Vehicle Routing Problem utilizando Algoritmos Metaheurísticos

Andrea Sofia Herrera Coss y León, Fernando de Jesus Rivera Reos

CENTRO UNIVERSITARIO DE CIENCIAS

EXACTAS E INGENIERÍAS, (CUCEI, UDG)

andrea.herrera4198@alumnos.udg.m

x

fernando.rivera8827@alumnos.udg.mx

***Abstract*— El Vehicle Routing Problem (VRP) es un problema clásico en logística y optimización de rutas para una flota de vehículos, considerando demandas de clientes y restricciones de capacidad y tiempo. Este documento presenta una revisión del VRP y sus variantes, así como los métodos de resolución incluyendo heurísticas, metaheurísticas y programación lineal entera (ILP). Se destacan las ventajas de optimización de costos y eficiencia, y las desventajas relacionadas con la complejidad computacional. Se profundiza en los algoritmos metaheurísticos, particularmente los algoritmos genéticos, describiendo su funcionamiento y aplicaciones. Además, se desarrolla un algoritmo genético para resolver el VRP, detallando las etapas de carga de datos, cálculo de distancias, función de evaluación, ajuste de la población y evolución mediante selección, cruce y mutación. Los resultados muestran que los algoritmos de Fuerza Bruta y Programación Dinámica encuentran rutas óptimas, mientras que el enfoque Greedy es eficiente en problemas de pequeña escala. La implementación del algoritmo genético demuestra ser efectiva, destacándose por su capacidad de ajustarse a restricciones de capacidad y evitar nodos repetidos, proporcionando soluciones factibles y eficientes. La calidad de las soluciones depende de la configuración de parámetros como el tamaño de la población y el número de iteraciones.**

***Palabras claves* – Análisis de Algoritmos, Algoritmos Metaheurísticos, Algoritmos Geneticos.**

Repositorio de código:
FernandooRR/VRP_Algoritmos_SwiftieWest (github.com)
Versión actual del código: 1.2
Licencia legal código:

I. INTRODUCCIÓN PROBLEMA

El Vehicle Routing Problem (VRP) es un problema clásico en el campo de la investigación operativa y logística. Se centra en la optimización de las rutas de una flota de vehículos que deben entregar o recoger bienes en varios lugares.

1. Descripción del Problema:

- Dado un conjunto de clientes (puntos de entrega o recogida) y un depósito (donde los vehículos comienzan y terminan sus rutas), el objetivoes encontrar las rutas óptimas para los vehículos.
- Cada cliente tiene una demanda (cantidad de bienes a entregar o recoger) y puede haber restricciones como capacidad de carga de los vehículos y ventanas de tiempo en las que se deben realizar los servicios.

2. Variantes del VRP:

- VRP Clásico: Un solo vehículo que visita todos los clientes y regresa al depósito.
- VRP con Capacidad: Múltiples vehículos con restricciones de capacidad.
- VRP con Ventanas de Tiempo: Los servicios deben realizarse dentro de ciertos intervalos de tiempo.
- VRP con Múltiples Depósitos: Vehículos que comienzan y terminan en diferentes depósitos.

3. Métodos de Resolución:

- Heurísticas: Algoritmos aproximados que encuentran soluciones rápidas pero no garantizan la óptima.
- Metaheurísticas: Algoritmos más avanzados (como algoritmos genéticos o búsqueda tabú) que exploran el espacio de soluciones de manera más eficiente.
- Programación Lineal Entera (ILP): Modelos matemáticos precisos que pueden resolver instancias pequeñas exactamente.

4. Ventajas y Desventajas:

Ventajas:

- Optimización de rutas para reducir costos y tiempo de entrega.
- Mejora en la eficiencia del servicio y satisfacción del cliente.

Desventajas:

- Complejidad computacional en instancias grandes.
- Soluciones no siempre óptimas debido a aproximaciones heurísticas.

5. Aplicaciones del VRP:

- Distribución de productos.
- Recolección de residuos.
- Mantenimiento de servicios (por ejemplo, reparación de equipos).

El VRP sigue siendo un área de investigación activa, con nuevos desarrollos que buscan incorporar más aspectos de la vida real y mejorar las metodologías de solución.

II. INVESTIGACIÓN ALGORITMOS METAHEURISTICOS

Los algoritmos metaheurísticos son técnicas avanzadas utilizadas para resolver problemas de optimización complejos que pueden ser difíciles de abordar con métodos exactos o tradicionales. A continuación, te explico en detalle qué son, sus características, y algunos ejemplos comunes.

A. Definición

Una metaheurística es un método de alto nivel que guía a otros algoritmos para explorar y explotar el espacio de soluciones en busca de la óptima (o una buena solución) en un tiempo razonable. No garantizan encontrar la solución óptima, pero suelen proporcionar buenas soluciones dentro de un tiempo práctico.

B. Características Principales

1.- Exploración y Explotación:

- Exploración (Exploration): La capacidad de buscar en diferentes regiones del espacio soluciones.
- Explotación (Exploitation): La capacidad de intensificar la búsqueda alrededor de las mejores soluciones encontradas hasta ahora.

2.- Adaptabilidad:

- Los algoritmos metaheurísticos son generalmente adaptables y pueden aplicarse a una amplia variedad de problemas sin cambios significativos en su estructura.

Probabilísticos:

- Emplean decisiones aleatorias en ciertos puntos del algoritmo, lo que ayuda a evitar caer en óptimos locales.

No Determinísticos:

- Debido a su componente aleatorio, pueden producir diferentes soluciones en diferentes ejecuciones.

Uso de Memoria:

- Muchos algoritmos metaheurísticos utilizan alguna forma de memoria para almacenar información sobre las soluciones ya evaluadas, lo que ayuda a mejorar la eficiencia de la búsqueda.

C. Tipos de Metaheurísticas

Las metaheurísticas se dividen generalmente en dos grandes categorías: mono-poblacionales y multi-poblacionales.

1.- Mono-poblacionales (basadas en trayectoria):

- Recocido Simulado (Simulated Annealing): Se basa en el proceso de enfriamiento lento de los metales. Empieza con una solución aleatoria y realiza pequeños cambios aleatorios, aceptando cambios que mejoran la solución o, con cierta probabilidad, aquellos que la empeoran, para escapar de los óptimos locales.
- Búsqueda Tabú (Tabu Search): Utiliza una lista "tabú" para recordar soluciones recientes y evitar ciclos, permitiendo la búsqueda en áreas nuevas del espacio de soluciones.

2.- Multi-poblacionales (basadas en población):

- Algoritmos Genéticos (Genetic Algorithms): Se inspiran en la evolución biológica, usando operaciones como la selección, el cruce (crossover), y la mutación para generar nuevas soluciones.
- Optimización por Enjambre de Partículas (Particle Swarm Optimization): Se basa en el comportamiento social de los animales, como bandadas de pájaros, donde cada solución (partícula) ajusta su posición en el espacio de soluciones basada en su experiencia y la de sus vecinos.
- Optimización por Colonia de Hormigas (Ant Colony Optimization): Se inspira en el comportamiento de las hormigas buscando comida. Las hormigas depositan feromonas para guiar a otras hormigas hacia soluciones prometedoras.

D. Algoritmos Genéticos

Los algoritmos genéticos (AG) son una clase de algoritmos de optimización inspirados en los principios de la selección natural y la genética. Fueron desarrollados por John Holland en los años 60 y 70 y han sido utilizados para resolver una amplia variedad de problemas de optimización y búsqueda.

Los AG simulan el proceso de evolución biológica utilizando conceptos como selección, cruce

(crossover), y mutación. A continuación, se describen los componentes y el funcionamiento básico de los algoritmos genéticos:

1. Representación de la Solución (Codificación)

Las soluciones a un problema se representan como individuos (cromosomas) dentro de una población. La representación más común es una cadena de bits (0s y 1s), aunque también se pueden usar otras representaciones como números enteros, reales, o estructuras más complejas.

2. Población Inicial

Se crea una población inicial de soluciones de forma aleatoria. Esta población puede tener un tamaño fijo determinado por el usuario.

3. Función de Aptitud (Fitness Function)

Cada individuo de la población se evalúa mediante una función de aptitud que mide qué tan buena es la solución en relación con el objetivo del problema. La función de aptitud proporciona una forma de comparar y seleccionar las mejores soluciones.

4. Selección

Se seleccionan individuos de la población actual para reproducirse y crear la próxima generación. Los métodos de selección más comunes incluyen:

- Ruleta de selección (Roulette Wheel Selection): Los individuos se seleccionan con una probabilidad proporcional a su aptitud.
- Torneo: Se seleccionan al azar grupos de individuos y el mejor de cada grupo es elegido para reproducirse.
- Selección por rango: Los individuos se clasifican según su aptitud, y se les asignan probabilidades de selección según su clasificación.

5. Operadores Genéticos

Cruce (Crossover): Dos individuos (padres) se combinan para producir uno o más descendientes. Los puntos de cruce pueden ser únicos o múltiples, determinando cómo se combinan las partes de los padres.

Mutación: Se aplican pequeñas modificaciones aleatorias a los individuos (descendientes) para mantener la diversidad genética y explorar nuevas áreas del espacio de soluciones. La mutación suele ser una inversión de bits en la representación binaria.

6. Reemplazo

Los descendientes generados reemplazan a algunos o a todos los individuos de la población actual, creando una nueva generación. Existen varios esquemas de reemplazo, como el reemplazo generacional (donde toda la población se reemplaza) o reemplazo elitista (donde los mejores individuos de la generación anterior se conservan).

7. Criterio de Parada

El algoritmo se ejecuta iterativamente hasta que se cumple un criterio de parada, que puede ser un número fijo de generaciones, una mejora mínima en la aptitud, o un tiempo de ejecución máximo.

Ventajas:

- Robustez: Los AG son robustos y pueden adaptarse a una amplia variedad de problemas.
- Paralelismo: Pueden aprovechar el paralelismo inherente, evaluando múltiples soluciones simultáneamente.
- Exploración global: Tienen una buena capacidad para explorar el espacio de búsqueda global y evitar caer en óptimos locales.

Desventajas:

- Coste computacional: Pueden requerir una gran cantidad de evaluaciones de la función de aptitud, lo que puede ser costoso en problemas complejos.
- Parámetros sensibles: Su rendimiento depende de la correcta configuración de parámetros como el tamaño de la población, la tasa de cruce y mutación.
- Convergencia lenta: En algunos casos, pueden converger lentamente hacia la solución óptima.

Los AG se utilizan en una amplia gama de áreas, incluyendo:

- Optimización de funciones: Para encontrar máximos o mínimos de funciones matemáticas complejas.
- Problemas de diseño: En ingeniería para optimizar diseños estructurales o de sistemas.
- Inteligencia artificial: En la evolución de redes neuronales y algoritmos de aprendizaje.
- Bioinformática: Para alineamiento de secuencias y modelado de estructuras proteicas.
- Logística: Para la optimización de rutas y la planificación de tareas.

III. Descripción del Desarrollo

A. Solución Fuerza Bruta

El algoritmo de fuerza bruta genera todas las permutaciones posibles de las ubicaciones (exceptuando el punto de inicio) y calcula la distancia total para cada ruta. Selecciona la ruta con la distancia mínima.

B. Solución Divide y Vencerás

El algoritmo divide la lista de ubicaciones en dos mitades, resuelve recursivamente el problema para cada mitad, y luego combina los resultados.

C. Solución Programación Dinámica

Este algoritmo utiliza una tabla de memorización para almacenar resultados intermedios y evitar cálculos repetidos. Calcula la ruta óptima considerando todas las posibles combinaciones de visitas a las ciudades.

D. Solución Técnica Voraz

El enfoque Greedy selecciona iterativamente la ciudad más cercana no visitada hasta que todas las ciudades hayan sido visitadas.

E. Solución Metaheurística

Esta parte del programa implementa un algoritmo genético para resolver el Problema del Enrutamiento de Vehículos (VRP, por sus siglas en inglés). A continuación, se describe cada función y su papel dentro del algoritmo genético.

1.- Carga de Datos

2.- Lectura del Archivo CSV

```
def read_csv(filepath):
```

Esta función lee un archivo CSV y carga los datos del VRP en un diccionario `vrp`. Verifica la validez de los datos, asegurándose de que las demandas y la capacidad del vehículo sean correctas.

3.- Cálculo de la Distancia

```
def distance(n1, n2):
    dx = n2['posX'] - n1['posX']
    dy = n2['posY'] - n1['posY']
    return math.sqrt(dx * dx + dy * dy)
```

Calcula la distancia euclidiana entre dos nodos dados.

4.- Función de Evaluación

```
def fitness(p):
    s = distance(vrp['nodes'][0], vrp['nodes'][p[0]])
    for i in range(len(p) - 1):
        prev = vrp['nodes'][p[i]]
        next = vrp['nodes'][p[i + 1]]
        s += distance(prev, next)
    s += distance(vrp['nodes'][p[len(p) - 1]], vrp['nodes'][0])
    return s
```

Calcula la longitud total de la ruta representada por la secuencia `p`, sumando las distancias entre nodos consecutivos y volviendo al depósito.

5.- Ajuste de la Población

```
def adjust(p):
```

Ajusta la ruta `p` para asegurar que no haya nodos repetidos y que se respeten las restricciones de capacidad del vehículo. Inserta paradas en el depósito según sea necesario.

6.- Inicialización y Evolución de la Población

```
popsize = 50
iterations = 100
```

```
pop = []

for i in range(popsize):

    p = list(range(1, len(vrp['nodes'])))

    random.shuffle(p)

    pop.append(p)

for p in pop:

    adjust(p)
```

Inicializa la población con `popsize` individuos, cada uno representando una posible solución al VRP. Cada individuo es una permutación aleatoria de los nodos, ajustada para cumplir las restricciones.

7.- Evolución del Algoritmo Genético

Evolución de la Población

Durante cada iteración del algoritmo genético, se sigue el siguiente proceso:

a. Selección por Torneo:

- Se seleccionan cuatro individuos aleatoriamente de la población.
- Se eligen dos padres de entre estos cuatro, seleccionando los que tengan mejor fitness (menor costo).

b. Cruce (Crossover):

- Se eligen dos puntos de corte aleatorios.
- Se generan dos hijos mezclando segmentos de los padres seleccionados.

c. Mutación Aleatoria:

- Con una probabilidad del 1/15, se selecciona un individuo al azar de la nueva población.
- Se intercambian dos nodos al azar en este individuo para introducir variabilidad.

Ajuste de los Hijos:

- Cada hijo generado es ajustado usando la función adjust() para asegurar que sea una solución válida.

Actualización de la Población:

- La nueva población reemplaza a la anterior para la siguiente iteración.

Selección de la Mejor Solución:

- Después de todas las iteraciones, se evalúa toda la población final.
- Se selecciona la solución con el menor costo total (mejor fitness) como la mejor solución encontrada.

Salida de Resultados:

La mejor ruta encontrada y su costo total se imprimen al final del programa.

8.- Selección de la Mejor Solución

```
better = None

bf = float('inf')

for p in pop:

    f = fitness(p)

    if f < bf:

        bf = f

        better = p
```

Selecciona la mejor solución de la población final, evaluando todas y eligiendo la de menor costo.

9.- Salida de Resultados

```
print('route:')

print('depot')

for nodeIdx in better:

    print(vrp['nodes'][nodeIdx]['label'])

print('depot')

print('cost:')

print('%f' % bf)
```

Imprime la mejor ruta encontrada y su costo total.

IV. Resultados Obtenidos del Proyecto

En este ejercicio, el algoritmo de Fuerza Bruta y el de Programación Dinámica lograron encontrar la ruta óptima para el problema dado, con una distancia mínima de 95. El enfoque Greedy también encontró una solución óptima, mientras que el método de Divide y Vencerás presentó una solución subóptima en comparación. Cada método tiene sus ventajas y desventajas en términos de complejidad computacional y eficiencia, siendo la Programación Dinámica y el enfoque Greedy los más eficientes para problemas de esta escala.

Los módulos implementados en el programa, retornaron a la interfaz gráfica soluciones que se pueden etiquetar como ‘factibles’, destacando la solución metaheurística.

V. Conclusiones

Este algoritmo genético proporciona una solución efectiva para el VRP, ajustando rutas para cumplir con las restricciones de capacidad y evitando nodos repetidos. La implementación muestra cómo combinar técnicas de selección, cruce y mutación para evolucionar la población de soluciones y encontrar rutas eficientes. La calidad de la solución depende de la configuración de los parámetros del algoritmo, como el tamaño de la población y el número de iteraciones.

Así como fue mencionado, los módulos desarrollados para el programa otorgaron soluciones que se pueden etiquetar como ‘factibles’, destacando la solución metaheurística, que rescata dos de las estrategias clásicas, que son técnica voraz para ajustar la ruta, y programación dinámica para evolucionar la población. Dicho esto, no utiliza las demas, debido a que los otros enfoques pueden llegar a ser un derroche de recursos en materia de memoria.

Referencias

[1] F. Herrera, “Introducción a los Algoritmos Metaheurísticos Metaheurísticos.” Available: https://sci2s.ugr.es/sites/default/files/files/Teaching/OtherPostGraduateCourses/Metaheuristicas/Int-Metaheuristicas-CAEPIA-2009.pdf

[2] “Algoritmos genéticos,” Conogasi, Sep. 21, 2018. https://conogasi.org/articulos/algoritmos-geneticos/

[3] Gilles Brassard, P. Bratley, Luis Joyanes Aguilar, Narciso Martí Oliet, and PeñaR., Fundamentos de algoritmia. Madrid Etc: Prentice Hall, 2004.

[4] A. Levitin, Introduction To Design And Analysis Of Algorithms, 2/E. Pearson Education India, 2008.

[5] [1]R. Sedgewick and Philippe Flajolet, An Introduction to the Analysis of Algorithms. Addison-Wesley, 2013.