

Entrega 2 Optimización

Grupo 4

01-07-2024

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

1. Introducción

Mediante un problema dado, se creará un modelo matemático definiendo conjuntos, parámetros, variables, función objetivo y restricciones necesarias para la solución de dicho problema. Además, se considerarán los posibles casos de infactibilidad del modelo determinando así una solución acertada.

Para la implementación y prueba del modelo se utilizará MiniZinc junto con un generador de modelo en Python con las respectivas instancias para poder así, obtener distintos datos como el número de variables, tiempo de ejecución, la cantidad de iteraciones, resultado de la función objetivo, cantidad de tareas y cantidad de trabajadores.

Finalmente, se presentarán gráficos con el fin de analizar los resultados obtenidos y se realizará un análisis de los tiempos de ejecución.

2. 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.

3. 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, $l \in \{1, 2, 3, \dots, 8\}$.

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 sobrecualificación 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 trabajador i a la tarea j por el costo por unidad de tiempo del trabajador i .

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

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:**

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

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

4. 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 límite.

5. Generador de Modelo en Python

```
1 import numpy as np
2
3 np.random.seed(12345)
4
5
6 def generar_instancia(I, J, N=4):
7     # Niveles de especializaci n (distribuci n triangular)
8     E = np.random.triangular(1, 8, 8, I).astype(int)
9     S = np.random.triangular(1, 1, 8, J).astype(int)
10
11     # Tiempo disponible por trabajador y tiempo requerido por tarea
12     inf_limit = 20 + 10*N
13     sup_limit = inf_limit + 15
14     H = np.random.randint(inf_limit, sup_limit, I)*10
15     T = np.random.randint(3, inf_limit, J)*10
16
17     # Costos por unidad de tiempo por trabajador
18     costo_tiempo = lambda n_e: np.random.uniform(5*n_e, 5*n_e+5)
19     U = costo_tiempo(E)
20
21     # Costos fijos y costos de sobrecalificaci n
22     C = np.random.randint(4000, 8001, size=(I, J))
23     O = np.random.randint(10000, 30001, size=(J))
24
25     # Presupuesto
26     P = (np.random.randint(low=(20000)//3, high=(40000)//5, size=1)*I)
27     [0]
28
29     return E.tolist(), S.tolist(), H.tolist(), T.tolist(), U.tolist(), C
30     .tolist(), O.tolist(), P, N, I, J
31
32 def MiniZinc_text(E:list,S:list,H:list,T:list,U:list,C:list,O:list,P:
33 float,I:int,J:int,N:int):
34
35     def list2d_to_minizinc2dstring(elements: list):
36         result = "["
37         for line in elements:
38             result += "|" + str(line)[1:-1]
39             result += " |]"
40         return(result)
41
42     list_of_lines = []
43     #Sets
44     list_of_lines.append(f"int: I = {I};")
45     list_of_lines.append(f"int: J = {J};")
46     list_of_lines.append(f"set of int: empleados = 1..I;")
47     list_of_lines.append(f"set of int: tareas = 1..J;")
48     list_of_lines.append(f"array[1..I] of float: E = {E};")
49     list_of_lines.append(f"array[1..J] of float: S = {S};")
50     list_of_lines.append(f"array[1..J] of float: T = {T};")
51     list_of_lines.append(f"array[1..I] of float: H = {H};")
52     list_of_lines.append(f"array[empleados,tareas] of float: C = {
53         list2d_to_minizinc2dstring(C)};")
54     list_of_lines.append(f"array[1..I] of float: U = {U};")
55     list_of_lines.append(f"array[tareas] of float: O = {O};")
56     list_of_lines.append(f"int: P = {P};")
57     list_of_lines.append(f"int: N={N};")
58
59     #Variables
60     list_of_lines.append(f"array[empleados,tareas] of var 0..1: x;")
61     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,
62 j] + U[i]*y[i,j] | j in 1..J]) | i in 1..I]);")
63
64 #restricciones
65 list_of_lines.append(f"constraint forall(j in 1..J)(sum([ y[i,j] | i
66 in 1..I]) >= T[j]);")
67 list_of_lines.append(f"constraint forall(i in 1..I)(sum([y[i,j] | j
68 in 1..J ]) <= H[i]);")
69 list_of_lines.append(f"constraint forall(i in 1..I)(forall(j in 1..J
70 ) ( x[i,j]*E[i] - x[i,j]*S[j] >= 0));")
71 list_of_lines.append(f"constraint forall(i in 1..I)(forall(j in 1..J
72 ) ( y[i,j] <= T[j]*x[i,j] ));")
73 list_of_lines.append(f"constraint forall(i in 1..I)( sum([ x[i,j] |
74 j in 1..J]) <= N );")
75 list_of_lines.append(f"constraint sum ([ sum ([ C[i,j]*x[i,j] + U[i
76 ]*y[i,j] + bool2int(E[i]>S[j])*O[j]*x[i,j]| j in 1..J]) | i in
77 1..I]) <= P;")
78 list_of_lines.append(f"constraint forall(j in 1..J)( sum([ x[i,j] |
79 i in 1..I]) <= 2 * I / 3 );")
80
81 #solve
82 list_of_lines.append("solve minimize funcob;")
83 list_of_lines.append("output [\"xij = \\\(x)\\n\\n\", \"yij = \\\(y)\\n
84 \", \"costos = \\\(funcob)\\n\"]")
85
86 return list_of_lines
87
88 #funcion generadora
89 range_func = lambda left_limit, distance: np.random.randint(left_limit,
90 left_limit+distance)
91
92 #instancias pequenas
93 I_pequena_i = np.arange(1,6)*10
94 I_pequena = range_func(I_pequena_i, 9)
95 J_pequena = np.random.randint(np.array([5,8,13,19,24]), np.array
96 ([7,12,18,23,30]))
97
98 #instancias medianas
99 I_mediana_i = np.arange(80,205,25)
100 I_mediana = range_func(I_mediana_i, 24)
101 J_mediana_i = np.arange(30,105,15)
102 J_mediana = range_func(J_mediana_i, 14)
103
104 #instancias grandes
105 I_grande_i = np.arange(300,600,60)
106 I_grande = range_func(I_grande_i, 59)
107 J_grande_i = np.arange(100,300,40)
108 J_grande = range_func(J_grande_i, 39)
109
110 I = np.concatenate((I_pequena,I_mediana,I_grande))
111 J = np.concatenate((J_pequena,J_mediana,J_grande))
112
113 #instancias
114 instancias = []
115 for i in range(len(I)):
116     instancias.append(generar_instancia(I[i], J[i]))
117
118 i = 1
119 for E, S, H, T, U, C, O, P, N, I, J in instancias:
120     file = open(f'instancia-{i}.mzn', 'w')
121     for line in MiniZinc_text(E=E, S=S, H=H, T=T, U=U, C=C, O=O, P=P, I=
122 I, J=J, N=N):
123         file.write(line+"\n")

```

```

111         file.close()
112         i+= 1

```

Listing 1: Código en Python para generar modelo

6. Resultados

En esta sección se presentan los resultados obtenidos al ejecutar las instancias pequeñas y medianas. Por la naturaleza exponencial del tiempo de ejecución en relación al tamaño de la instancia (esto se verá justificado en mayor detalle en la sección de análisis) no se han podido ejecutar las instancias grandes (instancias 11 a la 15).

Instancia	Tiempo [sec]	F.O	I	J	Iteraciones	Número Variables
1	0,16	54890,69	12	5	1	120
2	0,23	62535,99	25	9	1	450
3	0,30	U	21	14	1	588
4	2,34	148348,84	44	21	4	1848
5	3,74	187775,34	55	25	4	2750
6	53,20	204183,27	94	32	8	6016
7	492,00	343929,92	114	54	7	12312
8	1065,00	580538,77	153	71	8	21726
9	3201,00	605051,56	177	88	10	31152
10	11786,00	844721,24	196	100	11	39200

Cuadro 1: Datos de las instancias

7. Análisis de resultados y tiempos de resolución

En base a los datos recopilados, se puede estudiar cómo se comportan el tiempo de resolución, el valor de la función objetivo y el número de iteraciones en relación al número de variables, dado por $2 \times I \times J$.

Empezando con el tiempo, se puede ver una relación exponencial entre ambas variables. Esto se debe a que el método simple de programación lineal tiene un tiempo de ejecución de peor caso de $O(2^n)$, esto aún sin considerar las numerosas veces que se tiene que realizar para la resolución de variables binarias dependiendo del método que implemente el solver MiniZinc.

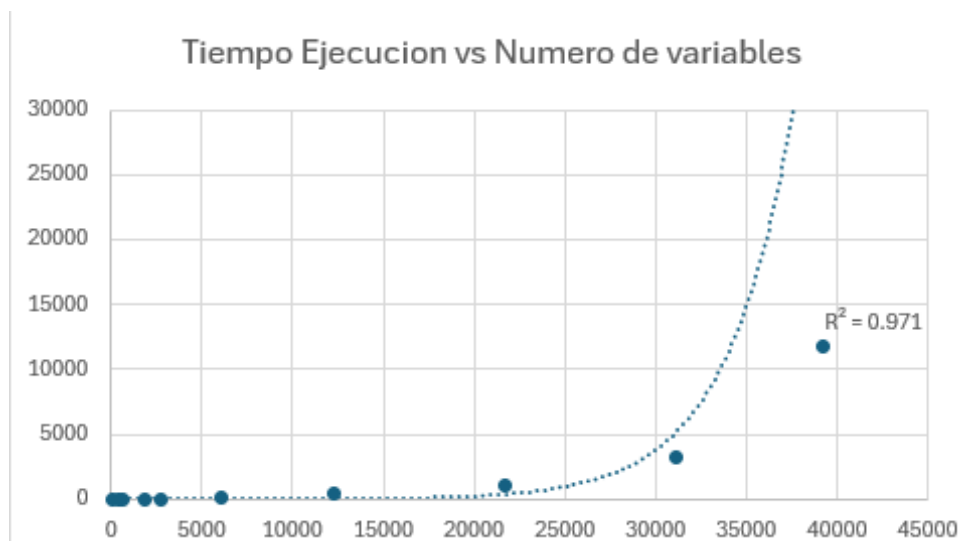


Figura 1: Gráfico Tiempo de Ejecución vs Número de variables

Como el número de variables es proporcional al número de la instancia, tiene sentido realizar el análisis de tiempo de ejecución en relación a este:

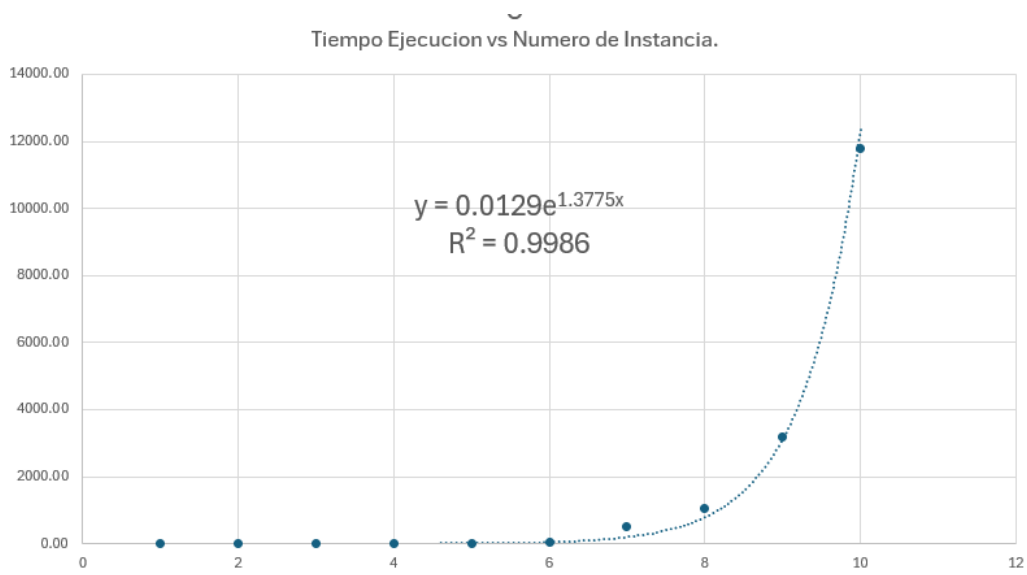


Figura 2: Gráfico Tiempo de Ejecución vs Número de Instancia

Se aprecia que el tiempo de ejecución con relación al número de instancia es extrapolable a una función exponencial con una alta precisión ($R^2 = 0,9986$). Por esto mismo es que se ha tomado la decisión de no ejecutar la resolución de las instancias grandes, se espera que el tiempo sea mayor al manejable. Esto se demostró empíricamente tratando de ejecutar la resolución de la instancia 14 y la instancia 15, ambas sin éxito.

En cuanto al valor de la función objetivo, el incremento del número de variables involucra el aumento del resto de parámetros, como el número de trabajadores, el de tareas, el de presupuesto y otros no dependientes del parámetro N. Específicamente en cuanto al número de tareas, cuando este aumenta, aumenta la cantidad de horas asignadas por trabajador (y potencialmente la asignación de más trabajadores) para completar esta mayor cantidad de tareas. Esto se puede ver en el gráfico, con el aumento del número de variables aumenta también el valor de la función objetivo encontrado en la solución óptima.

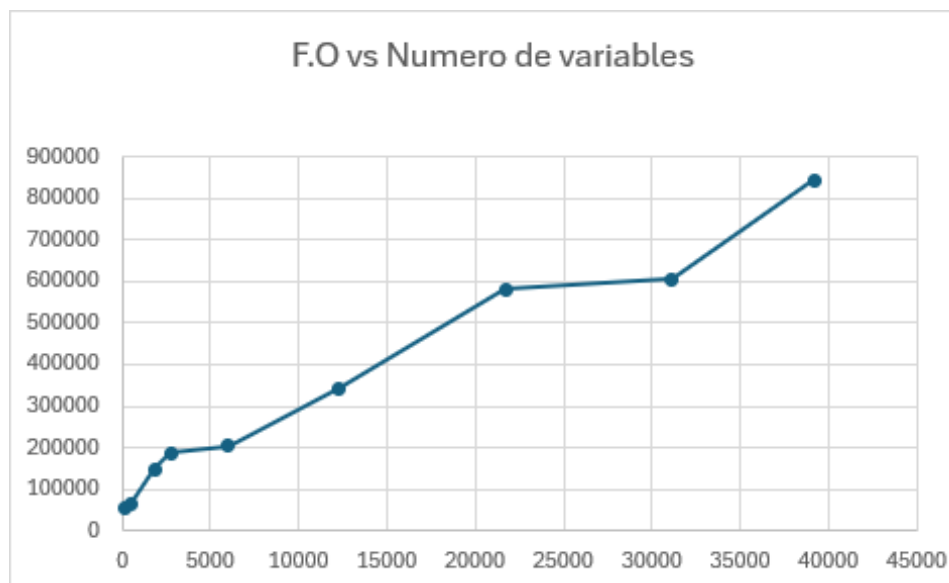


Figura 3: Gráfico Función Objetivo vs Número de variables

El número de iteraciones aumenta a medida que lo hace el número de variables. Más importante que esto es el hecho que las instancias pequeñas (1 a 5) tengan un número de iteraciones dentro del orden de

magnitud 10^0 , mientras que para instancias medianas (de 5 a 10) el número de iteraciones está dentro del orden de magnitud 10^1 . Con esto es esperable que instancias grandes y una hipotética lista de instancias "enormes" también tengan números de iteraciones con órdenes de magnitud de diferencia.

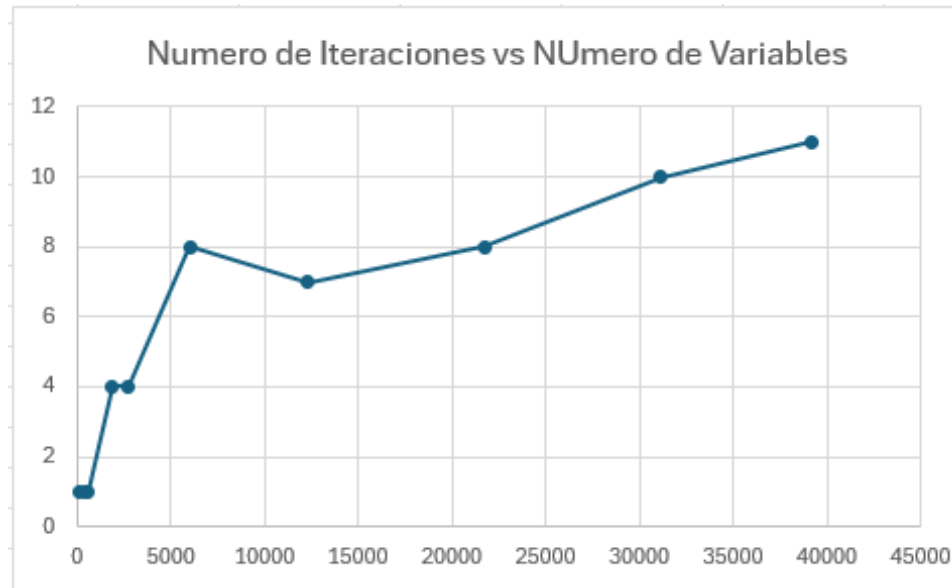


Figura 4: Gráfico Número Iteraciones vs Número de variables

En general también podemos ver que el valor de N no afecta el tamaño de las instancias, pues este no afecta el número de trabajadores o tareas. Si afecta, por otra parte, los valores de disponibilidad de horas de los trabajadores y el tiempo necesario para realizar tareas. Si nos fijamos en como se generan estos, independiente de N se va a tener que el rango de tiempo para realizar una tarea es $[0, Y]$ y el rango de disponibilidad de tiempo de los trabajadores es de $[Y, Y + 150]$, por lo que se espera que no afecte a la factibilidad de soluciones, pero si al resultado de la función objetivo.

8. Conclusión

El estudio anterior demuestra la importancia de considerar la relación exponencial del tiempo de ejecución al diseñar modelos matemáticos y optimizar problemas de gran escala. Los resultados sugieren que, para instancias grandes, es necesario mejorar la eficiencia del algoritmo para manejar la complejidad del problema.

Con respecto a la función objetivo, a medida que aumentan el número de variables, también lo hace el valor de la función. El aumento del número de tareas en particular, incrementa la cantidad de horas asignadas por trabajador y el número de trabajadores necesarios, lo que lleva a un mayor valor en la solución óptima de la función.

Finalmente, el valor del parámetro N no afecta directamente el tamaño de la instancias, pero sí influye en la disponibilidad de horas de los trabajadores y el tiempo necesario para realizar las tareas, afectando el resultado de la función objetivo sin comprometer la factibilidad de las soluciones.