Universidad Politécnica de Cartagena

TEORÍA DE REDES DE TELECOMUNICACIONES

Grado en Ingeniería Telemática
Grado en Ingeniería en Sistemas de Telecomunicación

Curso 2018-2019

Práctica #10. Problema de diseño de topología, capacidad y encaminamiento (TCFA)

(1 sesión)

Autor:

Pablo Pavón Mariño

1 Objetivo

Los objetivos de esta practica son:

1. Crear algoritmos en Net2Plan para resolver distintas variantes de problemas tipo TCFA (Topología, Flujo (routing), asignación de Capacidad) resolviendo la formulación con la librería *Java Optimization Modeler* (JOM).
2. Obtener experiencia con las distintas formas de escribir problemas de optimización con JOM.

2 Duración

Esta práctica esta diseñada para realizarse en una sesión de dos horas.

3 Evaluación

La práctica ha sido diseñada para guiar al estudiante en el aprendizaje de Net2Plan. Las anotaciones que los estudiantes hacen en el documento son para uso propio al estudiar la asignatura, y no deben entregar nada al profesor para su evaluación.

4 Documentación

- Documentación de la libreria JOM (ver `http://www.net2plan.com/jom`).
- Documentación de Net2Plan (ver `http://www.net2plan.com/`).
- Instrucciones de este boletín.

5 Trabajo previo

- Leer la sección 7.3 de [1], y los apuntes relacionados con la formulación de encaminamiento flujo-enlace.
- Leer la documentación de JOM en `http://www.net2plan.com/jom`, en particular, como manejar los vectores de variables y restricciones.

6 Problema TCFA

Sea $\mathcal{N}$ un conjunto de nodos, y $\mathcal{D}$ el tráfico ofrecido a la red. Para cada demanda d, se denota h_d como el trafico ofrecido. Estamos interesados en crear un algoritmo de Net2Plan que resuelva un problema TCFA que encuentra (i) los enlaces a instalar en la red, (ii) sus capacidades y (iii) como el trafico de las demandas se encamina sobre los enlaces. Dos nodos pueden conectarse a ningún o un enlace, y la capacidad máxima se denota como U.

El objetivo es minimizar el coste total de la red, dado por la suma de un coste fijo c_{km} por kilómetro de enlace instalado (independientemente de su capacidad) y un coste fijo c_u por unidad de capacidad instalado (independientemente de la longitud del enlace). El problema se formula como sigue:

- Parámetros de entrada (constantes conocidas):
  - $\mathcal{N}$: conjunto de nodos de la red.
  - $\mathcal{E}$: conjunto de enlaces *candidatos* en la red. Hay un enlace candidato por cada par de ndoos. De esta información se extrae $\delta^+(n)$, que es el conjunto de enlaces que salen del nodo n, y $\delta^-(n)$ el conjunto de enlaces que entran al nodo n.
  - $\mathcal{D}$: conjunto de demandas unicast ofrecidas.
  - $d_e, e \in \mathcal{E}$: Distancia en km del enlace e.
  - $h_d, d \in \mathcal{D}$: Tráfico ofrecido de la demanda d.
  - U: Capacidad máxima de un enlace.
- Variables de decisión:
  - $z_e, e \in \mathcal{E}$: 1 si el enlace candidato e se instala realmente, 0 en caso contrario (no hay enlace, y por tanto, la capacidad de ese enlace candidato debe ser cero.
  - $x_{de}, d \in \mathcal{D}, e \in \mathcal{E}$: Tráfico de la demanda d que atraviesa el enlace candidato e.
  - $u_e, e \in \mathcal{E}$: Capacidad del enlace candidato e.
- Formulación:

$$\min \quad c_{km} \sum_e d_e z_e + c_u \sum_e u_e, \quad \text{sujeto a:} \tag{1a}$$

$$u_e \leq U z_e, \quad \forall e \in \mathcal{E} \tag{1b}$$

$$\sum_{e \in \delta^+(n)} x_{de} - \sum_{e \in \delta^-(n)} x_{de} = \begin{cases} h_d, \text{ if } n = a(d) \\ -h_d, \text{ if } n = b(d) \\ 0, \text{ en caso contrario} \end{cases}, \quad \forall d \in \mathcal{D}, \forall n \in \mathcal{N} \tag{1c}$$

$$\sum_d x_{de} \leq u_e, \quad \forall e \in \mathcal{E} \tag{1d}$$

$$u_e \geq 0, x_{de} \geq 0, \quad \forall d \in \mathcal{D}, e \in \mathcal{E} \tag{1e}$$

La función objetivo (1a) minimiza el coste total de la red, sumando el coste de los enlaces y el coste de sus capacidades. Las restricciones (1b) hacen que (i) si un enlace candidato e no existe ($z_e = 0$), entonces no se le asigna capacidad ($u_e = 0$), y (ii) si un enlace candidato si existe ($z_e = 1$), entonces la capacidad del enlace esta limitada por U. Las restricciones (1c) son las restricciones de conservación de flujo. Las restricciones (1d) significan que, en cada enlace, el tráfico que circula en él es menor o igual a su capacidad (y por tanto, el enlace no esta saturado). Finalmente, (1e) significa que la capacidad de los enlaes y el trafico que circula no puede tomar valores negativos.

7 Algoritmo Net2Plan

El alumno debe desarrollar un algoritmo Net2Plan que resuelva el problema (1) siguiendo los siguientes pasos:

1. Copia el fichero `AlgorithmTemplate.java` del Aula Virtual y renómbralo como `FlowLinkUnicast.java`.
2. El algoritmo tiene los siguientes parámetros de entrada:
   - `maximumLinkCapacity`, con valor por defecto 1000 (U en (1)).
   - `costPerGbps`, con valor por defecto 1 (c_u en (1)).
   - `costPerKm`, con valor por defecto 1 (c_{km} en (1)).
3. Elimina todos los enlaces de la red.
4. Establece el tipo de routing como *source routing*.
5. Añade una red tipo full mesh, en el diseño `netPlan` (un enlace por cada par de nodos). Estos enlaces serán los enlaces candidatos $\mathcal{E}$ en (1). La capacidad de los enlaces deben establecerse a 0, su longitud en km se establece como la distancia euclidea entre los nodos (utilice `getNodePairEuclideanDistance`), y la velocidad de propagación es 200000 km/s.
6. Cree un objeto de tipo `OptimizationProblem` (p.ej. de nombre `op`).
7. Añada al problema las siguientes variables de decisión:
   - `z_e`: una variable por cada enlace candidato, que puede tomar el valor 0 o 1. *Double.MAX_ VALUE.*
   - `u_e`: una variable por cada enlace candidato, con una capacidad que puede tomar valores entre 0 y `maximumLinkCapacity`.
   - `x_de`: una variable por cada demanda y enlace candidato. El valor mínimo de la variable es 0, y el máximo *Double.MAX_ VALUE.*
8. Establezca la función objetivo del probleman.
9. Utilice un bucle for, con una iteración por cada enlace (candidato) para añadir las restricciones (1b). Se añade una restricción en cada iteración. Para añadir la restricción de un enlace candidato `e`, establezca un parámetro de entrada de JOM con nombre `e`, con el valor del índice del enlace de la iteración.
10. Utilice un bucle for doble con una iteración por cada nodo de la red y para cada nodo, una iteración para cada demanda, para añadir las restricciones (1c). Para añadir la restriccion de convesarción de flujo de una demanda `d` y un nodo `n`:
    - Establezca los parámetros de entrada de JOM:
      - `deltaPlus` con los índices de los enlaces candidatos de salida del nodo actual. Para ello utilice el método *getOutgoingLinks* del objeto node para obtener los enlaces salientes, y el método *NetPlan.getIndexes* para convertir la colección de enlaces a sus índices.
      - `deltaMinus` con los índices de los enlaces candidatos de entrada al nodo actual. Para ello utilice el método *getIncomingLinks* del objeto nodo para obtener los enlaces entrantes y el método *NetPlan.getIndexes* para convertir la colección de enlaces a sus índices.
      - `h_d` con la actual demanda de tráfico ofrecido.
      - `d` con el actual índice de la demanda.
    - Establezca la restricción utilizando la función `sum`, sobre `x_de`, pero restringiendo la suma a los elementos en la fila `d` y las columnas en `deltaPlus` o `deltaMinus`.
11. Utilice un bucle for con una iteración por cada enlace candidato para añadir las restricciones (1d). Se añade una restricción en cada iteración del bucle. Para añadir la restricción de un enlace candidato `e`:

- Establezca el parámetro de entrada JOM de nombre `e` con el índice del enlace candidato actual.
- Establezca la restricción utilizando la función `sum`, sobre todas las demandas (utilizar la palabra clave de JOM `all` en la coordenada de la demanda), y el enlace candidato de índice `e`.

12. Llame al solver para encontrar la solución numérica. Utilice la opción `maxSolverTimeInSeconds` para establecer el tiempo máximo de ejecución del solver a 10 segundos. El solver devolverá la mejor solución encontrada después de 10 segundos, si no la encuentra antes. Puede suceder que el solver no encuentre una solución factible en el tiempo indicado. Debe chequearlo utilizando el método `solutionIsFeasible` de `OptimizationProblem` y que salte la excepción si no se encontró una solución factible.

13. Extraiga la solución primal obtenida en `x_de`, y conviértala en un objeto *DoubleMatrix2D* utilizando el método *view2D*. Un objeto de la clase *DoubleMatrix2D* contiene una matriz 2D y permite hacer operaciones con ella eficientemente. Utilice el método de *NetPlan* llamado *setRoutingFromDemandLinkCarriedTraffic*. Este método crea automáticamente las rutas en la red que son consistentes con lo que aparece en la matriz x_{de} 2D. Tenga en cuenta que los valores obtenidos en x_{de}, cada coordenada (d, e) contiene la cantidad de tráfico de la demanda d en el enlace e, y *no* la fracción de tráfico respecto a h_d. Esto es importante cuando llama al método *setRoutingFromDemandLinkCarriedTraffic*. Esta formulación no puede crear bucles (soluciones con bucles no son nunca óptimas, ya que son estrictamente peores que las mismas rutas con bucles) por lo tanto establezca la opción que automáticamente los elimina, a false.

14. Extraiga la solución primal obtenida en `u_e`. Establezca la capacidad de los enlaces candidatos a la dada por `u_e`.

15. Elimina todos los enlaces candidatos que no tienen capacidad instalada, ya que no forman parte del diseño final. para ello utilice el método `removeAllLinksUnused` en el objeto `netPlan`. Este método elimina todos los enlaces con una capacidad menor que la indicada por un threshold. Establezca a 0.01 el valor del threshold. [1]

7.1 Chequear el algoritmo

Cargue la red `example4nodes.n2p`, y la matriz de tráfico `tm4nodes.n2p`. El algoritmo debería devolver una solución con 6 enlaces y una capacidad total instalada sumando todos los enlaces de la red de 108.18 unidades.

Quiz 1. Cargue la red `example4nodes.n2p`, y la matriz de tráfico en `tm4nodes.n2p` (tráfico ofrecido total de 100).

- Ejecute el algoritmo con el parámetro `costPerKm = 0`, y cualquier valor distinto de cero en `costtPerGbps`. ¿Cómo es la topología que se ha creado? ¿Por qué?
- Fija el parámetro `costPerKm = 1`, `maximumLinkCapacity = 1000` y rellena la tabla 1 ejecutando el algoritmo para distintos valores de `costPerGbps`. Explica como cambia la topología conforme aumenta `costPerGbps`.

[1] La razón por la que no se utiliza 0 como threshold es porque el solver puede llegar a asignar un valor muy pequeño cercano a cero a las capacidades de algunos enlaces, por una razón de precisión numérica. Esos enlaces deben ser eliminados del diseño.

Tabla para `example4nodes.n2p`

`c_Gbps`	Número de enlaces	Capacidad total instalada
0.01		
0.1		
1	6	108.18
10		
100		

8 Variaciones del problema

Quiz 2. Modifica la formulación en (1) para forzar a que la capacidad instalada en los enlaces sea un valor entero, múltiplo de C, la llamada capacidad modular. Resuelva la formulación:

$$\min \quad c_{km} \sum_e d_e z_e + c_u \sum_e C a_e, \quad \text{sujeto a:} \tag{2a}$$

$$C a_e \leq U z_e, \quad \forall e \in \mathcal{E} \tag{2b}$$

$$\sum_{e \in \delta^+(n)} x_{de} - \sum_{e \in \delta^-(n)} x_{de} = \begin{cases} h_d, \text{ if } n = a(d) \\ -h_d, \text{ if } n = b(d) \\ 0, \text{ otherwise} \end{cases}, \quad \forall d \in \mathcal{D}, \forall n \in \mathcal{N} \tag{2c}$$

$$\sum_d x_{de} \leq C a_e, \quad \forall e \in \mathcal{E} \tag{2d}$$

$$a_e \in \{0, 1, 2, \ldots\}, x_{de} \geq 0, \quad \forall d \in \mathcal{D}, e \in \mathcal{E} \tag{2e}$$

donde las variables de decisión $a_e, e \in \mathcal{E}$ son ahora el número (entero) de modulos de capacidad instalados en el enlace e, y Ca_e es ahora la capacidad del enlace e, y reemplaza a u_e en la formulación (1). Introduce C como un parámetro definido por el usuario con nombre `capacityModule`, valor por defecto 10.

Nota: El algoritmo con los valores de los parámetros por defecto debería instalar 8 enlaces y 120 unidades de capacidad en la red `example4nodes.n2p` con la matriz de tráfico `tm4nodes.n2p`.

Quiz 3. ¿La formulación (2) produce *siempre* resultados con mayor coste que la formulación (1)?¿Por qué?

9 Forma matricial de las restricciones del problema (opcional)

Como se ha visto en otras sesiones de prácticas, las restricciones de conservación de flujo (1c) y restricciones de capacidad (1d) se pueden establacer con una única llamada al método `addConstraint`, utilizando las matrices y vectores de las restricciones.

- Las restricciones de conservación de flujo se pueden establecer como:

$$A_{ne} \times x'_{de} = A_{nd} \times \mathbf{diag}(h) \tag{3}$$

donde A_{ne} es la matriz de incidencia nodo-enlace, que puede obtenerse utilizando el método `getMatrixNodeLinkIncidence` de `NetPlan`. A_{nd} es la matriz de incidencia nodo-demanda,

que se puede obtener con el método `getMatrixNodeDemandIncidence` de `NetPlan`. Finalmente `h` es el vector con el tráfico ofrecido de las demandas, que se puede obtener con el método `getVectorDemandOfferedTraffic`.

- Las restricciones de capacidad de los enlaces se puede establecer con la expresión:

```
sum(x_de,1) <= u_e
```

que hace que el vector de tráfico que lleva cada enlace, elemento a elemento, sea menor or igual que el vector de capacidad de los enlaces.

Las restricciones (1b), se pueden establecer de forma vectorial facilmente con:

```
u_e <= U * z_e
```

Quiz 4. Reescribe el algoritmo (1) utilizando la forma matricial.

10 Trabajo en casa después de la sesión de prácticas

El alumno puede completar todos los *Quizs* que no haya finalizado durante la sesión de prácticas.

Bibliography

[1] *P. Pavón Mariño, "Optimization of computer networks. Modeling and algorithms. A hands-on approach", Wiley 2016.*

Bibliography