Unicen Tandil Tudai

Programación 3:
Trabajo Práctico Especial 2

Emilio Colombo, Matias Sciancalepore
Grupo 50

-¿Cuál fue la estrategia Backtracking llevada adelante para resolver el problema?
¿Cuál es el costo computacional de dicha estrategia?

La estrategia de Backtracking utilizada para resolver el problema consistió en tomar los arcos como un conjunto de arcos, y así, armar todos los posibles conjuntos que se puedan formar teniendo, en cada recursión, una decisión de agregar/no agregar un túnel al conjunto de solución parcial.

```
Ask EasyCode | Explain | Refactor
private void back(Estado e) {
    this.metrica++;
    if (this.isSolucion(e)){
        if(this.esFactible(e)){
            this.solucionIdeal = new ArrayList<Arco<Integer>>(e.getSolucionParcial());
            this.sumaSolucionIdeal = e.getContador();
        }
    }
    else{
        Arco<Integer> tunel = e.getPosibilidades().get(index:0);
        e.getPosibilidades().remove(index:0);
        e.setContador(e.getContado  Integer sumaSolucionIdeal
        if(e.getContador() < this.sumaSolucionIdeal){
            e.solucionParcial.add(tunel);
            this.back(e);
        }
        e.solucionParcial.remove(tunel);
        e.setContador(e.getContador() - tunel.getEtiqueta());
        if(e.getContador() < this.sumaSolucionIdeal){
            this.back(e);
        }
        e.getPosibilidades().add(tunel);
    }
}
```

nuestro método isSolucion, chequeaba que no queden más túneles en el conjunto de candidatos que llevamos en la clase Estado.

```
Ask EasyCode | Explain | Refactor
private boolean isSolucion(Estado e) {
    //1ro me fijo si posibilidades esta vacio
    if (!e.getPosibilidades().isEmpty()){
        return false;
    }
    return true;
}
```

nuestro método esFactible, corroboraba que el "grafo de túneles" resultante era una solución válida, checkeando si todas las estaciones fueron alcanzadas, que la solución sea conexa, y que la distancia total de la solución sea menor a la que ya habíamos guardado anteriormente (inicialmente infinita).

```
Ask EasyCode | Explain | Refactor
//el metodo "es factible" originalmente se usa en greedy, pero debemos usarlo
//aqui tambien para controlar algunas cosas
public boolean esFactible(Estado e){
    //chekeamos que todos los las estaciones esten en nuestra solucion parcial
    ArrayList<String> verticesParciales = getVerticesParciales(e.getSolucionParcial());
    for (String vertice : this.verticesTotales){
        if(!verticesParciales.contains(vertice)){
            return false;
        }
    }
    Grafo  g = new GrafoNoDirigido(this.verticesTotales, e.getSolucionParcial());
    //para saber si los tuneles se conectan todos entre si, debemos corroborar que
    //el grafo que se formaria es conexo.
    if(!g.esConexo()){
        return false;
    }
    //y por ultimo verificaremos que la solucion ya guardada no es mejor que la actual
    if(e.getContador() > this.sumaSolucionIdeal){
        return false;
    }
    //si no, cumple con ninguno de los if anteriores, significa que es una solucion valida y mejor que
    //la que ya habiamos almacenado
    return true;
}
```

El costo computacional de esta estrategia fue, por mucho, mucho más elevado en comparación de la estrategia greedy. O(n!) n= {cantidad de estaciones}

```
tuneles entre estaciones :
dataset 1:
E1 <-> E2
E2 <-> E3
E2 <-> E4
metrica ( cantidad de veces q se entro a backtracking ): 65
metros a construir : 55
tuneles entre estaciones :
dataset 2:
E1 <-> E2
E4 <-> E3
E2 <-> E3
E2 <-> E5
E6 <-> E5
metrica ( cantidad de veces q se entro a backtracking ): 1727
metros a construir : 135
tuneles entre estaciones :
dataset 3:

no lo corrimos porque tarda mucho en llegar a la solucion pero:
metrica = 13!
metros a construir = 440
PS C:\tudai 2do\prog 3\TPE>
```

-¿Cuál fue la estrategia Greedy llevada adelante para resolver el problema? ¿Cuál es el costo computacional de dicha estrategia?

La estrategia Greedy utilizada para resolver el problema consistió en tomar los túneles como un conjunto de Arcos. (atributo de clase)

Para facilitar algunas cosas llevamos un array con “estaciones alcanzadas”, actualizando cuando agregamos una estación.

En cada iteración seleccionamos el arco que tenía la menor distancia y que tenga una una estación origen a la que nosotros ya hayamos agregado a nuestros “vértices parciales”, para evitar que el resultado no sea conexo. (a excepción de la primera vez, que no teníamos ningún arco en nuestro conjunto solución).

```
Ask EasyCode | Explain | Refactor
private Arco<Integer> seleccionar(ArrayList<String> verticesParciales ,ArrayList<Arco<Integer>> arcos){
    Arco<Integer> arco = null;
    if(verticesParciales.isEmpty()){
        arco = arcos.get(index:0);
        for(Arco<Integer> aux : arcos){
            if(arco.getEtiqueta() > aux.getEtiqueta()){
                arco = aux;
            }
        }
    }
    else{
        for (Arco<Integer> aux : arcos){
            if(this.contieneVerticeEnOrigen(aux, verticesParciales)){
                arco = aux;
            }
        }
        if(arco == null){
            return null;
        }
        for(Arco<Integer> aux : arcos){
            if(this.contieneVerticeEnOrigen(aux, verticesParciales)){
                if(arco.getEtiqueta() > aux.getEtiqueta()){
                    arco = aux;
                }
            }
        }
    }
    return arco;
}
```

luego, con el método esFactible(), corroboramos que el arco elegido no tenga como destino una estación a la que nosotros ya hayamos agregado a “vértices parciales”, evitando hacer un bucle y construir de mas.

```
Ask EasyCode | Explain | Refactor
private boolean esFactible(Arco<Integer> a, ArrayList<String> verticesParciales) {
    for(String vertice : verticesParciales){
        if(a.getVerticeDestino().equals(vertice)){
            return false;
        }
    }
    return true;
}
```

el costo computacional fue por mucho menor a backtracking, O(?).

```
tuneles entre estaciones :
dataset 1 :
E1 <-> E2
E2 <-> E3
E2 <-> E4
metrica ( cantidad de veces q se entro a greedy ): 6
metros a construir : 55
```

```
tuneles entre estaciones :
dataset 3 :
E9 <-> E13
E13 <-> E2
E2 <-> E3
E2 <-> E5
E5 <-> E11
E11 <-> E7
E11 <-> E8
E3 <-> E1
E8 <-> E4
E5 <-> E6
E8 <-> E10
E6 <-> E12
metrica ( cantidad de veces q se entro a greedy ): 48
metros a construir : 440
```

```
tuneles entre estaciones :
dataset 2 :
E4 <-> E3
E3 <-> E1
E1 <-> E2
E2 <-> E5
E5 <-> E6
metrica ( cantidad de veces q se entro a greedy ): 6
metros a construir : 135
```

Entrada	Resultado Backtracking	Resultado Greedy	Costo Backtracking	Costo Greedy
dataset 1	55	55	65	6
dataset 2	135	135	1727	6
dataset 3	440	440	13!	48

conclusión:

con Backtracking nos aseguramos que la solución siempre va a ser la óptima, pero sin dudar, el costo computacional es mayor al de greedy, sin embargo, encontramos una estrategia greedy que nos dio la mejor solución en este problema, para este ejercicio en particular, recomendamos utilizar greedy, ya que en casos en los que los que se ingresa un gran volumen de datos, backtracking no puede dar un tiempo razonable para resolver un problema de ese tamaño, ej, si tuviera 70 estaciones para ingresar la métrica seria 70!, no tiene sentido el tiempo y costo que generaría.