

Utilizando A^* para o melhor trajeto de um veículo entre bairros



Índice

1. Introdução
2. O problema do Caixeiro Viajante e um programa que dê uma rota entre bairros a um motorista (Mini-Waze)
3. O algoritmo A*
4. Código Prolog e execução

1. Introdução

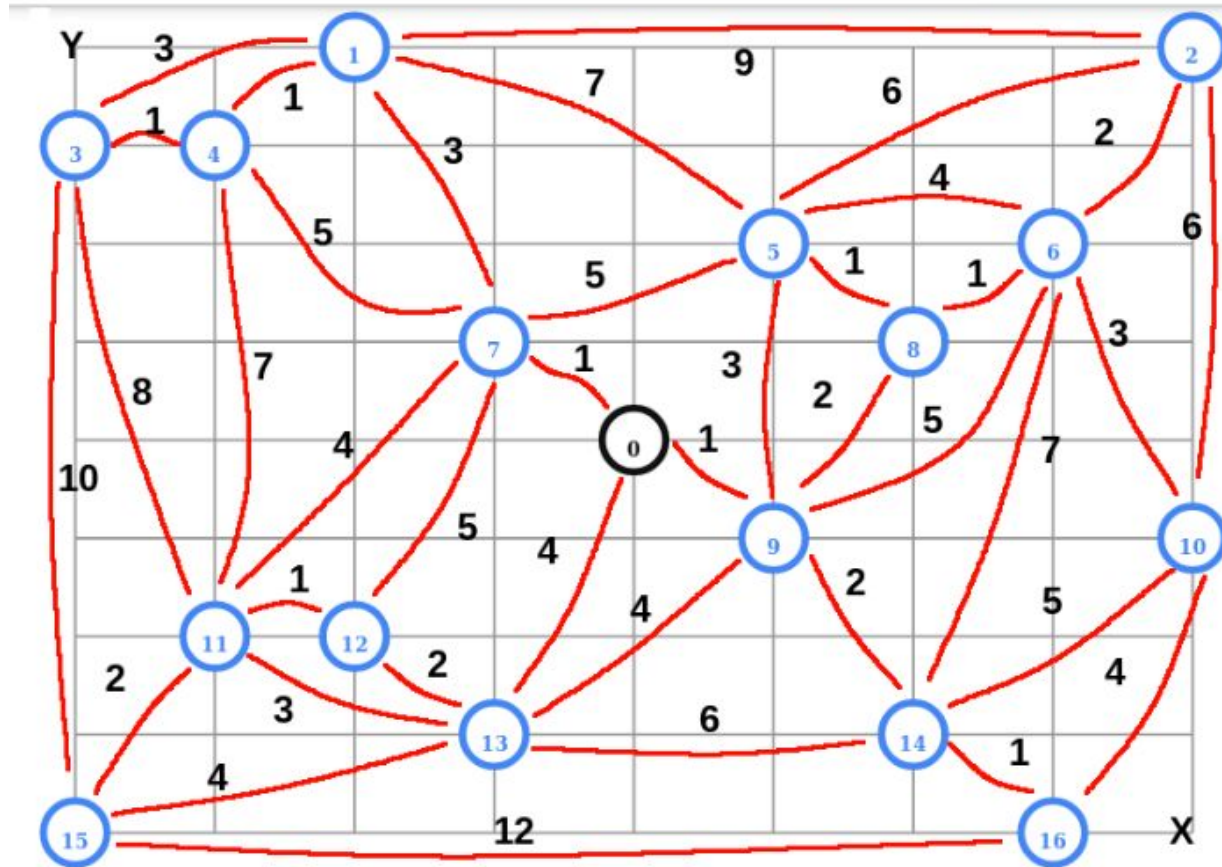
A idéia principal do trabalho é resolver um problema através de busca heurística. Para tal, foram estudados algoritmos básicos de procura e avaliados inúmeros problemas. O problema escolhido é baseado no Problema do Caixeiro Viajante, mas foi remodelado como o de um motorista que pretende percorrer bairros da forma mais eficiente possível, como num aplicativo tipo Waze..

2. O problema do Caixeiro Viajante e um programa que dê uma rota entre bairros a um motorista (Mini-Waze)

O problema do Motorista é uma abstração em cima do problema do Caixeiro Viajante. Para quem não se recorda, o Problema do Caixeiro Viajante é sobre um Caixeiro que sai de uma cidade e deve retornar à cidade de origem passando por 1 cidade de cada vez. No nosso não será necessário retornar à cidade de origem, mas o motorista poderá escolher um trajeto de um bairro a outro a partir do algoritmo A^* , e combinando uma função de avaliação e uma de custo para se chegar à melhor decisão.

No nosso programa cada número equivalerá a um bairro e são usadas as coordenadas X e Y dele e as distâncias de um bairro a outro por uma estrada.

Vale lembrar que as vias percorridas serão de “mão única”. Por exemplo, o trajeto do bairro 1 ao 9 dará um output diferente 9 ao 1, pois terá de fazer outro trajeto de carro.



A figura acima representa através de nós os bairros e seus devidos trajetos.

3. O algoritmo A*

Antes de se falar do algoritmo mencionado, vale ressaltar a motivação a se usar algum tipo de heurística em um problema de busca. Além de otimização de memória, dado que não há a necessidade de se guardar tantos estados, há um ganho em performance pois a procura se torna restrita e não precisa avaliar tantos estados.

O algoritmo A* combina busca em largura com a soma de uma função de avaliação com uma função de custo, onde durante a busca será escolhido o estado onde a soma de ambas as funções seja o menor possível.

$$F = g(n) + h(n)$$

g será o custo para se ir de um nó a outro, ou seja do estado inicial a um nó “target”.

h será a estimativa da distância de um nó a um nó “target”.

O objetivo do nosso programa será sempre o de termos a menor soma para a escolha do estado seguinte.

Em nosso programa a função de custo é dada entre cada bairro e uma função de avaliação é a distância euclidiana, usando coordenadas X e Y, entre os bairros (que são representados com números). O algoritmo A* empregado faz busca em largura e muda de estado de acordo com a menor soma entre a função de custo e de avaliação.

4. Código Prolog e execução

Vale lembrar que as vias percorridas serão de “mão única”. Por exemplo, o trajeto do bairro 1 ao 9 dará um output diferente 9 ao 1, pois terá de fazer outro trajeto de carro.

Para se fazer a execução é necessário usar o SWISH:

<https://swish.swi-prolog.org/>

Deve-se colocar o seguinte código na parte esquerda (lembrando que o código também foi enviado em arquivo separado):

```
% no swish, sem isso aqui embaixo ele não funciona o not  
:- op(900,fy,'not').
```

```
%as coordenadas X e Y são bairros
```

```
bairro(0,4,4).
```

```
bairro(1,2,8).
```

```
bairro(2,8,8).
```

```
bairro(3,0,7).
```

```
bairro(4,1,7).
```

```
bairro(5,5,6).
```

```
bairro(6,7,6).
```

```
bairro(7,3,5).
```

```
bairro(8,6,5).
```

```
bairro(9,5,3).
```

```
bairro(10,8,3).
```

```
bairro(11,1,2).
```

```
bairro(12,2,2).
```

```
bairro(13,3,1).
```

bairro(14,6,1).

bairro(15,0,0).

bairro(16,7,0).

% caminhos dos bairros e distâncias

caminho(0,9,1).

caminho(0,7,1).

caminho(0,13,4).

caminho(1,2,9).

caminho(1,3,3).

caminho(1,4,1).

caminho(1,5,7).

caminho(1,7,3).

caminho(2,5,6).

caminho(2,6,2).

caminho(2,10,6).

caminho(3,4,1).

caminho(3,11,8).

caminho(3,15,10).

caminho(4,7,5).

caminho(4,11,7).

caminho(5,6,4).

caminho(5,7,5).

caminho(5,8,1).

caminho(5,9,3).

caminho(6,8,1).

caminho(6,9,5).

caminho(6,10,3).

caminho(6,14,7).

caminho(7,11,4).

caminho(7,12,5).

caminho(8,9,2).

caminho(9,13,4).

caminho(9,14,2).
caminho(10,14,5).
caminho(10,16,4).
caminho(11,12,1).
caminho(11,13,3).
caminho(11,15,2).
caminho(12,13,2).
caminho(13,14,6).
caminho(13,15,4).
caminho(14,16,1).
caminho(15,16,12).

calcula(Origem,Chegada,Perc,Total):-

estimated(Origem,Chegada,H),

calcula_helper([c(H/0,[Origem])],Chegada,P,Total),

reverse(P,Perc).

calcula_helper(Trajetos,Chegada,Trajeto,Total):-

menor(Trajetos,Menor,Restantes),

proximos(Menor,Chegada,Restantes,Trajeto,Total).

proximos(c(_/Distancia,Trajeto),Chegada,_,Trajeto,Distancia):-

Trajeto=[Chegada|_].

proximos(c(_,[Chegada|_]),Chegada,Restantes,Trajeto,Total):-!

,

calcula_helper(Restantes,Chegada,Trajeto,Total).

proximos(c(_/Distancia,[Ult|T]),Chegada,Trajeto,Trajeto,Total):

-

findall(c(H1/D1,[Z,Ult|T]),next(Ult,T,Z,Distancia,Chegada,H1/D1),Lista),

append(Lista,Trajeto,NovosTrajetos),

calcula_helper(NovosTrajetos,Chegada,Trajeto,Total).

menor([Trajeto|Trajetos],Menor,[Trajeto|Resto]):-

menor(Trajeto,Menor,Resto),

menorWay(Menor,Trajeto),!.

menor([Trajeto|Z],Trajeto,Z).

menorWay(c(H1/D1,_),c(H2/D2,_)):-

C1 is H1+D1, C2 is H2 + D2, C1 < C2.

next(X,T,Y,Distancia,Chegada,H/Dist):-

(caminho(X,Y,Z);caminho(Y,X,Z)),

not member(Y,T),

Dist is Distancia + Z,

estimated(Y,Chegada,H).

estimated(B1,B2,Estimado):-

bairro(B1,X1,Y1),

bairro(B2,X2,Y2),

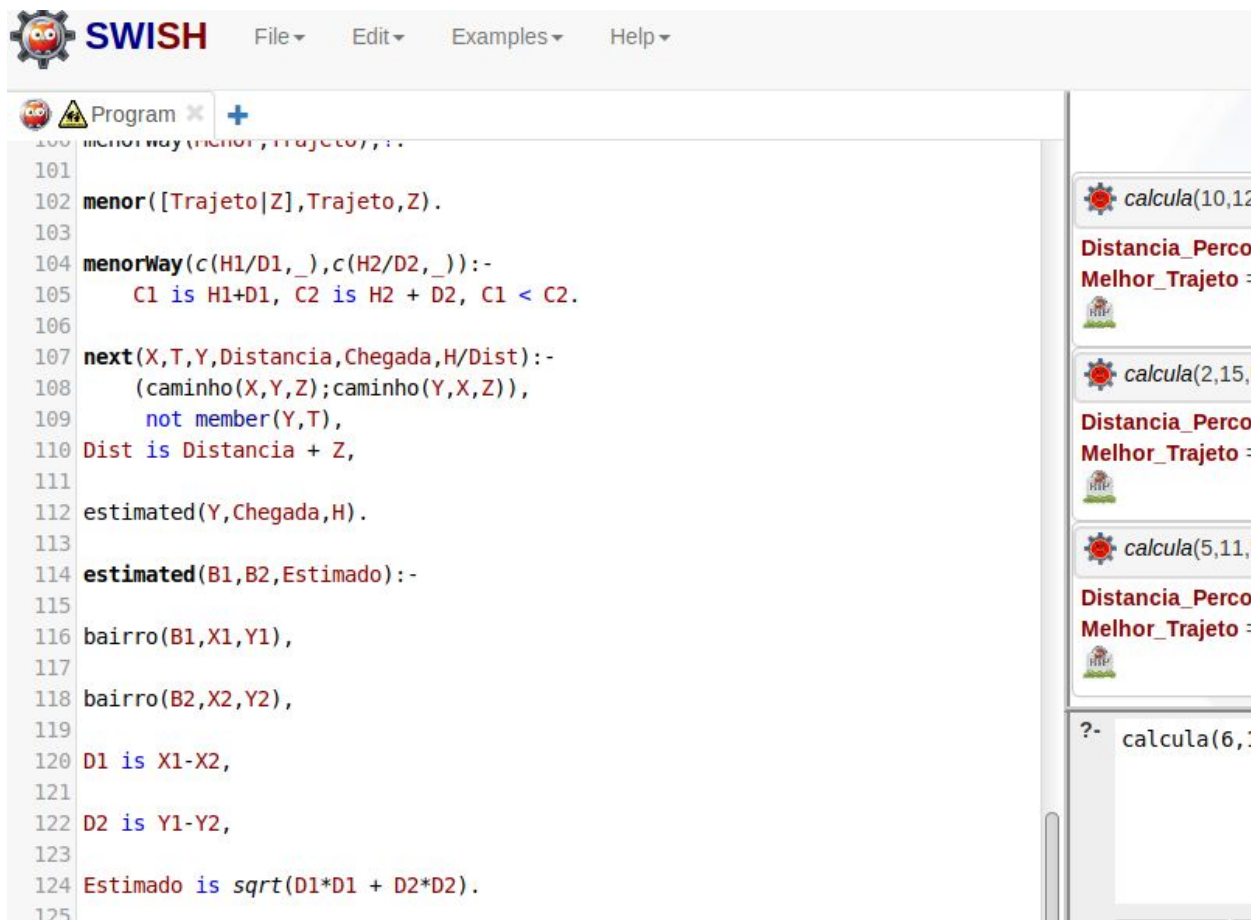
D1 is X1-X2,

D2 is Y1-Y2,

Estimado is $\sqrt{D1^2 + D2^2}$.

% um exemplo a seguir

% calcula(1,12,Melhor_Trajeto,Distância_Percorrida).



The image shows the SWISH Prolog IDE interface. The main window displays a Prolog program with the following code:

```
100 menorWay(menor,Trajeto,Z):-
101
102 menor([Trajeto|Z],Trajeto,Z).
103
104 menorWay(c(H1/D1,_),c(H2/D2,_)):-
105     C1 is H1+D1, C2 is H2 + D2, C1 < C2.
106
107 next(X,T,Y,Distancia,Chegada,H/Dist):-
108     (caminho(X,Y,Z);caminho(Y,X,Z)),
109     not member(Y,T),
110     Dist is Distancia + Z,
111
112 estimated(Y,Chegada,H).
113
114 estimated(B1,B2,Estimado):-
115
116 bairro(B1,X1,Y1),
117
118 bairro(B2,X2,Y2),
119
120 D1 is X1-X2,
121
122 D2 is Y1-Y2,
123
124 Estimado is sqrt(D1*D1 + D2*D2).
125
```

On the right side, there is a list of queries and their results:

- `calcula(10,12)`
- `Distancia_Percorrida`
- `Melhor_Trajeto`
- `calcula(2,15)`
- `Distancia_Percorrida`
- `Melhor_Trajeto`
- `calcula(5,11)`
- `Distancia_Percorrida`
- `Melhor_Trajeto`
- `?- calcula(6, ...)`

e na parte destinada a queries digitar:

`calcula(10,12,Melhor_Trajeto,Distância_Percorrida).`

Também pode-se usar inúmeras variantes, como as dos exemplos a seguir:

`calcula(5,15,Melhor_Trajeto,Distância_Percorrida).`

`calcula(6,11,Melhor_Trajeto,Distância_Percorrida).`

The screenshot shows a Prolog IDE interface. The top bar includes a 'Help' dropdown, a '305 users online' status, a search bar, and system icons. The main area displays a list of queries and their results:

- Query: `calcula(10,12,Melhor_Trajeto,Distancia_Percorrida).`
Results:
`Distancia_Percorrida = 12,`
`Melhor_Trajeto = [10, 6, 8, 9, 13, 12]`
- Query: `calcula(2,15,Melhor_Trajeto,Distancia_Percorrida).`
Results:
`Distancia_Percorrida = 13,`
`Melhor_Trajeto = [2, 6, 8, 9, 13, 15]`
- Query: `calcula(5,11,Melhor_Trajeto,Distancia_Percorrida).`
Results:
`Distancia_Percorrida = 9,`
`Melhor_Trajeto = [5, 7, 11]`

At the bottom, there is a query input field with the text: `?- calcula(6,11,Melhor_Trajeto,Distancia_Percorrida).`

Below the input field are tabs for 'Examples', 'History', and 'Solutions'. On the far right, there is a checkbox labeled 'table results'.

Lembrar que deve-se usar testando a figura já apresentada anteriormente, onde cada número equivale a um bairro:

