Universidad Politécnica de Cartagena

TEORÍA DE REDES DE TELECOMUNICACIONES

Grado en Ingeniería Telemática
Grado en Ingeniería en Sistemas de Telecomunicación

Curso 2018-2019

Práctica #5. Enrutamiento del Tráfico. Formulación Flujo-Camino

(1 sesión)

Autor:

Pablo Pavón Mariño
Juan Pedro Muñoz Gea
María Victoria Bueno Delgado
Ángel Antonio Pintado Sedano
Juan Carlos Sánchez Aarnoutse

1. Objetivos

Los objetivos de esta práctica son:

1. Crear algoritmos de Net2Plan que resuelvan algunas variantes de problemas de enrutamiento del tráfico, resolviendo formulaciones flujo-camino utilizando la librería *Java Optimization Modeler* (JOM).
2. Conseguir práctica en las diferentes formas de escribir problemas de optimización en JOM, aprovechando sus capacidades de representación vectorial.
3. Explorar el potencial de la representación matricial de las restricciones de enrutamiento en JOM y Net2Plan.

2. Duración

Esta práctica está diseñada para una sesión de dos horas.

3. Evaluación

Esta práctica ha sido diseñada para guiar al estudiante en su aprendizaje en Net2Plan. Las anotaciones que los estudiantes hagan en este documento son para su uso cuando repasen la asignatura y no es necesario que se entregue al profesor para su evaluación.

4. Documentación

Los recursos necesarios para este trabajo de laboratorio son:

- Documentación de la librería JOM (ver `http://www.net2plan.com/jom`).
- La herramienta Net2Plan y su documentación (ver `http://www.net2plan.com/`).
- Instrucciones contenidas en este guión de la práctica.

5. Trabajo previo antes de venir al laboratorio

- Leer la sección 4.2 y de la sección 4.6.2 a la 4.6.4 de la referencia [1], y los apuntes relacionados con la formulación flujo-camino.
- Refrescar las lecturas de la documentación de JOM en `http://www.net2plan.com/jom`, y en particular, como se manejan los vectores de variables y restricciones.

6. Ancho de banda mínimo para enrutamiento

Estamos interesados en crear algoritmos de Net2Plan que resuelvan el problema del ancho de banda mínimo para enrutamiento utilizando JOM. El algoritmo se define de la siguiente manera:

- Parámetros de entrada (constantes conocidas):
  - $\mathcal{N}$: Conjunto de nodos.
  - $\mathcal{E}$: Conjunto de enlaces.
  - $u_e, e \in \mathcal{E}$: Capacidad de un enlace e.
  - $\mathcal{D}$: Conjunto de demandas unicast.
  - $h_d, d \in \mathcal{D}$: Tráfico ofrecido por una demanda d.
  - $\mathcal{P}_d, d \in \mathcal{D}$: Para cada demanda, lista de los caminos admisibles. En nuestro caso, los k caminos más cortos sin bucles entre los nodos extremos de la demanda, en número de saltos. Estos son los caminos aceptados como rutas potenciales para llevar el tráfico de la demanda.
  - $\mathcal{P}$: El conjunto de todos los caminos admisibles en la red ($\mathcal{P} = \bigcup_{d \in \mathcal{D}} \mathcal{P}_d$). Dado un camino $p \in \mathcal{P}$, denotamos como $d(p)$ a la demanda asociado al camino. Denotamos como l_p el número de enlaces atravesados por el camino p. Dado un enlace e, denotamos como $\mathcal{P}_e$ al conjunto admissible de caminos (i.e. in $\mathcal{P}$) que atraviesan en enlace e.
- Variables de decisión:
  - $x_p, p \in \mathcal{P}$: Tráfico cursado por el camino p
- Formulación:

$$\text{mín} \quad \sum l_p x_p, \quad \text{sujeto a:} \tag{1a}$$

$$\sum_{d \in \mathcal{P}_d} x_p = h_d, \quad \forall d \in \mathcal{D} \tag{1b}$$

$$\sum_{e \in \mathcal{P}_e} x_p \leq u_e, \quad \forall e \in \mathcal{E} \tag{1c}$$

$$x_p \geq 0, \quad \forall p \in \mathcal{P} \tag{1d}$$

La función objetivo (1a) minimiza el total del tráfico en los enlaces: para cada camino, su tráfico cursado multiplicado por el número de enlaces atravesados. Las restricciones (1b) significan que para cada demanda, todo el tráfico se cursa (la suma del tráfico cursado por la ruta de las demandas totaliza el tráfico ofrecido por las demandas). Las restricciones (1c) significan que por cada enlace, el tráfico cursado en el enlace es inferior o igual que la capacidad (y por tanto ningún enlaces es demandado en exceso). Finalmente (1d) prohibe que un camino curse un tráfico negativo ya que esto no tiene ningún significado físico.

7. Algoritmo Net2Plan

El alumno debería desarrollar un algoritmo Net2Plan para resolver el problema (1) siguiendo los siguientes pasos:

1. Copiar el fichero `AlgorithmTemplate.java` que se encuentra en el Aula Virtual y renombrarlo como `FlowPathUnicast.java`.

2. El algoritmo debería tener un parámetro de entrada llamado `k`, con un valor por defecto de 10, con descripción *Maximum number of loopless admissible paths per demand.*

3. Seleccionar el tipo de enrutamiento del objeto `NetPlan` como source routing.

4. Eliminar todas las rutas existentes en el diseño de entrada.

5. Para cada demanda, calcular los `k` caminos sin bucles más cortos en número de saltos entre los nodos extremos de la demanda (`k` es el parámetro de entrada). Para ello, utilizar la función `GraphUtils.getKLooplessShortestPaths`. Todos estos caminos se deben añadir al diseño `NetPlan` de entrada como rutas con un tráfico cursado de cero. Todos ellos son los caminos admisibles. Señalar que entonces:

   - Dada una demanda Net2Plan `d`, el método `d.getRoutes` devuelve las rutas de la demanda (y de este modo el conjunto $\mathcal{P}_d$).
   - Dado un enlace Net2Plan `e`, el método `e.getTraversingRoutes` devuelve las rutas que atraviesan el enlace (y de este modo el conjunto $\mathcal{P}_e$).

6. Crear un objeto del tipo `OptimizationProblem` (p. e. de nombre `op`).

7. Añadir las variables de decisión del problema con nombre `x_p`: una variable por camino admisible (y de este modo, una por cada ruta en el objeto `NetPlan`). El valor mínimo de las variables se establece a cero, el maximo a *Double.MAX_ VALUE.*

8. Establecer la función objetivo del problema. Para esto:

   - Establecer un parámetro de entrada JOM (utilizando el método `setInputParameter`) de nombre `l_p`, que contenga un vector fila con tantos elementos como rutas admisibles, donde la coordenada p contenga el número de saltos de la ruta admisible de índice p. Utilizar el método *netPlan.getVectorRouteNumberOfLinks()* para obtener automáticamente dicho vector.
   - Establecer la función objetivo utilizando cualquiera de estas dos formas:
     - Utilizar la función `sum` de JOM combinada con el operador `.*` (multiplicación elemento a elemento).
     - Utilizar los operadores `*` (multiplicación de matrices) y `'` (matriz transpuesta), ya que la función objetivo es igual al vector fila l_p multiplicado por el vector x_p en forma de columna.

9. Utilizar un bucle for con una iteración por demanda para añadir las restricciones (1b). Dentro de cada iteración del bucle se añade una restricción. Para añadir la restricción de una demanda `d`:

   - Establecer un parámetro de entrada JOM de nombre `h_d`, con un valor igual al tráfico ofrecido de la demanda.
   - Establecer un parámetro de entrada JOM de nombre `P_d`, con un vector que contenga los índices de las rutas que cursan tráfico de la demanda. Utilizar el método *getRoutes* del objeto demanda para obtener las rutas, y el método estático *NetPlan.getIndexes* para convertir la colección de objetos demanda a una colección de sus índices.
   - Establecer la restricción utilizando la función `sum`, sobre `x_p`, pero restringiendo la suma a los elementos en `P_d`.

10. Utilizar un bucle for con una iteración por enlace, para añadir las restricciones (1c). Dentro de cada iteración del bucle se añade una restricción. Para añadir la restricción del enlace `e`:

- Establecer un parámetro de entrada JOM de nombre `u_e`, con un valor igual a la capacidad del enlace.
- Establecer un parámetro de entrada JOM de nombre `P_e`, con un vector que contenga los índices de las rutas que atraviesan el enlace. Utilizar el método *getTraversingRoutes* del objeto enlace para obtener las rutas, y el método estático *NetPlan.getIndexes* para convertir la colección de objetos demanda a una coleccion de sus índices.
- Establecer la restricción utilizando la función `sum`, sobre `x_p`, pero restringiendo la suma a los elementos en `P_e`.

11. Llamar al solver para obtener una solución numérica.
12. Obtener la solución primal obtenida. Entonces, para cada ruta en la red, establecer su tráfico cursado de acuerdo a la solución óptima encontrada.
13. Utilizar el metodo `removeAllRoutesUnused` del objeto `NetPlan` para eliminar las rutas que no se usan. Consideramos que las rutas que no se usan son las que cursan un tráfico inferior a 0.001 unidades de tráfico (rutas con un tráfico cursado diferente de cero, pero con un tráfico tan pequeño, pueden aparecer por errores numéricos en el solver).

7.1. Comprobar el algoritmo

Cargar la red `example7nodesWithTraffic.n2p`. El algoritmo debería generar una solución con una cantidad de ancho de banda consumido en los enlaces de 155,5.

Si reducimos la capacidad de los enlaces a 15 unidades (en lugar de 50), la solución óptima consumirá 174,1 unidades.

Si reducimos la capacidad de los enlaces a 10 unidades, el problema será no factible (no hay suficiente capacidad en los enlaces para cursar el tráfico de ninguna forma).

Quiz 1. ¿El tráfico circula por los caminos más cortos para todas las demandas en el primer caso ($u_e = 50$)? ¿Se mantiene para $u_e = 15$?

8. Variaciones al problema

Quiz 2. Modificar el cálculo de la lista admisible, considerando como admisibles los k caminos más cortos sin bucles medidos en km para cada demanda (es decir, sumando la longitud en km de los enlaces atravesados). Nota: Esto quiere decir que hay que pasarle al método *getKLooplessShortestPaths* un map con el coste de los enlaces, donde el coste de cada enlace es igual a su longitud en km.

Quiz 3. Añadir un parámetro de entrada al problema, para que el usuario pueda limitar el número máximo de saltos de cualquier camino admisible. Nombre del parámetro: `maxNumHops`, valor por defecto `4`, descripción "Máximo número de satos de un camino admisible".

Quiz 4. Modificar la formulación (1) para que ahora las variables x_p sean renombradas como $\hat{x}_p$, y midan la **fracción** ($\in [0; 1]$) del tráfico de la demanda $d(p)$, que es cursada a través del camino p.

- Variables de decisión:
  - $\hat{x}_p, p \in \mathcal{P}$: Fracción $\in [0, 1]$ del tráfico cursado de la demanda $d(p)$ que va a través del camino p.

- Formulación:

$$\text{mín} \quad \sum l_p \hat{h}_p x_p, \quad \text{sujeto a:} \tag{2a}$$

$$\sum_{d \in \mathcal{P}_d} \hat{x}_p = 1, \quad \forall d \in \mathcal{D} \tag{2b}$$

$$\sum_{e \in \mathcal{P}_e} h_p \hat{x}_p \leq u_e, \quad \forall e \in \mathcal{E} \tag{2c}$$

$$\hat{x}_p \geq 0, \quad \forall p \in \mathcal{P} \tag{2d}$$

Rehacer el algoritmo con las nuevas variables de decisión (cambiando su nombre en JOM a `xx_p`). Las constantes h_p proporcionan para cada camino p, el tráfico ofrecido de su demanda asociada $d(p)$. El vector h_p se puede obtener en Net2Plan utilizando el método de *NetPlan*:

`getVectorRouteOfferedTrafficOfAssociatedDemand`

Observe que si las variables de decisión `xx_p` se restringen a valores enteros (esto es, o 0 o 1), entonces el enrutamiento se restringe a no bifurcado (*non-bifurcated*). Añada un parámetro de entrada al problema con el nombre `isNonBifurcated`, cuyo valor por defecto será `false`. Use este parámetro para permitir que el usuario pueda elegir si el enrutamiento es restringido o no a bifurcado.

Quiz 5. Desarrolle un algoritmo de Net2Plan que resuelva una formulación de flujo-camino (*flow-path*) para encontrar el enrutamiento que maximice el ancho de banda en espera en el cuello de botella (*bottleneck*). Tome la formulación de los apuntes de clase o de la sección 4.2 de la referencia [1]:

$$\text{máx} \quad u, \quad \text{sujeto a:} \tag{3a}$$

$$\sum_{d \in \mathcal{P}_d} x_p = h_d, \quad \forall d \in \mathcal{D} \tag{3b}$$

$$\sum_{e \in \mathcal{P}_e} x_p \leq u_e - u, \quad \forall e \in \mathcal{E} \tag{3c}$$

$$x_p \geq 0, \quad \forall p \in \mathcal{P} \tag{3d}$$

9. Forma matricial de las restricciones del problema

9.1. Restricciones de satisfacción de demanda

Las restricciones de satisfacción de la demanda en (1) toman la forma de:

$$\sum_{p \in \mathcal{P}_d} x_p = h_d, \forall d \in \mathcal{D}$$

Todas las restricciones $|\mathcal{D}|$ pueden ser representadas por una única igualdad vectorial como la siguiente:

$$A_{dp} \times x'_p = h' \tag{4}$$

Donde:

- A_{dp} Es la matriz de asignación demanda a ruta (*demand-to-route*). Esto es, una matriz $|\mathcal{D}| \times |\mathcal{P}|$ con una fila por demanda y una columna por camino admisible. La coordenada (d, p) es 1 si el camino p es un camino admisible asociado a la demanda d, y 0 en el caso contrario.
    - La matriz de asignación demanda a ruta (*demand-to-route*) puede obtenerse como una matriz dispersa empleando el método de la clase `NetPlan`:

      `getMatrixDemand2RouteAssignment`

- x_p es un vector fila $1 \times |\mathcal{P}|$ que contiene las variables de decisión (el tráfico cursado por el camino p).
- h es un vector fila $1 \times |\mathcal{D}|$ con el tráfico ofrecido por cada demanda.
    - El vector de tráfico ofrecido puede obtenerse en Net2Plan empleando el método de la clase `NetPlan`:

      `getVectorDemandOfferedTraffic`

9.2. Restricciones de la capacidad de enlace

La restricciones de la capacidad de enlace en (1) toman la forma de:

$$\sum_{p \in \mathcal{P}_e} x_p \leq u_e, \forall e \in \mathcal{E}$$

Todas las restricciones de $|\mathcal{E}|$ pueden representarse como una única desigualdad de la siguiente forma:

$$A_{ep} \times x'_p \leq u' \tag{5}$$

Donde:

- A_{ep} en la matriz de asignación enlace a ruta (*link-to-route*). Esto es, una matriz $|\mathcal{E}| \times |\mathcal{P}|$ con una fila por enlace y una columna por cada camino admisible. La coordenada (e, p) es el número de veces que el camino p atraviesa el enlace e.
    - La matriz de asignación enlace a ruta (*link-to-route*) puede obtenerse como una matriz de dispersión en Net2Plan por medio del método de la clase `NetPlan`:

      `getMatrixLink2RouteAssignment`

- x_p es un vector fila $1 \times |\mathcal{P}|$ con las variables de decisión (el tráfico cursado por el camino p).
- u es un vector fila $1 \times |\mathcal{E}|$ con la capacidad de cada enlace.
    - El vector de la capacidad de enlace puede obtenerse en Net2Plan por medio del método de la clase `NetPlan`:

```
getVectorLinkCapacity
```

Quiz 6. Reescriba las restricciones de satisfacción de la demanda y de la capacida de enlace empleando la formulación matricial. Remarcar que el operador de JOM `*` realiza la multiplicación estándar de matrices.

10. Trabajo para casa después de la sesión de laboratorio

El estudiante debería completar todos los *Quizs* que no haya podido finalizar durante la sesión en el laboratorio.

Bibliografía

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