Complejidad de Enrutamiento de Suministros en Desastres Naturales: Un Análisis de NP-Completitud y un Algoritmo Greedy

Introducción

La rápida y eficiente distribución de suministros a las zonas afectadas por desastres naturales es crucial para mitigar el sufrimiento y salvar vidas. Este problema, aparentemente sencillo, se complica por diversas restricciones, como la capacidad limitada de los vehículos, las ventanas de tiempo para las entregas y la necesidad de minimizar los costos operativos. En este documento, se formaliza este problema como un problema de optimización combinatoria y se demuestra su pertenencia a la clase de problemas NP-completos. Además, se presenta un algoritmo greedy para abordar este desafío y se analiza su complejidad temporal.

1. El problema

El problema seleccionado es el de repartir determinados suministros a zonas afectadas por algún desastre natural (ciclón, terremoto, incendio). Disponemos de centros logísticos desde los cuales se pueden enviar vehículos para cubrir las necesidades de las zonas afectadas y existen varias restricciones como por ejemplo: la capacidad de carga de los vehículos y los intervalos de tiempo en los cuales es efectivo entregar a una zona recursos o no. El movimiento de los vehículos tiene un costo y nuestro objetivo es maximizar la eficacia de las entregas mientras minimizamos el costo de las operaciones realizadas.

1.1. Formalización del problema

Podemos abstraer el problema y formalizarlo a partir de la siguiente construcción:

- Un grafo dirigido $G = (V, E)$ con un vértice v_0 (centro logístico).
- Un conjunto $Z \subseteq V \setminus \{v_0\}$ de zonas afectadas, cada una con:
  - Demanda $d_z \in \mathbb{N}$ (suministros requeridos).
  - Ventana de tiempo $[a_z, b_z]$ (tiempo válido para la entrega).

- m vehículos con capacidad máxima $Q \in \mathbb{N}$.
- Costos $c_e \in \mathbb{N}$ para cada arista $e \in E$.
- Tiempos de traslado $t_e \in \mathbb{N}$ para cada arista $e \in E$.
- Un valor $K \in \mathbb{N}$.

Pregunta: ¿Existe un conjunto de m rutas (ciclos que parten y terminan en v_0) tal que:

1. Cada zona $z \in Z$ es visitada exactamente una vez por ruta.
2. La suma de demandas en cada ruta no excede Q.
3. Cada zona z es atendida dentro de su ventana $[a_z, b_z]$.
4. El costo total de todas las rutas es $\leq K$?

2. Demostración de que el problema es NP completo

Primero, demostremos que el problema es NP:
Una solución candidata consiste en un conjunto de rutas para los vehículos. Para verificar su validez en tiempo polinomial:

1. **Conexión y depósito**: Cada ruta inicia y termina en v_0.
2. **Cobertura**: Todas las zonas $z \in Z$ están incluidas exactamente una vez.
3. **Capacidad**: Para cada ruta, la suma de d_z no supera Q.
4. **Ventanas de tiempo**: Para cada zona z, el tiempo de llegada está en $[a_z, b_z]$.
5. **Costo total**: La suma de los costos de las aristas usadas es $\leq K$.

Cada verificación requiere tiempo lineal en el tamaño de la entrada, por lo que el problema es NP.

2.1. Reducción polinomial desde 3-Partition

3-Partition: Dados $3m$ enteros $a_1, \ldots, a_{3m}$, y un entero B, donde $\sum_{i=1}^{3m} a_i = mB$, ¿existe una partición de los enteros en m grupos de 3 elementos, cada uno sumando B?
Reducción al problema:

1. **Zonas**: Crear $3m$ zonas, cada una con demanda $d_z = a_i$.
2. **Vehículos**: m vehículos con capacidad $Q = B$.

3. **Grafo**: Conectar todas las zonas directamente al depósito v_0 y entre sí.
4. **Tiempos**:
   - Viaje $v_0 \to z$: tiempo 1.
   - Viaje $z \to v_0$: tiempo 1.
   - Viaje entre cualquier par de zonas: tiempo 1.
   - Ventanas de tiempo para cada zona z: $[0, 3]$.
5. **Costo**: $c_e = 0$ para todas las aristas.
6. **Umbral**: $K = 0$.

Equivalencia de la reducción: Si existe una 3-Partición, cada triple suma B, y se asignan a rutas de vehículos con capacidad $Q = B$. Si existe solución al problema de rutas:

- Existen m rutas donde en cada una se visita exactamente 3 zonas, pues la restricción de las ventanas de tiempo entre $[0, 3]$ impide que se visiten más de 3 en una ruta.
- Además, la suma por ruta debe ser exactamente B, pues la restricción de $Q = B$ garantiza que no se pueda transportar demanda mayor que B y la suma de todas las demandas es mB entonces cada ruta ha de transportar exactamente B unidades de demanda.

Como la reducción es polinomial el problema es NP-Hard. Luego al ser además NP, es NP completo.

3. Algoritmo greedy propuesto

Nuestro algoritmo recibe como entrada una matriz de costos de moverse entre zonas, un array con información acerca de los clientes (id, demanda, ventana de tiempo y tiempo de servicio), la capacidad homogénea de los vehículos y un parámetro extra que representa un costo fijo asociado a cada vehiculo empleado.

Input: Matriz de costos, clientes, capacidad del vehículo, costo fijo
Output: Rutas, costo total
// PASO 1: Cálculo de ahorros
savings = ListaVacía()
for *cada i en customers* **do**
    for *cada j en customers* **do**
        if *i.id* $\neq$ *j.id* **then**
            $costo_{i0}$ = cost_matrix[i.id, 0]
            $costo_{0j}$ = cost_matrix[0, j.id]
            $costo_{ij}$ = cost_matrix[i.id, j.id]
            ahorro = $costo_{i0} + costo_{0j} - costo_{ij}$
            Agregar (i.id, j.id, ahorro) a savings
        end
    end
end

// PASO 2: Ordenar los ahorros
Ordenar savings en orden descendente por ahorro
// Funciones auxiliares
Función *get_customer(cid)*
    for *cada c en customers* **do**
        if *c.id == cid* **then**
            Retornar c
        end
    end
Función *calculate_route_time(route, cost_matrix)*
    current_time = 0
    for *i desde 1 hasta Longitud(route) - 1* **do**
        from_node = route[i-1]
        to_node = route[i]
        travel_time = cost_matrix[from_node, to_node]
        current_time += travel_time
        if *to_node* $\neq$ *0* **then**
            customer = get_customer(to_node)
            if *current_time < customer.time_window[0]* **then**
                current_time = customer.time_window[0]
            end
            if *current_time > customer.time_window[1]* **then**
                Retornar Infinito
            end
            current_time += customer.service_time
        end
    end
    Retornar current_time

```
// PASO 3: Inicializar rutas
routes = ListaVacía()
for cada customer en customers do
    ruta = { nodes: [0, customer.id, 0], demand: customer.demand,
     time: calculate_route_time([0, customer.id, 0], cost_matrix) }
    Agregar ruta a routes
end

// PASO 4: Combinación de rutas
for cada (i, j, _) en savings do
    route_i = Nulo
    route_j = Nulo
    for cada r en routes do
        if r.nodes[-2] == i then
            route_i = r
        end
        if r.nodes[1] == j then
            route_j = r
        end
    end
    if route_i ≠ Nulo y route_j ≠ Nulo y route_i ≠ route_j then
        demanda_combinada = route_i.demand + route_j.demand
        if demanda_combinada ≤ vehicle_capacity then
            new_route = Concatenar(route_i.nodes[0:-1],
             route_j.nodes[1:])
            new_time = calculate_route_time(new_route, cost_matrix)
            if new_time ≠ Infinito then
                Remover route_i y route_j de routes
                Agregar nueva_ruta a routes
            end
        end
    end
end

// PASO 5: Cálculo de resultados
total_cost = 0
for cada ruta en routes do
    for i desde 0 hasta Longitud(ruta.nodes)-2 do
        total_cost += cost_matrix[ruta.nodes[i], ruta.nodes[i+1]]
    end
end
total_cost += fixed_cost × Longitud(routes)
return (routes, total_cost)
```

4. Análisis de complejidad temporal

Analicemos por separado la complejidad temporal de cada paso:

- **Paso 1**: El paso 1 es el cálculo de los ahorros. Existen aproximadamente n^2 tuplas (i,j) y por cada una se realiza una cantidad constante de operaciones por lo que este paso es $O(n^2)$.
- **Paso 2**: El paso 2 consiste en ordenar de forma descendente los ahorros y conocemos que ordenar una lista con k elementos es $O(klogk)$, por lo que este paso es $O(n^2logn)$.
- **Paso 3**: El paso 3 es la inicialización de las rutas. En este momento se crean n rutas y la creación de cada una requiere una cantidad constante de operaciones, por lo cual esto es $O(n)$.
- **Paso 4**: En el paso 4 se realiza la combinación de las rutas. En principio tenemos $O(n^2)$ ahorros y por cada uno se tratan de combinar rutas que pueden tener longitud como máximo cercanas a n. Cada vez que se van a combinar rutas **a** y **b** se realiza una verificación sobre cada uno de los nodos en **b**, por lo cual en el caso peor tenemos una complejidad de $O(n^3)$.
- **Paso 5**: En el paso 5 se realiza una iteración por los elementos de cada ruta para calcular los costos totales, por lo cual esta parte es $O(n)$.

Costo total del algoritmo: $\mathbf{O(n^3)}$.

Conclusiones

En este trabajo, hemos abordado el problema crítico de la distribución de suministros en zonas afectadas por desastres naturales. La formalización del problema reveló su naturaleza compleja, demostrando que pertenece a la clase NP-completa. Esta conclusión implica que no se conocen algoritmos eficientes (de tiempo polinomial) para encontrar la solución óptima en todos los casos, a menos que P=NP.

Frente a esta dificultad teórica, propusimos un algoritmo greedy como un enfoque práctico para obtener soluciones subóptimas en un tiempo razonable. El análisis de la complejidad temporal del algoritmo greedy reveló que tiene una complejidad de $O(n^3)$, lo que lo hace adecuado para problemas de tamaño moderado.