Ingeniería de la Salud
Escuela Técnica Superior de Ingeniería Informática
Universidad de Málaga

Práctica 6. Complejidad de los problemas combinatorios. Cuadrados latinos.

Arrabalí Cañete, Carmen Lucía

---

1. Introducción y objetivos

Los problemas de combinatoria son aquellos donde se halla la cantidad de objetos que poseen una determinada propiedad.

La transición de fase es un término tomado de la Física para denotar el paso de un sistema de un estado a otro, lo que da lugar a un cambio (a menudo abrupto y discontinuo) en sus propiedades.

La misma idea puede aplicarse a la resolución de problemas computacionales.

En este caso, el objetivo principal es estudiar la complejidad de la resolución de problemas combinatorios que presentan un cambio de fase. Se implementará un algoritmo de backtracking que resuelve el problema de los cuadrados latinos. Estos cuadrados son una matriz de *nxn* elementos en la que cada casilla está ocupada por uno de los *n* símbolos de tal modo que cada uno de ellos aparece exactamente una vez en cada columna y en cada fila.

2. Configuración del equipo

Se ha realizado la implementación en un equipo con un sistema operativo Windows 10 Home 64 bits, con un procesador Intel(R) Core (TM) i7-6500U CPU @ 2.50GHz 2.59 GHz (4 CPUs) y un disco SSD de 480GB y 12GB de RAM. La versión Java corresponde a la número 17.

3. Planteamiento del problema

Sea γ un parámetro de control que describa alguna propiedad estructural de las instancias del problema P considerado. La resolubilidad de la instancia e incluso el comportamiento de un algoritmo de búsqueda S aplicado a una instancia $\mathrm{I}(\gamma) \in \mathrm{P}$ puede depender de este parámetro de control.

Considerando el caso específico de los problemas de satisfacción con restricciones: Encontrar las variables $X = x_1, ..., x_n$ sujetas a las restricciones $C_1, ..., C_m$ donde cada C_i es una declaración lógica sobre los valores de las variables de X.

Un parámetro de control que describe el número y/o la estructura de restricciones puede definir dos subconjuntos homogéneos de instancias:

- **Instancias con pocas restricciones:** las restricciones son poco limitantes y por lo tanto estas instancias tienen muchas soluciones.
- **Instancias con exceso de restricciones:** las restricciones son muy limitantes y por lo tanto estas instancias no tienen ninguna solución.

4. Implemetación

4.1. Clase LatinSquareBacktracking.java

4.1.1. Método *protected Object initialState()*

Se encarga de crear la instancia inicial, un par con las coordenadas de la primera posición, el que se encuentra en el $(0, 0)$

```
1  protected Object initialState() {
2    Pair<Integer, Integer> initialLS = new Pair<Integer, Integer>(0, 0);
3    return initialLS;
4  }
```

En el segundo método, el algoritmo debe intentar llenar la tabla fila por fila, de izquierda a derecha. Teniendo en cuenta las siguientes restricciones:

- Si una posición no está especificada, el algoritmo debe comprobar qué valores son factibles para esa posición y continuar recursivamente.
- Si una posición es fija, el algoritmo debe seguir comprobando si ese valor es factible, porque la instancia inicial podría ser irresoluble.

Donde, las variables que se necesitan para su correcta implementación son:

- **p**: par de coordenadas en la que se encuentra el estado que se pasa como parámetro.
- **i**: primer valor del par.
- **j**: segundo valor del par.
- **n**: tamaño del cuadrado latino.

4.1.2. Método *protected boolean backtracking(Object state)*

```
1  protected boolean backtracking(Object state) {
2    Pair<Integer, Integer> p = (Pair<Integer, Integer>) state;
3    int i = p.getFirst();
4    int j = p.getSecond();
5    boolean ok;
6    int n = latinSquare.getSize();
7
8    if(i == n & j == 0) {
9      ok = true;
10   } else {
11     ok = false;
12     if(latinSquare.isFixed(i, j)) {
13       boolean value = latinSquare.test(i, j, latinSquare.get(i, j));
14       if(value) {
15         this.nodes++;
16         if(j < n-1) {
17           ok = backtracking(new Pair<Integer, Integer>(i, j+1));
18         } else {
19           ok = backtracking(new Pair<Integer, Integer>(i+1, 0));
20         }
21       }
22     } else {
23       int k = 1;
```

```
        while(!ok && k <= n) {
          boolean value2 = latinSquare.test(i, j, k);
          if(value2) {
            this.nodes++;
            latinSquare.set(i, j, k);
            if(j < n-1) {
              ok = backtracking(new Pair<Integer, Integer>(i, j+1));
            } else {
              ok = backtracking(new Pair<Integer, Integer>(i+1, 0));
            }
            if(!ok) {
              latinSquare.set(i, j, -1);
            }
          }
          k++;
        }
      }
    }
    return ok;
  }
```

5. Resultados y Conclusiones

Para poder analizar correctamente los resultados, el método main recibe como argumento de entrada un fichero llamado *ls10.txt* que incluye 1000 cuadrados latinos con diferentes valores del parámetro de control.

Al utilizar el algoritmo con este archivo como entrada, se generan dos archivos, uno con la solución a cada uno de los cuadrados latinos, *solution_ls10.txt* y otro con las estadísticas *stats_ls10.txt*

Por otro lado, entre los recursos aportados en el Campus Virtual se tiene un archivo llamado *analyze.R* que contiene un script el cual se encarga de generar unas gráficas para cada algoritmo haciendo uso del archivo que contiene las estadísticas.

Por un lado se obtiene una gráfica que contiene los nodos recorridos frente a la fracción de elementos fijos en el cuadrado latino (véase figura 1) y, por otro lado, se obtiene una gráfica con la solvencia que tiene cada una de las evaluaciones que se realiza (véase figura 2).

Se puede observar que los cambios se producen en un punto muy definido, el cuál se denomina como punto crítico, donde, a partir del mismo, en caso de ser un fluido como el agua, cambiaría de forma de manera drástica, como por ejemplo, pasar de líquido a gas, y que coincide con la transición entre esas dos regiones.

En este caso, el punto crítico sería $\gamma = 0{,}43$ ya que un punto inferior sería una instancia infra-restringidas y si es superior, estaría sobrecargada.

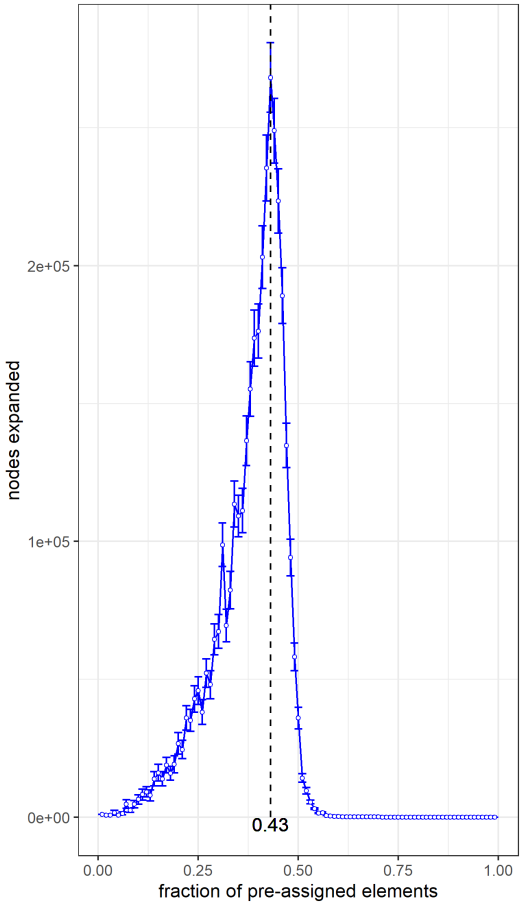

Figura 1: Resultado de los nodos recorridos.

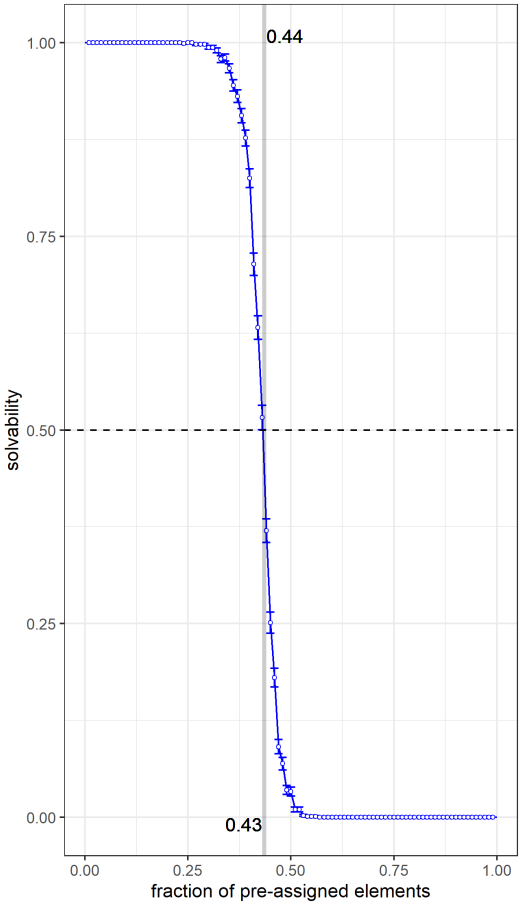

Figura 2: Resultado de la solvencia del problema.