TP1 - Problema do Vetor Curto (SVP)

Outubro, 2022

Bruno Miguel Ferreira Fernandes - a95972

Hugo Filipe de Sá Rocha - a96463

Variáveis

Inputs do Problema:

- m - número de componentes do vetor "e" e do número de linhas da matriz L.
- n - número de colunas de L e do vetor produto.
- q - número primo maior ou igual a 3 ao qual todos os elementos do vetor produto são múltiplos.
- d - número (q-1)/2 que define o intervalo de geração dos componentes da matriz L ([-d,d]).

Auxiliares

- $L_{j,i}$ - representa a matriz que será gerada aleatoriamente de dimensão m x n, com j∈M e i∈N.
- k_n - representa o vetor dos k's tais que multiplicados por q dão como resultado as componentes do vetor produto.
- e_m - representa o vetor "e" com m componentes pertencente a {-1,0,1}.
- $quadrados_m$ - representa o vetor com as componentes do vetor "e" ao quadrado.

Condições

Em notação matricial:

$$\begin{cases} \exists\, e \in \{-1,0,1\}^m \,.\, \exists\, k \in \mathbb{Z}^n & . \quad e \times \mathsf{L} \;=\; q\,k \\ \qquad\qquad\qquad\qquad\quad \exists\, i < n & . \quad e_i \;\neq\; 0 \end{cases}$$

Critérios de optimização

1. Minimizar o número de componentes não nulas.

Funções auxiliares:

- tam_bits(x) - calcula número de bits necessários para representar o inteiro x.

In [1]:
```
def tam_bits(x):
    contador = 0

    while (x >= 2**contador):
          contador+=1

    return contador
```

Definir valores de input do problema

In [2]:
```
# Importar biblioteca
from ortools.sat.python import cp_model
import random

# Cria o modelo CP-SAT
model = cp_model.CpModel()

n = 5
#print(tam_bits(n))
m =  16
#print(tam_bits(m))
q = 37
#print(tam_bits(q))
d = 18
```

In [3]:
```
#gera a matriz L de forma aleatória no intervalo [-d,d]

L = {}


for j in range(m):
    L[j] = {}
    for i in range(n):
        L[j][i] = random.randint(-d,d)

#dar print à matriz L
'''
for j in range(m):
      for i in range(n):
            print(f'L[{j}][{i}]:')
            print(L[j][i])
'''

#criar vetor e, com m componentes em [-1,0,1]
e = {}


for j in range(m):
    e[j] = model.NewIntVar(-1,1,f'e[{j}]')


#criar vetor k, vetor dos valores k tais que k[i]*q é igual à i-ésima
           #componente do vetor resultante do produto e*L

k = {}

for i in range(n):
    k[i] = model.NewIntVar(-10000, 10000, f'k[{i}]')

#vetor quadrados onde quadrados[i] tem guardado o valor do quadrado
           #da i-ésima componente do vetor e

quadrados = {}

for i in range(m):
        quadrados[i] = model.NewIntVar(0,1, f'quadrados[{i}]')
        model.AddMultiplicationEquality(quadrados[i], [e[i],e[i]])
```

Modelação das restrições

1. Restrição que garante que cada componente do vetor produto é múltiplo de q.

$$\forall_{i<n} \cdot (\sum_{j<m} e_j * L[j][i] == q * k_i)$$

In [4]:
```
for i in range(n):
    model.Add(    sum(   [ e[j]*L[j][i] for j in range(m)  ] )   == q*k[i]      )
```

1. Restrição que garante que o vetor "e" não é nulo.

$$\sum_{i<m} quadrados_i \geq 1$$

In [5]:
```
model.Add(sum([quadrados[i] for i in range(m)]) >= 1)
```

Optimização do problema

Condição de optimizacão que minimiza o número de componentes não nulas (minimiza a soma das componentes do vetor quadrados).

In [6]:
```
model.Minimize(sum([quadrados[i] for i in range(m)]))
```

Impressão do vetor e :

In [7]:
```
solver = cp_model.CpSolver()

# Invoca o solver com o modelo criado
status = solver.Solve(model)


if status == cp_model.OPTIMAL or status == cp_model.FEASIBLE:
      for i in range(m):
        print(solver.Value(e[i]))

else:
    print("Não foi possível")
```

```
1
0
-1
-1
0
1
-1
0
1
1
0
1
0
-1
-1
-1
```

In [ ]: