Universidade Federal de São Carlos
Bacharelado em Ciência da Computação
Disciplina de Inteligência Artificial

Aplicação de Estratégias de Busca na Resolução de jogos de Sudoku

Professor: Ricardo Cerri
Otavio Cesar Toma da Silva 726576
Rafael Sales Pavarina 726583
Rene Ferrante Neto 726587

São Carlos
16 de Novembro de 2018

SudokuAutoSolver

Solucionador de Sudoku usando estratégias de busca vistas na aula de Inteligência Artificial.

O projeto foi desenvolvido em javascript com o auxílio da biblioteca para desenhos p5.js.

Para executá-lo basta entrar na pasta *src* e abrir o arquivo *sudoku.html* no seu navegador.

É possível preencher a matriz do sudoku inserindo números pelo teclado, e mover-se a célula selecionada com as setinhas. Nota-se que não é possível preencher o sudoku de forma a quebrar sua regra de colocação de números.

Há dois botões, um para resolver o Sudoku por meio de busca cega, e outro por meio de informada. A partir do momento em que qualquer um é clicado não é possível mais inserir valores na matriz e os valores inseridos (não modificáveis) tem sua célula com fundo cinza.

Em ambas as buscas se contabiliza o tempo que decorreu desde seu início até o seu término, podendo este ser visualizado ao final da execução no log do console do navegador; se utiliza uma matriz fixa para ser desenhada na tela (matrizDesenho); e se separa as matrizes possíveis em categorias, sendo estas pertencentes ao array *explorados* caso já tenha-se passado por elas na busca e *fronteira*, para as próximas a serem exploradas.

Busca Cega - Busca em Profundidade

A busca cega implementada é a de DFS (Depth First Search), ou Busca em Profundidade.

A busca em profundidade implementada busca a primeira célula em branco e coloca em uma pilha todas as matrizes possíveis de acordo com os números que podem ser colocados nesta célula (fronteira), prosseguindo-se então para um próximo estado com uma dessas possíveis matrizes e voltando-se a buscar o primeiro espaço em branco. Ao não encontrar possibilidades de números e um espaço em branco ocorre o bracktracking, pois se retira uma das matrizes acumuladas na fronteira e a coloca como estado atual.

Busca Informada - A*

A busca A* implementada se inicia inserindo o estado inicial na fronteira e, a cada passo, é escolhido o melhor estado da fronteira para ser expandido (por isso a fronteira é sempre reordenada em cada iteração). Essa ordenação é feita baseando-se em duas restrições: a primeira é o custo de cada estado e a segunda é uma heurística definida para detectar estados para os quais não compensa continuar a busca.

A função de custo utilizada para cada estado foi definida como sendo a quantidade de valores possíveis que podem ser colocados na célula da matriz que possui a menor quantidade de valores possíveis para serem inseridos nela, enquanto que a heurística definida faz com que estados "impossíveis" não sejam inseridos na fronteira. Um estado é definido como impossível se a sua matriz contém alguma célula vazia que não consiga receber nenhum valor em um movimento válido do sudoku.

A busca A* implementada obteve resultados positivos, encontrando a meta (matriz com 81 elementos) mais rapidamente do que a busca cega em todos os casos testados.

Código Fonte

```
let width, height, x, y, qtdX, qtdY, squareSize;
let selectedCell = [0, 0];
let buscando = false;
let tipoBusca;
let bntCega, btnInformada;

let delay = 1; // velocidade da busca

let matrizDesenho = [
  [0, 0, 0, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0, 0],
```

```
  [0, 0, 0, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0, 0]
];

/*

Algumas matrizes para testar:

  Jogos considerados fáceis

  [4, 3, 5, 0, 0, 9, 7, 8, 1],
  [0, 0, 2, 5, 7, 1, 4, 0, 3],
  [1, 9, 7, 8, 3, 4, 0, 6, 2],
  [8, 2, 6, 1, 9, 5, 3, 4, 7],
  [3, 7, 0, 0, 8, 2, 0, 1, 5],
  [9, 5, 1, 7, 4, 3, 6, 2, 8],
  [5, 1, 9, 3, 2, 6, 8, 7, 4],
  [2, 4, 8, 9, 5, 7, 1, 3, 0],
  [0, 6, 0, 4, 0, 8, 2, 5, 9]

  [0, 0, 0, 2, 6, 0, 7, 0, 1],
  [6, 8, 0, 0, 7, 0, 0, 9, 0],
  [1, 9, 0, 0, 0, 4, 5, 0, 0],
  [8, 2, 0, 1, 0, 0, 0, 4, 0],
  [0, 0, 4, 6, 0, 2, 9, 0, 0],
  [0, 5, 0, 0, 0, 3, 0, 2, 8],
  [0, 0, 9, 3, 0, 0, 0, 7, 4],
  [0, 4, 0, 0, 5, 0, 0, 3, 6],
  [7, 0, 3, 0, 1, 8, 0, 0, 0]

  Jogo considerado intermediário

  [0, 2, 0, 6, 0, 8, 0, 0, 0],
  [5, 8, 0, 0, 0, 9, 7, 0, 0],
  [0, 0, 0, 0, 4, 0, 0, 0, 0],
  [3, 7, 0, 0, 0, 0, 5, 0, 0],
  [6, 0, 0, 0, 0, 0, 0, 0, 4],
```

```
  [0, 0, 8, 0, 0, 0, 0, 1, 3],
  [0, 0, 0, 0, 2, 0, 0, 0, 0],
  [0, 0, 9, 8, 0, 0, 0, 3, 6],
  [0, 0, 0, 3, 0, 6, 0, 9, 0]

  Jogo considerado difícil

  [0, 0, 0, 6, 0, 0, 4, 0, 0],
  [7, 0, 0, 0, 0, 3, 6, 0, 0],
  [0, 0, 0, 0, 9, 1, 0, 8, 0],
  [0, 0, 0, 0, 0, 0, 0, 0, 0],
  [0, 5, 0, 1, 8, 0, 0, 0, 3],
  [0, 0, 0, 3, 0, 6, 0, 4, 5],
  [0, 4, 0, 2, 0, 0, 0, 6, 0],
  [9, 0, 3, 0, 0, 0, 0, 0, 0],
  [0, 2, 0, 0, 0, 0, 1, 0, 0]

*/

let editavel = null;
let fronteira = null;
let explorados = null;

let tempoInicial = 0;

function buscaCegaInit(){
  tipoBusca = 'cega';
  buscaInit();
}

function buscaInformadaInit(){
  tipoBusca = 'informada';
  buscaInit();
}

function buscaInit(){
  tempoInicial = new Date().getTime();
  buscando = true;
```

```
  btnCega.remove();
  btnInformada.remove();
  editavel = [];
  for(let i = 0; i < qtdX; i++){
    editavel.push([]);
    for(let j = 0; j < qtdY; j++){
      editavel[i].push(matrizDesenho[i][j] === 0);
    }
  }
  concluido = false;
  fronteira = [];
  explorados = [];
  inicial = copiaMatriz(matrizDesenho);
  fronteira.push(inicial);
  buscaTimer = setInterval(buscaLoop, delay);
}

function buscaLoop(){
  if(fronteira.length == 0){ // busca falhou
    clearInterval(buscaTimer);
    return;
  }
  let atual = fronteira.pop();
  matrizDesenho = copiaMatriz(atual);
  explorados.push(atual);
  if(qtdElementos(atual) === 81){ // meta
    clearInterval(buscaTimer);
    console.log('Tempo total: ' + (new Date().getTime() - tempoInicial)/1000 + 's
    return;
  }
  if(tipoBusca == 'cega'){
    let sucessores = getSucessoresCega(atual);
    for(let i = 0; i < sucessores.length; i++){
        fronteira.push(sucessores[i]);
    }
  }
  else if(tipoBusca == 'informada'){
    let sucessores = getSucessoresInformada(atual);
```

```
    for(let i = 0; i < sucessores.length; i++){
      fronteira.push(sucessores[i]);
    }
    fronteira.sort(function(a,b){ return b.custo - a.custo});
  }
}

// retorna lista de matrizes sucessoras
function getSucessoresCega(matriz){
  let sucessores = [];
  let [i, j] = getFirstBlank(matriz);
  for(let k = 9; k >= 1; k--){
    if(matriz[i][j] == 0 && editavel[i][j] && movimentoValido(matriz, i, j, k)){
      let suc = copiaMatriz(matriz);
      suc[i][j] = k;
      if(!pertence(explorados, suc)){
        sucessores.push(suc);
      }
    }
  }
  return sucessores;
}

// atualiza custo se encontrado um sucessor que já pertence a fronteira com cust
function getSucessoresInformada(matriz){
  let sucessores = [];
  let [i, j] = getFirstBlank(matriz);
  for(let k = 9; k >= 1; k--){
    if(editavel[i][j] && movimentoValido(matriz, i, j, k)){
      let suc = copiaMatriz(matriz);
      suc[i][j] = k;
      if(!impossivel(suc) && !pertence(explorados, suc)){
        suc.custo = c(suc);
        sucessores.push(suc);
      }
    }
  }
  return sucessores;
```

```
}

// função de custo
function c(m){
  let r = 9;
  for(let i = 0; i < qtdX; i++){
    for(let j = 0; j < qtdY; j++){
      let v = {};
      for(let k = 1; k <= 9; k++) v[k] = 0;
      for(let a = 0; a < qtdX; a++) v[m[i][a]]++;
      for(let a = 0; a < qtdY; a++) v[m[a][j]]++;
      let I = Math.trunc(i/3)*3;
      let J = Math.trunc(j/3)*3;
      for(let a = I; a < I+3; a++)
        for(let b = J; b < J+3; b++) v[m[a][b]]++;
      let count = 0;
      for(let k = 1; k <= 9; k++) if(v[k]) count++;
      r = Math.min(r, count);
    }
  }
  return 9-r;
}

function impossivel(m){
  for(let i = 0; i < qtdX; i++){
    for(let j = 0; j < qtdY; j++){
      if(m[i][j] === 0){
        if(numerosPossiveis(m, i, j).length === 0) return true;
      }
    }
  }
  return false;
}

function numerosPossiveis(m, i, j){
  let lista = [], existe = {};
  for(let k = 1; k <= 9; k++) existe[k] = false;
```

```
    // linha
    for(let k = 1; k <= 9; k++){
      for(let a = 0; a < qtdX; a++)
        if(m[i][a] === k) existe[k] = true;

      // coluna
      for(let a = 0; a < qtdY; a++)
        if(m[a][j] === k) existe[k] = true;

      // quadrado
      let I = Math.trunc(i/3)*3;
      let J = Math.trunc(j/3)*3;
      for(let a = I; a < I+3; a++)
        for(let b = J; b < J+3; b++)
          if(m[a][b] === k) existe[k] = true;
    }
    for(let k = 1; k <= 9; k++) if(!existe[k]) lista.push(k);
    return lista;
}

// quantidade de celulas preenchidas de um tabuleiro
function qtdElementos(matriz){
    let count = 0;
    for(let i = 0; i < qtdX; i++){
      for(let j = 0; j < qtdY; j++){
        if(matriz[i][j] !== 0)
          count++;
      }
    }
    return count;
}

function getFirstBlank(matriz){
    for(let i = 0; i < qtdX; i++){
      for(let j = 0; j < qtdY; j++) {
        if(matriz[i][j] === 0){
          return [i, j];
        }
```

```
    }
  }
  return [-1, -1];
}

// verifica se pode colocar o numero k na posicao m[i][j]
function movimentoValido(m, i, j, k){
  if(k === 0) return true;

  // linha
  for(let a = 0; a < qtdX; a++)
    if(m[i][a] === k) return false;

  // coluna
  for(let a = 0; a < qtdY; a++)
    if(m[a][j] === k) return false;

  // quadrado
  let I = Math.trunc(i/3)*3;
  let J = Math.trunc(j/3)*3;
  for(let a = I; a < I+3; a++)
    for(let b = J; b < J+3; b++)
      if(m[a][b] === k) return false;

  return true;
}

// verifica se uma matriz pertence a uma lista
function pertence(lista, matriz){
  for(let i = 0; i < lista.length; i++){
    if(matrizIgual(lista[i], matriz))
      return true;
  }
  return false;
}

// compara matrizes
function matrizIgual(a, b){
```

```
  for(let i = 0; i < qtdX; i++){
    for(let j = 0; j < qtdY; j++){
      if(a[i][j] !== b[i][j])
        return false;
    }
  }
  return true;
}

// retorna a copia de uma matriz
function copiaMatriz(m){
  let c = [];
  for(let i = 0; i < qtdX; i++){
    c.push([]);
    for(let j = 0; j < qtdY; j++){
      c[i].push(m[i][j]);
    }
  }
  return c;
}

function setup(){
  width = windowWidth*0.97;
  height = windowHeight*0.97;
  canvas = createCanvas(width, height);
  qtdX = 9; // quantity of squares in row
  qtdY = 9; // quantity of squares in column
  squareSize = 50;
  x = width/2 - qtdX/2*squareSize;
  y = height/2 - qtdY/2*squareSize;

  btnCega = createButton('Busca em profundidade');
  btnCega.position(width/2-btnCega.size().width/2+10, 05);
  btnCega.mousePressed(buscaCegaInit);

  btnInformada = createButton('Busca A*');
  btnInformada.position(width/2-btnInformada.size().width/2+10,40);
  btnInformada.mousePressed(buscaInformadaInit);
```

```
}

function drawGrid(){

  // draw qtdX + 1 vertical lines
  for(let i = 0; i <= qtdX; i++){
    if(i % 3 == 0) strokeWeight(3);
    line(x + i*squareSize, y, x + i*squareSize, y+squareSize*qtdY);
    strokeWeight(1);
  }

  // draw qtdY + 1 horizontal lines
  for(let i = 0; i <= qtdY; i++){
    if(i % 3 == 0) strokeWeight(3);
    line(x, y + i*squareSize, x+squareSize*qtdX, y + i*squareSize);
    strokeWeight(1);
  }
}

function drawNumber(i, j, n){
  textSize(30);
  if(n != 0)
    text(n, x + j*squareSize + 16, y + i*squareSize + 38);
}

// fill sudoku grid with numbers
function fillGrid(){
  for(let i = 0; i < qtdX; i++){
    for(let j = 0; j < qtdY; j++){
      drawNumber(i, j, matrizDesenho[i][j]);
    }
  }
}

function fillSquare(i, j, squareColor){
  push();
  fill(squareColor);
  rect(x+j*squareSize, y+i*squareSize, squareSize, squareSize);
```

```
  pop();
}

function draw(){

  clear();

  if(!buscando){
    // pinta a célula selecionada
    let [i, j] = selectedCell;
    fillSquare(i, j, color(0, 0, 255, 100));
  }
  else{
    // pinta as células não editáveis
    for(let i = 0; i < qtdX; i++){
      for(let j = 0; j < qtdY; j++){
        if(!editavel[i][j])
          fillSquare(i, j, color(162, 172, 188, 130));
      }
    }
  }

  for(let i = 0; i < qtdX; i++){
    for(let j = 0; j < qtdY; j++){
      drawNumber(i, j, matrizDesenho[i][j]);
    }
  }

  drawGrid();

  fillGrid();

}

function keyPressed() {

  if(buscando){
    return;
```

```
  }

  let[i, j] = selectedCell;
  if(keyCode === LEFT_ARROW) {
    if(j > 0) j--;
  }
  else if(keyCode === RIGHT_ARROW) {
    if(j+1 < qtdX) j++;
  }
  else if(keyCode === UP_ARROW) {
    if(i > 0) i--;
  }
  else if(keyCode === DOWN_ARROW) {
    if(i+1 < qtdY) i++;
  }
  selectedCell = [i, j];
}

function keyTyped() {

  if(buscando){
    return;
  }

  let [i, j] = selectedCell;
  if('0' <= key && key <= '9' && movimentoValido(matrizDesenho, i, j, parseInt(key
    matrizDesenho[i][j] = parseInt(key);
  }
}
```