Universidade de São Paulo - USP

Instituto de Ciências Matemáticas e de Computação - ICMC
Departamento de Ciências de Computação - SCC
SCC-5900 - Projeto de Algoritmos

Professor: Gustavo Batista

Aluno: Jorge Andoni Valverde Tohalino

Projeto 01 - Backtracking

1. Implementação do Sudoku

a. Compilação:

Os algoritmos de backtracking foram feitos em Java. O código está organizado em duas classes Utils e Sudoku. A classe Utils tem algumas funções auxiliares. A classe principal Sudoku resolve cada um das matrices do arquivo de entrada de acordo com o tipo de heurística indicada.

Para compilar o programa os seguintes passos são seguidos:

- Na terminal executar: java -jar Sudoku.jar
- Depois, escolher o tipo de heurística: 0 (Backtracking Simple) , 1 (Forward Checking) , 2 (MVR)
- Finalmente, escolher o tipo de arquivo de entrada: 0 (entrada.txt) , 1 (entrada10.txt)

b. Backtracking Simple:

Implementação sem heurísticas de poda. Foram utilizadas algumas funções auxiliares:

- searchFreeSpace: encontra a primeira posição disponível na matriz.
- isPerfect: verifica se um valor está correto numa linha, coluna e box. Utiliza as funções auxiliares isInRow, isInCol, isInBox

c. Backtracking com Forward Checking:

Para usar Backtracking com Forward Checking é criada a estrutura Map<String,Vector> probable_values. A chave do Map representa uma posição livre da matriz (ex. 00 , 11 , 12, etc.) . O value é um Vector, a primeira posição do Vector representa o número de valores prováveis disponíveis e os seguintes elementos são flags que identificam os valores remanescentes. Antes de avançar para a posição seguinte, o algoritmo primeiro verifica se na estrutura probable_values existe uma posição com 0 valores prováveis, então, se foi encontrada uma posição com 0 valores prováveis, o algoritmo faz backtracking, caso contrário, avança para a próxima posição.

d. Backtracking com Forward Checking e MVR:

Neste caso, a nova função findMVR() é usada. A função busca na estrutura probable_values a posição com o menor número de valores prováveis. Então, para encontrar a posição seguinte para ser utilizada, o algoritmo usa a função findMVR().

A implementação está disponível em https://github.com/andoniVT/Sudoku

2. Avaliação

a. Tempos de execução:

Os gráficos seguintes mostram o tempo de execução dos três algoritmos. Eles foram avaliados com os 95 jogos de Sudoku.

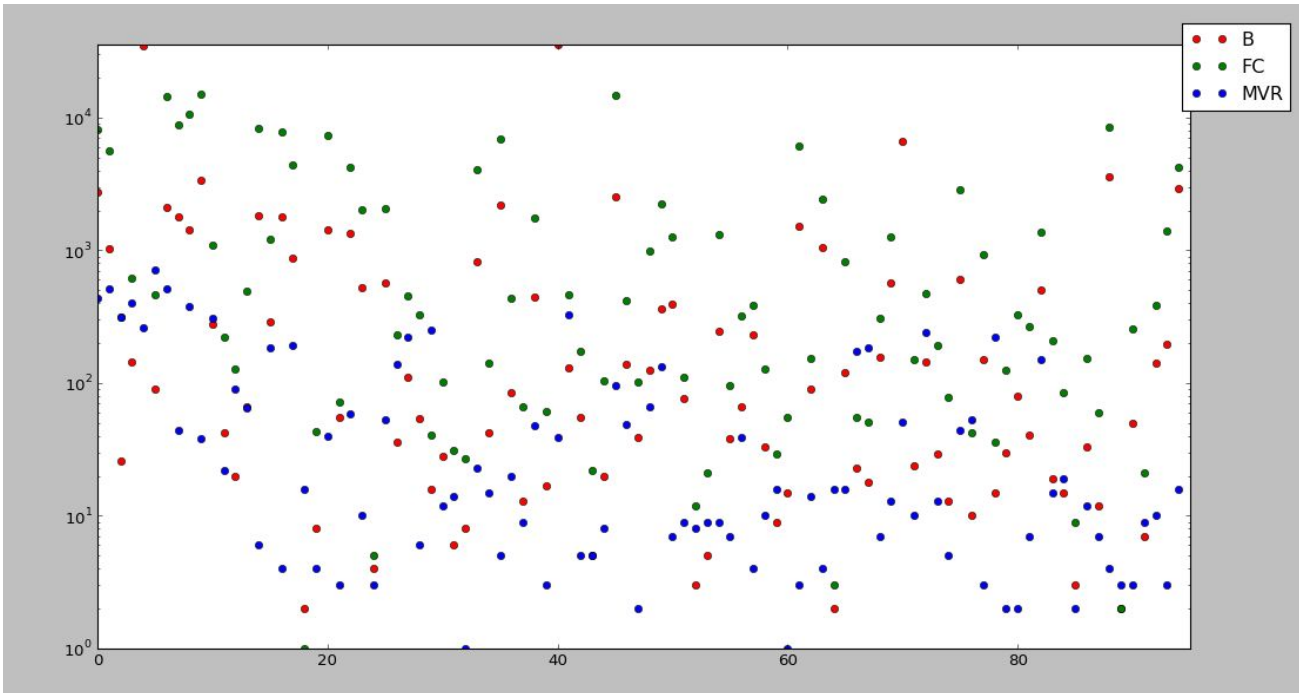

Gráfico 1: Tempo de execução(milissegundos) - N° de jogo no arquivo “entrada.txt”

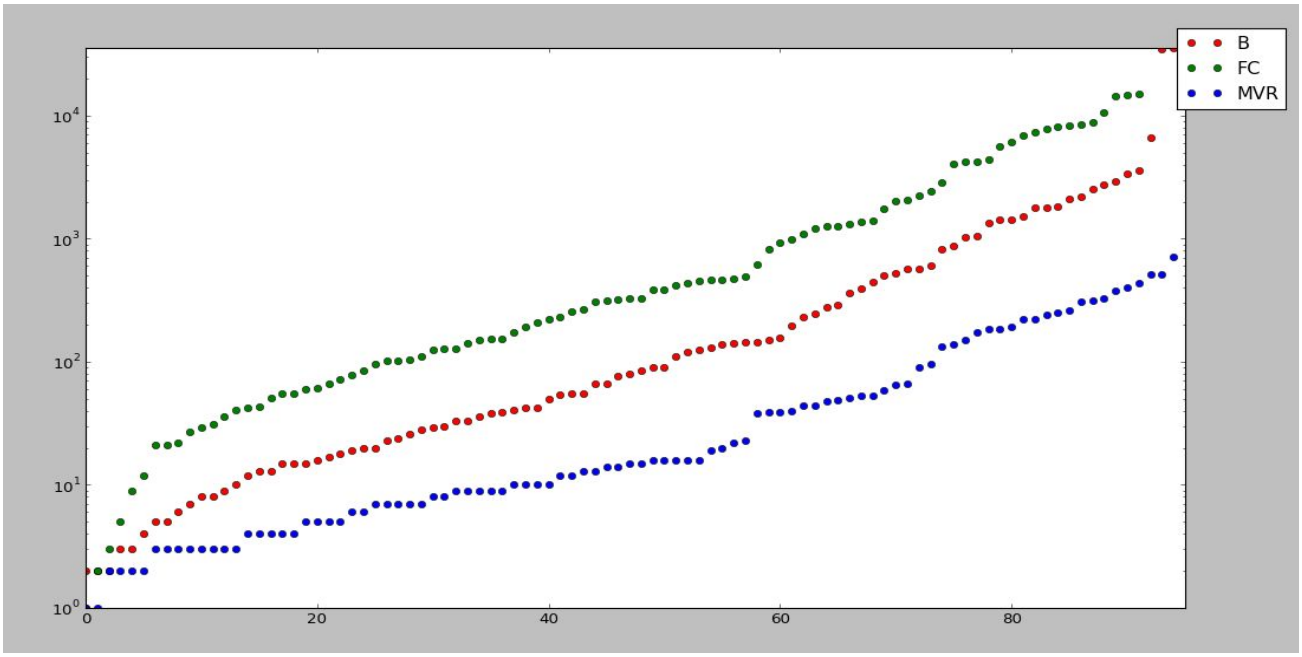

Gráfico 2: Tempo de execução(milissegundos) - Enésimo Jogo ordenado por tempo

b. Número de atribuições:

Os gráficos seguintes mostram o número de atribuições a variáveis dos três algoritmos. Eles foram avaliados com os 95 jogos de Sudoku.

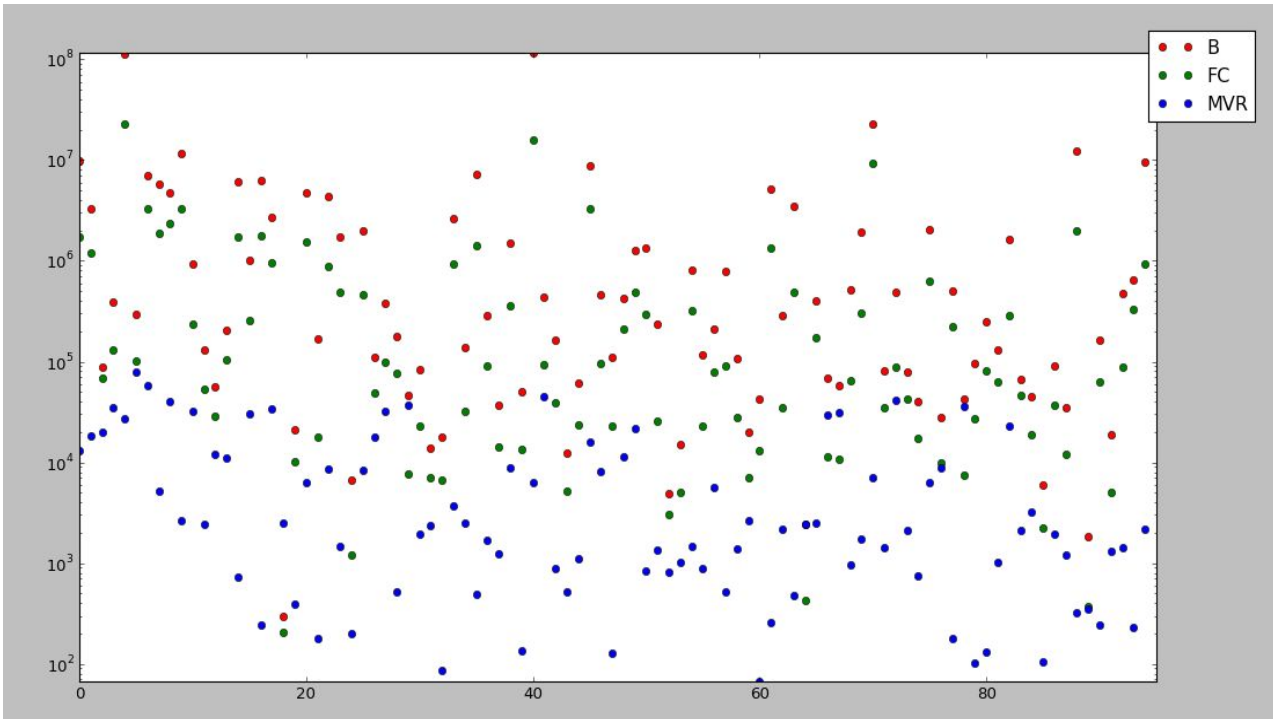

Gráfico 3: Número de atribuições - N° de jogo no arquivo “entrada.txt”

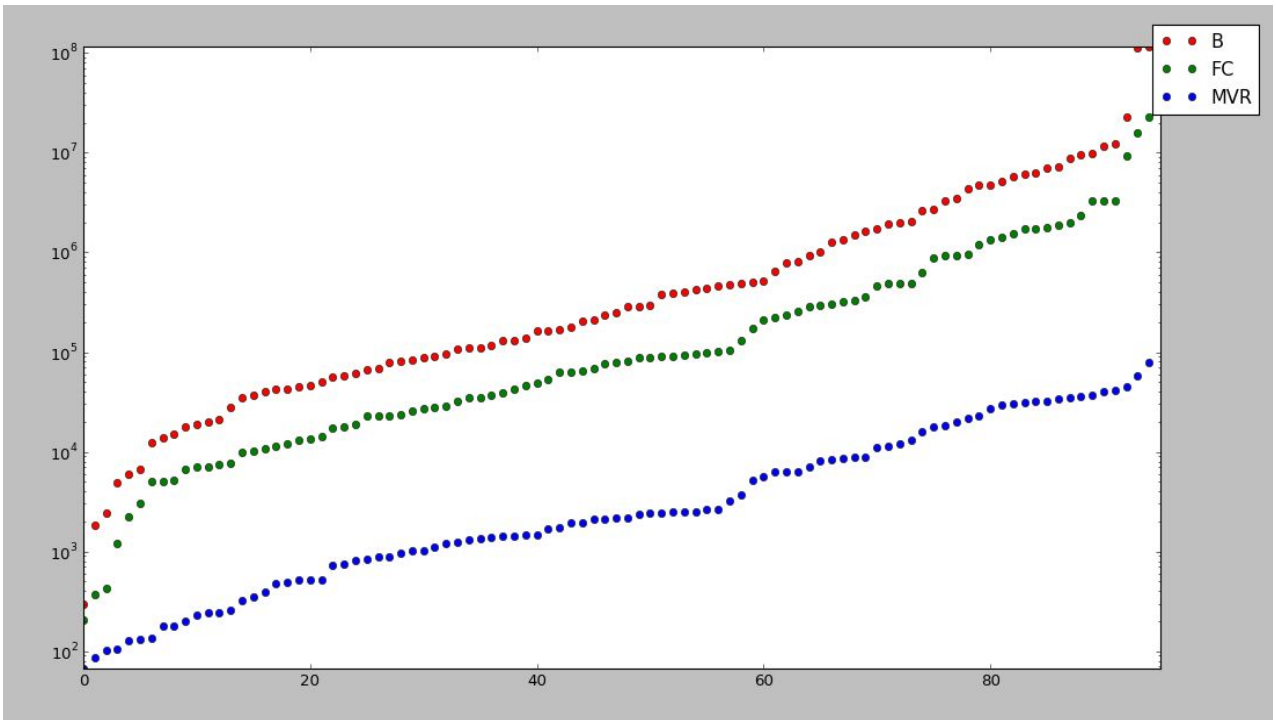

Gráfico 4: Número de atribuições - Enésimo Jogo ordenado por Número de atribuições

3. Conclusão

Na Figura 1 pode-se ver que os pontos azuis (MVR) estão mais abaixo dos pontos verdes (FC) e vermelhos (B). Também pode ser visto que os pontos verdes (FC) são normalmente encontrados em cima dos outros pontos. Isto significa que a heurística MVR leva menos tempo do que as outras técnicas e que FC é a mais lenta. Na Figura 2, é mostrado de forma mais clara os rendimentos de os três algoritmos.

Da mesma forma, na Figura 3, é possível notar que os pontos azuis (MVR) ainda são mantidos abaixo e que o algoritmo de backtracking simple é o que faz mais atribuições. Na Figura 4, pode-se ver melhor o numero de atribuições de cada um dos algoritmos.

Concluiu-se que a heurística MVR com Forward Checking é a melhor, porque resolve o jogo Sudoku em menos tempo e com menos atribuições.