Ex2_HardLattices

October 17, 2022

1 Exercício 2 (Hard Lattices) - Trabalho Prático 1

Grupo 4: Carlos Costa-A94543 Ruben Silva-A94633

2 Problema:

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

$$\forall\, i < n \,\centerdot\, \sum_{j<m} e_j \,\times\, \mathsf{L}_{j,i} \;\equiv\; 0 \mod q$$

1. Pretende-se resolver o SVP por programação inteira dentro das seguintes condições 1. Os valores m , n , q são escolhidos com $n > 30$, $|m| > 1 + |n|$ e $|q| > |m|$. 2. Os elementos $\mathsf{L}_{j,i}$ são gerados aleatória e uniformemente no intervalo inteiro $\{-d \cdots d\}$ sendo $d \equiv (q-1)/2$. 2. Pretende-se determinar em, em primeiro lugar, se existe um vetor e não nulo (pelo menos um dos e_j é diferente de zero). Se existir e pretende-se calcular o vetor que minimiza o número de componentes não nulas.

Notas:

1. Se $x \geq 0$, representa-se por $|x|$ o tamanho de x em bits: o menor ℓ tal que $x < 2^\ell$.
2. Um inteiro x verifica $x \equiv 0 \mod q$ sse x é um múltiplo de q.

$$x \equiv 0 \mod q \qquad \iff \qquad \exists\, k \in \mathbb{Z} \,\centerdot\, x \;=\; q \times k.$$

Por isso, escrito de forma matricial, as relações que determinam o vetor $e \neq 0$ são:

$$\left\{\begin{array}{rcl} \exists\, e \in \{-1,0,1\}^m \,\centerdot\, \exists\, k \in \mathbb{Z}^n & \centerdot & e \times \mathsf{L} \;=\; q\, k \\ \exists\, i < n & \centerdot & e_i \;\neq\; 0 \end{array}\right.$$

3 Análise do problema

Este é problema sobre SVP que possui uma relação matricial que tem de ser respeitada. Para se resolver este problema é necessário um dicionário com uma matriz $L^{m \times n}$ Também será necessário desenvolver um algoritmo que faça a multiplicação tanto entre Vetor e Matriz, como valor e Vetor. Algumas variáveis que useramos para descrever as limitações e as obrigações do nosso problema: * $e \in \{-1, 0, 1\}^m$ $\rightarrow$ Vetor de inteiros não nulo * $q \in \mathbb{P} \geq 3; \rightarrow$ Numero inteiro primo maior que 3 * $m \in \mathbb{N}; \rightarrow$ Linhas * $n \in \mathbb{N}; \rightarrow$ Colunas * $j \in \mathbb{N}; \rightarrow$ Linhas * $i \in \mathbb{N}; \rightarrow$ Colunas * $x = (\forall_{i<n} : \sum_{j<m} e_j \,\times\, \mathsf{L}_{j,i}\ ) \rightarrow$ Simplificação do Somatório

4 Limitações e obrigações

1. A seguinte relação matricial tem de ser respeitada

$$\forall_{i<n} : \sum_{j<m} e_j \times \mathsf{L}_{j,i} \equiv 0 \mod q \iff \exists k \in \mathbb{Z} \centerdot x = q\,k$$

2. A seguinte equação paramétrica tem de ser respeitada

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

5 Implementaçao do Problema

Importar o solver 1. Instalar o o ortools a partir da libraria do PyPi (pip) 2. Importar o cp_model do ortools (contem o solver que iremos usar neste problema) 3. Importar o next_prime do ortools (contem o solver que iremos usar neste problema)

[ ]:
```
!pip install ortools
!pip install gmpy2
from ortools.sat.python import cp_model
from random import randint
from gmpy2 import next_prime
```

Função Auxiliar para visualizar a matriz formada

[ ]:
```
#Função Para visualizar a matriz
def print_array(L, m, n):
    for j in range(m):
        str1 = "Row: "+str(j+1)+" -> "
        for i in range(n):
            str1 += str(L[j, i])+" "
```

Função para multiplicar matriz e vetores Este função tem o parâmetro "valor" que tem como objeto servir de flag e transportar o valor de q. Se este valor for 0 (por default), então será uma multiplicação de matrizes, se for diferente de 0, então é a multiplicação de um vetor por q

[ ]:
```
def multiplicar_matriz_vetor(matriz, j, i, vetor, valor=0):

    # So pode entrar aqui um vetor de cada vez
    #Se 0, entao é Vetor x Matriz
    #Se !=0, então é Valor "q" x Matriz

    #NAO FUNCIONA
    if valor == 0:
        final = [0]*i
        for i1 in range(i):
            #final[i1] = sum(vetor[j1] * matriz[j1, i1] for j1 in range(j))
            final[i1] = sum(vetor * matriz[j1, i1] for j1 in range(j))
        return final
```

```
    if valor != 0:
        final = [0]*j
        for j1 in range(j):
            #final[j1] = valor * vetor[j1]
            final[j1] = valor * vetor
        return final
    return 0
```

Função que resolve o problema geral (hard_lattices) 1. Criar o model 2. Criar a Matriz $L^{m \times n}$ 3. Criar o Vetor e e o vetor u 4. Adicionar ao model todas Limitaçôes e Obrigações 5. Criar o solver, usar o nosso model e no fim executá-lo

[ ]:
```
def hard_lattices(n, m, q, d):
    model = cp_model.CpModel()

    #Criação da Matriz Inicial
    L = {}
    for j in range(m):
        for i in range(n):
            L[j, i] = randint(-d, d+1)

    # Vetor de alocação
    e = [model.NewIntVar(-1, 1, f"e[{j}]") for j in range(m)]
    # Vetor Booleano
    u = [model.NewIntVar(0, 1, f'U[{j}]') for j in range(m)]
    for j in range(m):
        model.AddAbsEquality(u[j], e[j])

    # Limitação/Obrigação 1
    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)) == K * q)

    # Limitação/Obrigação 2.1 (Parte Superior da Paramétrica)
    K1 = [model.NewIntVar(-q, q, f'K1[{i}]') for j in range(m)]
    p1 = []
    p2 = []
    for j in range(m):
        p1.append(multiplicar_matriz_vetor(L, j, i, e[j]))
        p2.append(multiplicar_matriz_vetor(L, j, i, K1[j], q))
    model.Add((p1[j1] for j1 in range(m)) == (p2[j2] for j2 in range(m)))

    # Limitação/Obrigação 2.2 (Parte Inferior da Paramétrica)
    model.Add(sum(u) >= 1)

    #Executar o solver
    model.Minimize(sum(u))
```

```
    solver = cp_model.CpSolver()
    status = solver.Solve(model)
    if status == cp_model.OPTIMAL or status == cp_model.FEASIBLE:
        print('e = ', end='')
        for j in range(m):
            print(solver.Value(e[j]), end=' ')
        print()

        print('u = ', end='')
        for j in range(m):
            print(solver.Value(u[j]), end=' ')

        print()
    else:
        print('Sem solução!')
```

6 Vetores Finais

[ ]:
```
n = next_prime(2)
m = next_prime(8)
q = next_prime(11)
d = (q - 1) // 2

hard_lattices(n, m, q, d)
```