Projeto Backtracking

SCC-0218 Algoritmos Avançados e Aplicações
Prof. Gustavo Batista

Amanda Lutzenberger Ruiz
8532227

Anna Paula Pawlicka Maule
4624650

1. Backtracking Simples

Começando da posição (0,0) na matriz (Sudoku) , verifica se existe algum número entre 1 e 9 atribuido a esta coordenada. Caso exista, chama-se novamente esta função, mas para a coordenada seguinte (recur_backtracking(grid, pos+1, steps)). Caso o valor nessa posição seja ZERO, verifica-se quais dos valores entre 1 e 9 pode ser atribuido a ela. A verificação de validade são as regras do Sudoku, ou seja, um dado número não pode se repetir na mesma coluna, linha, e no sub quadrado 3x3 a que ele pertence. Ao achar um número válido para a coordenada, atribui-se a ela este valor. Então, a função é novamente chamada dentro dela mesma (recursivamente).

O critério de parada da recursão é que todos os elementos estejam preenchidos, ou não exista número válido para a coordenada que esta sendo processada. No caso de não existir elementos válidos, retorna-se um passo na recursão, atribuindo ZERO a coordenada anterior e achando um próximo número válido pra ela. Este processo será repetido até todas as coordenadas estarem preenchidas com valor diferente de ZERO.

Esta implementação é significativamente mais devagar do que as próximas, as quais utilizam eurísticas. O único caso em que o backtracking simples rodou mais rápido que a verificação adiante foi em que toda a matriz estava zerada. Nos outros casos o backtracking foi de 10 a 200 vezes mais lento nos testes realizados.

Com o seguinte exemplo o código demora 9727396 interações para chegar ao sudoku resolvido:

4 0 0 0 0 0 8 0 5	4 1 7 3 6 9 8 2 5
0 3 0 0 0 0 0 0 0	6 3 2 1 5 8 9 4 7
0 0 0 7 0 0 0 0 0	9 5 8 7 2 4 3 1 6
0 2 0 0 0 0 0 6 0	8 2 5 4 3 7 1 6 9
0 0 0 0 8 0 4 0 0	7 9 1 5 8 6 4 3 2
0 0 0 0 1 0 0 0 0	3 4 6 9 1 2 7 5 8
0 0 0 6 0 3 0 7 0	2 8 9 6 4 3 5 7 1
5 0 0 2 0 0 0 0 0	5 7 3 2 9 1 6 8 4
1 0 4 0 0 0 0 0 0	1 6 4 8 7 5 2 9 3

Pseudo-código:

```
backtracking(grid, pos)
      if (pos ==81)
            return 1
end-if

if(grid[pos]!=ZERO)
      backtracking(grid, pos++)
end-if

for( i = ZERO to i = 9)
      if (verify_if_valid(grid, pos, i))
            grid[pos] = i;
            if(backtracking(grid, pos++))
                  return 1;
            end-if
            grid[pos]=ZERO
      end-if
end for

return ZERO
```

2. *Backtracking com Verificação Adiante*

Nesta implementação é utilizado o backtracking assim como o que foi descrito acima, porém utilizando a eurística “verificacao adiante”. Isto significa que quando uma coordenada recebe um valor válido, verifica se alguma das coordenadas da mesma linha, coluna, e sub quadrado 3x3 ficou sem possibilidades válidas. Caso isso ocorra, volta-se na recursão e se procura por outro número válido. Essa verificação serve pra diminuir o número de chamadas recursivas e acelerar a resolução do problema. Nesta implementação, é utilizada uma lista, para cada posição (coordenada) no sudoku, para guardar todos os números que são válidos naquela posição, dessa forma conseguimos verificar as possibilidades das colunas, linhas e quadrados 3x3 sem a necessidade de recalcular as possibilidades do Sudoku inteiro a cada interação. Dessa forma, apenas manipulamos as listas correspondentes a 27 coordenadas e não todas elas (81).

Esta implementacao do V.A. demora 4652288 interações para o mesmo exemplo.

3. Backtracking com Verificação Adiante e Mínimos Valores Remanescentes

Esta terceira implementação utiliza duas eurísticas para melhorar o Backtraking. A primeira é o mínimo valor remanescente (MVR), que consiste em verificar qual posição tem o menor número de possibilidades válidas para aquela coordenada. Ou seja, ela simula o que um humano normalmente faz, procurando preencher primeiro a posição com menor número de possibilidades. Conforme uma posição é preenchida, busca-se a próxima menor, seguindo em ordem crescente de possibilidades. Assim, enquanto houver posições com apenas 1 possibilidade, o backtracking não é feito, só sendo utilizado quando há mais que isso.

Assim que uma posição é preenchida, é utilizada a verificação adiante, que como mencionado anteriormente, ela é responsável por verificar que nenhuma das coordenadas da mesma coluna, linha, e quadrado 3x3 fiquem sem nenhuma possibilidade. Se nenhuma posição ficar sem nenhuma possibilidade, as listas de possibilidades são atualizadas, e é chamada a função novamente, que em seu início calcula a próxima posição de menor número de possibilidades. Esta implementação com duas eurísticas tem o processamento mais rápido entre as implementadas.

Esta implementacao do M.V.R. + V.A. demora 432 interações com o exemplo a cima.