Universidad Politécnica de Cartagena

PLANIFICACIÓN Y GESTIÓN DE REDES

Grado en Ingeniería Telemática
Curso 2012-2013

Práctica 2. Algoritmos de enfriamiento simulado (*simulated annealing*)

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 basado en un heurístico de tipo *enfriamiento simulado* (*simulated annealing*). 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.

5. Algoritmos de tipo *Simulated Annealing* (SAN)

Los algoritmos SAN son algoritmos iterativos, que parten de una solución inicial x_0, y en cada iteración se salta de una solución al problema a otra. Dada una solución actual x, se elige aleatoriamente una solución vecina v. Si el coste de la solución vecina mejora el coste de la solución actual ($c(v) < c(x)$), se adopta la nueva solución vecina como solución actual. En caso contrario, todavia es posible aceptar la nueva solución vecina (aunque sea de coste peor), y esto se hace con una probabilidad P dada por:

$$P = e^{-\frac{c(v)-c(x)}{T}}$$

Donde T es una variable global del sistema llamada *temperatura del sistema*, cumpliéndose que:

- Cuando la temperatura es un valor alto, la probabilidad P de aceptar una solución que empeore la actual es alta. Por ejemplo, si T es p.e. 100 veces la cantidad $c(v) - c(x)$ en que se empeora el coste, entonces la probabilidad es $P \approx 0{,}99$.

- Cuando la temperatura es un valor bajo, la probabilidad de aceptar una solución que empeore la solución actual es muy pequeña. Por ejemplo, si T es p.e. 10 veces inferior a $c(v) - c(x)$, entonces la probabilidad de aceptar la solución es $P \approx 0{,}000045$.

A continuación, mostramos un esquema habitual que ilustra el funcionamiento del algoritmo. Las iteraciones se organizan en dos bucles anidados, uno interno y uno externo. En cada iteración del bucle externo se disminuye la temperatura (en este caso de manera multiplicativa), y en cada iteración del bucle interno se itera la solución siempre con la misma temperatura.

Algorithm 5.1: SAN()

main
- $x = x_0$ **comment:** solución actual igual a la inicial
- $x_{best} = x_0$ **comment:** mejor solución igual a la inicial
- $T = T_0$ **comment:** temperatura igual a la temperatura inicial

while stopping criteria in outer loop does not hold
do
- **while** stopping criteria in inner loop does not hold
  do
  - $v =$ solución vecina de x elegida aleatoriamente
  - $\Delta c = c(v) - c(x)$
  - **if** $\Delta c < 0$
    - **then**
      - **comment:** la solución vecina v mejora la solución actual x
      - $x = v$
      - **if** $c(v) < c(x_{best})$
        then $x_{best} = v$
    - **else** $x = v$, con probabilidad $e^{-\frac{\Delta c}{T}}$
- $T = \alpha T$ **comment:** reducir la temperatura

return (x_{best})

5.1. Problema de encaminamiento no bifurcado

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.

Para cada demanda, calculamos una lista de caminos admisibles formado por los k caminos más cortos *en km* entre los nodos extremos de la demanda. El parámetro k es un parámetro de entrada para el algoritmo.

Se trata de encontrar el encaminamiento no bifurcado que minimice la congestión de red, medida como la utilización en el enlace de la red que mayor utilización tiene.

Es posible formular el problema de la siguiente manera:

Parámetros de entrada:

$P_d, d \in D = \{$Conjunto de caminos admisibles para la demanda $d\}$

$P = \{$Unión de todos los caminos admisibles para las demandas$\}$

$h_p, p \in P = \{$tráfico ofrecido por la demanda $d(p)$ asociada al camino $p\}$

Variables de decisión:

$x_p, p \in P = \{1$ si la demanda $d(p)$ se encamina por el camino p, 0 en caso contrario$\}$

$\rho = \{$Utilización en el enlace más cargado$\}$

$$\text{mín}\, \rho \tag{1a}$$

$$\sum_{p \in P_d} x_p = 1 \quad \forall d \in D \tag{1b}$$

$$\sum_{p \in P_e} h_p x_p \leq \rho u_e \quad \forall e \in E \tag{1c}$$

La función objetivo (1a) intenta minimizar la congestión. Las restricciones (1b), una para cada demanda, hacen que todas las demandas se encaminen por un y sólo un camino (encaminamiento no bifurcado). Las restricciones (1c), hacen que la utilización de cualquier enlace sea siempre menor o igual a ρ, y por tanto hará que ρ tome el valor de la utilización del enlace más cargado. El sumatorio de la parte izquierda acumula el tráfico que circula por los caminos que atraviesan el enlace e.

5.2. Algoritmo de tipo SAN para el problema de encaminamiento no bifurcado

El problema de encaminamiento no bifurcado descrito es un problema NP-completo, y por tanto no existen 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 *Simulated Annealing* para este problema.

Las características del algoritmo pedido son:

- El algoritmo debe implementarse en una clase de nombre `FA_SAN_minCongestion.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.
- Los parámetros de entrada definidos por el usuario serán:
  - `k`: Número máximo de caminos admisibles para cada demanda. El alumno deberá utilizar la clase `CandidatePathList` incluida en `net2plan` para crear la lista de k caminos sin ciclos más cortos para cada demanda, que serán los caminos admisibles.
  - `san_numOuterIterations`: Número de iteraciones a ejecutar en el bucle externo del algoritmo. En cada iteración de este bucle, la temperatura del sistema se reduce, tal y como se indicará a continuación.
  - `san_numInnerIterations`: Número de iteraciones a ejecutar en el bucle interno del algoritmo. En cada iteración de este bucle, la temperatura del sistema es siempre la misma.
- La temperatura inicial del sistema se calculará de tal manera que una solución que empeora la congestión en 0.05, tenga una probabilidad de 0.99 de ser aceptada. Es decir:

$$e^{\frac{0{,}05}{T_{init}}} = 0{,}99 \Rightarrow T_{init} = -\frac{0{,}05}{\log 0{,}99}$$

- La temperatura final del sistema se calculará de tal manera que una solución que empeora la congestión en 0.05, tenga una probabilidad de 0.01 de ser aceptada. Es decir:

$$e^{\frac{0,05}{T_{end}}} = 0{,}01 \Rightarrow T_{end} = -\frac{0{,}05}{\log 0{,}01}$$

- En cada iteración del bucle externo, la temperatura se reduce multiplicativamente según factor α. Es decir:

$$T_{i+1} = T_i \times \alpha$$

  El factor α se calcula tal que, si empezamos en la temperatura T_{init}, después de `san_numOuterIterations`, la temperatura sea T_{end}. Por tanto:

$$T_{end} = T_{init} \times \alpha^{\texttt{san\_numOuterIterations}-1} \Rightarrow$$
$$\alpha = \left(\frac{T_{end}}{T_{init}}\right)^{\frac{1}{\texttt{san\_numOuterIterations}-1}}$$

- La solución inicial será aquella en la que cada demanda se encamina por el camino más corto en km.

5.3. Ayudas para la realización del algoritmo

Se sugiere que el alumno utilice como plantilla el código que se incluye en Aula Virtual. En ese código:

- Se incluyen las líneas que recogen los parámetros de entrada del algoritmo.
- Se incluyen las líneas que calculan la lista de caminos admisibles de la demanda. Nótese que para cada demanda, el número *máximo* de caminos admisibles es k. Si k es un valor alto y/o la topología de red es pequeña, puede haber demandas con menos de k caminos admisibles (al no existir k caminos sin ciclos distintos para esa demanda).
- Se incluye la función `computeNetCongestion` que calcula la congestión de red, a partir de una solución dada. La solución se codifica con un vector de enteros `pathId_d`, con un elemento para cada demanda. Para una demanda d, el valor asociado indica el identificador p del camino que cursa el 100 % del tráfico de esa demanda. Ese identificador apunta a la lista de caminos admisibles.
- Use la clase `java.util.Random` para generar números aleatorios.

Se recuerda al alumno que antes de añadir las rutas al objeto `netPlan` con la solución final, debe eliminar todas las rutas que el objeto pudiera tener anteriormente, p.e. llamando al método `removeAllRoutes`.

5.4. Análisis: ajuste de los parámetros del algoritmo

El algoritmo SAN a desarrollar depende de varios parámetros, como el número de iteraciones en el bucle externo y en el interno. Realizando varias ejecuciones con distintos parámetros, podremos llegar a soluciones distintas. El alumno puede observar esto p.e. ejecutando el algoritmo sobre la red NSFNET, utilizando el fichero `NSFNet_N14_E42_complete.n2p`, que incluye la topología y el tráfico.

Se recomienda que, cuando quiera realizar ejecuciones de prueba con el objetivo de analizar la evolución del algoritmo, imprima en cada iteración del bucle interno.

- La congestión de la solución actual al finalizar el bucle
- La mejor congestión encontrada hasta el momento
- Si se acepta la solución vecina.

En distintas ejecuciones podrá observar como:

- A menudo sucede que las soluciones vecinas probadas no varían la congestión de red, y son aceptadas. Esto es debido a que le congestión es la utilización del enlace más cargado. Si se varía el camino de una demanda de tal manera que no se ven afectados los enlaces "cuello de botella", la congestión de red no varía, y el coste de la solución actual y la vecina son la misma.
- A temperaturas más altas, se observará en general que se aceptan prácticamente todas las soluciones vecinas. Sólo a temperaturas más bajas esto no se cumple.
- Valores mayores de k incrementan el espacio de soluciones, y provocan que las soluciones encontradas, si no se deja el algoritmo ejecutarse durante el tiempo suficiente, sean a menudo peores que las encontradas para menores valores de k. El motivo es que el algoritmo puede "perderse" iterando continuamente por soluciones claramente malas, sin inteligencia suficiente (debido a su comportamiento muy aleatorio) para corregir esta situación.

5.5. Posibles variaciones (opcional)

Se sugieren algunas variaciones al algoritmo que pueden ser intentadas:

- Fijar un parámetro de entrada `san_maxNumInnerIterationsWithoutImprovement` y que el bucle interno se detenga sólo cuando hayan pasado ese número de iteraciones consecutivas sin que la solución mejore.
- Cada 10 iteraciones del bucle externo, el bucle interno empezará como solución inicial, con la mejor solución encontrada hasta el momento. De esta manera se intensifica la búsqueda en las soluciones que parecen más prometedoras.
- Variar la función objetivo del problema, se tal manera que sea la media de la utilización de los 5