Práctica 5. Algoritmos ByB - Backtracking - Problema del viajante de comercio

Noelia Escalera Mejías Alejandro Menor Molinero
Javier Núñez Suárez Adra Sánchez Ruiz
Jesús Torres Sánchez

May 25, 2019

1 Descripción del problema

En este caso, hemos propuesto una solución para el problema del viajante de comercio por medio de un algortimo Branch and Bound, y después hemos creado la versión Backtracking para comparar como varía la eficiencia entre ambos algoritmos.

2 Cotas global y cota local

Como para cada algoritmo Branch and Bound, necesitamos dos cotas:
Por un lado tenemos la **cota global**, que determina la mínima distancia encontrada hasta el momento para un circuito generado. Si aún no se ha encontrado ninguna solución, estará inicializada de acuerdo a una versión Greedy del algoritmo (en este caso, se ha usado la aproximación basada en Inserción).

Por otro lado, también necesitamos una **cota local**, que nos servirá de referencia tanto a la hora de comprobar la factibilidad (criterio de poda), como a la hora de insertar los nodos en la pila, de modo que los nodos se almacenarán con una prioridad de menor a mayor cota local.

3 Representación de la solución

A la hora de representar el problema, hemos usado un TDA *Solucion*, para implementar nuestro algoritmo de *Branch and Bound* de forma más clara y organizada.

- **Información almacenada en el TDA Solucion**:
  - x: almacena la solución generada hasta el momento. Se representa por medio de un vector de enteros que contendrá el conjunto de ciudades seleccionadas en cada instante.

- n: es el número de ciudades.
- *distancias*: representadas con una matriz, en la que cada elemento d[i][j] tiene asociado la distancia entre la ciudad i y la ciudad j. Por tanto, deben coincidir d[i][j] con d[j][i].
- *cotaLocal*: contiene la cota local de acuerdo a las ciudades seleccionadas hasta el momento. La cota local es una distancia optimista calculada a partir de la distancia entre las ciudades ya seleccionadas y una estimación de aquellas que están aún por seleccionar.
- *distanciaActual*: contiene la distancia acumulada para las ciudades seleccionadas hasta el momento. Se usa para el cálculo de la cota local.
- *ciudadesYaSeleccionadas*: representadas como un vector de booleanos, almacena si la ciudad i ya ha sido seleccionada ($true$) o no ($false$).
- *mejorSolución*: es un atributo de clase (público), y por tanto único, representado por medio de un vector que almacena la mejor solución encontrada hasta el momento. Este valor se puede inicializar a partir de una solución Greedy del problema.
- *cotaGlobal*: almacena la menor distancia de las soluciones generadas hasta el momento, es decir, la distancia de la mejor solución encontrada.

- **Operaciones del TDA Solucion**:
  - *Constructor*: realiza una serie de operaciones:
    - Inicializa las variables del TDA a sus valores por defecto. Debemos tener en cuenta que, a la hora de inicializar la matriz distancias, el primer elemento es el $d[1][1]$, por lo que debemos añadirle una fila y columna basura para poder acceder con índices que empiecen por 1.
    - Seleccionamos en primer lugar la ciudad 0 para no generar soluciones equivalentes y reducir así el número de nodos hoja de $n!$ a $(n-1)!$.
  - *factible(k)*: únicamente comprueba si la cotaLocal es menor que la cotaGlobal. No es necesario comprobar si el hijo estaba ya seleccionado o no, ya que esto se garantiza por la naturaleza de nuestro algoritmo ByB.
  - *generaHijos(k)*: este método devuelve un vector con todos los hijos derivados de añadir cualquiera de las ciudades no seleccionadas al vector solución. En el caso de que a la solución generada le falte solamente una ciudad, se completa con la ciudad en cuestión. Debemos tener en cuenta la correcta actualización de los atributos de los nodos hijos generados (cotaLocal, ciudadesYaSeleccionadas, etc).
  - *actualizarCotaLocal()*: provisional.

- *esSolucion()*: devuelve si hemos llegado a una solución, es decir, si el tamaño del vector solución coincide con el número de ciudades a explorar. En este caso, siempre se actualiza la cota global y la mejor solución encontrada, ya que para que se invoque al método antes se ha debido comprobar la factibilidad en nuestro algoritmo ByB.
- *operator ¡*: necesario para almacenar las soluciones en la cola con prioridad dentro del algoritmo ByB. Devuelve si la cota local del objeto implícito es menor que la del objeto pasado como parámetro. En caso de que las cotas locales sean iguales, usa como criterio de ordenación el que tenga una ciudad anterior.
- *inicializarCotaGlobal*: inicializa la cotaGlobal y la mejorSolucion a partir de una solución Greedy del problema (en nuestro caso, hemos usado la aproximación basada en Inserción).
- *obtenerSoluciónÓptima*: devuelve la solución óptima.

4 Algoritmo ByB

El algoritmo Branch and Bound sigue el siguiente esquema:

```
void branchAndBound (Solucion sol){
  set<Solucion> cola;
  cola.insert(sol);
  bool fin = false;

  while (!cola.empty() && !fin){
        Solucion enodo = *cola.begin();
        cola.erase(cola.begin());

        fin = !enodo.factible();

        if (!fin){
                vector<Solucion> hijosDelEnodo =
                                enodo.generaHijos();

                for (Solucion hijo : hijosDelEnodo){
                        if (hijo.factible()){
                                if (!hijo.esSolucion())
                                   cola.insert(hijo);
                        }
                }
        }
        else
        {
                cout << "elfin" << endl;
        }
```

```
    }
}
```

5 Algoritmo Backtracking VS ByB

6 Estudio empírico