71.14 - Modelos y Optimización I - 3er entrega

Facultad de Ingeniería
Universidad de Buenos Aires

Mermet, Ignacio Javier
98153

Junio 2022

Índice

1 Análisis de las soluciones

Si bien ambas soluciones proveen una solución con valor del objetivo ≈ 5249.622, hay diferencias en los tiempos de ejecución, los cuales se detallan a continuación en 1.1 y 1.2. También se detallan las diferencias en las estrategias para eliminar subtours, mediante las restricciones de cada modelo.

1.1 MTZ

Uno de los dos modelos dados por la cátedra es el ya conocido MTZ. En esta estrategia, se emplean variables enteras $U_i, i = 2, \ldots, n$ que indican el orden en el cual se pasa por la ciudad i.

$$u_i - u_j + (n-1) \times Y_{ij} \leqslant n - 2 \;\forall\; i, j \in \text{cities} \mid i > 1,\; j > 1,\; j \neq i \tag{1}$$

La idea detrás de esta estrategia es crear un recorrido entre todas las ciudades menos la de partida, en la cual no se pueda volver para atrás. Por eso notemos que estando las ciudades indexadas desde 1, las variables U arrancan para $i = 2$. Se requiere una ciudad fuera de este recorrido que cierre el tour. Esta ciudad será nuestra ciudad de partida.

La restricción 1 no permite que se pueda ir de una ciudad con número de orden x a una con número de orden $y < x$. Solo se puede volver a la ciudad inicial, por lo dicho anteriormente que no tiene esta restricción.

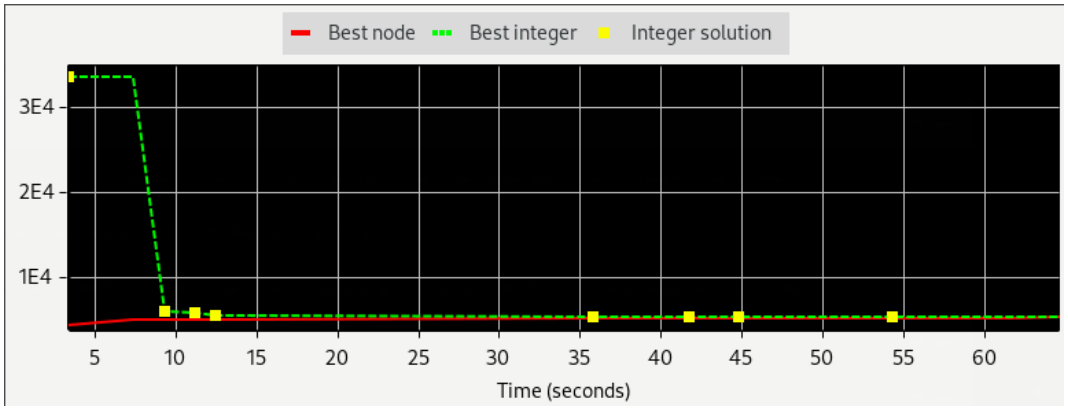

Figura 1: Gráfico de la solapa `Statistics`

```
Root node processing (before b&c):
  Real time             =    5.65 sec. (3495.92 ticks)
Parallel b&c, 32 threads:
  Real time             =   80.17 sec. (41031.31 ticks)
  Sync time (average)   =   23.10 sec.
  Wait time (average)   =    0.08 sec.
                          ------------
Total (root+branch&cut) =   85.82 sec. (44527.24 ticks)
```

Figura 2: Fragmento de la salida bajo la solapa `Engine Log`

Notemos que la solución se alcanza en un tiempo de 85.82 segundos (más algunos segundos de inicialización).

El valor óptimo inicial corresponde al costo de la solución con las ciudades ordenadas, pero todas las soluciones óptimas encontradas luego son estrictamente mejores que la anterior.

1.2 Subtours

Este modelo es conocido como `DFJ` por Dantzig, Fulkerson, Jhonson. En cada ejecución (ya que se implementa de modo iterativo) se toma el subtour S de menor cardinalidad. Para ese subtour, se agregan restricciones que dicen que se sale del subtour al menos una vez.

$$\forall S \sum_{i \in S} \sum_{j \notin S} X_{ij} \leqslant \mid S \mid -1 \tag{2}$$

Por ello podemos tener solo las variables $X_{ij},\ j > i$, ya que $\forall i \in S$, necesitamos solo las aristas de entrada o de salida de las mismas, y tanto X_{ij} como X_{ji} sirven para indicar que se sale del subtour por i.

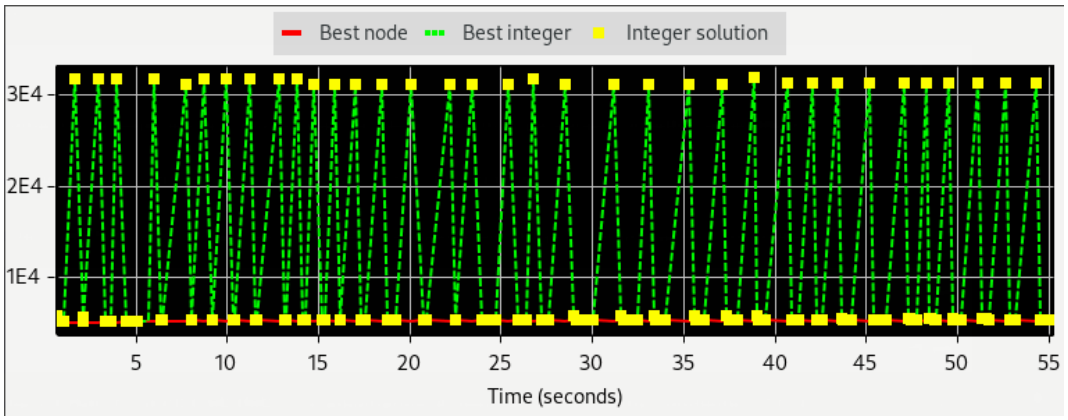

Figura 3: Gráfico de la solapa `Statistics`

```
Root node processing (before b&c):
Parallel b&c, 32 threads:
  Real time             =    1.46 sec. (305.17 ticks)
  Real time             =    0.00 sec. (0.00 ticks)
  Sync time (average)   =    0.00 sec.
  Wait time (average)   =    0.00 sec.
                          ------------
Total (root+branch&cut) =    1.46 sec. (305.17 ticks)
```

Figura 4: Fragmento de la salida bajo la solapa `Engine Log`. Notar que esto se repite más de 30 veces.

Los óptimos encontrados oscilan entre valores muy bajos en la escala (notar que va de 10.000 en 10.000) a valores superiores a 30.000. Esto se debe a que luego de cada paso de

eliminación de subtours, al conectarse el subtour eliminado, la distancia total suele ser grande. Luego, al ejecutarse el motor de optimización, notemos que siempre posterior a cada uno de estos pasos se acerca al valor óptimo.

2 Modificación de la heurística de la segunda entrega

La heurística modificada se puede consultar en el notebook `Greedy+2opt-Problema3.ipynb`. Como resumen, utilizando una heurística de vecino más cercano, se genera un camino que empieza desde cada ciudad. Luego, se toma el de menor costo (distancia total recorrida) y se aplica 2-OPT como optimización local. En la versión de la segunda entrega se generaba una cantidad razonable de candidatos y se optimizaba localmente durante una hora como máximo. Para cumplir con las capacidades, se revisaba también que el camino fuera válido.

En esta instancia, no es necesario revisar que el camino cumpla con las restricciones de capacidad. Y dado el tamaño de la instancia del problema, se pueden generar caminos para todas las ciudades como candidatos iniciales. Esta heurística da un resultado con distancia total recorrida de 5411.213, apenas un 3 % por encima del óptimo.

2.1 MTZ inicializado

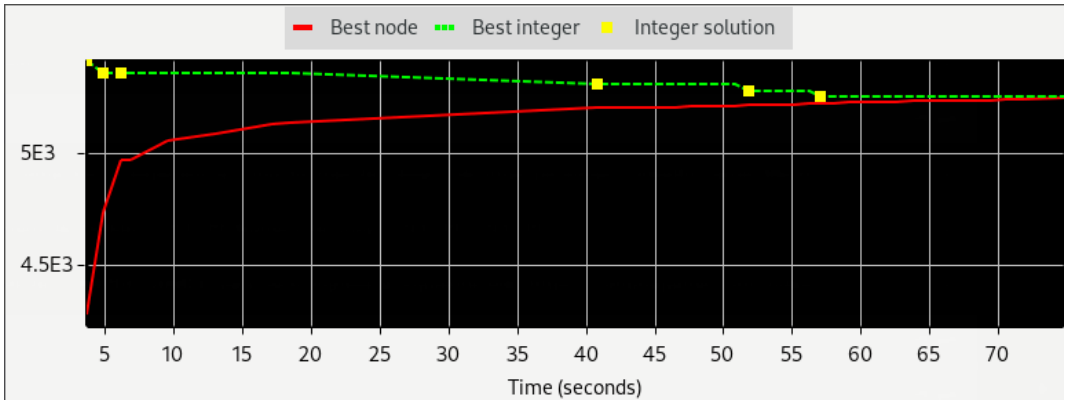

Figura 5: Gráfico de la solapa `Statistics`

```
Root node processing (before b&c):
  Real time             =    9.67 sec. (6852.01 ticks)
Parallel b&c, 32 threads:
  Real time             =   64.71 sec. (31281.19 ticks)
  Sync time (average)   =   20.94 sec.
  Wait time (average)   =    0.04 sec.
                          ------------
Total (root+branch&cut) =   74.38 sec. (38133.19 ticks)
```

Figura 6: Fragmento de la salida bajo la solapa `Engine Log`

Vemos, al comparar con 1.1, que agregar la salida de la heurística como inicialización

el modelo converge $\approx 13\,\%$ más rápido. Las soluciones iniciales encontradas tienen valores mucho más cercanos al óptimo, dado que el valor inicial es mas cercano al optimo.

2.2 Subtours inicializado

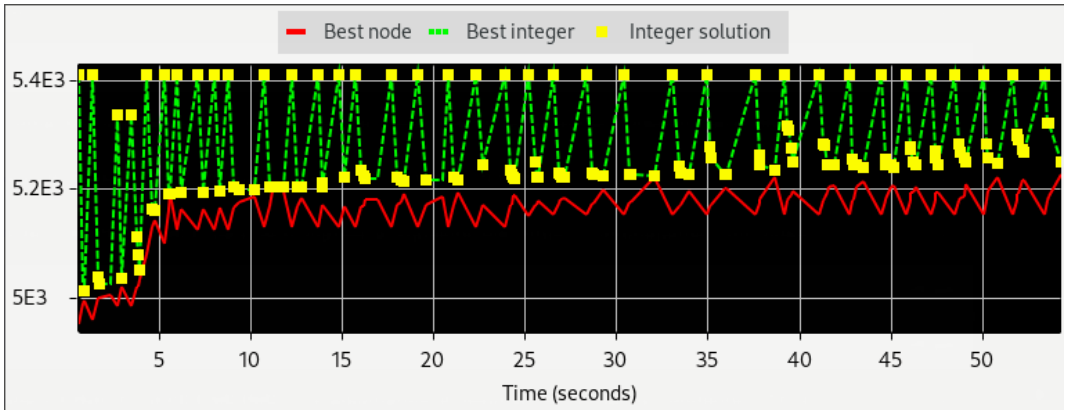

Figura 7: Gráfico de la solapa `Statistics`

```
Root node processing (before b&c):
  Real time              =    1.56 sec. (276.30 ticks)
Parallel b&c, 32 threads:
  Real time              =    0.00 sec. (0.00 ticks)
  Sync time (average)    =    0.00 sec.
  Wait time (average)    =    0.00 sec.
                           ------------
Total (root+branch&cut) =    1.56 sec. (276.30 ticks)
```

Figura 8: Fragmento de la salida bajo la solapa `Engine Log`. Notar que esto se repite más de 30 veces.

En este caso notamos grandes diferencias con 1.2 en cuanto al gráfico de la solapa `Statistics`. Por empezar, la escala utilizada es un orden de magnitud más chica. Por ello podemos apreciar mejor los valores de las soluciones obtenidas. Vemos como el rango de valores entre los cuales se mueven es mucho más chico.

Al tener una inicialización *razonable*, romper los subtours y conectarlos aumenta las distancias, pero jamas podrá superar el costo de 5411 de la heurística. En cambio, en esta instancia del problema, el tour que va en orden por las ciudades tiene costo aproximado de 30.000 (que vimos en 1.2).

3 Conclusiones

En esta instancia particular del problema del viajante, el algoritmo DFJ encuentra el óptimo más rápidamente.

Es importante remarcar que MTZ introduce más variables de decisión y una cantidad cuadrática de restricciones, pero permite resolver el problema en una sola ejecución. DFJ introduce una cantidad exponencial de restricciones al ir introduciendo nuevos subtours que se deban eliminar. Se ejecuta más de 30 veces en esta instancia hasta encontrar el óptimo.