[Entrega-AD-EC-1] Problema Asignación Contenedores

Grupo1: Naroa Iparraguirre, Xiomara Cáceres y Iraida Abad

21 de enero de 2025

1. Enuncidado:

La optimización de problemas logísticos es un problema muy complejo que trae de cabeza a más de un operador y planicador profesional. En este ejercicio te proponemos que trates de completar el algoritmo para la optimización de la asignación de contenedores a camiones. Disponemos de k camiones y m contenedores que deben ser transportados. Cada contenedor tiene un peso delimitado por wi. Tu tarea consiste en asignar todos los contenedores entre todos los camiones de manera que se reparta el peso total de manera equitativa. Tienes como ayuda auxiliar parte del código implementado. Recuerda que existe más de una manera correcta de realizar el ejercicio.

2. Pseudocódigo en LaTeX

2.1. repartir_cajas_a_camiones_pesos_ordenados

Este algoritmo asigna los contenedores a los camiones siguiendo un orden ascendente o descendente del peso, intentando equilibrar el peso total entre los camiones.

Algorithm 1 Asignación de contenedores a camiones por pesos ordenados

Require: k (número de camiones), w (pesos de contenedores), m (número de contenedores), is_reverse (orden descendente si es verdadero)
Ensure: camion_por_caja (asignación de contenedores a camiones)

Crear lista lista_peso_indice = $[(w[i], i)\ \forall\ i \in [0, m-1]]$
Ordenar lista_peso_indice por peso en orden ascendente/descendente según is_reverse
Inicializar peso_por_camion = $[0, 0, \ldots, 0]$ (longitud k)
Inicializar camion_por_caja = $[0, 0, \ldots, 0]$ (longitud m)
for $i \leftarrow 0$ to $m-1$ **do**
    (pesoactual, indiceactual) $\leftarrow$ lista_peso_indice[i]
    imin $\leftarrow$ arg mín(peso_por_camion)
    peso_por_camion[imin] $\leftarrow$ peso_por_camion[imin] + pesoactual
    camion_por_caja[indiceactual] $\leftarrow$ imin
end for
    return camion_por_caja

2.2. repartir_cajas_en_orden_ascendente

Este algoritmo asigna los contenedores a los camiones en un orden secuencial ascendente, asegurando que cada camión reciba contenedores de manera cíclica.

Algorithm 2 Asignación de contenedores en orden ascendente

Require: k (número de camiones), w (pesos de contenedores), m (número de contenedores)
Ensure: camion_por_caja (asignación de contenedores a camiones)

Inicializar lista vacía camion_por_caja
for peso_index $\leftarrow 0$ to $m-1$ **do**
    Agregar peso_index mód k a camion_por_caja
end for
    return camion_por_caja

2.3. repartir_cajas_en_orden_descendente

Este algoritmo asigna los contenedores a los camiones en un orden secuencial descendente, asegurando que cada camión reciba contenedores de manera cíclica comenzando desde el contenedor más pesado.

Algorithm 3 Asignación de contenedores en orden descendente

Require: k (número de camiones), w (pesos de contenedores), m (número de contenedores)
Ensure: camion_por_caja (asignación de contenedores a camiones)
1: Inicializar lista vacía camion_por_caja
2: **for** peso_index $\leftarrow m-1$ to 0 (en orden descendente) **do**
3: Agregar peso_index mód k a camion_por_caja
4: **end for**
return camion_por_caja

2.4. repartir_cajas_random_uniform

Este algoritmo asigna los contenedores a los camiones de forma completamente aleatoria, sin seguir ningún patrón fijo.

Algorithm 4 Asignación aleatoria de contenedores a camiones

Require: k (número de camiones), w (pesos de contenedores), m (número de contenedores)
Ensure: camion_por_caja (asignación de contenedores a camiones)
1: Inicializar lista vacía camion_por_caja
2: **for** peso_index $\leftarrow 0$ to $m-1$ **do**
3: Agregar un valor aleatorio entre 0 y $k-1$ a camion_por_caja
4: **end for**
return camion_por_caja

2.5. calcula_desviacion

Este algoritmo calcula cuánto se desvía el peso asignado a cada camión respecto a la media ideal, proporcionando una medida de equilibrio en la distribución.

Algorithm 5 Cálculo de desviación de pesos por camiones

Require: candidato (asignación de contenedores), w (pesos de contenedores), k (número de camiones)
Ensure: desvio_sobre_media (desviación total sobre la media)

Descripción: Este algoritmo calcula cuánto se desvía el peso asignado a cada camión respecto a la media ideal, proporcionando una medida de equilibrio en la distribución.
pesos_camiones ← calcula_peso_camiones(candidato, w, k)
media_perfecta ← sum(w)/k
desvio_sobre_media ← 0
for peso_de_este_camion ∈ pesos_camiones **do**
    desvio_sobre_media ← desvio_sobre_media + |media_perfecta − peso_de_este_camion|
end for
    return desvio_sobre_media

2.6. calcula_peso_camiones

Este algoritmo calcula el peso total que lleva cada camión sumando los pesos de los contenedores que le han sido asignados según la lista candidato.

Algorithm 6 Cálculo de pesos por camiones

Require: candidato (asignación de contenedores), w (pesos de contenedores), k (número de camiones)
Ensure: pesos_camiones (pesos totales por camión)

Inicializar pesos_camiones ← $[0, 0, \ldots, 0]$ (longitud k)
for $i \leftarrow 0$ to longitud(*candidato*) − 1 **do**
    camion_asignado ← candidato[i]
    pesos_camiones[camion_asignado] ← pesos_camiones[camion_asignado] + $w[i]$
end for
    return pesos_camiones

2.7. encontrar_mejor_solucion

Este algoritmo selecciona la mejor asignación de contenedores a camiones de una lista de soluciones posibles.

Algorithm 7 Encontrar la mejor solución

Require: solution_list (lista de posibles asignaciones), w (pesos de los contenedores), k (número de camiones)
Ensure: best_solution (asignación con menor desviación de peso)

```
Inicializar best_solution ← None
Inicializar best_solution_desviation ← ∞
for solution ∈ solution_list do
    this_solution_desviation ← calcula_desviacion(solution, w, k)
    if this_solution_desviation < best_solution_desviation then
        best_solution ← solution
        best_solution_desviation ← this_solution_desviation
    end if
end for
    return best_solution
```

3. Flujo inicial del programa

El flujo inicial del programa se encarga de definir las variables necesarias, calcular las diferentes soluciones para la asignación de contenedores a camiones, y determinar cuál de estas soluciones es la óptima según el criterio de mínima desviación. Este flujo incluye los siguientes pasos principales:

- Definir las variables clave:
  - k: Número de camiones.
  - w: Lista de pesos de los contenedores.
  - m: Número total de contenedores.
- Calcular las soluciones con diferentes estrategias de asignación:
  - Reparto por pesos ordenados ascendente.
  - Reparto por pesos ordenados descendente.
  - Reparto en orden ascendente de índices.
  - Reparto en orden descendente de índices.
  - Reparto aleatorio.
- Determinar cuál de estas soluciones es la mejor en términos de mínima desviación.

El código de este flujo se muestra a continuación:

```
# Define variables
k = 5
w = [77,72,89,30,29,19,34,68,35,44,55,92,93,78,14,36,95,56,87,18,63,51,85,37,37,
     1,89,16,43,44,35,16,6,12,56,92,51,64,49,93,22,77,4,74,64,40,9,73,77,9]
m = len(w)

# Conseguir soluciones
solucion_1 = repartir_cajas_a_camiones_pesos_ordenados(k, w, m, True)
solucion_2 = repartir_cajas_a_camiones_pesos_ordenados(k, w, m, False)
solucion_3 = repartir_cajas_en_orden_ascendente(k, w, m)
solucion_4 = repartir_cajas_en_orden_descendente(k, w, m)
solucion_5 = repartir_cajas_random_uniform(k, w, m)

# Crear una lista de soluciones
solution_list = [solucion_1, solucion_2, solucion_3, solucion_4, solucion_5]

mi_solucion = encontrar_mejor_solucion(solution_list, w, k)
```

4. Código:

```

import random

def repartir_cajas_a_camiones_pesos_ordenados(k, w, m, is_reverse =
     False):

    #ordenar lista de contenedores sin perder el indice del peso
    lista_peso_indice = [(w[i], i) for i in range(m)]
    lista_peso_indice = sorted(lista_peso_indice, key=lambda x: x
    [0], reverse = is_reverse)
    #definir el camion de menor peso y su peso
    imin = 0
    pesomin = 0
    #crear una lista de camiones y sus pesos
    peso_por_camion = [0 for x in range(k)]
    #crear una lista que representa cada caja y el valor sera el
    numero del camion asignado
    camion_por_caja = [0 for x in range(m)]


    #iterar por cada peso
    for i in range(m):
        (pesoactual, indiceactual)  = lista_peso_indice[i]
        peso_por_camion[imin] = pesomin + pesoactual
        #asignar en qu  camion va la caja actual
        camion_por_caja[indiceactual] = imin

        #actualizar peso min
        pesomin = min(peso_por_camion)
```

```
        imin = peso_por_camion.index(pesomin)

    return camion_por_caja

def repartir_cajas_en_orden_ascendente(k, w, m):

    camion_por_caja = []
    for peso_index in range(m):
        camion_por_caja.append(peso_index % k)

    return camion_por_caja

def repartir_cajas_en_orden_descendente(k, w, m):

    camion_por_caja = []
    for peso_index in reversed(range(m)):
        camion_por_caja.append(peso_index % k)

    return camion_por_caja

def repartir_cajas_random_uniform(k, w, m):

    camion_por_caja = []
    for peso_index in range(m):
        camion_por_caja.append(random.randint(0,k-1))

    return camion_por_caja

# Implementaci n de la funcion objetivo
def calcula_desviacion(candidato, pesos, camiones):

    # Conseguir array del peso total que lleva cada camion
    pesos_camiones = calcula_peso_camiones(candidato, pesos,
    camiones)

    # Cual ser a el peso ideal si esta se distribuyese
    uniformemente
    media_perfecta = sum(pesos) / camiones

    # Sumatorio de respuesta
    desvio_sobre_media = 0

    # Por cada camion se suma la diferencia entre la media perfecta
     y el peso real del camion
    for peso_de_este_camion in pesos_camiones:
        desvio_sobre_media += abs(media_perfecta -
    peso_de_este_camion)

    return desvio_sobre_media

def calcula_peso_camiones(candidato, pesos, camiones):
    pesos_camiones = [0 for i in range(camiones)]

    # A cada camion se le suma el peso de las cajas que tiene
    asignadas
    for indice_peso, camion_asignado  in enumerate(candidato):
        pesos_camiones[camion_asignado] += pesos[indice_peso]
```

```
79
80      return pesos_camiones
81
82  ## Desarrollo del algoritmo
83  def encontrar_mejor_solucion(solution_list, w, k):
84
85      # Inicializar la mejor solucion
86      best_solution = None
87      best_solution_desviation = float("inf")
88
89      # Iterar por todas las soluciones
90      for solution in solution_list:
91          this_solution_desviation = calcula_desviacion(solucion_1, w
        , k)
92          # Actualizar la mejor solucion
93          if this_solution_desviation < best_solution_desviation:
94              best_solution = solution
95              best_solution_desviation = this_solution_desviation
96
97      return best_solution
98
99
100 ###### MAIN ######
101
102 # Define variables
103 k = 5
104 w = [77, 72, 89, 30, 29, 19, 34, 68, 35, 44, 55, 92, 93, 78, 14,
        36, 95, 56, 87, 18, 63, 51, 85, 37, 37, 1, 89, 16, 43, 44, 35,
        16, 6, 12, 56, 92, 51, 64, 49, 93, 22, 77, 4, 74, 64, 40, 9,
        73, 77, 9]
105 m = len(w)
106
107 # Crear una lista de soluciones
108 solucion_1 = repartir_cajas_a_camiones_pesos_ordenados(k, w, m,
        True)
109 solucion_2 = repartir_cajas_a_camiones_pesos_ordenados(k, w, m,
        False)
110 solucion_3 = repartir_cajas_en_orden_ascendente(k, w, m)
111 solucion_4 = repartir_cajas_en_orden_descendente(k, w, m)
112 solucion_5 = repartir_cajas_random_uniform(k, w, m)
113 solution_list = [solucion_1, solucion_2, solucion_3, solucion_4,
        solucion_5]
114
115 # Mejor soluci n:
116 mi_solucion = encontrar_mejor_solucion(solution_list, w, k)
117 print("Soluci n  ptima :" , mi_solucion)
118 print("Interpretaci n de la soluci n: ")
119 for peso_index, peso in enumerate(w):
120     print(f"La caja n mero {peso_index} que pesa {peso}kg va al
        camion n mero {mi_solucion[peso_index]}")
```