PRÁCTICA IV

Algoritmos Backtracking

Por: Antonio Gámiz

1 Problema

Se va a celebrar una cene de gala a la que asistirán n invitados. Todos se van a sentar alrededor de una única gran mesa rectangular, de forma que cada invitado tendrá sentados junto a él a otros dos comensales (uno a su izquierda y otro a su derecha). En función de las característiscas de cada invitado (por ejemplo po categoría, puesto, lugar de procedencia...) existen unas normas de protocolo que indican el nivel de conveniencia de que dos invitados se siente en lugares contiguos (supondremos que dicho nivel es un número entero entre 0 y 100). El nivel de conveniencia total de una asignación de invitados a su puesto en la mesa es la suma de todos los niveles de conveniencia de cada invitado con cada uno de los dos invitados sentados a su lado.

Se desea sentar a los invitados de forma que el nivel de conveniencia global sea lo mayor posible. Diseñar e implementar un algoritmo vuelta atrás para resolver este problema. Realizar un estudio empírico de su eficiencia.

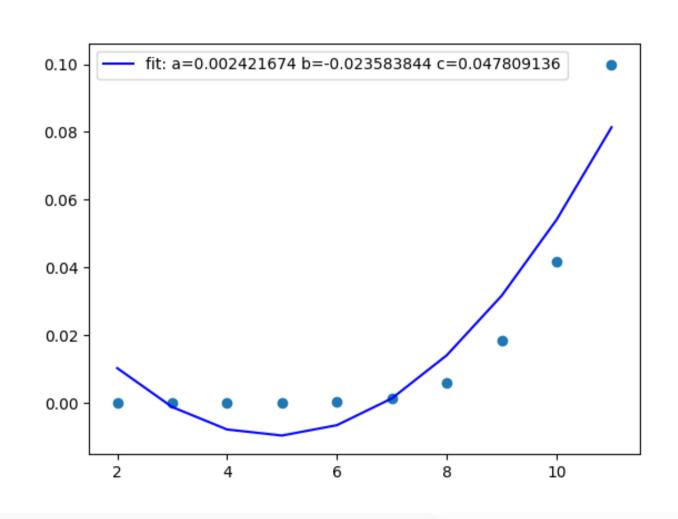
```
class ConvenienceMatrix
  private:
    vector<vector<conv> > c;
  public:
    ConvenienceMatrix(){};
    ConvenienceMatrix(int n)
      c.resize(n);
      for(int i=0; i<n; i++) c[i].resize(n);</pre>
      int k=0;
      for(int i=0; i<c.size(); i++)</pre>
        for(int j=0; j<c.size(); j++)</pre>
            c[i][j].person=j;
            c[i][j].convenience=(i!=j) ? rand() % 100 : 0; //"""pseudo-random""""
        k+=1;
    }
```

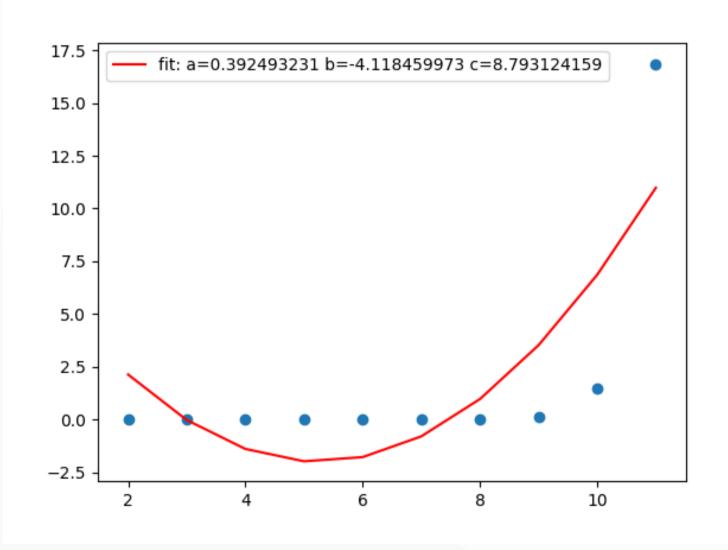
```
void backtracking(ConvenienceMatrix & c, vector<int> v)
  vector<int> available=supplementary(v, c.size()); //definida en 'auxliar.cpp'
  for(int i=0: i<available.size(): i++)</pre>
    v.push_back(available[i] ); //aniadimos el siguiente numero que no este ya en la solucion
    if(v.size() == c.size()) ++count;//to s(v); //descomentar para ver todas las posibilidades
    aux cost=c.costs(v);
    if(aux cost > cost ) //guardamos la solucion con el mayor coste
      cost=aux cost;
      solution=v;
    backtracking(c, v); //recursividad
    v.pop back(); //eliminamos el elemento aniadido antes para calcular
                  //la siguiente posibilidad
```

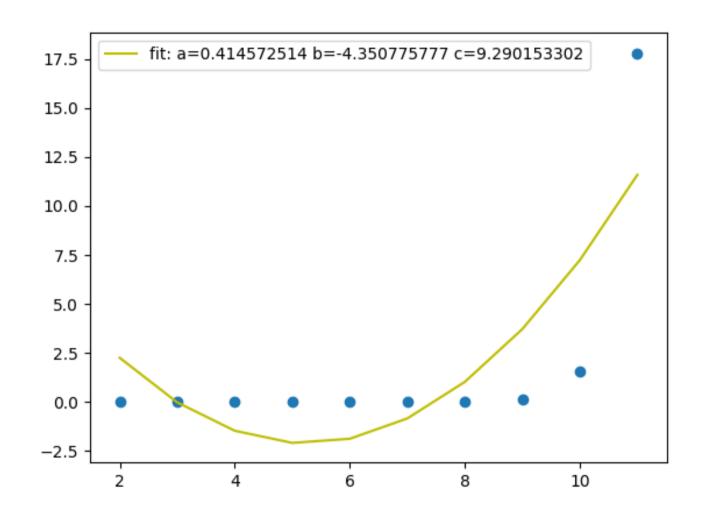
```
void backtracking(ConvenienceMatrix & c, vector<int> v)
 vector<int> available=supplementary(v, c.size()); //definida en 'auxliar.cpp'
 for(int i=0; i<available.size(); i++)</pre>
    v.push_back(available[i] ); //aniadimos el siguiente numero que no este ya en la solucion
    if(v.size() == c.size()) ++count;//to s(v); //descomentar para ver todas las posibilidades
    aux cost=c.costs(v):
    if(aux cost > cost ) //quardamos la solucion con el mayor coste
      cost=aux cost;
      solution=v;
    else backtracking(c, v); //recursividad
    v.pop_back(); //eliminamos el elemento aniadido antes para calcular
                  //la siguiente posibilidad
```

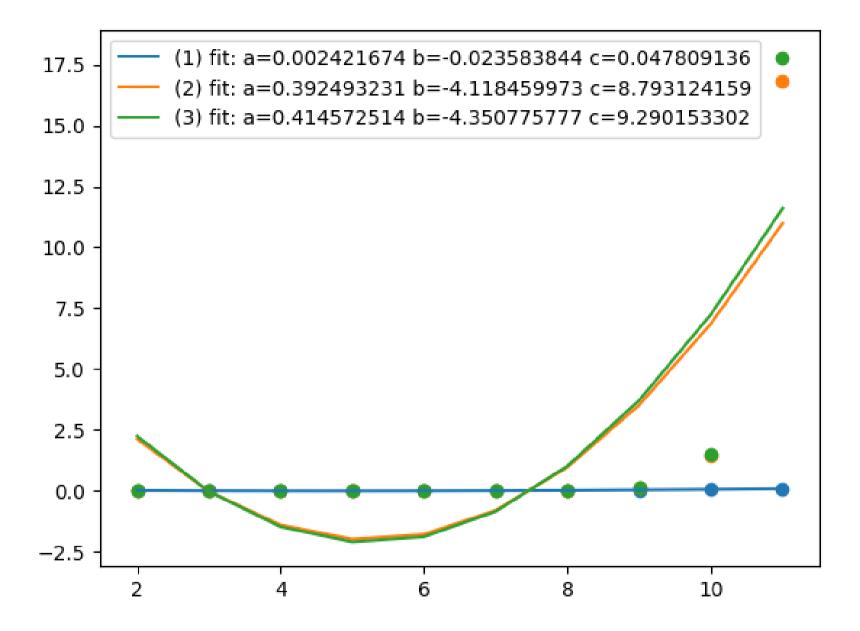
```
void backtracking(ConvenienceMatrix & c, vector<int> v)
  vector<int> available=supplementary(v, c.size()); //definida en 'auxliar.cpp'
  for(int i=0: i<available.size(): i++)</pre>
    v.push_back(available[i] ); //aniadimos el siguiente numero que no este ya en la solucion
    if(v.size() == c.size()) ++count;//to s(v); //descomentar para ver todas las posibilidades
    aux cost=c.costs(v);
    if(aux cost > cost ) //guardamos la solucion con el mayor coste
      cost=aux cost;
      solution=v;
    backtracking(c, v); //recursividad
    v.pop back(); //eliminamos el elemento aniadido antes para calcular
                  //la siguiente posibilidad
```

```
vector<int> solution; //mejor solucuion
int cost: //coste de la solucion
int aux cost; //coste auxiliar para calcular el maximo
void backtracking(ConvenienceMatrix & c, vector<int> v)
  vector<int> available=supplementary(v, c.size()); //definida en 'auxliar.cpp'
  for(int i=0: i<available.size(): i++)</pre>
    v.push back(available[i] ); //aniadimos el siguiente numero que no este ya en la solucion
    if(v.size() == c.size()) ++count;//to s(v); //descomentar para ver todas las posibilidades
    aux cost=c.costs(v);
    if(aux cost > cost ) //quardamos la solucion con el mayor coste
      cost=aux cost;
      solution=v:
      backtracking(c, v); //recursividad
    if(v.size()<7) backtracking(c, v);</pre>
    v.pop_back(); //eliminamos el elemento aniadido antes para calcular
                  //la siguiente posibilidad
```









FIN