Problema original

Optimización de Ayuda Humanitaria en el Huracán Aurora

En septiembre de 2023, la región costera de un país ficticio llamado **Costa del Sol** fue devastada por el **Huracán Aurora**, una tormenta de categoría 5 que dejó a su paso inundaciones extremas, carreteras destruidas y más de **50,000 personas afectadas**. La tormenta impactó especialmente a comunidades rurales donde la infraestructura era débil, lo que ocasionó retrasos significativos en la llegada de ayuda humanitaria.

En las primeras **48 horas del desastre**, organizaciones humanitarias como la **Agencia de Respuesta Internacional (ARI)** y las autoridades locales enfrentaron **graves problemas logísticos**:

- **Distribución desigual de recursos:** Centros de distribución enviaron alimentos y medicinas a zonas que no eran las más críticas, mientras que otras áreas quedaron sin suministros esenciales.
- **Infraestructura colapsada:** Las carreteras principales estaban inundadas o bloqueadas por escombros, complicando el transporte y ralentizando la entrega.
- **Falta de predicción de la demanda:** A medida que surgían problemas adicionales, como enfermedades derivadas del agua estancada, la demanda cambió rápidamente, generando un desbalance en la asignación de recursos.

La combinación de estos factores **ralentizó la entrega** de recursos críticos como agua potable, alimentos y medicinas, poniendo **vidas en riesgo** y destacando la necesidad urgente de un sistema eficiente de optimización logística.

La **Agencia de Respuesta Internacional (ARI)** y las autoridades locales enfrentan el desafío de optimizar la distribución de ayuda humanitaria tras el **Huracán Aurora**, con el objetivo de **maximizar la entrega eficiente de recursos** y **minimizar tiempos, costos y riesgos operativos** en una situación crítica. Es necesario determinar qué suministros deben enviarse a cada área priorizando la **gravedad del impacto** y la **población afectada**, mientras se consideran las **limitaciones de transporte y almacenamiento** en centros logísticos para evitar escasez en las zonas más críticas o exceso en áreas menos afectadas. La **infraestructura parcialmente colapsada**, con carreteras inundadas o bloqueadas, obliga a encontrar **rutas alternativas** que reduzcan el tiempo y el costo de transporte para garantizar la entrega rápida de recursos esenciales. Además, es vital anticipar

nuevos riesgos como réplicas sísmicas, inundaciones adicionales o fallos en el transporte y el personal, implementando **planes de contingencia** que minimicen interrupciones en la cadena de suministro.

Subproblemas seleccionados

- Luego del huracán Aurora, algunas calles y caminos fueron dañados, impidiendo el paso de las ayudas humanitarias a las zonas afectadas. Se tiene un presupuesto B, una cantidad de suministros S y un grafo ponderado de la zona afectada. Se tiene un conjunto D, tal que cada D_i representa una zona de desastre y una lista P tal que P_i es la prioridad de la zona i. Se desea saber si existe un conjunto de caminos que se pueden arreglar con el presupuesto B tal que el suministro S llegue a la mayor cantidad de zonas afectadas posibles, priorizando las zonas de mayor demanda.
- La agencia humanitaria tiene varias sucursales distribuidas en el territorio, representadas por el conjunto S. Cada sucursal i posee una flota heterogénea de vehículos F_i, donde cada vehículo j tiene capacidad de carga c_{ij} y consumo d_{ij}. Se dispone de un grafo ponderado que modela el mapa de la región afectada, donde algunos nodos pueden ser zonas afectadas o sucursales y dada una arista e, $w(e)$ representa la distancia del viaje entre los nodos extremos.
  Cada zona afectada D_k, tiene una demanda específica de suministros que debe ser satisfecha exactamente una vez por un único vehículo. Un vehículo asignado debe partir de su sucursal de origen y visitar un conjunto de zonas $\{D_{k_1}, D_{k_2}, ...\}$ y regresar a su sucursal. La suma de las demandas q_k en las zonas visitadas por un vehículo no puede exceder la capacidad c_{ij}. De ser necesario no todos los vehículos deben utilizarse, ya que esto podría minimizar costos. Se desea minimizar el costo de combustible de todos los vehículos, donde el costo esta está como: d_{ij} * distancia recorrida por F_{ij} (vehículo j de la flota i).

Subproblema 1

Nuestro problema se basa en optimizar la llegada de suministros a las zonas afectadas luego del desastre, lo que implica determinar el conjunto de calles que se pueden reparar con el presupuesto disponible y, al mismo tiempo, maximizar la atención a las zonas de mayor prioridad a través de los caminos reparados. Es decir, buscamos identificar un conjunto de calles que, al ser reparadas, permitan conectar el centro de distribución con las zonas críticas, de manera que se alcance a la mayor cantidad posible de áreas afectadas, priorizando aquellas con mayor demanda.

Nota: Es importante aclarar que algunas zonas de desastre, incluso aquellas de alta prioridad, pueden ser accesibles a través de vías que no han resultado dañadas y, por lo tanto, no requieren

reparación. En nuestro modelo, asumimos que sólo se reparan aquellas calles afectadas que impiden la conexión entre el centro de distribución y las zonas críticas. Si una zona ya es accesible mediante vías operativas, se considera automáticamente conectada, y los recursos se destinan exclusivamente a la reparación de los tramos dañados.

Modelo matemático 1

Para formalizar el problema, partimos de un grafo ponderado que representa la red de la zona afectada. Como planteamos anteriormente, el objetivo es determinar cuáles calles reparar (dentro de un presupuesto B) de modo que se conecte el centro de distribución con las zonas de desastre más críticas (priorizadas según un valor P_i). Para ello, definiremos variables que indican la reparación de calles y la atención de zonas, y utilizaremos un modelo de flujo para certificar la conectividad del centro con las zonas que deben ser atendidas.

Dado un grafo $G = (V, E)$, V el conjunto de nodos que incluye intersecciones, centros de distribución y zonas de desastre y E el conjunto de aristas, donde cada arista (i, j) tiene un costo de reparación $c_{i,j}$.

Sea $D \subset V$ el conjunto de zonas de desastre. Para cada zona $i \in D$, se asigna una prioridad P_i; un valor mayor de P_i indica que la zona tiene mayor urgencia de atención.

Existe un nodo fuente $s \in V$ que representa el centro de distribución. Desde este nodo se inyecta el suministro, y se asume que la cantidad de suministros es suficiente para atender a todas las zonas conectadas; por ello, el criterio principal es lograr la conectividad.

Sea B el monto total disponible para reparar calles. La suma de los costos de las calles seleccionadas (aristas reparadas) no debe exceder B.

Variables de Decisión

Para cada $(i, j) \in E$:

- $x_{i,j} \in \{0, 1\}$:
  - $x_{i,j} = 1$ indica que la calle (arista) (i, j) se repara.
  - $x_{i,j} = 0$ indica que la calle no es reparada.

Para cada $i \in D$:

- $y_i \in \{0, 1\}$:
  - $y_i = 1$ indica que la zona de desastre i es atendida (es decir, se conecta al centro de distribución s mediante calles reparadas).

- $y_i = 0$ en caso contrario.

Variables de flujo:

Para cada arco (i, j) (versión dirigida de E):

- $f_{ij} \geq 0$: Representa la cantidad de flujo que circula por la arista (i, j).
  - El flujo solo puede circular por aquellas aristas que se han reparado, lo cual se garantiza mediante restricciones adicionales.

    Nota: Más adelante explicaremos por qué utilizamos un modelo de flujo para garantizar la conectividad y cómo se demuestra que, si se marca una zona como atendida (es decir, si $y_i = 1$), entonces existe un camino de conexión entre el centro de distribución s y dicha zona i a través de las calles reparadas.

Restricciones y Función Objetivo del modelo

Restricción Presupuestaria

La suma de los costos de reparación de las calles seleccionadas no debe exceder el presupuesto:

$$\sum_{(i,j)\in E} c_{i,j}\, x_{i,j} \leq B$$

Restricción de Conectividad mediante Flujo

Para garantizar que, si se decide atender una zona i (es decir, si $y_i = 1$), exista un camino efectivo desde el nodo fuente s hasta i formado únicamente por calles reparadas, incorporamos un modelo de flujo con las siguientes condiciones:

Inyección de Flujo en s:

Se “inyecta” un total de F unidades de flujo en el nodo fuente s. Este valor F se elige de forma que sea suficiente para cubrir las demandas de todos los nodos que se pretenden conectar.

Conservación de Flujo en los Nodos:

Para cada nodo v distinto de s (es decir, $v \in V \setminus \{s\}$), se impone una restricción de conservación de flujo modificada. En un modelo de flujo estándar, la cantidad de flujo que entra en un nodo es igual a la que sale, de modo que el balance neto o "exceso" es cero. Sin embargo, en nuestro caso, queremos que la diferencia neta (flujo que entra menos flujo que sale) sea igual a y_v. Esto significa que:

- Si $y_v = 0$ (la zona v no está marcada para ser atendida), el flujo que entra a v debe ser igual al que sale, es decir, el balance neto es cero.
- Si $y_v = 1$ (la zona v debe ser atendida), el balance neto en v debe ser 1. Esto obliga a que v reciba una unidad extra de flujo en comparación con la que envía, lo que equivale a decir que v tiene una "demanda" de 1 unidad. La única forma de cumplir esta restricción es que exista un camino por el cual la unidad de flujo inyectada en s pueda pueda llegar hasta v.

Para formalizar esto, utilizamos la siguiente ecuación de conservación de flujo para cada nodo $v \neq s$

$$\sum_{i:(i,v)\in E'} f_{iv} - \sum_{j:(v,j)\in E'} f_{vj} = y_v$$

En esta ecuación, E' representa la versión dirigida de las aristas del grafo, y f_{ij}(como planteamos) es la cantidad de flujo que circula por el arco (i, j). Si $y_v = 1$, la ecuación exige que el flujo neto en v sea exactamente 1, lo que implica que 1 unidad de flujo ha sido transportada desde s hasta v.

Sin embargo, para que el flujo pueda circular, debe existir una condición adicional: el flujo solo puede transitar por las calles que se han reparado. Esto se garantiza mediante la restricción:

$$f_{ij} \leq M\, x_{ij}, \quad \forall (i,j) \in E'$$

donde x_{ij} es la variable binaria que vale 1 si la calle (o arco) (i, j) se repara, y M es una constante suficientemente grande para que no limite el flujo cuando la calle esté reparada. Si una calle no se repara, $x_{ij} = 0$, y la restricción impone $f_{ij} = 0$, es decir, no se permite el paso de flujo por ese arco.

Al imponer estas restricciones, el modelo obliga a que, para que se cumpla que un nodo v reciba una unidad neta de flujo (cuando $y_v = 1$), debe existir un camino formado únicamente por calles reparadas que conecte s con v. Si no existiera dicho camino, v no podría acumular la unidad de flujo requerida y la restricción de conservación se violaría, lo que significa que no sería posible marcar v como atendida.

Al diseñar el modelo de esta forma, asumimos únicamente los caminos que contienen calles reparadas, ya que la restricción de capacidad

$$f_{ij} \leq M\, x_{ij}$$

evita que cualquier flujo circule por una calle dañada (o no reparada). Asegurando que la única forma de satisfacer la condición de que el flujo neto en v sea 1 es que se encuentre un camino compuesto exclusivamente por calles que han sido reparadas con el presupuesto disponible.

Función Objetivo

Como hemos comentado el objetivo es maximizar la suma de las prioridades de las zonas conectadas:

$$\max \sum_{i \in D} P_i \, y_i$$

Esta formulación integra la restricción presupuestaria, la garantía de conectividad mediante el flujo (que certifica que las zonas atendidas están conectadas al centro de distribución) y la maximización de la cobertura de zonas de desastre de mayor prioridad.

Reduccion BMC

Para demostrar que nuestro problema es, al menos, tan difícil como Budgeted Maximum Coverage (BMC), un problema conocido por ser NP-hard, realizaremos una reducción polinomial. En esta reducción transformaremos cualquier instancia de BMC en una instancia de nuestro problema de tal manera que resolverlo permita, resolver BMC.

Antes de realizar la reducción debemos demostrar que nuestro problema está en NP ya que para que un problema sea NP-hard, primero debe estar en NP. Esto significa que si tomamos una solución por ejemplo un conjunto de calles reparadas y un conjunto de zonas atendidas, podemos verificar en tiempo polinomial que la solución es válida. Verificar que el costo total de reparación no excede B es $O(m)$ siendo m el número de arista o calles, nos es mas que sumar los costos de reparacion de todas estas aristas. Verificar que cada zona atendida está conectada al centro de distribución puede hacerse con BFS o DFS en $O(n+m)$ siendo n el número de nodos. Y por ultimo la evaluación de las prioridades sería $O(n)$
Como todas estas verificaciones pueden hacerse en tiempo polinomial, nuestro problema pertenece a NP.

Descripción del Problema Budgeted Maximum Coverage (BMC)

En el problema Budgeted Maximum Coverage se tiene un conjunto universo de elementos, denotado por $U = \{u_1, u_2, \ldots, u_n\}$. Cada elemento u tiene un peso $w(u)$. Una colección de subconjuntos $S = \{S_1, S_2, \ldots, S_m\}$ donde cada $S_i \subseteq U$ y cada subconjunto S_i tiene un costo $c(S_i)$ asociado. Un valor B que limita la suma de los costos de los subconjuntos que se pueden seleccionar.
El objetivo es seleccionar una colección de subconjuntos $S' \subseteq S$ tal que la suma de los costos de los subconjuntos elegidos no exceda B

$$\sum_{S \in S'} c(S) \leq B$$

y la suma de los pesos de los elementos cubiertos (es decir, la suma de $w(u)$ para u perteneciente a la unión de los subconjuntos seleccionados) sea máxima.

Reducción de BMC a Nuestro Problema

Nuestro problema se centra en elegir un conjunto de calles (aristas) para reparar con un presupuesto B de modo que se conecte el centro de distribución con las zonas de desastre de mayor prioridad. La idea es transformar una instancia de BMC en una instancia de nuestro problema.

Construimos un grafo $G' = (V', E')$ de la siguiente forma:

- Creamos un nodo fuente s que representa el centro de distribución.
- Para cada subconjunto S_i en la colección, creamos un nodo intermedio s_i y se añade una arista (s, s_i) con costo $c(s, S_i) = c(S_i)$ (es decir el costo del subconjunto $S_i$en la instancia de BMC). La idea es que, si se "repara" la calle que conecta s con s_i se estará seleccionando el subconjunto S_i en la solución.
- Para cada elemento u del universo U, creamos un nodo $v(u)$ ó $u \in V'$. Estos nodos representarán las "zonas de desastre" y se les asigna una prioridad $P(v(u))$ igual al peso $w(u)$ del elemento u
- Para cada elemento u y para cada subconjunto S_i que contenga a u (es decir, $u \in S_i$) se añade una arista desde el nodo s_i al nodo $v(u)$ con costo cero (o, en la práctica, un costo despreciable). Estas aristas no afectan el presupuesto, pero permiten saber que si se selecciona S_i (si se "repara" (s, s_i)), todos los nodo $v(u)$ correspondientes a los elemntos de S_i se conecten al centro s.

Según esta construcción podemos decir que reparar una arista (s, s_i) equivale a seleccionar el subconjunto S_i de la instancia de BMC, ya que ello implica que los nodos $v(u)$ correspondientes a los elementos $u \in S_i$ quedarán conectados al nodo fuente s, estas aristas de costo cero no afectan el presupuesto y garantizan que, una vez seleccionada la arista (s, s_i), todos los nodos $v(u)$ queden conectados al nodo fuente s.
El presupuesto B se aplica a la suma de los costos de las aristas (s, s_i) que es exactamente la suma de los costos de los subconjuntos seleccionados. Y el objetivo de maximizar la suma de las prioridades de las zonas conectadas se corresponde con maximizar la suma de los pesos de los elementos cubiertos en BMC.

Demostremos que:

Si existe una solución para BMC, entonces existe una solución para nuestro problema

Supongamos que en la instancia de BMC se selecciona una colección de subconjuntos $S' \subseteq S$ tal que el costo total

$$\sum_{S_i \in S'} c(S_i) \leq B$$

y la suma de los pesos de los elementos cubiertos (la suma de $w(u)$ para u en la unión de los subconjuntos seleccionados) es W

En nuestra construcción, para cada subconjunto $S_i \in S'$, repararemos la arista (s, s_i). Esto implica que, a través de las aristas de costo cero, todos los nodos $v(u)$ correspondientes a los elementos u que pertenecen a algún $S_i \in S'$ quedarán conectados al nodo fuente s. Por lo tanto, en la solución de nuestro problema, se obtendrá que para cada elemento u cubierto por la solución de BMC, el correspondiente nodo $v(u)$ estará atendido (se asignará $y_{v(u)} = 1$). La suma de las prioridades de los nodos atendidos será, entonces, W.

Además, el costo total para reparar las calles (s, s_i) es exactamente la suma de los costos $c(S_i)$ para $S_i \in S'$, la cual no excede B. Así, la solución factible para BMC se transforma en una solución factible para nuestro problema con el mismo presupuesto y un valor objetivo igual a la suma de los pesos de los elementos cubiertos.

Si existe una solución para nuestro problema, entonces existe una solución para BMC

Supongamos que en nuestro problema se ha obtenido una solución en la que se repara un conjunto de aristas de modo que el nodo fuente s queda conectado con un conjunto de nodos $v(u)$ (cada uno correspondiente a un elemento u de U). Llamemos S' al conjunto de nodos intermedios s_i para los cuales la arista (s, s_i) fue reparada. La reparación de (s, s_i) significa que se “selecciona” el subconjunto S_i correspondiente.

Debido a que, mediante las aristas de costo cero, los nodos $v(u)$ se conectan a s solo si existen algunos s_i que los cubren, la colección de subconjuntos S' cubre los elementos correspondientes a los nodos $v(u)$ conectados. Además, el costo total de las aristas (s, s_i) reparadas es menor o igual a B.

Por lo tanto, la solución de nuestro problema da una colección de subconjuntos S' de la instancia de BMC que cumple la restricción presupuestaria y cubre un conjunto de elementos cuya suma de pesos es igual al valor objetivo obtenido en nuestro problema.

Podemos concluir que la transformación de una instancia de Budgeted Maximum Coverage (BMC) a nuestro problema se realiza de manera polinomial, la creación de nodos y aristas tal y como lo

describimos, el tiempo es proporcional al tamaño del universo U y de la colección S La correspondencia entre la selección de subconjuntos en BMC y la reparación de aristas (s, s_i) en nuestro problema es directa y la cobertura de elementos en BMC se traduce en la conectividad de los nodos $v(u)$ (con prioridad igual a $w(u)$) a s.

Dado que hemos demostrado que toda solución factible para BMC se puede transformar en una solución factible para nuestro problema y viceversa, podemos decir que nuestro problema es, al menos, tan difícil como el problema Budgeted Maximum Coverage, el cual es **NP-hard**.

Propuesta de algoritmo 1

Dado que es un problema NP-hard, se emplea un algoritmo heurístico basado en una estrategia greedy. En cada iteración, se elige la arista que proporcione la mejor relación beneficio/costo, es decir, aquella que permita conectar la mayor cantidad de zonas de desastre nuevas por cada unidad de presupuesto invertida.

El beneficio se mide en términos de la ganancia marginal, que corresponde a la suma de las prioridades de los nodos de desastre que se lograrían conectar al reparar una arista. El costo es simplemente el costo de reparación de la arista. La razón beneficio/costo se obtiene dividiendo la ganancia marginal entre el costo de la arista.

Algoritmo

Comenzamos definiendo el conjunto de nodos Z, que al principio contiene solo a s ($Z = \{s\}$), el centro de distribución. Además, se asigna un presupuesto total B, y por otro lado, se crea el conjunto E de aristas reparadas, que se mantendrá actualizado con las aristas que se hayan reparado.

Para gestionar el proceso de selección de aristas, se emplea una cola de prioridad, que mantiene ordenadas las aristas que conectan Z con nodos fuera de Z. Estas aristas las denominaremos “frontera” y serán evaluadas para determinar cuál ofrece la mejor relación beneficio/costo.

Para cada arista candidata e=(u,v) tal que $u \in Z$ y $v \notin Z$, se realiza una búsqueda BFS partiendo de v en el subgrafo de aristas reparadas para simular qué nodos serían alcanzables en caso de que se repare la arista e y para determinar cuántos de esos nodos alcanzables son zonas de desastre (nodos en D), ya que estos aportan una ganancia en términos de prioridad.

Una vez identificados los nodos alcanzables, se calcula la ganancia marginal, que corresponde a la suma de las prioridades de los nodos de desastre alcanzables tras la reparación de e. Luego, se determina la razón beneficio/costo, dividiendo la ganancia marginal entre el costo de la arista e. Toda esta información se almacena en la cola de prioridad, de modo que la arista con la mejor relación beneficio/costo quede en la primera posición, facilitando su selección en la siguiente etapa.

Entonces, se extrae de la cola de prioridad la arista con la mayor razón beneficio/costo. Se verifica que su costo no exceda el presupuesto restante. Si la arista es viable, se repara:

- Se agrega al conjunto E.
- Se descuenta su costo del presupuesto.
- Se actualiza el conjunto de nodos conectados Z sin necesidad de realizar otro BFS. En su lugar, se usa la información guardada anteriormente en la cola, permitiendo que, además de v, se incorporen a Z todos los nodos alcanzables desde v, garantizando una expansión eficiente de la conectividad en el grafo.

Tras actualizar Z, se revisan todas las aristas que parten de los nodos recién incorporados y que conducen a nodos que aún no han sido alcanzados. Estas nuevas aristas se añaden a la frontera y se someten al mismo proceso de evaluación. El procedimiento de evaluación, selección y actualización se repite mientras existan aristas en la frontera y haya presupuesto disponible para seguir reparando.

El algoritmo finaliza en dos posibles escenarios:

- No queda suficiente presupuesto para reparar ninguna arista en la frontera.
- No existen más aristas que conecten Z con nodos fuera de Z, o la ganancia marginal de todas las aristas restantes es cero, es decir, no se pueden conectar nuevas zonas de desastre.

Al concluir, el algoritmo devuelve el conjunto E con las aristas reparadas y Z con todos los nodos que quedaron conectados al centro de distribución mediante calles reparadas.

Correctitud

Como vimos el algoritmo está diseñado para conectar el centro de distribución (nodo s) con la mayor cantidad de zonas de desastre (nodos en D con prioridad) sin exceder un presupuesto B. La estrategia consiste en cada paso, seleccionar la arista que ofrezca el mejor “retorno” (es decir, la mayor ganancia en prioridades de nodos nuevos) por cada unidad de presupuesto invertida. Para demostrar que el algoritmo produce una solución válida, se verifica que:

- Cada vez que el algoritmo evalúa una arista candidata, se comprueba que su costo no exceda el presupuesto restante. Al reparar una arista, se descuenta su costo del presupuesto. De esta forma, la suma de los costos de todas las aristas reparadas nunca supera B.
- Como el algoritmo empieza con $Z = \{s\}$, en cada iteración, evalúa las aristas que conectan Z con nodos aún no alcanzados. Al reparar una arista, se usa el resultado del BFS previo para actualizar Z, asegurando que el nodo recién agregado y sus conexiones queden unidos a s. Así, cualquier zona de desastre marcada como atendida ($y = 1$) queda conectada al centro.

- El algoritmo evalúa cada arista candidata según su razón beneficio/costo, priorizando la que maximiza la cobertura de zonas de desastre por unidad de presupuesto. Aunque es una heurística greedy y no siempre garantiza la solución óptima, genera una solución factible que optimiza la cobertura dentro del presupuesto disponible.

Ejemplo de código

```python
import networkx as nx
import heapq

'''
Input:
  G: Grafo que representa la red de calles
  s: Nodo central (origen de distribución)
  D: Conjunto de nodos que son zonas de desastre (por ejemplo, {'a', 'b', 'c'})
  P: Diccionario con las prioridades asignadas a cada zona de desastre,
  por ejemplo, {'a': 5, 'b': 3, 'c': 8}
  B: Presupuesto total disponible para reparar calles

Output:
  E_repaired: Conjunto de aristas reparadas
  Z: Conjunto de nodos que quedaron conectados a s luego de reparar las calles
'''
def bfs_simulation(start_node, candidate_edge, E_repaired, Z, G):
    """
    Realiza un BFS en el subgrafo formado por las aristas ya reparadas
    junto con la arista candidata (que se asume "reparada" para la simulación)
    """
    u, v = candidate_edge
    # Para simular la reparación de la arista candidata, se agrega al conjunto temporal
    simulated_edges = E_repaired.union({tuple(sorted(candidate_edge))})
    visited = set()
    queue = [v]
    reachable = set()

    while queue:
        node = queue.pop(0)
        if node not in visited:
            visited.add(node)
            reachable.add(node)
            for neighbor in G.neighbors(node):
                edge = tuple(sorted((node, neighbor)))
                if edge in simulated_edges:
                    queue.append(neighbor)
    # También se incluye el nodo v si es zona de desastre
    if v in D and v not in Z:
        reachable.add(v)
    return reachable

def greedy_repair_algorithm(G, s, D, P, B):
```

```
"""
- Se inicia con Z = {s} y un presupuesto B.
- Se construye una cola de prioridad con las aristas de "frontera"
evaluadas por la razón (beneficio marginal / costo).
- Se selecciona la arista de mayor razón
- Se actualiza E_repaired, Z, y el presupuesto; luego se agregan
  nuevas aristas de frontera.
- El proceso se repite hasta agotar la frontera o el presupuesto.
"""
Z = {s}                   # Nodos conectados
E_repaired = set()        # Aristas reparadas
budget_remaining = B
frontier = []             # Cola de prioridad: ( -razon, costo, (u,v), nodos alcanz
# Inicializa la frontera para todas las aristas que salen de s
for v in G.neighbors(s):
    if v in Z:
        continue
    cost = G[s][v]['cost']
    if cost > budget_remaining:
        continue
    # Simula el resultado de reparar la arista (s, v)
    reachable = bfs_simulation(s, (s, v), E_repaired, Z, G)
    benefit = sum(P.get(node, 0) for node in reachable if node in D and node not in
    ratio = benefit / cost if cost != 0 else float('inf')
    heapq.heappush(frontier, (-ratio, cost, (s, v), reachable))

while frontier and budget_remaining > 0:
    neg_ratio, cost, edge, reachable = heapq.heappop(frontier)
    u, v = edge
    # Si ambos extremos ya se encuentran en Z, descartar la arista
    if u in Z and v in Z:
        continue
    new_node = v if v not in Z else u
    if cost > budget_remaining:
        continue

    # Repara la arista y actualiza presupuesto
    E_repaired.add(tuple(sorted(edge)))
    budget_remaining -= cost
    # Actualiza el conjunto de nodos conectados Z usando la simulación
    Z.update(reachable)
    Z.add(new_node)
```

```
        # Agrega a la frontera las aristas que salen de los nuevos nodos
        for node in list(reachable.union({new_node})):
            for neighbor in G.neighbors(node):
                if neighbor in Z:
                    continue
                candidate_edge = (node, neighbor)
                candidate_cost = G[node][neighbor]['cost']
                if candidate_cost > budget_remaining:
                    continue
                # Simula la reparación de candidate_edge(arista candidata)
                candidate_reachable = bfs_simulation(node, candidate_edge, E_repaired,
                candidate_benefit = sum(
                  P.get(n,0) for n in candidate_reachable
                    if n in D and n not in Z
                )
                candidate_ratio = candidate_benefit / candidate_cost
                if candidate_cost == 0:
                  candidate_ratio = float('inf')
                heapq.heappush(
                  frontier,
                  (-candidate_ratio, candidate_cost, candidate_edge, candidate_reachabl

    return E_repaired, Z
```

Análisis de complejidad

Podemos decir que el tiempo de ejecución del algoritmo depende principalmente de la evaluación de las aristas candidatas, la ejecución del BFS y la gestión de la cola de prioridad.

El algoritmo trabaja iterativamente seleccionando aristas que conectan el conjunto de nodos alcanzados Z con nodos aún no conectados. En cada iteración, se evalúan las aristas en la frontera, y en el peor de los casos, se pueden considerar hasta $O(m)$ aristas. Para cada una, se ejecuta un BFS en el subgrafo de las calles ya reparadas. La complejidad de este BFS es $O(n + m')$, donde n es el número de nodos y m' es el número de aristas reparadas hasta ese momento. Como el algoritmo guarda el resultado del BFS, se evita hacer dos búsquedas para el mismo candidato, reduciendo el costo promedio.

Las aristas candidatas se almacenan en una cola de prioridad, la cual permite extraer la mejor opción en $O(\log m)$. La inserción de nuevas aristas en esta cola también tiene un costo de $O(\log m)$, donde m es el número total de aristas.

El número total de iteraciones del algoritmo depende de cuántos nodos se agregan a Z. En el peor caso, se pueden agregar hasta $O(n)$ nodos, lo que implica $O(n)$ iteraciones. Como en cada iteración se evalúan nuevas aristas en la frontera y se ejecuta una búsqueda BFS, el peor caso sin optimizaciones podría llegar a $O(n \times m)$. Sin embargo, con el uso de la cola de prioridad para seleccionar la mejor arista en cada iteración y el almacenamiento de los resultados del BFS, el tiempo global se reduce significativamente en la práctica.

Finalmente, la complejidad total del algoritmo es $O(n(n+m))$, lo que indica que el rendimiento depende del tamaño del grafo y del número de conexiones entre los nodos. Gracias a las optimizaciones en la gestión de la frontera y el uso eficiente del BFS, el algoritmo resulta eficiente para grafos de tamaño moderado.

Subproblema 2

Este problema consiste en minimizar los costos de transportación de recursos desde las sucursales de suministros hacia las zonas afectadas, priorizando las zonas con mayor demanda, puesto que en nuestra modelación esto implica mayor prioridad.
Dichos costos estan contemplados como combustible y tiempo, de ahí que sea un problema tanto de selección de vehículos, como de optimización de rutas.
Se agrega un conjunto de restricciones tanto a los vehículos, como a las condiciones de selección de casos válidos, dado que es necesario que se asemeje lo más posible a las necesidades de un problema de la vida real.
Este problema consiste en minimizar los costos de transportacion de recursos desde las sucursales de suministros hacia las zonas afectadas y encontrar asignaciones validas de sucursales capaces de suplir la demanda de los distintos conjuntos de zonas afectadas.

El criterio para una zona ser capaz de suplir la demanda de un conjunto de zonas esta dado por su flota:

$$\sum_{v \in F} c_v \geq \sum_{u \in C} q_u$$

Donde C es una asignacion de zonas a la sucursal S con flota F.

Dichos costos estan contemplados como combustible y tiempo, pero se asume un costo fijo basado en distancia para relajar el problema y llegar a una solucion polinomial.

Se agrega un conjunto de restricciones tanto a los vehiculos, como a las condiciones de seleccion de casos validos, dado que es necesario que se asemeje lo mas posible a las necesidades de un problema de la vida real.

Modelo matemático 2

Dominios

- S: Conjunto de sucursales de la agencia humanitaria.
- F: Conjunto de flotas, donde F_i representa la flota asociada a la sucursal i. Cada vehículo de la flota tiene las siguientes propiedades:
  - c: Que representa la capacidad de carga del vehículo.
  - d: Que representa el consumo de combustible por unidad de distancia.
- D: Conjunto de zonas afectadas. Cada zona tiene una demanda q.

Definición del grafo

- $V = S \cup D$: donde los nodos son tanto las sucursales como las zonas afectadas.
- E: Las aristas entre dos nodos i, j representan que existe un camino de i a j. De donde siendo e dicha arista, $w(e)$ representa la distancia de dicho camino.

Variables del problema de optimización

- $x_{ijkl} \in \{0, 1\}$: Toma el valor 1 si el vehículo j de F_i viaja del nodo l al nodo k.
- $y_{ij} \in \{0, 1\}$: Toma el valor 1 si el vehículo j de F_i es utilizado.

Función objetivo

Definimos la función objetivo como el problema de minimizar:

$$\sum_{i \in S} \sum_{j \in F_i} \sum_{k \in V} \sum_{l \in V} d_j \times w(< k, l >) \times x_{ijkl} \times y_{ij}$$

Reducción HFVRP

Definicion del problema base HFVRP (Heterogeneous fleet vehicle routing problem)

Una instancia del *HFVRP* se define de la siguiente manera:

- Depósito único: Un único depósito d.
- Flota Heterogénea: Un conjunto $V = v_1, v2, \ldots, v_m$ de vehículos, donde cada vehículo v_i tiene una capacidad q_i y un costo asociado c_i.
- Clientes: Un conjunto $D = d_1, d_2, \ldots, d_n$ de clientes, cada uno con una demanda δ_j.
- Distancias: Una función de distancia $dist(u, v)$ definida entre cualquier par de nodos (depósito y clientes, o entre clientes).

Objetivo
Determinar un conjunto de rutas, donde cada ruta:

- Comienza y termina en el depósito d.
- Es asignada a un vehículo v_i de la flota V.
- Satisface la restricción de capacidad (la suma de demandas de los clientes en la ruta no excede q_i), de tal forma que se minimice el costo total (por ejemplo, la suma de los costos de las rutas o de los vehículos utilizados).

Se sabe que el *HFVRP* es **NP-hard**.

Definición del problema dado

La instancia del problema a reducir se define como:

- Depósitos: Un conjunto S de depósitos.
- Flotas Heterogéneas: Para cada depósito s in S se tiene una flota F_s de vehículos donde cada vehículo cuenta con una capacidad c_f y un costo k_f.
- Clientes: Un conjunto D de clientes con necesidades q_d.

Objetivo

Asignar una ruta a cada vehículo de cada flota de cada depósito, de tal forma que:

- Cada ruta comienza y termina en el depósito correspondiente.
- Se respetan las capacidades de los vehículos.
- Cada cliente $d \in D$ debe ser visitado (es decir, debe aparecer en la ruta de al menos un vehículo de algún depósito).

Transformando una instancia de *HFVRP* en una instancia de nuestro problema
Sea I una instancia arbitraria de HFVRP, definida por:

- Depósito: d
- Flota: $V = v_1, v_2, \ldots, v_m$
- Clientes: $D = d_1, d_2, \ldots, d_n$
- Función de distancia: $dist(v, u)$

Definimos la función de transformación R de la siguiente manera para construir $I' = R(I)$ instancia del problema dado:

- Conjunto S:
  Se define $S = d$, o sea, nuestro conjunto de depósitos tiene solamente un elemento.

- Conjunto $F_s | s \in S$:
  Para el único depósito s definimos su flota como $F_s = V$.
- Conjunto de clientes D:
  Es exactamente el mismo conjunto D de *HFVRP*.
- Distancias y demandas:
  También se toman las mismas del problema *HFVRP*.

Con lo anterior, la instancia I' queda completamente definida para el problema dado.

Correctitud de la reducción

Demostraremos que I tiene solución factibe si y sólo si I' tiene solución factible (con un costo equivalente).

I tiene solución factible, entonces I' tiene solución factible

Supongamos que existe una solución factibe para I, es decir existen rutas $R_1, R_2, \ldots, R_k$ que cumplen las restricciones del problema. Como la instancia I' es la misma que I, con los cambios definidos en R para adaptarla a nuestro problema, entonces la solución $R_1, R_2, \ldots, R_k$ satisface también las restricciones del problema dado. Por tanto I' es una solución factible.

I' tiene solución factible, entonces I tiene solución factible

Supongamos que I' tiene solución factible, es decir, existe un conjunto de rutas asignadas a vehículos de F_d que cumplen las restricciones. Como en F solamente hay un elemento (una flota) y todos los nodos de D son visitados por algún vehículo de dicha flota, respetando las respectivas capacidades, entonces esa solución factible también para I.

Análisis de complejidad de R

Como construir el conjunto $S = d$ se realiza en tiempo constante $O(1)$ y tanto el conjunto $F_d = V$ como D se construyen en tiempo lineal con respecto a los tamaños de V y D respectivamentes, entonces la función de transformación se efectua en tiempo polinomial.

Conclusión

Como se probó la validez tanto de una instancia I como una I' en la factibilidad de las soluciones propuestas en ambos sentidos y la función de transformación R del problema *HFVRP* al problema dado se efectua en tiempo polinomial, además sabiendo que el problema base *HFVRP* es **NP-Hard**, se puede concluir que el problema propuesto es al menos **NP-Hard**.

Propuesta de algoritmo 2

Propuesta Greedy:

La idea general del algoritmo es asociar zonas afectadas a sucursales siempre que se cumpla la restricción de que la sucursal pueda suplir la demanda de dicha selección de zonas.
Inicialmente asumiendo como heurística la asociación a la sucursal más cercana a dicha zona, esto por cada vértice del grafo, por lo que se hace uso del algoritmo de Floyd-Warshall para caminos de costo mínimo.
Este proceso termina creando una especie de clusterización de las zonas afectadas, las cuales pueden ser reasignadas en caso de que la sucursal más cercana no sea capaz de suplir toda la demanda de ese clúster.
El proceso de selección de rutas consiste en buscar el camino de costo mínimo desde una zona v a una sucursal s para cada nodo v asociado a dicha sucursal.
Para ello se hace un recorrido por la lista de predecesores que genera el algoritmo Floyd-Warshall, agregando nodos secuencialmente los cuales representan caminos de costo mínimo hacia la sucursal s.
Como dichos caminos no son disjuntos, el resultado es un MST donde el nodo raíz sería la sucursal asignada.
Luego, se verifica la condición de que la capacidad de la flota de la sucursal asignada a un conjunto de zonas pueda proveer de los recursos necesarios a dicha asignación.
En caso de que no se pueda, se hace un cambio de asignación con la sucursal más cercana a s que pueda hacerse cargo de la demanda.

Correctitud:

Asignando la sucursal más cercana a cada nodo, garantizamos de primera mano que el total del recorrido a las zonas afectadas asignadas a la sucursal será el menor. Supongamos que existe una ruta C asignada a una sucursal S_1 que tiene un nodo v cuya distancia a otra sucursal S_2 es menor a S_1, entonces en la matriz de distancias del algoritmo Floyd-Warshall se cumple que $d(v, S_1) > d(v, S_2)$, luego el nodo v pertenece a la ruta asignada a la sucursal S_2, lo cual es una contradicción.
Además, a la hora de verificar que la ruta asignada a una sucursal cumple las restricciones de capacidad de la flota de la sucursal, se realiza un cambio de ruta con la más cercana que pueda satisfacer esa demanda, lo cual sigue garantizando, por lo anterior, que será la mas corta, puesto que sería el costo total de recorrer la ruta adicionando el costo de ir de S_1 a S_2. Esto siempre bajo la hipótesis de relajación que se le aplica al problema al introducir el concepto de clusters, puesto que, en caso contrario, es sabido que pueden existir mejores asignaciones de zonas a sucursales, pero eso implica que nuestro problema pasaría a ser un problema combinatorio no resoluble en tiempo polinomial determinista.

Ejemplo de codigo:

```python
import sys
import heapq as hq
from scipy.sparse.csgraph import floyd_warshall

'''
Input
F: conjunto de flotas: F[i] = (c, d), asumimos que F[i] esta ordenado bajo
el criterio de F[i] >= F[j] si y solo si ci/di >= cj/dj, o sea, la capacidad es mayor q
D: conjunto de zonas: D[i] = q
M: matriz de distancia, donde los primeros k nodos representan las sucursales
y los restantes las zonas de desastre, con k = len(F)

Output
A: diccionario de asignacion, donde las llaves son las sucursales y el valor es la asig
b: bool que representa si se existe una asignacion válida o no
'''
def assign_routes(F, D, M):
  # se calculan las distancias minimas entre los nodos,
  # asi como las rutas y fuentes de los caminos de costo minimo utilizando
  # floyd_warshall
  distances, predecessors = floyd_warshall(csgraph=graph, directed=False, return_predec
  # key = zona
  # value = sucursal
  depot_assign = {}
  # dict que contiene las rutas de cada asignacion
  paths = {}
  # dict que contiene el costo de las demandas de las zonas de la ruta
  path_cost = {}
  # lista que contiene la suma para cada sucursal, la suma de los valores de su respect
  depot_fleet = [(sum([v[0] for v in f]), sum([v[1] for v in f])) for f in F]
  # lista que contiene las asignaciones de vehiculos a las rutas
  path_assign = []
  # a cada zona se le asocia su sucursal mas cercana
  for i in range(len(F), len(M[0])):
    min_dist = sys.float_info.max
    closest_depot = 0
    for j in range(len(F)):
      if distances[i][j] < min_dist:
        min_dist = distances[i][j]
        closest_depot = j
    depot_assign[i] = closest_depot
  # ordenamos las asignaciones por prioridad
  depot_assign.sort(key=lambda x: D[x[0]], reverse=True)
```

```
# formamos la ruta desde la zona i hasta la sucursal j
for i, j in depot_assign:
  if i in merged:
    continue
  # si existe un camino hacia la sucursal j, se agrega i a dicho camino
  if paths.get(j):
    paths[j].append(i)
    continue
  # se asume que siempre hay camino de i a j
  path = [i]
  current = i
  while current != j:
    current = predecessors[j, current]
    path.append(current)
    if current in path:
      break
  paths[j] = path
# por cada ruta se le calcula el costo total y se verifica si su sucursal
# puede encargarse de proveer dicha demanda
for depot, path in paths:
  path_cost[depot] = sum([D[i] if i >= len(F) else 0 for i in path])
  # si el costo de la ruta es mayor que la capacidad de la flota de la sucursal
  # entonces se cambia con la sucursal mas cercana que tenga una flota capaz de
  # encargarse de la demanda
  if path_cost[depot] > depot_fleet[depot][0]:
    closest_depots = [i for i in range(len(F))]
    closest_depots.sort(key=lambda x: distances[depot, x])

    valid_depot = False

    for closest in closest_depots:
      if depot_fleet[closest][0] >= path_cost[depot]:
        temp = paths[depot]
        paths[depot] = paths[closest]
        paths[closest] = temp
        break
    if not valid_depot:
      return {}, False
return paths, True
```

Análisis de complejidad:

La implementación el algoritmo de Floyd-Warshall de la biblioteca scipy que se utiliza en el código tiene una complejidad de $O(V^3)$, donde V es la dimensión de la matriz cuadrada M de adyacencia que representa el grafo del problema.
El ciclo para crear los *clústers* tiene una complejidad de $O(SD)$, tal que $SD \leq V^2$, por lo que asintóticamente tiene una complejidad de $O(V^2)$.
La creación de las rutas tiene una complejidad de $O(V)$, ya que solamente pasa una vez por cada vértice del grafo a la hora de moverse por la lista de predecedores de Floyd-Warshall.
A la hora de verificar que las asignaciones son válidas para el cambio la sección tiene una complejidad de $O(S^2 log S)$.

Luego, utilizando el principio de suma de complejidad temporal para algoritmos se tiene que $O(V^3) + O(V^2) + O(V) + O(S^2 log S) = O(V^3)$, la cual es la complejidad final de nuestro algoritmo.

Críticas y limitantes

Problema 1

La utilización del enfoque *greedy* y el criterio de beneficio/costo, garantiza la resolución del problema **NP-Hard** en un tiempo razonable, pero con la limitante de solo poder esperar una solución que sea óptimo local del problema. Además, se tiene como premisa que solamente es una sucursal la que tiene el presupuesto, lo que podría extrapolarse a varias, aunque no se tiene en cuenta en nuestro algoritmo.
También, si tenemos en cuenta parámetros más complejos como la distancia entre zonas, carreteras múltiples con nodos intermedios que no necesariamente necesitan ayuda, etc, nuestro problema puede generalizarse e incluso aplicarse a otros escenarios, como por ejemplo, el diseño de circuitos o redes de flujo de otro tipo (eléctricas, oleoductos, hidráulicas, etc).

Problema 2

Dado que la heurística que se decidió utilizar asume el uso de *clústers* para la agrupacion de conjuntos de zonas afectadas para asignarlas a sucurusales de ayuda humanitaria, no es posible encontrar un óptimo global para el problema original. También cabe recalcar que dicho enfoque, asumiendo la capacidad total de la flota como un todo, no permite la asignación de vehículos independientes de una sucursal a un conjunto de zonas de otra sucursal en caso de que bajo nuestro criterio el problema sea irresoluble.