Entrega 1 Optimización

Grupo 4

30-05-2024

Carlos Arévalo - 202173501-1
Nangel Coello - 202173591-7
Daniel Maturana - 202173575-5
Benjamin Pavez - 202173628-K

1. Enunciado

En una empresa se busca asignar de forma óptima a sus trabajadores en las diversas tareas que deben llevarse a cabo. Se sabe que los trabajadores están clasificados según su nivel de especialización. Además, las tareas están clasificadas según los mismos niveles de especialización de tal manera que una tarea de cierto nivel de especialización puede ser ejecutada solo por trabajadores de tal nivel, o bien, uno superior, pero no inferior.

Cada tarea requiere de cierta cantidad de tiempo para ser completada y cada trabajador puede ser asignado por una cantidad determinada de tiempo. Todas las tareas deben ser completadas en su totalidad.

Todos los trabajadores de cierto nivel de especialización suponen un costo por unidad de tiempo y un costo fijo asociado a asignar a cierto trabajador a cierta tarea en específico. A estos costos los llamaremos *costos 1*. Además, existe un costo por sobrecalificación asociado a cada tarea.

Cada trabajador puede ser asignado a un máximo de N tareas, pero para completar una tarea puede requerirse más de un trabajador.
No obstante, no se pueden asignar más de dos tercios del total de los trabajadores a una sola tarea.

La empresa cuenta con un presupuesto P para la asignación de tareas que no puede ser superado, y busca:

1. Minimizar *costos 1*.
2. Minimizar costos de sobrecalificación.

2. Formulación del Modelo Matemático

1. Conjuntos:

   I : Conjunto de trabajadores, i $\in I$.
   J : Conjunto de tareas, j $\in J$.
   L : Conjunto de niveles de especialización, l $\in \{1, 2, 3, \ldots, 8\}$.

2. Parámetros:

   E_i: Nivel de especialización que tiene el trabajador i.
   S_j: Nivel de especialización requerido para la tarea j.
   T_j: Tiempo requerido para completar la tarea j.
   H_i: Tiempo disponible del trabajador i.
   C_{ij}: Costo fijo asociado a asignar al trabajador i a la tarea j.
   U_i: Costo por unidad de tiempo del trabajador i.
   O_j: Costo por sobrecalificación de asignar al trabajador i a la tarea j (si $E_i > S_j$).
   P: Presupuesto disponible.
   N: Máximo número de tareas a las que puede ser asignado un trabajador.

3. Variables:

   x_{ij} : Variable binaria que representa si el trabajador i es asignado a la tarea j.
   y_{ij} : Variable que representa el tiempo que el trabajador i dedica a la tarea j.

4. Función objetivo:
   Se busca minimizar los *costos 1*: si el trabajador i es asignado a la tarea j entonces va a tener un costo fijo C_{ij}, esto más la cantidad de tiempo que va a dedicar el trabajor i a la tarea j por el costo por unidad de tiempo del trabajador i.

$$\text{Min} \quad Z = \sum_{i \in I} \sum_{j \in J} (C_{ij} \cdot x_{ij} + U_i \cdot y_{ij})$$

5. Restricciones:

   a) **Asignación completa de tareas**: Cada tarea debe ser completada en su totalidad.

   $$\sum_{i \in I} y_{ij} = T_j \quad \forall j \in J$$

   b) **Tiempo disponible por trabajador**: Un trabajador no puede ser asignado más allá de su tiempo disponible.

   $$\sum_{j \in J} y_{ij} \leq H_i \quad \forall i \in I$$

   c) **Nivel de especialización**: Un trabajador puede ser asignado a una tarea solo si su nivel de especialización es igual o superior al requerido.

   $$x_{ij}(E_i - S_j) \geq 0 \quad \forall i \in I, \forall j \in J$$

   d) **Relación entre x_{ij} y y_{ij}**: Si un trabajador i no está asignado a una tarea j, el tiempo dedicado es cero.

   $$y_{ij} \leq T_j \cdot x_{ij} \quad \forall i \in I, \forall j \in J$$

   e) **Límite de tareas por trabajador**: Un trabajador puede ser asignado a un máximo de N tareas.

   $$\sum_{j \in J} x_{ij} \leq N \quad \forall i \in I$$

f) **Presupuesto**: El costo total no debe superar el presupuesto disponible.

$$\sum_{i \in I} \sum_{j \in J} (C_{ij} \cdot x_{ij} + U_i \cdot y_{ij} + O_j \cdot (E_i > S_j) \cdot x_{ij}) \leq P$$

g) **Límite de asignación por tarea**: No se pueden asignar más de dos tercios de los trabajadores a una sola tarea.

$$\sum_{i \in I} x_{ij} \leq \frac{2}{3}|I| \quad \forall j \in J$$

h) **Naturaleza de las variables**:

$$x_{ij} \in \{0, 1\} \quad \forall i \in I, \forall j \in J$$

$$y_{ij} \geq 0 \quad \forall i \in I, \forall j \in J$$

3. Posibles Casos de Infactibilidad

1. **Insuficiente tiempo disponible**: Si el tiempo total requerido por todas las tareas es mayor que la suma del tiempo disponible de todos los trabajadores.

2. **Presupuesto insuficiente**: Si el menor costo posible para asignar todas las tareas es estrictamente mayor al presupuesto.

3. **Desajuste en niveles de especialización**: Si hay tareas para las cuales no hay trabajadores con un nivel de especialización adecuado.

4. **Exceso de tareas por trabajador**: Si existe varias tareas pero pocos trabajadores

5. **Límite de asignación por tarea**: Si existe alguna tarea que requiera una mayor cantidad de trabajadores de los que hay disponibles sin exceder el limite.

4. Generador de Instancias en Python

```python
import numpy as np

np.random.seed(12345)


def generar_instancia(I, J, N=4):
    # Niveles de especializaci n (distribuci n triangular)
    E = np.random.triangular(1, 8, 8, I).astype(int)
    S = np.random.triangular(1, 1, 8, J).astype(int)

    # Tiempo disponible por trabajador y tiempo requerido por tarea
    inf_limit = 20 + 10*N
    sup_limit = inf_limit + 15
    H = np.random.randint(inf_limit, sup_limit, I)*10
    T = np.random.randint(3, inf_limit, J)*10

    # Costos por unidad de tiempo por trabajador
    costo_tiempo = lambda n_e: np.random.uniform(5*n_e, 5*n_e+5)
    U = costo_tiempo(E)

    # Costos fijos y costos de sobrecalificaci n
    C = np.random.randint(4000, 8001, size=(I, J))
    O = np.random.randint(10000, 30001, size=(J))

    # Presupuesto
    P = (np.random.randint(low=(20000)//3, high=(40000)//5, size=1)*I)
        [0]

    return E.tolist(), S.tolist(), H.tolist(), T.tolist(), U.tolist(), C
        .tolist(), O.tolist(), P, N, I, J


def MiniZinc_text(E:list,S:list,H:list,T:list,U:list,C:list,O:list,P:
    float,I:int,J:int,N:int):

    def list2d_to_minizinc2dstring(elements: list):
        result = "["
        for line in elements:
            result += "|"+str(line)[1:-1]
        result += "|]"
        return(result)

    list_of_lines = []
    #Sets
    list_of_lines.append(f"int: I = {I};")
    list_of_lines.append(f"int: J = {J};")
    list_of_lines.append(f"set of int: empleados = 1..I;")
    list_of_lines.append(f"set of int: tareas = 1..J;")
    list_of_lines.append(f"array[1..I] of float: E = {E};")
    list_of_lines.append(f"array[1..J] of float: S = {S};")
    list_of_lines.append(f"array[1..J] of float: T = {T};")
    list_of_lines.append(f"array[1..I] of float: H = {H};")
    list_of_lines.append(f"array[empleados,tareas] of float: C = {
        list2d_to_minizinc2dstring(C)};")
    list_of_lines.append(f"array[1..I] of float: U = {U};")
    list_of_lines.append(f"array[tareas] of float: O = {O};")
    list_of_lines.append(f"int: P = {P};")
    list_of_lines.append(f"int: N={N};")

    #Variables
    list_of_lines.append(f"array[empleados,tareas] of var 0..1: x;")
    list_of_lines.append(f"array[empleados,tareas] of var float: y;")
```

```
59
60              #func. objetivo
61              list_of_lines.append(f"var float: funcob = sum ([ sum ([ C[i,j]*x[i,
                    j] + U[i]*y[i,j] | j in 1..J]) | i in 1..I]);")
62
63              #restricciones
64              list_of_lines.append(f"constraint forall(j in 1..J)(sum([ y[i,j] | i
                     in 1..I]) >= T[j]);")
65              list_of_lines.append(f"constraint forall(i in 1..I)(sum([y[i,j] | j
                    in 1..J ]) <= H[i]);")
66              list_of_lines.append(f"constraint forall(i in 1..I)(forall(j in 1..J
                    ) ( x[i,j]*E[i] - x[i,j]*S[j] >= 0));")
67              list_of_lines.append(f"constraint forall(i in 1..I)(forall(j in 1..J
                    )( y[i,j] <= T[j]*x[i,j] ));")
68              list_of_lines.append(f"constraint forall(i in 1..I)( sum([ x[i,j] |
                    j in 1..J]) <= N );")
69              list_of_lines.append(f"constraint  sum ([ sum ([ C[i,j]*x[i,j] + U[i
                    ]*y[i,j] + bool2int(E[i]>S[j])*O[j]*x[i,j]| j in 1..J]) | i in
                    1..I]) <= P;")
70              list_of_lines.append(f"constraint forall(j in 1..J)( sum([ x[i,j] |
                    i in 1..I]) <= 2 * I / 3 );")
71
72              #solve
73              list_of_lines.append("solve minimize funcob;")
74              list_of_lines.append("output [\"xij = \\(x)\\n\", \"yij = \\(y)\\n
                    \", \"costos = \\(funcob)\"]")
75
76              return list_of_lines
77          #funcion generadora
78          range_func = lambda left_limit, distance: np.random.randint(left_limit,
                left_limit+distance)
79
80
81          #instancias pequenas
82          I_pequena_i = np.arange(1,6)*10
83          I_pequena = range_func(I_pequena_i, 9)
84          J_pequena = np.random.randint(np.array([5,8,13,19,24]), np.array
                ([7,12,18,23,30]))
85
86          #instancias medianas
87          I_mediana_i = np.arange(80,205,25)
88          I_mediana = range_func(I_mediana_i, 24)
89          J_mediana_i = np.arange(30,105,15)
90          J_mediana = range_func(J_mediana_i, 14)
91
92          #instancias grandes
93          I_grande_i = np.arange(300,600,60)
94          I_grande = range_func(I_grande_i, 59)
95          J_grande_i = np.arange(100,300,40)
96          J_grande = range_func(J_grande_i, 39)
97
98          I = np.concatenate((I_pequena,I_mediana,I_grande))
99          J = np.concatenate((J_pequena,J_mediana,J_grande))
100
101         #instancias
102         instancias = []
103         for i in range(len(I)):
104             instancias.append(generar_instancia(I[i], J[i]))
105
106         i = 1
107         for E, S, H, T, U, C, O, P, N, I, J in instancias:
108             file = open(f'instancia-{i}.txt', 'w')
109             for line in MiniZinc_text(E=E, S=S, H=H, T=T, U=U, C=C, O=O, P=P, I=
                    I, J=J, N=N):
110                 file.write(line+"\n")
```

```
111                 file.close()
112                 i+= 1
```

Listing 1: Codigo en Python para generar instancias