

# TP1\_\_Problema\_2\_\_

October 18, 2022

## 1 TP1

### 1.1 Problema 2

2. Na criptografia pós-quântica os reticulados inteiros (“hard lattices”) e os problemas a eles associados são uma componente essencial. Um reticulado inteiro pode ser definido por uma matriz

$$L \in \mathbb{Z}^{m \times n}$$

(com  $m > n$ ) de inteiros e por um inteiro primo  $q \geq 3$ .

O chamado problema do vetor curto (SVP) consiste no cálculo de um vetor de inteiros

$$e \in \{-1, 0, 1\}^m$$

não nulo que verifique a seguinte relação matricial

$$\forall i < n.$$

$$\sum_{j < m} e_j \times L_{j,i} \equiv 0 \pmod{q}$$

#### 1.1.1 Implementação

Começamos por importar a biblioteca de programação linear do OR-Tools e criar uma instância do model. Entretanto, inicializamos o `model`.

```
[126]: from ortools.sat.python import cp_model
import random

#solver = pywraplp.Solver.CreateSolver('SCIP')

model = cp_model.CpModel()
```

Pretende-se resolver o SVP por programação inteira dentro das seguintes condições

- i. Os valores  $m, n, q$  são escolhidos com  $n > 30$ ,  $|m| > 1 + |n|$  e  $|q| > |m|$ .
- ii. Os elementos  $L_{j,i}$  são gerados aleatória e uniformemente no intervalo inteiro  $\{-d \dots d\}$  sendo

$$d \equiv (q - 1)/2$$

```
[127]: q = 5
d = 2
n = 2
m = 4

L = {}
for j in range(m):
    for i in range(n):
        L[j,i] = random.randint(-d, d+1) # retorna um inteiro dentro do limite
        ↪ dado (-d...d)

e = [model.NewIntVar(-1, 1, f'e[{j}]') for j in range(m)] # estamos a criar o
        ↪ vetor "e" entre os limites -1 e 1
```

$$\forall i < n.$$

$$\sum_{j < m} e_j \times L_{j,i} \equiv 0 \pmod{q}$$

```
[128]: for i in range(n):
    K = model.NewIntVar(0, q, f'K[{i}]')
    model.Add(sum(e[j] * L[j, i] for j in range(m)) == q * K)
```

b. Pretende-se determinar, em primeiro lugar, se existe um vetor  $e$  não nulo (pelo menos um dos  $e_j$  é diferente de zero).

\$ \{j < m\}. \ e\{j\} \neq 0 \$

```
[129]: e2 = [model.NewIntVar(0, 1, f'e[{j}]') for j in range(m)]
for j in range(m):
    model.AddAbsEquality(e2[j], e[j])

model.Add(sum(e2) > 0)
pass
```

Minimizar o número de componentes não nulas, i.e., maximizar o número de componentes nulas.

```
[130]: model.Minimize(sum(e2))

solver = cp_model.CpSolver()
status = solver.Solve(model)

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

    print("e2: ")
```

```
    for j in range(m):  
        print(solver.Value(e2[j]))  
else:  
    print("unsat")
```

```
e:  
0  
0  
-1  
-1  
e2:  
0  
0  
1  
1
```