Trabalho Prático 1

Ex.2 : Sudoku

O jogo "Sudoku" é generalizado para a dimensão tradicional $N = 3$. O objetivo do Sudoku é preencher uma grelha de $N^2 \times N^2$ com inteiros positivos no intervalo 1 até N^2, satisfazendo as seguintes regras:

- Cada inteiro no intervalo 1 até N^2 ocorre só uma vez em cada coluna, linha e secção $N \times N$.
- No início do jogo uma fração $0 \leq \alpha < 1$ das N^4 casas da grelha são preenchidas de forma consistente com a regra anterior.

Inicialização da grelha através de N e alpha e criação da instância do solver para resolver o problema:

```
from ortools.linear_solver import pywraplp
# Criar a instância do solver para definir o horário
sudoku_solver = pywraplp.Solver.CreateSolver('SCIP')
#Variáveis para a resolução
N = 3
alpha = 0.2

#Criar a matriz
grelha = {}
for l in range(1,N*N+1):
    grelha[l] = {}
    for c in range(1,N*N+1):
        grelha[l][c] = {}
        for n in range(1,N*N+1):
            grelha[l][c][n] = sudoku_solver.BoolVar('grelha[%i][%i][%i]' % (l,c,n))
```

Numa linha (l), cada número (n) aparece uma e só uma vez. Então, em cada linha, o sumatório do número de vezes que n aparece é 1.

$$\forall_l . \forall_n . \sum_{c=1}^{N^2} grelha_{l,c,n} = 1$$

```
#Cada número só aparece uma vez na linha
for l in range(1,N*N+1):
    for n in range(1,N*N+1):
        sudoku_solver.Add(sum([grelha[l][c][n] for c in range(1,N*N+1)]) == 1)
```

Numa coluna (c), cada número (n) aparece uma e só uma vez. Então, em cada coluna, o sumatório do número de vezes que n aparece é 1.

$$\forall_c.\forall_n.\sum_{l=1}^{N^2} grelha_{l,c,n} = 1$$

```
#Cada número só aparece uma vez na coluna
for c in range(1,N*N+1):
    for n in range(1,N*N+1):
        sudoku_solver.Add(sum([grelha[l][c][n] for l in range(1,N*N+1)]) == 1)
```

Em cada quadrado de N por N, cada número (n) aparece uma e só uma vez. Então, em cada quadrado, o sumatório do número de vezes que n aparece é 1. Sendo que, para verificar esse acontecimento, usamos p e q que variam entre 1 e N-1, multiplicando por N, de seguida, subtraindo N e por fim somando 1, o que nos dá os limites do quadrado em questão.

$$\forall_l.\forall_c.\forall_n.\forall_q.\forall_p.\sum_{n=1}^{N^2} grelha_{l,c,n,q,p} = 1$$

```
#Cada número só aparece uma vez no quadrado N*N
for n in range(1,N*N+1):
    for q in range(1,N):
        for p in range(1,N):
            sudoku_solver.Add(sum([grelha[l][c][n] for l in range(N*p-N+1,N*p+1) for c in
```

Numa casa (l)(c), cada número (n) aparece uma e só uma vez. Então, em cada casa, o sumatório do número de vezes que n aparece é 1.

$$\forall_l.\forall_c.\sum_{n=1}^{N^2} grelha_{l,c,n} = 1$$

```
#Todas as casas têm de ter um número associado
for l in range(1,N*N+1):
    for c in range(1,N*N+1):
        sudoku_solver.Add(sum([grelha[l][c][n] for n in range(1,N*N+1)]) == 1)
```

Construir um programa para inicializar a grelha a partir dos parâmetros N e α.

O programa utiliza o solver para resolver o problema e, em seguida, utilizando o α multiplicamos por N^4 para sabermos o número de casas que serão inicializadas. Enquanto esse valor não é atingido, são escolhidos números aleatóriamente e guardados na matriz $known$ para serem impressos na grelha inicial.

```
import random
```

```
#Invocar o solver para criar a grelha inicial
status = sudoku_solver.Solve()
#criar a matriz para guardar os valores conhecidos(iniciais)
known = {}
for l in range(1,N*N+1):
    known[l] = {}
    for c in range(1,N*N+1):
        known[l][c] = 0

#Se possivel preencher a grelha inicial
if status == pywraplp.Solver.OPTIMAL:
    #Invocar números random para as linhas e colunas
    for i in range(1,round(alpha*N*N*N*N)+1):
        l = random.randint(1,N*N)
        c = random.randint(1,N*N)
        while(known[l][c] != 0):
            l = random.randint(1,N*N)
            c = random.randint(1,N*N)
        for n in range(1,N*N+1):
            #Para uma linha e coluna, verificar qual o número da própria e guardar na matr
            if grelha[l][c][n].solution_value() == 1:
                known[l][c] = n
    #Programa que percorre a matriz para imprimir os valores '*' no caso de ser 0(Não tem
    for l in range(1,N*N+1):
        for c in range(1,N*N+1):
            if c != N*N:
                if known[l][c] == 0:
                    print(' *',end=' ')
                else:
                    if known[l][c] < 10:
                        print('',known[l][c], end=' ')
                    else:
                        print(known[l][c], end=' ')
            else:
                if known[l][c] == 0:
                    print(' *')
                else:
                    print('',known[l][c])

else:
    print('Erro')
```

```
 1  *  8  *  *  *  4  3  *
 *  5  4  *  3  *  *  6  *
 *  3  *  *  *  *  *  *  *
 *  *  6  *  *  *  *  *  *
 *  4  *  *  *  *  *  *  *
 *  *  *  *  9  *  *  *  *
 *  *  9  *  *  *  *  *  *
 *  *  7  *  *  *  2  *  *
 *  1  *  *  *  *  *  *  *
```

Output

Utilizando as regras do enunciado, imprime o sudoku resolvido.

```
#Variáveis para o auxilio da impressão do sudoku
p = 1
o = 1
#Se existir resultado imprimir o sudoku
if status == pywraplp.Solver.OPTIMAL:
    #Impressão do sudoku
    for l in range(1,N*N+1):
        for c in range(1,N*N+1):
            for n in range(1,N*N+1):
                if grelha[l][c][n].solution_value() == 1:
                    if (l == 1  and c == 1):
                        for t in range(1,N*N*3+N*2+N+1):
                            print('-',end='')
                        print('-')
                    if l == N*o+1:
                        for t in range(1,N*N*3+N*2+N+1):
                            print('-',end='')
                        print('-')
                        o += 1
                    if c == 1:
                        if n < 10:
                            print('| ', n,end=' ')
                        else:
                            print('|', n,end=' ')
                    elif c == N*N:
                        if n < 10:
                            print('',n,end='  |\n')
                            p = 1
                        else:
                            print(n,end='  |\n')
                            p = 1
                    elif c != N*N and c != N*p:
                        if n < 10:
                            print('',n, end=' ')
                        else:
                            print(n,end=' ')
                    else:
                        if n < 10:
                            print('',n, end='  | ')
                        else:
                            print(n, end='  | ')
                        p +=1
    for t in range(1,N*N*3+N*2+N+1):
        print('-',end='')
    print('-')
else:
    print('Erro')
```

```
 -------------------------------------
 |  1  9  8  |  7  6  5  |  4  3  2  |
```

```
|  7  5  4  |  9  3  2  |  8  6  1  |
|  6  3  2  |  8  4  1  |  9  7  5  |
-------------------------------------
|  9  7  6  |  5  2  8  |  3  1  4  |
|  8  4  3  |  6  1  7  |  5  2  9  |
|  5  2  1  |  4  9  3  |  7  8  6  |
-------------------------------------
|  4  8  9  |  2  7  6  |  1  5  3  |
|  3  6  7  |  1  5  9  |  2  4  8  |
|  2  1  5  |  3  8  4  |  6  9  7  |
-------------------------------------
```

À medida que o trabalho foi realizado verificamos que, com o aumento do N e do alpha, a complexidade do problema aumenta e, com isto, o tempo de execução aumenta exponencialmente, notando-se logo apartir de N=5 que, apesar de executar, já tem um longo tempo de espera, mas principalmente em N = 6 que se torna inviável o uso de um solver (programação linear) para a resolução do sudoku devido ao excessivo tempo de espera.