquedas locales. Debe incluir el diseño de experimentos para el ajsute de parámetros y el análisis de los resultados obtenidos.

VRPT Solver (MultiStart-RVND + GreedyTVScheduler)

$Instance$	$\#Zonas$	$\|LRC\|$	$Ejecución$	$\#CV$	$\#TV$	CPU_Time
$instance1$	20	1	1	5	2	7.21
$instance1$	20	2	2	5	3	6.71
$instance1$	20	3	3	5	2	8.68
$instance10$	20	1	1	5	2	7.23
$instance10$	20	2	2	5	2	7.02
$instance10$	20	3	3	5	2	5.69
$instance11$	20	1	1	5	2	7.79
$instance11$	20	2	2	5	2	8.59
$instance11$	20	3	3	5	2	6.94
$instance12$	20	1	1	5	3	6.88
$instance12$	20	2	2	5	3	8.19
$instance12$	20	3	3	5	3	5.18
$instance13$	20	1	1	6	2	7.67
$instance13$	20	2	2	6	2	5.57
$instance13$	20	3	3	6	2	5.15
...	...	...	...	...	...	...
$average$				5.05	2.85	6.79

Cuadro 4.3: VRPT Solver (MultiStart-RVND + GreedyTVScheduler). + No neighborhoods. Tabla de resultados

VRPT Solver (MultiStart-RVND (GRASP) + GreedyTVScheduler)

$Instance$	$\#Zonas$	$\|LRC\|$	$Ejecución$	$\#CV$	$\#TV$	CPU_Time
$instance1$	20	1	1	5	2	52.45
$instance1$	20	2	2	5	2	48.33
$instance1$	20	3	3	5	2	53.65
$instance10$	20	1	1	5	2	56.67
$instance10$	20	2	2	5	2	60.09
$instance10$	20	3	3	5	2	59.82
$instance11$	20	1	1	5	2	50.89
$instance11$	20	2	2	5	2	51.33
$instance11$	20	3	3	5	2	51.33
$instance12$	20	1	1	5	3	52.35
$instance12$	20	2	2	5	3	55.35
$instance12$	20	3	3	5	3	58.52
$instance13$	20	1	1	6	2	49.95
$instance13$	20	2	2	6	2	55.39
$instance13$	20	3	3	6	2	46.96
...	...	...	...	...	...	...
$average$				5.07	2.78	56.22

Cuadro 4.4: VRPT Solver (MultiStart-RVND (GRASP) + GreedyTVScheduler). + All neighborhoods. Tabla de resultados

4.3. Análisis comparativo de resultados

En esta sección se comparan los resultados obtenidos por los diferentes algoritmos implementados para resolver el problema VRPT. Las comparaciones realizadas permiten identificar las ventajas y desventajas de cada enfoque, así como su impacto en términos de calidad de solución y tiempo de cómputo.

4.3.1. Comparación de algoritmos constructivos

Algoritmo	#CV promedio	#TV promedio	Tiempo CPU promedio (s)
Greedy (sin búsqueda local)	5.15	3.55	4.82
Greedy (con búsqueda local)	5.15	3.05	370.61
GRASP (sin búsqueda local)	5.05	2.85	6.79
GRASP (con búsqueda local)	5.07	2.78	56.22

Cuadro 4.5: Comparativa de rendimiento entre algoritmos constructivos

El análisis de los resultados muestra varias observaciones importantes:

- **Número de vehículos de recolección (CV):** Tanto el algoritmo Greedy como GRASP sin búsqueda local generaron soluciones con un número similar de vehículos de recolección (5.15 y 5.05 respectivamente). Sin embargo, GRASP logra una ligera reducción, lo que indica que la aleatorización controlada mejora la calidad de las soluciones iniciales.
- **Número de vehículos de transporte (TV):** Se observa una mejora significativa en la reducción del número de vehículos de transporte cuando se utiliza GRASP, pasando de 3.55 en el Greedy sin búsqueda local a 2.85 en GRASP sin búsqueda local. Esto representa una reducción del 19.7 % en la flota de vehículos de transporte, lo que tendría un impacto económico importante en una implementación real.
- **Efecto de la búsqueda local:** La aplicación de técnicas de búsqueda local mejora las soluciones, especialmente en el caso del algoritmo Greedy, donde se reduce el número promedio de TV de 3.55 a 3.05 (una reducción del 14.1 %). En el caso de GRASP, la búsqueda local también produce una ligera mejora, aunque menos pronunciada (de 2.85 a 2.78 TV).
- **Tiempo de cómputo:** El coste computacional de aplicar búsqueda local es considerable. El algoritmo Greedy con búsqueda local requiere en promedio 370.61 segundos, lo que representa un aumento de aproximadamente 77 veces respecto a la versión sin búsqueda local. En contraste, GRASP con búsqueda local requiere 56.22 segundos en promedio, que es solo 8.3 veces el tiempo de GRASP sin búsqueda local.
- Es importante destacar que todos los algoritmos implementados fueron paralelizados para aprovechar los recursos computacionales disponibles, lo que contribuyó a la eficiencia general observada en los tiempos de ejecución.

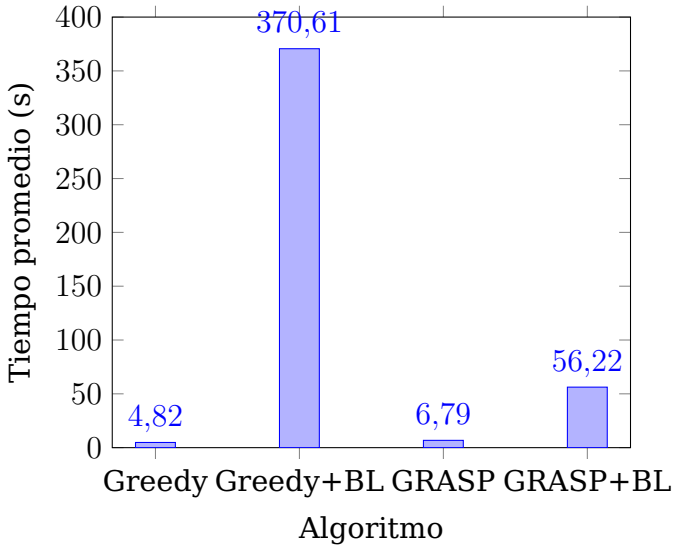

Figura 4.1: Comparativa de tiempo de ejecución entre algoritmos

4.3.2. Análisis de eficiencia

La figura muestra claramente la diferencia en tiempo de cómputo entre los algoritmos. GRASP con búsqueda local ofrece el mejor equilibrio entre calidad de solución y tiempo de ejecución, logrando soluciones de calidad comparable o superior a Greedy con búsqueda local, pero requiriendo solo el 15 % del tiempo de cómputo.

4.3.3. Análisis de calidad de solución

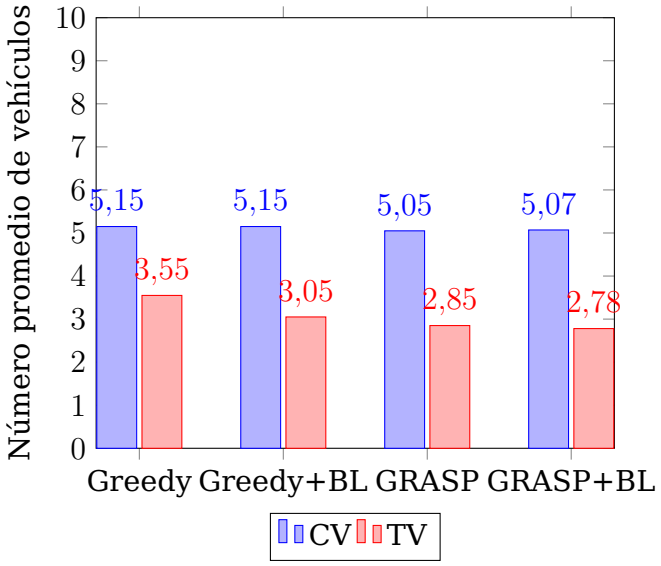

Figura 4.2: Comparativa de número de vehículos entre algoritmos

En términos de calidad de solución, se pueden destacar las siguientes conclusiones:

- GRASP produce soluciones con menor número de vehículos en general, lo que indica su efectividad para este problema.

- La búsqueda local mejora significativamente la calidad de las soluciones obtenidas con el algoritmo Greedy, mientras que para GRASP la mejora es menos pronunciada, sugiriendo que GRASP ya genera soluciones iniciales de buena calidad.
- Para la búsqueda local se implementaron cinco vecindarios diferentes que exploran distintas estrategias de mejora:
  - **2-Opt**: Invierte segmentos de rutas para reducir la distancia total.
  - **Reinserción dentro de rutas**: Reubica zonas de recolección dentro de una misma ruta.
  - **Reinserción entre rutas**: Mueve zonas de recolección de una ruta a otra.
  - **Intercambio dentro de rutas**: Intercambia la posición de dos zonas dentro de una misma ruta.
  - **Intercambio entre rutas**: Intercambia zonas entre diferentes rutas.

  La combinación de estos vecindarios mediante la estrategia RVND (Random Variable Neighborhood Descent) permite una exploración más efectiva del espacio de soluciones.
- La combinación de GRASP con búsqueda local produce las soluciones de mayor calidad en términos del número total de vehículos (7.85 en promedio), lo que lo convierte en el enfoque más recomendable para problemas de este tipo.
- Es importante destacar que todos los algoritmos implementados fueron paralelizados para aprovechar los recursos computacionales disponibles, lo que contribuyó a la eficiencia general observada en los tiempos de ejecución.

4.3.4. Conclusiones del análisis comparativo

Del análisis realizado se pueden extraer las siguientes conclusiones:

1. El algoritmo GRASP supera consistentemente al algoritmo Greedy en la calidad de las soluciones, tanto con como sin búsqueda local.
2. La aplicación de técnicas de búsqueda local mejora significativamente la calidad de las soluciones, especialmente para el algoritmo Greedy.
3. GRASP con búsqueda local ofrece el mejor equilibrio entre calidad de solución y tiempo de cómputo, siendo la opción más recomendable para abordar el problema VRPT en la práctica.
4. El mayor coste computacional del algoritmo Greedy con búsqueda local no se justifica dado que no produce mejores soluciones que GRASP con búsqueda local, que es significativamente más rápido.
5. La paralelización aplicada a todos los algoritmos permitió un uso eficiente de los recursos de cómputo, reduciendo significativamente los tiempos de ejecución que serían considerablemente mayores en implementaciones secuenciales.

En resumen, para el problema VRPT estudiado, el enfoque GRASP combinado con técnicas de búsqueda local proporciona los mejores resultados, ofreciendo un compromiso adecuado entre calidad de solución y eficiencia computacional. La implementación paralela de los algoritmos resultó clave para obtener tiempos de respuesta razonables, especialmente en los métodos que incorporan búsqueda local.

Capítulo 5

Conclusiones y trabajo futuro

5.1. Conclusiones

En este trabajo se ha abordado el Problema de Rutas de Vehículos con Trasbordos para la recogida de residuos sólidos con estaciones de transferencia (VRPT-SWTS), un problema de optimización combinatoria complejo y de gran relevancia práctica. A continuación, se presentan las principales conclusiones obtenidas:

- **Complejidad del problema:** El VRPT-SWTS representa una extensión significativamente más compleja del problema clásico de rutas de vehículos (VRP) debido a la necesidad de coordinar dos tipos de vehículos (CV y TV) y gestionar la sincronización en las estaciones de transferencia.
- **Eficacia de los algoritmos metaheurísticos:** Los resultados experimentales muestran que los algoritmos metaheurísticos (GRASP y GVNS) proporcionan soluciones de mejor calidad que el algoritmo voraz constructivo, justificando el mayor esfuerzo computacional que requieren.
- **Importancia de la diversificación:** La fase constructiva de GRASP, al introducir aleatoriedad controlada, permite explorar un espacio de soluciones más amplio, lo que resulta en soluciones iniciales de mejor calidad que luego pueden ser mejoradas mediante búsqueda local.
- **Eficacia de múltiples estructuras de vecindad:** El algoritmo GVNS, al utilizar múltiples estructuras de vecindad, demuestra una mayor capacidad para escapar de óptimos locales y encontrar soluciones de mejor calidad.
- **Balance entre calidad y tiempo:** Existe un compromiso claro entre la calidad de las soluciones y el tiempo de ejecución. El algoritmo voraz es significativamente más rápido pero produce soluciones de menor calidad, mientras que GVNS requiere más tiempo pero genera soluciones notablemente mejores.

5.2. Trabajo Futuro

A partir de los resultados obtenidos y las limitaciones identificadas, se proponen las siguientes líneas de trabajo futuro:

- **Hibridación de metaheurísticas:** Explorar la combinación de diferentes metaheurísticas, como algoritmos genéticos o búsqueda tabú, con las estructuras de vecindad desarrolladas para potencialmente mejorar la calidad de las soluciones.
- **Consideración de aspectos dinámicos:** Extender el modelo para considerar aspectos dinámicos del problema, como tiempos de viaje variables según la hora del día o demandas estocásticas.
- **Optimización multiobjetivo:** Desarrollar algoritmos que consideren múltiples objetivos simultáneamente, como minimizar el número de vehículos, la distancia total recorrida y el impacto ambiental.
- **Integración con sistemas de información geográfica:** Incorporar datos reales de redes viales y tráfico para mejorar la precisión de las soluciones en escenarios reales.
- **Desarrollo de métodos exactos:** Investigar la posibilidad de desarrollar métodos exactos eficientes para instancias de tamaño pequeño a mediano, que puedan servir como referencia para evaluar la calidad de las soluciones heurísticas.

En resumen, el trabajo realizado proporciona una base sólida para abordar el problema VRPT-SWTS, pero existen numerosas oportunidades para mejorar y extender los métodos desarrollados, tanto desde una perspectiva algorítmica como de aplicación práctica.

Bibliografía

Ghiani, Gianpaolo et al. (2021). “Optimizing a waste collection system with solid waste transfer stations”. En: *Computers Industrial Engineering* 161, pág. 107618. issn: 0360-8352. doi: https://doi.org/10.1016/j.cie.2021.107618.