Projeto 1 - Backtracking

SCC-5900 - Projeto de Algoritmos

Aluno: Afonso Matheus Sousa Lima

Nº USP: 11357812

1 Explicação da Implementação

O algoritmo de *backtracking* para resolução do problema Sudoku foi implementado seguindo os passos do modelo genérico apresentado em aula. A função *backtrack* presente no arquivo **sudoku_solver.py** recebe como argumentos de entrada o Sudoku a ser resolvido e a heurística que deve ser utilizada, sendo o Sudoku lido de um arquivo estruturado conforme descrito no documento do projeto e a heuristica podendo receber os valores: *None, fv, mvs*.

Primeiro passo do algoritmo é criar um dicionário (estrutura chave-valor do Python) onde cada chave armazena a posição dos valores não preenchidos (0) dentro do Sudoku. Na medida que se adiciona valores nessas posições, a chave respectiva é retirada do dicionário. Quando esse dicionário estiver vazio, então a tarefa é completada. Escolhendo o primeiro espaço vazio do Sudoku contido no dicionário, é atribuído à ele valores de 1 até 9, que são os valores possíveis dentro do Sudoku 9x9. A cada atribuição, é verificado se esse valor já é atribuído em algum espaço na mesma linha ou coluna ou subgrade do espaço vazio. Caso exista, o próximo valor é atribuido ao espaço vazio. Caso não exista, então o valor é atribuído a esse espaço e o *backtrack* é chamado recursivamente. Isso é feito para todos os espaços vazios até que o dicionário não tenha mais nenhuma chave, com isso retornando o jogo de Sudoku solucionado. Caso ocorra de um espaço vazio não ser preenchido com nenhum valor entre 1 e 9, o algoritmo retornará para um estado onde um espaço vazio ainda pode ser atribuído um valor nesse intervalo, até encontrar a solução.

Para a heurística verificação adiante, a função *foward_verification*, baseado no espaço vazio escolhido, seleciona valores que podem ser atribuídos aquele espaço, utilizando funções semelhantes às usadas para verificar a presença do valor em linhas, colunas e subgrades. Então, ao atribuir valores aos espaços, é considerado só esses possíveis valores, não mais sendo de 1 até 9. Caso em um momento os valores possíveis de um espaço seja vazio, então o algoritmo retornará para um estado anterior, e tentará com outro possível valor. Para a heurística MVS, utilizada em conjunto com a verificação adiante, a função *mvs* retorna, além dos possíveis valores, o espaço vazio

com menor número de valores possíveis, ditando a ordem em que os espaços vazios serão preenchidos.

2 Executando o Algoritmo

O algoritmo foi desenvolvido utilizando Python3. Ele pode ser executado utilizando o comando:

```
python3 sudoku_solver.py <nome-do-arquivo> <heuristica>
```

As heurísticas possíveis são: *None, fv, mvs*, representando: sem heurística, verificação adiante e verificação adiante com MVS, respectivamente. É apresentado o Sudoku antes de ser resolvido, a quantidade de atribuições de variáveis, o Sudoku resolvido e o tempo de execução do algoritmo.

3 Resultados

Para o Sudoku proposto pelo projeto, obteve-se os seguintes resultados:

Algoritmo	*Backtracking*	*Back. + Verificação adiante*	*Back. + V.a. + MVS*
Número de atribuições	14.096	1.589	54
Tempo de execução (s)	0.042	0.052	0.019

A próxima tabela mostra os resultados para outros exemplos presentes no arquivo anexado *sudoku_test*. Ressaltando que o Sudoku 1 é o que foi apresentado anteriormente.

Sudoku	**Nível de Dificuldade**[1]	**Backtracking**		**Back. + V.a.**		**Back. + V.a. + MVS**	
		Tempo de execução (s)	**Núm. de atribuições**	**Tempo de execução (s)**	**Núm. de atribuições**	**Tempo de execução (s)**	**Núm. de atribuições**
2	Fácil	0.0010	278	0.0014	48	0.0085	36
3	Fácil	0.0011	237	0.0019	42	0.0095	36
4	Médio	0.1047	37.038	0.1304	4137	0.0301	73
5	Difícil	0.2784	100.553	0.3225	11198	0.1006	206
6	Difícil	0.1382	47.059	0.1626	5252	0.0252	68

[1] Dificuldade baseada no site: Sudoku - Racha Cuca

O algoritmo também pode ser encontrado neste repositório do GitHub.