Tp1

November 6, 2019

1 Trabalho Prático 1

1.0.1 Exercicio 1

O problema em questão passa por preencher um tabuleiro de (N*N)(N*N) quadrados de sudoku seguindo as regras do jogo. Cada quadrado tem um unico numero de 1 a N*N. Para além disso, não podem existir numeros repetidos na mesma linha, coluna ou "quadrado de quadrados" N*N. A nossa solução passa por criar as variáveis, tendo elas valor booleano. Tal acontece porque para cada quadrado(linha x coluna), para cada valor possivel, a variavel é 1 se naquela posição for colocado o numero em questão (caso contrário é 0). Depois de as variaveis estarem definidas, aplicam-se as condiçoes do sudoku utilizando o quicksum do Scip, uma vez que, a soma num grupo de variáveis ser igual a 1 é o mesmo que dizer que nesse grupo não existe um numero repetido. Para resolver o problema usa-se o método optimize para se encontrar uma soluçao para o sudoku.

[1]:
```python
import numpy as np
from pyscipopt import Model,quicksum
import random
```

[2]:
```python
def sudoku (N):
    M = N*N
    X = {}
    S = Model()

    #Criação de Variáveis
    for l in range(M):
        X[l] = {}
        for c in range(M):
            X[l][c] = {}
            for d in range(M):
                X[l][c][d] = S.addVar('',vtype='B')


    #Condições

    #Cada quadrado pode ter apenas um numero
    for l in range(M):
        for c in range(M):
            S.addCons(quicksum([X[l][c][d] for d in range(M)]) == 1)
```

```
    #Não há numeros repetidos na mesma linha
    for l in range(M):
        for d in range(M):
            S.addCons(quicksum([X[l][c][d] for c in range(M)]) == 1)

    #Nao há numeros repetidos na mesma coluna
    for c in range(M):
        for d in range(M):
            S.addCons(quicksum([X[l][c][d] for l in range(M)]) == 1)


    #Nao há numeros repetidos no mesmo quadrado N*N
    for x in range(N):
        for y in range(N):
            for d in range(M):
                S.addCons(quicksum([X[l][c][d] for l in range(x*N,(x*N)+N) for␣
↪c in range(x*N,(x*N)+N)]) == 1)


    #Criar tabuleiro introduzindo numeros aleatoriamente
    sol = np.zeros((M,M),dtype=int)
    x = random.choice(range(1,M))
    while x != 0 :
        l = random.choice(range(M))
        c = random.choice(range(M))
        d = random.choice(range(M))
        S.addCons(X[l][c][d] == 1)
        sol[l][c] = d+1
        x = random.choice(range(M))
    #print(sol)

    #Imprimir solução
    S.optimize()
    if S.getStatus() == 'optimal':
        for l in range(M):
            for c in range(M):
                for d in range(M):
                    if (S.getVal(X[l][c][d]) == 1):
                        sol[l][c] = d+1
        print(sol)
    else:
        print("Sem Solução")

sudoku(3)
```

```
[[1 2 3 9 4 5 6 7 8]
 [4 5 6 8 7 1 9 3 2]
 [7 8 9 2 3 4 5 6 1]
 [5 6 7 1 2 3 8 9 4]
 [2 9 8 4 5 6 3 1 7]
 [6 3 1 7 8 9 2 4 5]
 [9 7 4 5 6 8 1 2 3]
 [8 1 2 3 9 7 4 5 6]
 [3 4 5 6 1 2 7 8 9]]
```

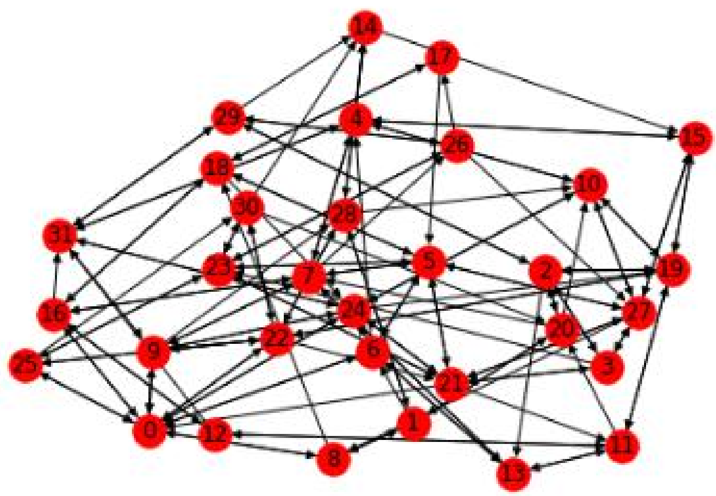
1.0.2 Exercício 2

Este problema consiste em ver um sistema de tráfego como um grafo orientado e ligado em que os arcos denotam vias de comunicação(com um ou dois sentidos) e os nodos denotam pontos de acesso. O primeiro objetivo é gerar um grafo ligado, ou seja: para cada par de nodos <n1,n2> existe um caminho de n1 para n2 e de n2 para n1. Consideramos que existe igual probabilidade de cada via ter um só sentido ou os 2 sentidos. Assumindo N = 32 e uma probabilidade de p = 2^(-k) para a existência de k descendentes adicionais.

[3]:
```
import networkx as nx
import random
import matplotlib.pyplot as plt
```

[4]:
```
    #geração de um grafo de forma aleatória

    G = nx.DiGraph() #criação de um grafo orientado vazio
    G.add_nodes_from([i for i in range(32)]) #adicionar N = 32 nodos ao grafo
    ligado = nx.is_strongly_connected(G)
    while ligado == False: #enquanto o grafo não for ligado vai continuar a␣
↪adicionar edges

        for o in G.nodes(): #1º 'for' para ggarantir que todos os nodos tem␣
↪pelo menos 1 ligação a um outro qualquer nodo
            d = o
            while d == o:
                d = random.randint(0,31)
            G.add_edge(o,d)

        for o in G.nodes(): #2º for para "atirar uma moeda ao ar" através da␣
↪função randint e se
                                    # calhar 1 ligamos o nodo em que estamos a um outro␣
↪qualquer
            r = random.randint(0,1)
            if r == 1:
                d = random.randint(0,31)
                while d == o:
                    d = random.randint(0,31)
                G.add_edge(o,d)
```

```
                r = random.randint(0,1)

        for (o,d) in G.edges():#3º for onde se volta a "atirar a moeda" para␣
↪ver se vamos ter uma ligação num sentido
                              # ou se temos em ambos os sentidos
            if G.has_edge(d,o) == False:
                s = random.randint(0,1)
                if s == 1:
                    G.add_edge(d,o)

        ligado = nx.is_strongly_connected(G)

    nx.draw(G,with_labels = True)
```



O próximo problema consiste em ir ao grafo gerado anteriormente e verificar o número máximo de edges que podem ser retirados e o grafo continuar a ser ligado. Trata-se de um problema de otimização em que vamos ter em conta as seguintes restrições:

1)Para cada par de vértices existe pelo menos um caminho : para todo $o \in V$ e todo $d \in V$

$$\sum_{c \in C} c[(o,d]) >= 1$$

2)O produto das arestas de um dado caminho c de cada par de vértices é igual ao valor da variável c: para todo $o \in V$ e todo $d \in V$ com 'o' diferente de 'd' , para todo $c \in C$

$$arestas[c] == caminhos[o][d][c]$$

[5]:
```
from pyscipopt import Model, quicksum
```

[ ]:
```
def cortado(G):
    #criação das variáveis

    final = nx.DiGraph()
    m = Model()
    arestas = {}
    caminhos = {}
    for x in G.edges():
        arestas[x] = m.addVar('',vtype = 'B')
    print(arestas)

    for o in G.nodes():
        for d in G.nodes():
            if o != d:
                caminhos[(o,d)] = {}
                for p in nx.all_simple_paths(G,o,d):
                    caminhos[(o,d)] = m.addVar('',vtype = 'B')

    # estabelecimento de restrições
    # restrição 1)
    for o in G.nodes():
        for d in G.nodes():
            if o != d:
                m.addCons(quicksum([caminhos[o,d][x] for x in caminhos[(o,d)]])␣
↪>=1)

    # restrição 2)
    for o in G.nodes():
        for d in G.nodes():
```

```python
            if o != d:
                for c in caminhos[(o,d)]:
                    l = []
                    for k in range(len(l)-1):
                        l.append((c[k],c[k+1]))
                    m.addCons(quickprod([arestas[a] for a in l]) == caminhos[(o,d)][c])

    m.setObjetive(quicksum([arestas[a] for a in arestas]),sense = 'minimze')
    m.optimize()
    for (o,d) in arestas:
        if m.getVal(arestas[(o,d)]) == 1:
            final.add_edge(o,d)

    return final
print(nx.is_strongly_connected(G))
final = cortado(G)
nx.draw(final, with_labels = True)
```

```
True
{(0, 25): x1, (0, 24): x2, (0, 6): x3, (0, 9): x4, (0, 16): x5, (0, 8): x6, (0,
22): x7, (1, 6): x8, (1, 8): x9, (1, 28): x10, (1, 20): x11, (2, 19): x12, (2,
3): x13, (2, 20): x14, (2, 13): x15, (2, 29): x16, (3, 27): x17, (3, 2): x18,
(3, 23): x19, (3, 21): x20, (4, 7): x21, (4, 14): x22, (4, 10): x23, (4, 15):
x24, (4, 31): x25, (5, 21): x26, (5, 6): x27, (5, 23): x28, (5, 7): x29, (5,
18): x30, (5, 27): x31, (6, 0): x32, (6, 24): x33, (6, 4): x34, (6, 5): x35, (6,
13): x36, (7, 16): x37, (7, 26): x38, (7, 9): x39, (7, 4): x40, (7, 5): x41, (7,
13): x42, (8, 18): x43, (8, 1): x44, (8, 0): x45, (9, 22): x46, (9, 0): x47, (9,
28): x48, (9, 7): x49, (9, 24): x50, (10, 27): x51, (10, 24): x52, (10, 19):
x53, (10, 4): x54, (11, 20): x55, (11, 12): x56, (11, 13): x57, (11, 19): x58,
(12, 31): x59, (12, 11): x60, (12, 16): x61, (13, 6): x62, (13, 7): x63, (13,
11): x64, (14, 28): x65, (14, 15): x66, (15, 27): x67, (15, 4): x68, (15, 19):
x69, (16, 31): x70, (16, 0): x71, (16, 7): x72, (16, 18): x73, (16, 12): x74,
(17, 18): x75, (17, 5): x76, (18, 16): x77, (18, 5): x78, (18, 24): x79, (18,
17): x80, (19, 2): x81, (19, 22): x82, (19, 11): x83, (19, 15): x84, (19, 10):
x85, (20, 7): x86, (20, 10): x87, (20, 1): x88, (20, 2): x89, (21, 24): x90,
(21, 11): x91, (21, 5): x92, (21, 27): x93, (21, 0): x94, (21, 3): x95, (21,
23): x96, (22, 21): x97, (22, 25): x98, (22, 19): x99, (22, 0): x100, (22, 9):
x101, (22, 28): x102, (22, 30): x103, (23, 30): x104, (23, 26): x105, (23, 25):
x106, (23, 21): x107, (23, 5): x108, (24, 10): x109, (24, 0): x110, (24, 6):
x111, (24, 19): x112, (24, 9): x113, (24, 21): x114, (24, 31): x115, (25, 0):
x116, (25, 23): x117, (25, 30): x118, (25, 22): x119, (26, 29): x120, (26, 17):
x121, (26, 27): x122, (26, 23): x123, (27, 21): x124, (27, 3): x125, (27, 10):
x126, (27, 15): x127, (27, 8): x128, (27, 5): x129, (28, 22): x130, (28, 10):
x131, (28, 1): x132, (28, 14): x133, (29, 31): x134, (29, 26): x135, (29, 2):
x136, (29, 14): x137, (30, 22): x138, (30, 20): x139, (30, 14): x140, (30, 23):
x141, (31, 9): x142, (31, 4): x143, (31, 24): x144, (31, 29): x145}
```

```
[ ]:
```