Universidad Peruana de Ciencias Aplicadas

Ciclo 2021-2

CC58 - Tópicos en Ciencias de la Computación

Sección: CC58

TB1

Tema: Orchestra rehearsal

PROFESOR DEL CURSO: Willy Gustavo Ugarte Rojas

Integrantes:

1. Bill Brandon Chavez Arias
2. Mosqueira Chacón, César Manuel
3. Peralta Ireijo, Sebastián Fernando

1. Definición del problema:

El concierto consta de 9 composiciones musicales. Cada composición involucra a 5 *players* de la orquesta. Cada *player* puede llegar inmediatamente antes de su primera composición en la que participa y retirarse inmediatamente después de su última composición.

Para esto es necesario encontrar el orden de las composiciones para un ensayo de modo que se minimice el tiempo total de espera de los *players*, los datos del problema se representan en la siguiente tabla

Composition number	1	2	3	4	5	6	7	8	9
Player 1	×		×	×		×	×		×
Player 2	×		×	×	×	×		×	
Player 3	×	×			×			×	
Player 4	×				×		×		×
Player 5		×		×	×	×	×		
Composition duration	2	4	1	3	3	2	5	7	6

Tabla.1

La Tabla 1 presenta las composiciones en las que cada *player* debe estar presente (con una ×). Además se nos da 2 restricciones. La composición 2 debe ser tocada antes de la composición 8 y la composición 6 debe ser tocada inmediatamente después de la composición 5. Cabe mencionar que, el tiempo que se calcula es el tiempo donde es estrictamente necesario que el *player* esté en el ensayo. Es decir, si se tocara la composición 5 primero, el player1 podría llegar al finalizar la composición, igualmente con la última composición.

Si el orden fuese (1, 2, 3, 4, 5, 6, 7, 8, 9), el tiempo de espera sería:

- player 1: 14 = (4 + 3 + 7)
- player 2: 9 = (4 + 5)
- player 3: 11 = (1 + 3 + 2 + 5)
- player 4: 17 = (4 + 1 + 3 + 2 + 7)
- player 5: 1

Tiempo de espera total = 52.

2. Variables y Restricciones:

Datos de Entrada:

En la Fig. 1 se tiene como dato de entrada una matriz, esta nos permite identificar la relación de los *players* y sus composiciones.

```
tabpxc = [[1, 0, 1, 1, 0, 1, 1, 0, 1],
          [1, 0, 1, 1, 1, 1, 0, 1, 0],
          [1, 1, 0, 0, 1, 0, 0, 1, 0],
          [1, 0, 0, 0, 1, 0, 1, 0, 1],
          [0, 1, 0, 1, 1, 1, 1, 0, 0]]
```

Fig.1

En la Fig. 2 se tiene como dato de entrada un arreglo, el cual nos permite identificar la relación de la duración de su respectiva composición.

```
arrcxd = [2, 4, 1, 3, 3, 2, 5, 7, 6]
```

Fig.2

Variables y Dominios:

- Arreglo tabcxo

La variable *tabcxo* representa el número de la composición (eje x) por el orden de la composición (eje y), está implementada mediante un bool, lo que le permite solamente tomar valores entre 1 y 0.la representación en código se muestra a continuación.

```
tabcxo = []
for i in range(len(tabpxc[0])):
    fila = []
    for j in range(len(tabpxc[0])):
      fila += [model.NewBoolVar('x'+str(i)+str(j))]
    tabcxo += [fila]
```

Fig.3

Por ejemplo:

```
Relacion de que composicion toca cada hora del evento
Comp1-> 0       0       0       0       0       1       0       0
Comp2-> 0       0       0       1       0       0       0       0
Comp3-> 0       0       1       0       0       0       0       0
Comp4-> 0       0       0       1       0       0       0       0
Comp5-> 0       0       0       0       0       0       1       0
Comp6-> 0       0       0       0       0       0       0       1
Comp7-> 0       1       0       0       0       0       0       0
Comp8-> 0       0       0       0       1       0       0       0
Comp9-> 1       0       0       0       0       0       0       0
```

Fig. 4: Output de *tabpxc*

La interpretación de la matriz *tabpxc* vendría a ser de la siguiente manera: la columna con un uno en cada fila representa el número de composición y el turno el orden de uno a 9. En este caso, el primer turno, es decir la primera composición, es la número 9. En segundo lugar la 7, tercera la 3 y así según corresponda (Fig. 4).

- Arreglo arrcxo

La variable *arrcxo* representa la duración de cada composición, donde el índice es el número de la composición, y el valor representa la duración, está comprendida entre los valores 1 y 9, también se puede observar el llenado de datos, este será igual a j+1 siempre y cuando exista el número y orden de composición en la matriz tabcxo, la representación en código se muestra a continuación.

```
arrcxo = []
for i in range(len(tabcxo)):
    arrcxo += [model.NewIntVar(1, 9, 'x'+str(i))]
    for j in range(len(tabcxo)):
      model.Add(arrcxo[i] == j + 1).OnlyEnforceIf(tabcxo[i][j])
      model.Add(arrcxo[i] != j + 1).OnlyEnforceIf(tabcxo[i][j].Not())
```

Fig.5

- Matriz tmp y invtmp

La variable *tmp* nos permite reorganizar las composiciones según el orden óptimo (arrcxd) del modelo. Esto se logra haciendo un channeling con *OnlyEnforceIf()* que se activa cuando se está evaluando los datos de la composición con el nuevo orden de la programación. Está implementada por un variables booleanas, lo que le permite tomar valores entre 0 y 1. De manera similar invtmp, captura los la inversa de *tmp* (donde tmp = 0, *invtmp* = 1). La representación en código se muestra a continuación.

```
tmp = []
invtmp = []
for i in range(9):
    tmp += [[model.NewIntVar(0,1,'pcomp_'+str(i)+'_'+str(j)) for j in range(5)]]
    invtmp += [[model.NewIntVar(0,1,'invwaitTime_'+str(i)+'_'+str(j)) for j in range(5)]]

for i in range(9):
    for j in range(9):
        for z in range(5):
            model.Add(tmp[i][z] == tabpxc[z][j]).OnlyEnforceIf(tabcxo[j][i])
            model.Add(invtmp[i][z] == -1*tabpxc[z][j]+1).OnlyEnforceIf(tabcxo[j][i])
```

Fig.6

```
Composicion x player: donde las composiciones donde participa son 1
P1->    1       1       1       1       0       0       1       0       1
P2->    0       0       1       1       0       1       1       1       1
P3->    0       0       0       0       1       1       1       1       0
P4->    1       1       0       0       0       0       1       1       0
P5->    0       1       0       1       1       0       0       1       1
```

Fig.7: Output de *tmp*

- Matrices tmp2 y tmp3

  La variable tmp2 y tmp3 están creadas de la misma forma, son matrices comprendidas por valores entre 0 y 100. La variable tmp2 servirá para almacenar todos los valores multiplicados entre la matriz invertirá *invtmp* (la cual nos permitirá multiplicar los espacios comprendidos por 1 en su matriz, los que representan cuando un *player* no pertenece a la composición) y la el arreglo *arrcxd* (el cual almacena el tiempo de espera de cada composición). Por último la matriz tmp3 tomará los valores de la matriz tmp2 siempre y cuando cumplan con el orden y número de composición de la matriz *tabcxo.*

```
tmp2 = []
tmp3 = []
for i in range(5):
    tmp2 += [[model.NewIntVar(0,100,'waitTime_'+str(i)+'_'+str(j)) for j in range(9)]]
    tmp3 += [[model.NewIntVar(0,100,'waitTime_'+str(i)+'_'+str(j)) for j in range(9)]]

for i in range(9):
    for j in range(9):
        for z in range(5):
            model.AddMultiplicationEquality(tmp2[z][j], [invtmp[j][z], arrcxd[j]])
            model.Add(tmp3[z][j] == tmp2[z][j]).OnlyEnforceIf(tabcxo[i][j])
```

Fig.8

3. Optimización

Respecto al problema, se exige que se minimice el tiempo de espera total de los *players*. Para esto declaramos la variable *waiti*, la cual almacena el tiempo total de espera para todos los *players,* dicho esto, la función de optimización se basa en minimizar el valor de la variable *waiti* como se observa a continuación.

```
wait = []
for i in range(5):
    wait += [sum(tmp2[i])]
model.Add(sum(wait) == waiti)
model.Minimize(waiti)
```

Fig.9

4. Heurísticas

Heurística de valor	Heurística de variable	Solución	Tiempo Total de espera (Minimizado)	Tiempo de ejecución (s)
Menor Valor	Primera Variable	7 5 3 4 8 9 2 6 1	62	0,4109
Menor Valor	Primera Variable Mínimo más pequeño	7 5 3 4 8 9 2 6 1	62	0,3953
Menor Valor	Variable Valor Máximo más alto	7 5 3 4 8 9 2 6 1	62	0,4190
Mayor Valor	Primera Variable	7 5 3 4 8 9 2 6 1	62	0,3864
Mayor Valor	Primera Variable Mínimo más pequeño	7 5 3 4 8 9 2 6 1	62	0,4037
Mayor Valor	Variable Valor Máximo más alto	7 5 3 4 8 9 2 6 1	62	0,3899

Tabla.2

Como se puede observar en la tabla, se muestra una comparación de los resultados y tiempo de ejecución para diferentes tipos de heurísticas aplicadas al modelo. Para ambos, el tiempo total de espera y el tiempo de ejecución, no hay mucha variación. En el primer caso, siempre se consigue la misma solución, mientras que para el tiempo de ejecución hay

una variación mínima de tiempo de segundos. Por lo tanto, no se provoca una diferencia significativa entre las diferentes heurísticas, desde el punto de vista del tiempo de ejecución.

5. Conclusiones

El modelo desarrollado fue desarrollado de manera parcial. El modelo es capaz de reorganizar la programación de las composiciones según las condiciones dadas (composición 2 antes que la 9 y 6 después de 5) y de calcular la suma de los tiempos de espera. Sin embargo, nos cruzamos con un detalle bastante complicado que fue, excluir los tiempos previos a la llegada del artista y posteriores a su salida del cálculo del tiempo total. Se empleó amplio tiempo para solucionar el detalle para cumplir los requisitos del problema, no obstante, no se consiguió resolver el detalle. Tenemos en cuenta que es un problema con complejidad y aparenta que nos falta más experiencia en la programación con restricciones. Creemos que hay metodologías y funciones propias de la programación con restricciones que nos hace falta poder aprovechar en su máximo potencial para poder resolver con eficacia este tipo de problemas.