Projeto I - Buscas

CTC- 17 Inteligência Artificial

Prof. Paulo André Castro

Shelly Leal

**1. Objetivo**

Implementar algoritmos de busca para resolução de problemas, como de encontrar o menor caminho entre duas cidades e o problema de solução do objetivo de vitória em um jogo contra um usuário humano, como o “jogo da velha”. Para o primeiro problema de menor distância entre duas cidades foi utilizada a linguagem Java, enquanto para a solução do “jogo da velha”, a linguagem C.

**2. Descrição**

* 1. **Encontre o menor caminho entre as cidades 203 e 600 do Uruguai (arquivo Uruguay.csv). O arquivo tem os seguintes campos: ID da cidade, coordenada x, coordenada y e lista de adjacências. Cada lista de adjacências da cidade C contém os IDs das cidades para as quais há ligação a partir da cidade C. A distância entre as cidades pode ser calculada a partir das coordenadas cartesianas (x,y) disponibilizadas no arquivo Uruguay.csv. Utilize os algoritmos greedy e A\* para fazer o trabalho, compare os resultados e explicite as funções de avaliação usadas heurística e no relatório.**

O algoritmo greedy utilizado foi baseado no Best First Search, o qual se caracteriza por ser um tipo de algorimo greedy não otimizado. Este utiliza uma função que incorpora custo aos nós do grafo de forma a estimar o melhor caminho para o estado objetivo.

O pseudo-código base para a implementação foi como a seguir*: (fonte: ime.usp.br)*

**função** BestFirstSearch (problema, estratégia) **devolve** uma solução,

ou falha

inicializa a árvore de busca com o estado inicial do *problema*

**laço faça**

se não há mais candidatos para expandir **então devolve** falha

escolha o primeiro nó da lista de nós terminais

**se** o nó contém um estado gol **então devolve** a solução

**senão** expanda o nó de acordo com a *estratégia*

**fim**

A heurística utilizada se baseia em uma função h para selecionar os nós a serem expandidos.

A implementação parcial do programa em Java – correspondente à parte da fila a ser percorrida - pode ser observada a seguir:

**while**(!sucessores.isEmpty()){

actualCity = sucessores.removeFirst();

**if**(actualCity.id == dest){

**int** index = dest;

**int** counter = 0;

**while**(index != start){

counter++;

cost = cost+actualCity.costTo;

index = actualCity.fromID;

actualCity = cidades.get(index);

}

System.***out***.println("Custo = "+cost);

System.***out***.println("Contador = "+counter);

**return** cost;

}

**else**{

vizIDs = actualCity.adjList;

viz.clear();

**for**(**int** i = 0; i < actualCity.adjList.size(); i++){

ID actViz = cidades.get(vizIDs.get(i));

viz.add(actViz);

}

**for**(**int** i = 0; i < actualCity.adjList.size(); i++){

**int** auxID = viz.get(i).id;

**if**(!visited[auxID]){

ID actViz = viz.get(i);

actViz.costTo = distBetween(actViz.id, actualCity.id);

actViz.h = distBetween(actViz.id, dest);

actViz.fromID = actualCity.id;

visited[auxID] = **true**;

sucessores.add(actViz);

}

}

Collections.*sort*(sucessores);

Para a etapa do algoritmo A\*, versão otimizada em relação ao greedy, baseou-se na mesma ideia do algoritmo de Dijkstra de cálculo de menor caminho possível. A diferenção entre os dois está apenas na fórmula heurística a ser utilizada, sendo que no segundo, é nula, e para o A\*, a função heurística que foi utilizada para a estimativa do menor caminho foi a distância do nó atual no percurso até o nó do objetivo final.

Portanto, a função de avaliação para o algoritmo A\* equivale à uma função f(n) = g(n) + h(n), onde f(n) é o custo até o objetivo via n, g(n) é o custo para alcançar n e o h(n) o custo mínimo de n ao objetivo. No caso do Dijkstra, h(n) seria igual à 0.

**public** **static** **void** computePaths(Vertex source)

{

source.minDistance = 0.;

PriorityQueue<Vertex> vertexQueue = **new** PriorityQueue<Vertex>();

vertexQueue.add(source);

**while** (!vertexQueue.isEmpty()) {

Vertex u = vertexQueue.poll();

// Visit each edge exiting u

**for** (Edge e : u.adjacencies)

{ **if**(e==**null**)

**break**;

Vertex v = e.target;

**double** weight = e.weight;

**double** distanceThroughU = u.minDistance + weight;

**double** d = distanceThroughU;

**if** (d < v.minDistance) {

vertexQueue.remove(v);

v.minDistance = distanceThroughU ;

v.previous = u;

vertexQueue.add(v);

}

}

}

}

**public** **static** List<Vertex> getShortestPathTo(Vertex target)

{

List<Vertex> path = **new** ArrayList<Vertex>();

**for** (Vertex vertex = target; vertex != **null**; vertex = vertex.previous)

path.add(vertex);

Collections.*reverse*(path);

**return** path;

}

**2.2 Crie um agente capaz de jogar o tic tac toe (jogo da velha) contra um usuário humano. A interface gráfica pode ser bastante simples inclusive em modo texto, porém deve permitir ao usuário perceber qual a situação atual do jogo e selecionar sua próxima jogada. O usuário humano é o "X" e sempre dá o primeiro lance.**

O programa implementado utilizando a ideia do minimax buscava eliminar dinamicamente as jogadas que gerassem perda no final para o computador. A heurística utilizada foi de forma que as jogadas do computador gerassem a pior possibilidade para o usuário (menos chance da pessoa ganhar).

A seguir um trecho implementado em linguagem C:

int minimax(int board[9], int player) {

**// turno do jogador**

**int winner = win(board);**

**if(winner != 0) return winner\*player;**

**int move = -1;**

**int score = -2;//descarta movimentos que geram perda do jogo**

**int i;**

**for(i = 0; i < 9; ++i) {**

**if(board[i] == 0) {**

**board[i] = player;**

**int thisScore = -minimax(board, player\*-1);**

**if(thisScore > score) {**

**score = thisScore;**

**move = i;**

**}//escolha a jogada pior pro jogador**

**board[i] = 0;**

**}**

**}**

**if(move == -1) return 0;**

**return score;**

**}**

O programa foi simplificado para que as jogadas do usuário fossem resumidas a uma seleção das casas do “jogo da velha”, entre 0 e 8. O usuário sempre usava o X, mas poderia escolher entre assumir a primeira jogada, ou esperar a máquina jogar primeiro.

**3. Resultados**

**3.1.** O menor caminho encontrado entre as cidades 203 e 600 pode ser visto a seguir:

Algoritmo Greedy (BestFirstSearch)

Custo = 121.7399864918759

Contador = 116

Algoritmo A\*

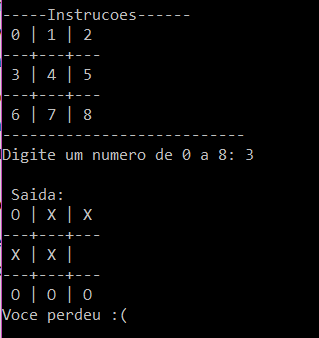
Distance to 600.0: 93.5607275848353

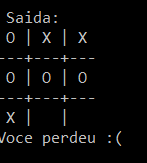
Path: [203.0, 206.0, 211.0, 217.0, 214.0, 218.0, 223.0, 225.0, 230.0, 232.0, 235.0, 239.0, 241.0, 244.0, 246.0, 248.0, 253.0, 257.0, 262.0, 265.0, 268.0, 272.0, 277.0, 280.0, 283.0, 288.0, 293.0, 297.0, 302.0, 307.0, 308.0, 312.0, 317.0, 321.0, 325.0, 330.0, 335.0, 339.0, 342.0, 344.0, 349.0, 354.0, 359.0, 364.0, 365.0, 371.0, 373.0, 378.0, 382.0, 383.0, 388.0, 393.0, 396.0, 398.0, 400.0, 404.0, 408.0, 411.0, 413.0, 419.0, 422.0, 424.0, 426.0, 431.0, 432.0, 435.0, 438.0, 439.0, 442.0, 447.0, 451.0, 454.0, 458.0, 461.0, 466.0, 470.0, 476.0, 478.0, 483.0, 485.0, 490.0, 494.0, 499.0, 500.0, 504.0, 508.0, 513.0, 515.0, 519.0, 523.0, 526.0, 527.0, 530.0, 535.0, 537.0, 542.0, 546.0, 549.0, 554.0, 556.0, 560.0, 563.0, 561.0, 566.0, 569.0, 574.0, 577.0, 581.0, 584.0, 586.0, 590.0, 595.0, 600.0]

Comparativamente, percebe-se que a otimização com a heurística do algoritmo A\* torna o caminho muito mais eficiente (distância de 121.739 para 93.560).

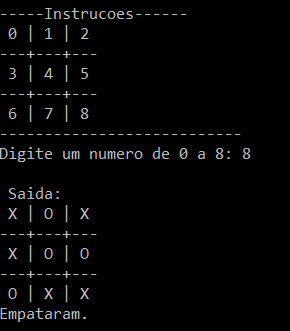
* 1. A seguir se apresentam possíveis resultados das jogadas entre o homem e a máquina:

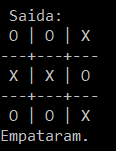
**Perda do usuário**





**Empate**





**Ganhar**

Não tem essa opção, já que o computador sempre busca a opção do grafo que gere perda do usuário ou empate, desde sua primeira jogada.

1. **Conclusão**

Os algoritmos de busca são mais eficientes com o uso de funções heurísticas mais otimizadas, como visto anteriormente.

A implementação foi realizada em Java, porém é possível realizar implementações mais sucintas em linguagem C utilizando-se fila a partir de métodos de pilha (push e pop).

O algoritmo de minimax foi interessante de ser analisado, visto que limita o suficiente as jogadas possíveis do usuário de formar que chegue ao objetivo final. Porém, este algoritmo é mais recomendado para jogos em que a forma de alacançar o objetivo é mais simples de se determinar – o que não ocorre em jogos, por exemplo, como damas ou xadrez.