Universidad Politécnica de Cartagena

PLANIFICACIÓN Y GESTIÓN DE REDES

Grado en Ingeniería Telemática
Curso 2015-2016

Práctica 3. Algoritmos de tipo búsqueda tabú (*tabu search*) para el diseño de un encaminamiento con protección 1+1

Autor:

Pablo Pavón Mariño

1 Objetivos

Los objetivos de esta práctica son:

1. Desarrollar en `net2plan` un algoritmo para el diseño de encaminamiento no bifurcado con protección 1+1, basado en un heurístico de tipo *búsqueda tabú* (*tabu search*). Analizar los resultados y calidad de la solución. Desarrollar posibles variaciones del algoritmo.

2 Duración

Esta práctica tiene una duración de 1 sesión, cumpliendo un total de 3 horas de laboratorio.

3 Evaluación

Los alumnos no tienen que entregar ningún material al finalizar esta práctica. Este boletín es para el estudio del alumno. En él, el alumno deberá resolver los problemas planteados y anotar las aclaraciones que estime oportunas para su posterior repaso en casa.

4 Documentación empleada

La información necesaria para resolver esta práctica se encuentra en:

- Ayuda de la herramienta `net2plan` (`http://www.net2plan.com/`).
- Instrucciones básicas presentes en este enunciado.
- Apuntes de la asignatura. Sección 12.5 del libro.

Se recomienda a los alumnos que, antes de realizar la práctica, vean la documentación relativa a la *Resilience simulation tool* de `net2plan`. En particular, se recomienda el video tutorial *Getting started with Net2Plan: Resilience Simulation tool*, que se puede encontrar en el canal de Youtube de `net2plan`.

5 Problema de encaminamiento no bifurcado con protección 1+1

Sea una topología $G(N, E)$ dada por un conjunto de nodos N, y un conjunto E de enlaces entre ellos. Las capacidades u_e de los enlaces son valores conocidos. El tráfico ofrecido a la red está compuesto por un conjunto conocido D de demandas. Para cada demanda d, conocemos el tráfico ofrecido h_d, y sus nodos origen y destino.

El objetivo de este problema es encontrar para cada demanda d, dos caminos p_p, p_b: uno que actúe de camino primario (p_p), y otro de camino de protección o *backup* (p_b). Los caminos p_p y p_b no deben tener enlaces en común. De esta manera, en caso de que un enlace de la red caiga, se garantiza que para todas las demandas, uno de los dos caminos (principal o *backup*) siempre sobrevive.

En los esquemas de protección 1+1, los caminos primarios y de respaldo transmiten simultáneamente una copia del mismo tráfico. Es el nodo destino el encargado de aceptar la copia proveniente

del camino principal, descartando la copia del camino de respaldo, salvo cuando no se reciba tráfico (por un fallo en la red) del camino principal.

En el caso que veremos en esta práctica, el objetivo de diseño será encontrar el encaminamiento con protección 1+1 que minimice la congestión de red. Es decir, que minimice la utilización en el enlace de la red que más utilización tiene. Naturalmente, a la hora de calcular la utilización de un enlace, debemos cpmsiderar el tráfico que atraviesa el enlace perteneciente a los caminos principales y a los de respaldo.

6 Algoritmo TS para el pb. de encaminamiento con protección 1+1

El problema de encaminamiento no bifurcado con protección 1+1 descrito es un problema $\mathcal{NP}$-completo, y por tanto no se han encontrado algoritmos de complejidad polinomial que lo resuelvan óptimamente. El objetivo de este apartado de la práctica es desarrollar en `net2plan` un algoritmo heurístico de tipo *Tabu Search* para este problema.

6.1 Diseño básico

El objetivo de esta sección es implementar un esquema TS básico (sin criterio de ambición ni memoria a largo plazo) para este problema.

Las características del algoritmo pedido son:

- El algoritmo debe implementarse en una clase de nombre `Offline_fa_tsMinCongestion11.java`. Recibirá como entrada una topología con los nodos y los enlaces de la red, y un conjunto de demandas con el tráfico ofrecido. Se conocen también las capacidades en los enlaces de la red. El algoritmo devolverá un diseño con el encaminamiento no bifurcado encontrado como solución, donde para cada ruta se añade un segmento de protección con un ancho de banda reservado igual al tráfico de la demanda[1].
- Los parámetros de entrada definidos por el usuario serán:
  - `k`: Número máximo de caminos admisibles para cada demanda, con valor por defecto `k=10`. El código incluido como plantilla, incluye las llamadas del objeto *NetPlan* que (i) generan los k caminos sin ciclos más cortos para cada demanda, y (ii) a partir de ellos, genera todos los posibles pares de caminos que sean disjuntos enlace a enlace. Se sugiere al alumno que analice este código para entender cómo manejar la lista de pares 1+1 candidatos (objeto `cpl11` en el código).
  - `ts_maxNumIt`: Número de iteraciones a realizar por el algoritmo. Se utiliza como condición de parada. El valor por defecto de este parámetro es `ts_maxNumIt = 1000`.
  - `ts_tabuListTenureFraction`: Tamaño máximo de la lista tabú, medido como un porcentaje respecto al número total de demandas en la red. El valor por defecto para este parámetro será `ts_tabuListTenureFraction=0.5`, que indica que el tamaño será igual al 50% (redondeado hacia abajo) del número de demandas en la red.
- **Codificación de la solución:** La solución se codificará como un array de enteros, con tantos elementos como demandas. La coordenada d, corresponde a la demanda con índice d. El valor que contiene es un número entre 0 y el número de pares 11 admisibles para esa demanda (menos 1), con la posición en la lista `cpl11` de ese par de caminos. Es decir (ejemplo):

[1] Para añadir un segmento de protección, utilice el método `addProtectionSegment` de la clase `NetPlan`

  - Si `demand` es un objeto de tipo `Demand`.
  - `int dIndex = demand.getIndex ()`: devuelve el índice de la demanda.
  - `cpl11.get(demand)` es una lista de pares 1+1 de caminos asociados a la demanda.
  - Si `currentSol_d` es el array `int []` que asocia a cada demanda el identificador del par 1+1 asociado, entonces `currentSol_d [dIndex]` es el identificador del par elegido para la demanda de índice `dIndex`.
  - Entonces, `cpl11.get(demand).get(currentSol_d [dIndex])` es el par de caminos 1+1 que deben cursar tráfico de la demanda, según la solución codificada en `currentSol_d`.

- **Lista tabú**: La lista tabú no contendrá soluciones completas, sino atributos de esa solución. En concreto, cuando de una solución x_i pasemos a una solución vecina x_{i+1}, introduciremos en la lista tabú la demanda d en la que el encamnamiento de x_i y x_{i+1} difieren. Igualmente, si v es una solución vecina de la solución actual x, comprobaremos que la demanda d en la que difieren x y v no está en la lista tabú.

  El resultado obtenido es que una vez se cambia el encaminamiento de una demanda d, esa demanda no varía su encaminamiento durante al menos `ts_tabuListTenure` iteraciones.

6.2 Ayudas para la realización del algoritmo

El alumno debe utilizar como plantilla el código de la clase
`Offline_fa_tsMinCongestion11_template.java` que se incluye en Aula Virtual.

6.3 Variación del algoritmo: criterio de ambición

Modifique el código de su algoritmo, para incluir el siguiente criterio de ambición: dada una solución vecina, si esta solución mejora la congestión de la mejor solución encontrada hasta el momento (`best_solution`), entonces se aceptará como posible vecino, aunque se encuentre en la lista tabú.

7 Posibles variaciones (opcional)

Se sugieren algunas variaciones al algoritmo que pueden ser intentadas:

- Modificar la función objetivo del algoritmo, de tal manera que se busque minimizar una función F dada por la suma del ancho de banda consumido en la red, a la que se le sume una penalización elevada, cuando el tráfico en un enlace excede su capacidad.
- Modificar la estructura básica, de tal manera que cuando se encuentre un vecino (no prohibido por la lista tabú) que mejore la solución actual, se salte inmediatamente a él.
- Modificar la definición de vecindario de una solución: dos soluciones se consideran vecinas si se diferencian en la forma de encaminar una o dos demandas (no sólo una, como en la práctica realizada).
- Modificar la generación de la lista de pares 1+1 candidatos. El usuario pasará un parámetro de entrada `maxPathLengthKm` con la distancia máxima (en km) admitida para cualquier camino (principal o de respaldo). El valor por defecto será `maxPathLengthKm = 4500`. Los caminos admisibles (a partir de los cuales se generan los pares 1+1 admisibles) serán los `k` primeros caminos sin ciclos más cortos en km, siempre y cuando no superen la limitación de distancia dada por `maxPathLengthKm`.

- Implementar la funcionalidad de que si el algoritmo lleva más de un número dado de iteraciones (parámetro de entrada) sin mejorar la solución incumbente (la mejor solución histórica encontrada), salte a una nueva solución elegida aleatoriamente (todas las demandas eligen aleatoriamente un par 1+1 entre los disponibles).
- Añadir un sistema de memoria a largo plazo, que almacene para cada posible par 1+1, el número de veces que ha aparecido en alguna solución. Cuando el algoritmo lleve un número dado de iteraciones sin mejorar la solución incumbente (la mejor encontrada hasta la fecha), se cambia la solución actual cogiendo para la mitad de las demandas, elegidas aleatoriamente, el par 1+1 que menos frecuencia tiene.