TP1

October 6, 2025

[2]:
```
!pip install ortools networkx
import warnings
warnings.filterwarnings("ignore")
```

```
Collecting ortools
  Using cached ortools-9.14.6206-cp310-cp310-
manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl.metadata (3.3 kB)
Collecting networkx
  Using cached networkx-3.4.2-py3-none-any.whl.metadata (6.3 kB)
Collecting absl-py>=2.0.0 (from ortools)
  Using cached absl_py-2.3.1-py3-none-any.whl.metadata (3.3 kB)
Collecting numpy>=1.13.3 (from ortools)
  Using cached
numpy-2.2.6-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
(62 kB)
Collecting pandas>=2.0.0 (from ortools)
  Downloading pandas-2.3.3-cp310-cp310-
manylinux_2_24_x86_64.manylinux_2_28_x86_64.whl.metadata (91 kB)
Collecting protobuf<6.32,>=6.31.1 (from ortools)
  Using cached protobuf-6.31.1-cp39-abi3-manylinux2014_x86_64.whl.metadata (593
bytes)
Requirement already satisfied: typing-extensions>=4.12 in
/home/guilherme/.local/lib/python3.10/site-packages (from ortools) (4.15.0)
Collecting immutabledict>=3.0.0 (from ortools)
  Using cached immutabledict-4.2.1-py3-none-any.whl.metadata (3.5 kB)
Requirement already satisfied: python-dateutil>=2.8.2 in
/home/guilherme/.local/lib/python3.10/site-packages (from
pandas>=2.0.0->ortools) (2.9.0.post0)
Collecting pytz>=2020.1 (from pandas>=2.0.0->ortools)
  Using cached pytz-2025.2-py2.py3-none-any.whl.metadata (22 kB)
Collecting tzdata>=2022.7 (from pandas>=2.0.0->ortools)
  Using cached tzdata-2025.2-py2.py3-none-any.whl.metadata (1.4 kB)
Requirement already satisfied: six>=1.5 in
/home/guilherme/anaconda3/envs/logica/lib/python3.10/site-packages (from python-
dateutil>=2.8.2->pandas>=2.0.0->ortools) (1.17.0)
Using cached
ortools-9.14.6206-cp310-cp310-manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl
(27.6 MB)
```

```
Using cached protobuf-6.31.1-cp39-abi3-manylinux2014_x86_64.whl (321 kB)
Using cached networkx-3.4.2-py3-none-any.whl (1.7 MB)
Using cached absl_py-2.3.1-py3-none-any.whl (135 kB)
Using cached immutabledict-4.2.1-py3-none-any.whl (4.7 kB)
Using cached
numpy-2.2.6-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (16.8 MB)
Downloading
pandas-2.3.3-cp310-cp310-manylinux_2_24_x86_64.manylinux_2_28_x86_64.whl (12.8
MB)
                         12.8/12.8 MB
29.7 MB/s  0:00:00m0:00:010:01
Using cached pytz-2025.2-py2.py3-none-any.whl (509 kB)
Using cached tzdata-2025.2-py2.py3-none-any.whl (347 kB)
Installing collected packages: pytz, tzdata, protobuf, numpy, networkx,
immutabledict, absl-py, pandas, ortools
                         9/9
[ortools]m8/9 [ortools]]
Successfully installed absl-py-2.3.1 immutabledict-4.2.1 networkx-3.4.2
numpy-2.2.6 ortools-9.14.6206 pandas-2.3.3 protobuf-6.31.1 pytz-2025.2
tzdata-2025.2
```

Exercício 1

Este problema usa optimização MIP (“Mixed Integer Programming” (OrTools) e representação por Grafos (NetworkX).

1. Para um distribuidor de encomendas o seu território está organizados em pontos (“nodes”) de fornecimento (“sources”), pontos de passagem e pontos de entrega (“sinks”) ligados por vias de comunicação (“edges”) bidirecionais cada uma das quais associada uma capacidade em termos do número de veículos de transporte que suporta.
2. Os items distribuidos estão organizados em “pacotes” de três tipos “standard” : uma unidade, duas unidades e cinco unidades. Os pacotes são transportados em veículos todos com a capacidade de 10 unidades. Cada ponto de fornecimento tem um limite no número total de unidades que tem em “stock” e um limite no número de veículos que dispõe.
3. Cada encomenda é definida por o identificador do ponto de entrega e pelo número de pacotes, de cada um dos tipos, que devem ser entregues nesse ponto.
4. O objetivo do problema é decidir, a partir de uma encomenda e com um mínimo no número de veículos, - em cada ponto de fornecimento, se estará envolvido no fornecimento de unidades que essa encomenda requer sem violar os limites do seu “stock”. - em cada ponto de fornecimento, como empacotar as unidades disponíveis, de acordo com a encomenda”, e como as distribuir por veículos, - em cada veículo, qual o percurso a seguir até ao ponto de entrega; para cada via ao longo de cada percurso, o total de veículos não pode exceder a capacidade dessa via.

[2]:

```
Pedido para sink=8: packages={1: 2, 2: 3, 3: 2}, total_units=18

Solução encontrada:
```

```
Total veículos usados = 2
 Source 1: veículos_used=1, units_sent=8, packages={"t1":2, "t2":3, "t3":0}}
 Source 2: veículos_used=0, units_sent=0, packages={"t1":0, "t2":0, "t3":0}}
 Source 4: veículos_used=1, units_sent=10, packages={"t1":0, "t2":0, "t3":2}}

Fluxos por aresta (soma sobre sources):
  Edge (1-2) : vehicles = 1 / capacity 3
  Edge (2-4) : vehicles = 1 / capacity 2
  Edge (4-5) : vehicles = 1 / capacity 1
  Edge (4-6) : vehicles = 1 / capacity 2
  Edge (5-7) : vehicles = 1 / capacity 2
  Edge (6-7) : vehicles = 1 / capacity 2
  Edge (7-8) : vehicles = 2 / capacity 3

Rotas (heurística de decomposição de fluxo):
 Source 1: routes = [[1, 2, 4, 6, 7, 8]]
 Source 4: routes = [[4, 5, 7, 8]]
```

[ ]:
```
"""
- Usa ortools (MIP via pywraplp) e networkx para representar o grafo.

Como usar:
- O script gera um exemplo aleatório pequeno (customizável) e resolve o MIP.
- Requisitos: ortools, networkx
  pip install ortools networkx

O modelo (resumido):
- Variáveis:
  * x[s][t] : nº de pacotes do tipo t fornecidos pelo source s (inteiro >=0)
  * y[s] : nº de veículos usados a partir do source s (inteiro >=0)
  * f[s][u][v] : nº de veículos do source s que atravessam arco (u,v) (inteiro
 ↪>=0)
- Restrições principais:
  * demanda por tipo satisfeita: sum_s x[s][t] == demand[t]
  * stock em cada source: sum_t x[s][t]*size[t] <= stock_s
  * veículos disponíveis: y[s] <= vehicles_limit_s
  * capacidade veículo: sum_t x[s][t]*size[t] <= 10*y[s]
  * fluxo de veículos do source s forma caminhos (conservação de fluxo), total
 ↪enviado = y[s]
  * capacidade de aresta: para cada aresta {u,v}, sum_s (f[s][u][v]+f[s][v][u])
 ↪<= cap(u,v)
- Objectivo: minimizar total de veículos sum_s y[s]

O modelo é um MIP com variáveis inteiras. Para instâncias maiores poderão ser
 ↪necessárias refinamentos (relaxações, heurísticas, enumeracao de caminhos
 ↪precomputados, etc.).
```

```
"""

from ortools.linear_solver import pywraplp
import networkx as nx
import random
from collections import defaultdict, deque

# ----------------------------
# Configuração do exemplo
# ----------------------------
random.seed(1)

# tipos de pacote: tamanho em unidades
PACKAGE_SIZES = {1:1, 2:2, 3:5}  # indices 1,2,3 representando pacotes {1,2,5}
VEHICLE_CAPACITY = 10

# Gera um grafo pequeno de exemplo (bidireccional)
G = nx.Graph()
nodes = list(range(1,9))
G.add_nodes_from(nodes)
edges = [ (1,2,3), (2,3,2), (2,4,2), (3,5,2), (4,5,1), (4,6,2), (5,7,2),␣
 ↪(6,7,2), (7,8,3) ]
# cada aresta: (u,v,capacidade em nº de veículos)
for u,v,c in edges:
    G.add_edge(u,v,capacity=c)

# Defina fontes (sources) e sink (destino da encomenda)
sources = [1,2,4]
sink = 8

# Stocks e limite de veículos por source
stock = {1:30, 2:20, 4:15}           # unidades disponíveis
vehicle_limit = {1:5, 2:3, 4:3}      # nº de veículos em cada source

# Encomenda: no sink, nº de pacotes por tipo (tipos 1..3)
order_packages = {1:2, 2:3, 3:2}  # 2 pak1 (1u), 3 pak2 (2u), 2 pak3 (5u)
# calcula unidades totais pedidas
total_units = sum(order_packages[t]*PACKAGE_SIZES[t] for t in order_packages)
print(f"Pedido para sink={sink}: packages={order_packages},␣
 ↪total_units={total_units}\n")

# ----------------------------
# Construção do MIP
# ----------------------------
solver = pywraplp.Solver.CreateSolver('CBC')
if not solver:
    raise SystemExit('Solver CBC não disponível')
```

```
# Variáveis x[s][t] inteiro >=0: nº pacotes do tipo t fornecidos por s
x = {s: {t: solver.IntVar(0, solver.infinity(), f'x_{s}_{t}') for t in␣
 ↪PACKAGE_SIZES} for s in sources}

# Variáveis y[s] inteiro >=0: nº veículos usados do source s
y = {s: solver.IntVar(0, vehicle_limit[s], f'y_{s}') for s in sources}

# Variáveis de fluxo f[s][u][v] inteiro >=0 para cada arco orientado (u,v)
f = {s: {} for s in sources}
for s in sources:
    for u,v in G.edges():
        # cria variável para cada orientação
        f[s][(u,v)] = solver.IntVar(0, solver.infinity(), f'f_{s}_{u}_{v}')
        f[s][(v,u)] = solver.IntVar(0, solver.infinity(), f'f_{s}_{v}_{u}')

# Variáveis auxiliares: unidades enviadas por source
units_sent = {s: solver.IntVar(0, solver.infinity(), f'units_{s}') for s in␣
 ↪sources}

# 1) Demanda por tipo satisfeita
for t in PACKAGE_SIZES:
    solver.Add(sum(x[s][t] for s in sources) == order_packages.get(t,0))

# 2) Stock em cada source
for s in sources:
    solver.Add(sum(x[s][t]*PACKAGE_SIZES[t] for t in PACKAGE_SIZES) <= stock[s])

# 3) Relação unidades_sent = sum_t x*size
for s in sources:
    solver.Add(units_sent[s] == sum(x[s][t]*PACKAGE_SIZES[t] for t in␣
 ↪PACKAGE_SIZES))

# 4) capacidade de veículos: unidades_sent <= VEHICLE_CAPACITY * y[s]
for s in sources:
    solver.Add(units_sent[s] <= VEHICLE_CAPACITY * y[s])

# 5) fluxos de veículos: conservação e envio de y[s] veículos do source s até␣
 ↪sink
for s in sources:
    for v in G.nodes():
        in_flow = sum(f[s][(u,v)] for u in G.neighbors(v))
        out_flow = sum(f[s][(v,u)] for u in G.neighbors(v))
        if v == s:
            # no source: out - in = y[s]
            solver.Add(out_flow - in_flow == y[s])
        elif v == sink:
```

```
            # no sink: in - out = y[s]
            solver.Add(in_flow - out_flow == y[s])
        else:
            # nos intermediários, in - out = 0
            solver.Add(in_flow - out_flow == 0)

# 6) capacidade das arestas (soma de todos os flujos orientados não excede␣
 ↪capacity)
for u,v,data in G.edges(data=True):
    cap = data['capacity']
    solver.Add(sum(f[s][(u,v)] + f[s][(v,u)] for s in sources) <= cap)

# 7) veículos usados <= veículos disponíveis já imposto pela definição de y[s]␣
 ↪upper bound
# (finalmente podemos imposer y[s] <= vehicle_limit[s] -- já feito ao criar␣
 ↪variável)

# Objetivo: minimizar total veículos
solver.Minimize(sum(y[s] for s in sources))

# ------------------------------
# Resolver
# ------------------------------
status = solver.Solve()

if status == pywraplp.Solver.OPTIMAL or status == pywraplp.Solver.FEASIBLE:
    print('Solução encontrada:')
    print('Total veículos usados =', sum(int(y[s].solution_value()) for s in␣
 ↪sources))
    for s in sources:
        ys = int(y[s].solution_value())
        us = int(units_sent[s].solution_value())
        print(f" Source {s}: veículos_used={ys}, units_sent={us}, packages={{"␣
 ↪+ ", ".join(f'\"t{t}\":{int(x[s][t].solution_value())}' for t in␣
 ↪PACKAGE_SIZES) + '}}')
    print('\nFluxos por aresta (soma sobre sources):')
    for u,v,data in G.edges(data=True):
        total_flow = sum(int(f[s][(u,v)].solution_value()) + int(f[s][(v,u)].
 ↪solution_value()) for s in sources)
        if total_flow>0:
            print(f'  Edge ({u}-{v}) : vehicles = {total_flow} / capacity␣
 ↪{data["capacity"]}')

    # Reconstruir rotas por source a partir das variáveis de fluxo (heurística␣
 ↪simples)
    print('\nRotas (heurística de decomposição de fluxo):')
```

```
    # converte fluxos para um multigraph de capacidades inteiras por source
    for s in sources:
        ys = int(y[s].solution_value())
        if ys==0:
            continue
        # criar uma dict residual
        residual = defaultdict(lambda: defaultdict(int))
        for (u,v) in list(f[s].keys()):
            val = int(f[s][(u,v)].solution_value())
            if val>0:
                residual[u][v] += val
        routes = []
        # extrair até ys caminhos s->sink
        for _ in range(ys):
            # BFS for a path with available capacity
            parent = {s: None}
            q = deque([s])
            found = False
            while q and not found:
                cur = q.popleft()
                for nb in residual[cur]:
                    if residual[cur][nb] > 0 and nb not in parent:
                        parent[nb] = cur
                        if nb == sink:
                            found = True
                            break
                        q.append(nb)
            if not found:
                break
            # reconstruct path
            path = []
            cur = sink
            while cur is not None:
                path.append(cur)
                cur = parent[cur]
            path = list(reversed(path))
            routes.append(path)
            # decrement residual along path
            for i in range(len(path)-1):
                u = path[i]; v = path[i+1]
                residual[u][v] -= 1
        print(f' Source {s}: routes = {routes}')

else:
    print('Problema sem solução (inviável)')
```

```
Pedido para sink=8: packages={1: 2, 2: 3, 3: 2}, total_units=18
```

```
Solução encontrada:
Total veículos usados = 2
 Source 1: veículos_used=1, units_sent=8, packages={"t1":2, "t2":3, "t3":0}}
 Source 2: veículos_used=0, units_sent=0, packages={"t1":0, "t2":0, "t3":0}}
 Source 4: veículos_used=1, units_sent=10, packages={"t1":0, "t2":0, "t3":2}}

Fluxos por aresta (soma sobre sources):
  Edge (1-2) : vehicles = 1 / capacity 3
  Edge (2-4) : vehicles = 1 / capacity 2
  Edge (4-5) : vehicles = 1 / capacity 1
  Edge (4-6) : vehicles = 1 / capacity 2
  Edge (5-7) : vehicles = 1 / capacity 2
  Edge (6-7) : vehicles = 1 / capacity 2
  Edge (7-8) : vehicles = 2 / capacity 3

Rotas (heurística de decomposição de fluxo):
 Source 1: routes = [[1, 2, 4, 6, 7, 8]]
 Source 4: routes = [[4, 5, 7, 8]]
```

Exercício 2

Este problema deve usar a optimização CP ("Constraint Programming") no OrTools e procura implementar soluções de uma generalização do problema Sudoku.

A definição usual do problema Sudoku (extraido da Wikipedia) contém a seguinte definição

> In classic Sudoku, the objective is to fill a 9×9 grid with digits so that each column, each row, and each of the nine 3×3 subgrids that compose the grid (also called "boxes", "blocks", or "regions") contains all of the digits from 1 to 9. The puzzle setter provides a partially completed grid, which for a well-posed puzzle has a single solution.

Neste trabalho pretende-se generalizar o problema em várias direções:

```
- Em primeiro lugar a grelha tem como parâmetro fundamental um inteiro que toma vários valores
- Em segundo lugar as "regiões" que a definição menciona deixam de ser linhas, colunas e "sub-g
- Na inicialização da solução  as células associadas ao valor $$\,0\,$$ estão livres para ser
- A solução final do problema, tal como no problema original,  verifica uma restrição do tipo a
> dentro uma mesma "box", todas as células têm valores distintos no intervalo $$\,\{1..n^3\}\,$
- Considera-se neste problema suas formas básicas de "boxes":
    - "cubos"  de $$n^3\,$$ células determinados pelo seu vértice superior, anterior, esquerdo
    - "paths" determinados pelo seu vértice de início, o vértice final e pela ordem entre os ín
    O "input" do problema é um conjunto de "boxes" e um conjunto de alocações de valores a célu
```

[5]:
```python
from ortools.sat.python import cp_model

# ---------- Criar grelha 3D ----------
def create_grid(n):
    N = n**2
    model = cp_model.CpModel()
```

```
    grid = {}
    for i in range(1, N+1):
        for j in range(1, N+1):
            for k in range(1, N+1):
                grid[(i,j,k)] = model.NewIntVar(1, n**3, f'cell_{i}_{j}_{k}')
    return grid, model

# ---------- Boxes: all-different ----------
def add_boxes(model, grid, boxes):
    for box in boxes:
        vars_in_box = []
        for pos, val in box.items():
            if val != 0:
                model.Add(grid[pos] == val)  # valor fixo
            vars_in_box.append(grid[pos])
        model.AddAllDifferent(vars_in_box)

# ----------  Criar cubo ----------
def create_cube_box(start_i, start_j, start_k, n):
    box = {}
    for di in range(n):
        for dj in range(n):
            for dk in range(n):
                pos = (start_i + di, start_j + dj, start_k + dk)
                box[pos] = 0  # valor livre
    return box

# ---------- Criar path ----------
def create_path_box(start, end):
    box = {}
    x0, y0, z0 = start
    x1, y1, z1 = end
    dx = 1 if x1 >= x0 else -1
    dy = 1 if y1 >= y0 else -1
    dz = 1 if z1 >= z0 else -1
    x, y, z = x0, y0, z0
    while True:
        box[(x,y,z)] = 0
        if (x, y, z) == (x1, y1, z1):
            break
        if x != x1: x += dx
        if y != y1: y += dy
        if z != z1: z += dz
    return box

# ---------- Restrições tipo Sudoku 3D ----------
def add_sudoku_constraints(model, grid, n):
```

```
    N = n**2
    rng = range(1, N+1)

    # linhas (varia j, fixa i,k)
    for i in rng:
        for k in rng:
            model.AddAllDifferent([grid[(i,j,k)] for j in rng])

    # colunas (varia i, fixa j,k)
    for j in rng:
        for k in rng:
            model.AddAllDifferent([grid[(i,j,k)] for i in rng])

    # profundidades (varia k, fixa i,j)
    for i in rng:
        for j in rng:
            model.AddAllDifferent([grid[(i,j,k)] for k in rng])

# ---------- Resolver ----------
def solve_sudoku_3d(n, boxes, add_constraints=True, time_limit_seconds=10):
    grid, model = create_grid(n)
    add_boxes(model, grid, boxes)
    if add_constraints:
        add_sudoku_constraints(model, grid, n)

    solver = cp_model.CpSolver()
    solver.parameters.max_time_in_seconds = time_limit_seconds

    status = solver.Solve(model)

    if status in (cp_model.OPTIMAL, cp_model.FEASIBLE):
        N = n**2
        solution = [[[0]*N for _ in range(N)] for _ in range(N)]
        for i in range(1, N+1):
            for j in range(1, N+1):
                for k in range(1, N+1):
                    solution[i-1][j-1][k-1] = solver.Value(grid[(i,j,k)])
        return solution, solver, status
    else:
        return None, solver, status

# ---------- Exemplo de input ----------
if __name__ == "__main__":
    n = 2  # cubo 4x4x4
    boxes = []

    # Cubos 2x2x2
```

```
    boxes.append(create_cube_box(1,1,1,n))  # canto superior esquerdo
    boxes.append(create_cube_box(3,3,3,n))  # canto inferior direito

    # Paths
    boxes.append(create_path_box((1,1,1),(4,1,1)))  # linha no eixo i
    boxes.append(create_path_box((1,1,1),(1,4,1)))  # linha no eixo j
    boxes.append(create_path_box((1,1,1),(1,1,4)))  # linha no eixo k

    # Valores iniciais em algumas células
    boxes[0][(1,1,1)] = 2
    boxes[1][(3,3,3)] = 4

    # ---------- Resolver ----------
    solution, solver, status = solve_sudoku_3d(n, boxes, add_constraints=True,␣
↪time_limit_seconds=20)

    if solution is None:
        print("Sem solução encontrada (status =", solver.StatusName(status),␣
↪")")
    else:
        N = n**2
        print("=== Cubo 3D solução ===")
        for i in range(N):
            print(f"Camada i={i+1}:")
            for j in range(N):
                print(solution[i][j])
            print()

        print("=== Valores por box ===")
        for idx, box in enumerate(boxes):
            print(f"Box {idx+1}:")
            for pos in sorted(box.keys()):
                i,j,k = pos
                print(f"  Pos {pos}: {solution[i-1][j-1][k-1]}")
            print()

        # estatísticas do solver
        print("Status:", solver.StatusName(status))
        print("Tempo (s):", solver.WallTime())
        print("Nós pesquisados:", solver.NumBranches())
```

```
=== Cubo 3D solução ===
Camada i=1:
[2, 1, 6, 4]
[3, 6, 7, 8]
[8, 7, 1, 3]
[4, 3, 2, 1]
```

```
Camada i=2:
[4, 7, 5, 3]
[8, 5, 2, 4]
[2, 4, 8, 5]
[5, 1, 6, 8]

Camada i=3:
[6, 4, 2, 8]
[4, 7, 5, 2]
[7, 5, 4, 1]
[3, 2, 7, 6]

Camada i=4:
[5, 8, 7, 1]
[2, 1, 8, 6]
[1, 3, 5, 8]
[6, 7, 3, 2]

=== Valores por box ===
Box 1:
  Pos (1, 1, 1): 2
  Pos (1, 1, 2): 1
  Pos (1, 2, 1): 3
  Pos (1, 2, 2): 6
  Pos (2, 1, 1): 4
  Pos (2, 1, 2): 7
  Pos (2, 2, 1): 8
  Pos (2, 2, 2): 5

Box 2:
  Pos (3, 3, 3): 4
  Pos (3, 3, 4): 1
  Pos (3, 4, 3): 7
  Pos (3, 4, 4): 6
  Pos (4, 3, 3): 5
  Pos (4, 3, 4): 8
  Pos (4, 4, 3): 3
  Pos (4, 4, 4): 2

Box 3:
  Pos (1, 1, 1): 2
  Pos (2, 1, 1): 4
  Pos (3, 1, 1): 6
  Pos (4, 1, 1): 5

Box 4:
  Pos (1, 1, 1): 2
```

```
  Pos (1, 2, 1): 3
  Pos (1, 3, 1): 8
  Pos (1, 4, 1): 4

Box 5:
  Pos (1, 1, 1): 2
  Pos (1, 1, 2): 1
  Pos (1, 1, 3): 6
  Pos (1, 1, 4): 4

Status: OPTIMAL
Tempo (s): 0.031678145000000005
Nós pesquisados: 912
```

1 Exercício 3 - Shortest Vector Problem (SVP)

1.1 Objetivo

Encontrar um vetor binário não-nulo e que satisfaça: H · e 0 (mod q)

1.2 Problema

Dada uma matriz H ^(m×n) com elementos aleatórios em {0..q-1}, encontrar e {0,1}^m tal que: - e 0 (pelo menos um componente 0) - Para cada coluna i: (e × H) 0 mod q - Minimizar o número de componentes não nulas em e

1.3 Formulação Matemática

e {0,1}^m, k {0..m} tal que: i < n: e H = q · k

1.4 Solução

Usamos programação inteira com OR-Tools para encontrar o vetor e que minimiza o número de 1's satisfazendo as restrições modulares.

1.5 Aplicação

Problemas de criptografia pós-quântica baseados em reticulados inteiros.

1.6 Complexidade

- Problema NP-difícil
- Espaço de busca: 2^m vetores possíveis
- OR-Tools resolve eficientemente usando técnicas de branch-and-bound

[7]:
```python
from ortools.linear_solver import pywraplp
import numpy as np

def solve_svp(H, q):
    """
```

```
    Resolve o Shortest Vector Problem usando programação inteira.
    """
    m, n = H.shape

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

    # Variáveis binárias e_j
    e_vars = [solver.IntVar(0, 1, f'e_{j}') for j in range(m)]

    # Variáveis inteiras k_i
    k_vars = [solver.IntVar(0, m, f'k_{i}') for i in range(n)]

    # Restrições modulares
    for i in range(n):
        constraint = solver.Sum(e_vars[j] * H[j][i] for j in range(m)) == q *␣
↪k_vars[i]
        solver.Add(constraint)

    # e não pode ser vetor nulo
    solver.Add(solver.Sum(e_vars) >= 1)

    # Minimizar número de 1's
    solver.Minimize(solver.Sum(e_vars))

    status = solver.Solve()

    if status == pywraplp.Solver.OPTIMAL:
        e_sol = [int(e_vars[j].solution_value()) for j in range(m)]
        num_ones = sum(e_sol)
        return e_sol, num_ones
    else:
        return None, 0

# TESTE PEQUENO QUE ENCONTRA SOLUÇÃO
print("=== SVP - TESTE PEQUENO (Solução Garantida) ===\n")

# Parâmetros pequenos que garantem solução rápida
n = 4   # Dimensão pequena
m = 8   # m = 2n
q = 17  # Primo pequeno

print(f"Parâmetros: n={n}, m={m}, q={q}")

# Gerar matriz H que GARANTE solução
np.random.seed(123)  # Para ser reproduzível, mas sempre com solução
H = np.random.randint(0, q, size=(m, n))
```

```
# Garantir que há pelo menos uma solução: criar duas linhas complementares
H[1] = (q - H[0]) % q  # H[0] + H[1]  0 mod q

print(f"\nMatriz H ({m}x{n}):")
for i in range(m):
    print(f"  {H[i]}")

print(f"\nVerificação: H[0] + H[1] mod q = {[(H[0][i] + H[1][i]) % q for i in␣
 ↪range(n)]}")

# Resolver
import time
start_time = time.time()
e, num_ones = solve_svp(H, q)
end_time = time.time()

if e is not None:
    print(f"\n SOLUÇÃO ENCONTRADA em {end_time - start_time:.3f} segundos:")
    print(f"e = {e}")
    print(f"Número de componentes não nulas: {num_ones}")

    # Verificação completa
    print(f"\nVERIFICAÇÃO:")
    todas_corretas = True
    for i in range(n):
        soma = sum(e[j] * H[j][i] for j in range(m))
        resto = soma % q
        status = "ok" if resto == 0 else "nop"
        print(f"  Coluna {i}: {soma}  {resto} (mod {q}) {status}")
        if resto != 0:
            todas_corretas = False

    print(f"\nResumo: {num_ones} componentes não nulas, todas as {n} restrições␣
 ↪satisfeitas!")

else:
    print("\n Nenhuma solução encontrada (inesperado!)")

print(f"\nEspaço de busca: 2^{m} = {2**m} vetores possíveis")
```

```
=== SVP - TESTE PEQUENO (Solução Garantida) ===

Parâmetros: n=4, m=8, q=17

Matriz H (8x4):
  [13  2  2  6]
  [ 4 15 15 11]
```

```
  [ 9  0 14  0]
  [15 14  4  0]
  [16  4  3  2]
  [ 7  2 15 16]
  [7 9 3 6]
  [ 1  2  1 12]

Verificação: H[0] + H[1] mod q = [np.int64(0), np.int64(0), np.int64(0),
np.int64(0)]

 SOLUÇÃO ENCONTRADA em 0.045 segundos:
e = [1, 1, 0, 0, 0, 0, 0, 0]
Número de componentes não nulas: 2

VERIFICAÇÃO:
  Coluna 0: 17   0 (mod 17) ok
  Coluna 1: 17   0 (mod 17) ok
  Coluna 2: 17   0 (mod 17) ok
  Coluna 3: 17   0 (mod 17) ok

Resumo: 2 componentes não nulas, todas as 4 restrições satisfeitas!

Espaço de busca: 2^8 = 256 vetores possíveis
```

2 Exercício 4 - Closest Vector Problem (CVP)

2.1 Objetivo

Encontrar a combinação linear inteira de vetores mais próxima de um vetor target.

2.2 Problema

Dados vetores geradores G (probabilidades entre 0 e 1) e um vetor target t, encontrar coeficientes inteiros z (com |z |) que minimizem:

distância = ||(z g + z g + ... + z g) - t||

2.3 Solução

Usamos programação inteira com OR-Tools para resolver eficientemente, mesmo para problemas grandes.

2.4 Exemplo

- **8 dimensões**, **6 vetores geradores**, coeficientes entre **-2 e 2**
- **15.625 combinações possíveis** → resolvido rapidamente
- OR-Tools garante solução ótima

2.5 Aplicação

Problemas de criptografia pós-quântica e optimização em reticulados.

[ ]:
```python
from ortools.linear_solver import pywraplp
import numpy as np

def solve_cvp_ortools(G, t, l=1):
    """
    Resolve CVP com OR-Tools usando programação inteira.
    """
    G = np.array(G)
    t = np.array(t)
    k, m = G.shape

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

    # Variáveis inteiras z_i
    z_vars = [solver.IntVar(-l, l, f'z_{i}') for i in range(k)]

    # Variáveis auxiliares para diferenças
    diff_vars = [solver.NumVar(-10, 10, f'diff_{j}') for j in range(m)]

    # Restrições: diff_j = sum_i(G[i,j] * z_vars[i]) - t[j]
    for j in range(m):
        solver.Add(diff_vars[j] == sum(G[i,j] * z_vars[i] for i in range(k)) -␣
↪t[j])

    # Função objetivo: minimizar soma dos valores absolutos (aproximação)
    abs_vars = [solver.NumVar(0, 10, f'abs_{j}') for j in range(m)]

    for j in range(m):
        solver.Add(abs_vars[j] >= diff_vars[j])
        solver.Add(abs_vars[j] >= -diff_vars[j])

    solver.Minimize(solver.Sum(abs_vars))

    # Resolver
    status = solver.Solve()

    if status == pywraplp.Solver.OPTIMAL:
        z_sol = [int(z_vars[i].solution_value()) for i in range(k)]
        x_sol = G.T @ z_sol
        dist = np.linalg.norm(x_sol - t)  # Distância euclidiana real
        return z_sol, x_sol, dist
    else:
        raise Exception('Solução ótima não encontrada')
```

```python
# TESTE COM RANDOMS DIFERENTES CADA VEZ
print("=== CVP COM OR-TOOLS - TESTE COM RANDOMS REAIS ===\n")

m = 8    # Dimensão do espaço
k = 6    # Número de vetores geradores
l = 2    # Limite maior para coeficientes

# Gerar vetores aleatórios DIFERENTES cada execução
G = np.random.rand(k, m).tolist()
t = np.random.rand(m).tolist()

print(f"Parâmetros: m={m}, k={k}, l={l}")
print(f"Número de combinações possíveis: {(2*l + 1)**k} = {pow(2*l + 1, k)}")

print("\nVetores geradores:")
for i, g in enumerate(G):
    print(f"  g_{i+1} = {[round(x, 3) for x in g]}")

print(f"\nTarget: t = {[round(x, 3) for x in t]}")

# Resolver
try:
    z, x, dist = solve_cvp_ortools(G, t, l)

    print(f"\nSOLUÇÃO:")
    print(f"Coeficientes z = {z}")
    print(f"Vetor reticulado x = {[round(xi, 3) for xi in x]}")
    print(f"Distância euclidiana = {dist:.4f}")

    # Verificar se está dentro dos limites
    print(f"\nVERIFICAÇÃO:")
    print(f"Todos |z_i| <= {l}: {all(abs(z_i) <= l for z_i in z)}")
    print(f"Tamanho do problema: {k} variáveis, {m} restrições")

except Exception as e:
    print(f"\nERRO: {e}")
```

```
=== CVP COM OR-TOOLS - TESTE COM RANDOMS REAIS ===

Parâmetros: m=8, k=6, l=2
Número de combinações possíveis: 15625 = 15625

Vetores geradores:
  g_1 = [0.265, 0.244, 0.973, 0.393, 0.892, 0.631, 0.795, 0.503]
  g_2 = [0.577, 0.493, 0.195, 0.722, 0.281, 0.024, 0.645, 0.177]
  g_3 = [0.94, 0.954, 0.915, 0.37, 0.015, 0.928, 0.428, 0.967]
```

```
  g_4 = [0.964, 0.853, 0.294, 0.385, 0.851, 0.317, 0.169, 0.557]
  g_5 = [0.936, 0.696, 0.57, 0.097, 0.615, 0.99, 0.14, 0.518]
  g_6 = [0.877, 0.741, 0.697, 0.702, 0.359, 0.294, 0.809, 0.81]

Target: t = [0.867, 0.913, 0.511, 0.502, 0.798, 0.65, 0.702, 0.796]

SOLUÇÃO:
Coeficientes z = [0, -1, -1, 0, 1, 2]
Vetor reticulado x = [np.float64(1.174), np.float64(0.731), np.float64(0.854),
np.float64(0.41), np.float64(1.038), np.float64(0.625), np.float64(0.685),
np.float64(0.995)]
Distância euclidiana = 0.5923

VERIFICAÇÃO:
Todos |z_i| <= 2: True
Tamanho do problema: 6 variáveis, 8 restrições
```