Relatório Algoritmia Avançada

2023 / 2024

1180727 - Ruben Martins
1211171 – Pedro Mesquita
1210816 - João Castro
1191831 - Rui Gonçalves
1210913 - Pedro Mendes

ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

Introdução

No âmbito da disciplina de Algoritmia Avançada (ALGAV), este relatório apresenta o desenvolvimento do Sprint B, focando-se no planeamento de trajetória de robôs, especificamente dentro e entre edifícios por corredores e elevadores.

O projeto **ProjetoAlgav.pl**, desenvolvido em Prolog pelo nosso grupo, visa abordar os desafios associados à representação de conhecimento espacial, planeamento de rotas otimizadas e movimentação autônoma de robôs em um ambiente complexo.

O domínio do problema engloba a movimentação de robôs em diferentes pisos de edifícios, que navegam por corredores de ligação, acedem a elevadores e transitam por corredores externos. Além disso, o projeto contempla a movimentação interna dos robôs em um único piso de um edifício, considerando a eficiência e a otimização de trajetórias.

Neste contexto, utilizamos algoritmos clássicos de pesquisa e otimização, tais como Primeiro em Profundidade (Depth-First Search - DFS), Primeiro em Largura (Breadth-First Search - BFS) e o algoritmo A*, adaptados para lidar com a complexidade do ambiente e as especificidades do domínio.

Um aspeto inovador do projeto é a inclusão de movimentos diagonais na movimentação do robô, o que introduz uma camada adicional de complexidade na representação do espaço e no cálculo das rotas.

Além da implementação prática, este relatório inclui um estudo detalhado da complexidade dos algoritmos utilizados, avaliando a viabilidade de encontrar soluções ótimas em diferentes cenários e configurações.

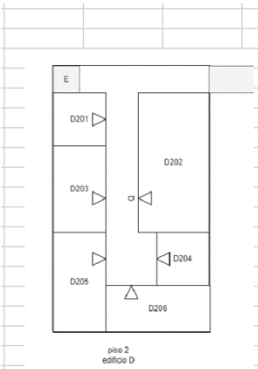
Através deste trabalho, tentamos não apenas desenvolver uma solução técnica robusta para o problema proposto, mas também aprofundar nosso entendimento teórico e prático sobre representação de conhecimento, algoritmos de procura e otimização, e as suas aplicações em contextos de inteligência artificial e robótica.

Representação do Conhecimento do domínio

No contexto do nosso projeto **ProjetoAlgav.pl**, abordamos esta representação com uma estratégia sistemática e detalhada, focando na estruturação espacial de edifícios e na movimentação acessível para os robôs.

Estruturação das matrizes

Inicialmente, o ambiente de cada piso de um edifício foi modelado utilizando matrizes compostas por células representadas por 0s e 1s. Neste modelo, os 0s representam espaços pelos quais o robô pode navegar, enquanto os 1s indicam obstáculos ou áreas inacessíveis. Esta abordagem matricial permite uma representação clara e precisa dos espaços internos dos edifícios, essencial para o planeamento de rotas.



Linha/coluna	1	2	3	4	5	6	7	8	9	10
1	1	1	1	1	1	1	1	1	1	1
2	1	0	0	0	0	0	0	0	0	0
3	1	1	1	1	0	1	1	1	1	1
4	1	0	0	0	0	1	0	0	0	1
5	1	0	0	1	0	1	0	0	0	1
6	1	1	1	1	0	1	0	0	0	1
7	1	0	0	1	0	0	0	0	0	1
8	1	0	0	1	0	1	1	1	1	1
9	1	0	0	0	0	0	0	0	0	1
10	1	1	1	1	0	0	1	0	0	1
11	1	0	0	0	0	1	1	1	1	1
12	1	0	0	1	0	0	0	0	0	1
13	1	0	0	1	0	0	0	0	0	1
14	1	1	1	1	1	1	1	1	1	1

Passagem edificios
Portas salas
Elevador

Figura 1 Exemplo matriz Edifício D Piso 2

```
%Edificio D piso 2
m(d, 2, 1, 1, 1).
m(d, 2, 1, 2, 1).
m(d, 2, 1, 3, 1).
m(d, 2, 1, 4, 1).
m(d, 2, 1, 5, 1).
m(d, 2, 1, 6, 1).
m(d, 2, 1, 7, 1).
m(d, 2, 1, 8, 1).
m(d, 2, 1, 9, 1).
m(d, 2, 1, 10, 1).
```

Figura 2 m(Edifício, Piso, X, Y, Valor) D2

Para incorporar esta estrutura no Prolog, definimos um conjunto de factos representando cada célula da matriz. Cada facto é definido por **m(Edifício, Piso, X, Y, Valor)**, onde **Edifício** e **Piso** especificam a localização do edifício e do piso, **X** e **Y** denotam as coordenadas da célula na matriz, e **Valor** determina se a célula é transitável (0) ou não (1).

Conexão entre pisos

Após estruturar os espaços individuais de cada piso, o próximo passo foi conectar estes diferentes níveis dentro de um mesmo edifício. Para isso, utilizamos ligacao_piso, que é definido por **ligação_piso (Edifício, Celula1, Celula2)**. Por exemplo, ligacao_piso (c, cel (c, 1, 10, 2), cel (c, 2, 11, 2)) cria uma ligação entre o primeiro e o segundo piso do edifício 'C' através do elevador.

```
ligacao_piso(c, cel(c, 1, 10, 2), cel(c, 2, 11, 2)).
ligacao_piso(c, cel(c, 2, 11, 2), cel(c, 3, 12, 1)).
ligacao_piso(c, cel(c, 3, 12, 1), cel(c, 4, 11, 1)).
ligacao_piso(d, cel(d,1,2,2), cel(d,2,2,2)).
ligacao_piso(d, cel(d,2,2,2), cel(d,3,2,2)).
ligacao_piso(a, cel(a,1,2,15), cel(a,2,2,17)).
ligacao_piso(b, cel(b,1,8,14), cel(b,2,6,18)).
ligacao_piso(b, cel(b,2,6,18), cel(b,3,5,16)).
```

Figura 3 Ligações entre pisos

Conexão entre edifícios

Além das conexões internas, também era necessário estabelecer ligações entre diferentes edifícios. Para tal, usamos ligacao_edificio, que é definido por **ligação_edificio (Celula1, Celula2)** que conecta células específicas localizadas nos corredores externos entre edifícios. Por exemplo, ligacao_edificio (cel (c, 2, 13, 1), cel (d, 2, 2, 10)) representa uma conexão entre os edifícios 'C' e 'D'.

```
ligacao_edificio(cel(c, 2, 13, 1), cel(d, 2, 2, 10)).
ligacao_edificio(cel(a, 2, 5, 18), cel(b, 2, 4, 1)).
ligacao_edificio(cel(b,2,7,19), cel(c,3,8,1)).
ligacao_edificio(cel(b,2,8,18), cel(d,3,1,8)).
ligacao_edificio(cel(c,3,14,1), cel(d,3,2,9)).
ligacao_edificio(cel(b,3,7,17), cel(c,4,7,1)).
```

Figura 4 Ligações entre edifícios

Com esta abordagem, conseguimos mapear de forma eficaz o ambiente tridimensional composto por múltiplos edifícios e pisos. A representação detalhada permite que o sistema de navegação do robô interprete o ambiente com precisão, facilitando a geração de trajetórias viáveis e eficientes.

Solução Ótima para o Planeamento de Movimentação entre Pisos

Criação de grafos

Para a criação dos grafos para representar as possíveis rotas de movimentação dentro de cada piso, foi utilizado o predicado **cria_grafo(Edificio, Piso, Col, Lin)**. Este predicado, juntamente com **cria_grafo_lin**, definem as ligações possíveis entre células adjacentes, incluindo **movimentos diagonais**, para cada piso de cada edifício.

Extensão da ligação das células

Para abranger as diversas formas de conexão, o predicado **ligacel** foi expandido para incluir ligações diretas, bidirecionais entre células no mesmo piso, ligações entre pisos e entre edifícios. Esta extensão garante que todas as possíveis rotas de movimentação sejam consideradas no planeamento de trajetórias**.**

```
:- dynamic ligacel/2.
:- dynamic ligacao_piso/3.
:- dynamic ligacao_edificio/2.

% Função para criar os grafos de um piso específico

cria_grafo(Edificio, Piso, _, 0) :- !.
cria_grafo(Edificio, Piso, Col, Lin) :-
    cria_grafo_lin(Edificio, Piso, Col, Lin),
    Lin1 is Lin - 1,
    cria_grafo(Edificio, Piso, Col, Lin1).

cria_grafo_lin(_, _, 0, _) :- !.
cria_grafo_lin(Edificio, Piso, Col, Lin) :-
    m(Edificio, Piso, Col, Lin, 0), !,
    ColS is Col + 1, ColA is Col - 1, LinS is Lin + 1, LinA is Lin - 1,
    cria_ligacoes(Edificio, Piso, Col, Lin, ColS, ColA, LinS, LinA),
    Col1 is Col - 1,
    cria_grafo_lin(Edificio, Piso, Col1, Lin).

cria_grafo_lin(Edificio, Piso, Col, Lin) :-
    Col1 is Col - 1,
    cria_grafo_lin(Edificio, Piso, Col1, Lin).

% Função auxiliar para criar ligações entre células adjacentes
cria_ligacoes(Edificio, Piso, Col, Lin, ColS, ColA, LinS, LinA) :-
    ((m(Edificio, Piso, ColS, Lin, 0), assertz(ligacel(cel(Edificio, Piso, Col, Lin), cel(Edificio, Piso, ColS, Lin))); true)),
    ((m(Edificio, Piso, ColA, Lin, 0), assertz(ligacel(cel(Edificio, Piso, Col, Lin), cel(Edificio, Piso, ColA, Lin))); true)),
    ((m(Edificio, Piso, Col, LinS, 0), assertz(ligacel(cel(Edificio, Piso, Col, Lin), cel(Edificio, Piso, Col, LinS))); true)),
    ((m(Edificio, Piso, Col, LinA, 0), assertz(ligacel(cel(Edificio, Piso, Col, Lin), cel(Edificio, Piso, Col, LinA))); true)),
    % Ligações diagonais
    ((m(Edificio, Piso, ColS, LinS, 0), assertz(ligacel(cel(Edificio, Piso, Col, Lin), cel(Edificio, Piso, ColS, LinS))); true)),
    ((m(Edificio, Piso, ColS, LinA, 0), assertz(ligacel(cel(Edificio, Piso, Col, Lin), cel(Edificio, Piso, ColS, LinA))); true)),
    ((m(Edificio, Piso, ColA, LinS, 0), assertz(ligacel(cel(Edificio, Piso, Col, Lin), cel(Edificio, Piso, ColA, LinS))); true)),
    ((m(Edificio, Piso, ColA, LinA, 0), assertz(ligacel(cel(Edificio, Piso, Col, Lin), cel(Edificio, Piso, ColA, LinA))); true)).

% Extensão do ligacel para lidar com ligações entre pisos e edifícios
ligacel(Cel1, Cel2) :-
    ligacel_piso(Cel1, Cel2);  % Verifica se existe ligação direta
    ligacel_piso(Cel2, Cel1);  % Verifica se a ligação é bidirecional
    ligacao_piso(_, Cel1, Cel2);  % Verifica ligações entre pisos
    ligacao_piso(_, Cel2, Cel1);  % Verifica se a ligação entre pisos é bidirecional
    ligacao_edificio(Cel1, Cel2);  % Verifica ligações entre edifícios
```

Figura 5 Criação dos grafos

Inicialização automática dos grafos

Para a eficiência e praticidade, o projeto foi estruturado para criar automaticamente os grafos necessários para a representação do ambiente ao inicializar o ficheiro Prolog. O predicado cria_grafos é chamado automaticamente, configurando os grafos para cada piso dos edifícios definidos, como demonstrado nas chamadas de cria_grafo para diferentes combinações de edifícios e pisos.

```
% Predicado de inicialização
:- initialization(cria_grafos).

cria_grafos :-
    cria_grafo(c,1,16,10),
    cria_grafo(d,1,17,10),
    cria_grafo(a,1,10,16),
    cria_grafo(b,1,11,15),
    cria_grafo(c,2,17,10),
    cria_grafo(d,2,14,10),
    cria_grafo(a,2,9,18),
    cria_grafo(b,2,8,19),
    cria_grafo(c,3,18,9),
    cria_grafo(d,3,14,9),
    cria_grafo(b,3,8,17),
    cria_grafo(c,4,15,9).
```

Figura 6 Inicialização automática dos grafos

Essa metodologia detalhada e sistemática de representação espacial e de conexões permite uma análise abrangente e precisa das rotas possíveis para a movimentação autônoma dos robôs, formando a base sobre a qual os algoritmos de procura e otimização foram aplicados para encontrar as melhores trajetórias.

Movimentação do Robot

Primeiro em profundidade (DFS)

No projeto **ProjetoAlgav.pl**, a pesquisa em profundidade (DFS) foi implementada para explorar as trajetórias possíveis entre pisos e edifícios. O DFS é particularmente eficaz em ambientes complexos, pois permite uma exploração completa das rotas disponíveis, garantindo que nenhuma possibilidade seja deixada de lado.

A implementação do DFS começa com a função dfs(Orig, Dest, Cam), que inicia a procura pelo caminho entre o ponto de origem e destino. A função dfs2 é a implementação recursiva do algoritmo, que explora cada ligação possível, evitando ciclos e acumulando o caminho percorrido. Esta abordagem garante uma pesquisa completa do espaço, embora possa não ser a mais eficiente em termos de tempo ou distância percorrida.

Porém o algoritmo de DFS não se adequa às nossas matrizes, pois são matrizes muito grandes. Quando executada uma pesquisa em profundidade de um piso ao outro o algoritmo devolve algumas listas de caminhos possíveis, porém fica parado e provavelmente demoraria horas a devolver os restantes caminhos.

```
%Inicia a pesquisa em profundidade
dfs(Orig, Dest, Cam):-
    dfs2(Orig, Dest, [Orig], Cam).

%Função recursiva para a pesquisa em profundidade
dfs2(Dest, Dest, LA, Cam):-
    reverse(LA, Cam).

dfs2(Act, Dest, LA, Cam):-
    (ligacel(Act, Next); ligacao(Act, Next)),  % Verifica as ligações regulares e especiais
    \+ member(Next, LA),                       % Evita ciclos
    dfs2(Next, Dest, [Next|LA], Cam).

%Verifica se existe uma ligação especial (piso ou edifício)
ligacao(Act, Next) :-
    (ligacao_piso(_, Act, Next); ligacao_piso(_, Next, Act);
     ligacao_edificio(Act, Next); ligacao_edificio(Next, Act)).
```

Figura 7 Algoritmo de DFS

Melhor solução de DFS (Better_DFS)

Para encontrar o caminho mais curto, o better_dfs foi desenvolvido. Este método utiliza o DFS para gerar todas as possíveis soluções (all_dfs) e, em seguida, seleciona a melhor solução (shortlist) com base no critério de menor número de passos. Este método é crucial para garantir que, apesar da natureza exaustiva do DFS, a solução mais eficiente seja escolhida.

Como seria de esperar o algoritmo fica travado a calcular o melhor caminho devido á sua complexidade. Dentro do mesmo piso conseguimos obter o melhor caminho de uma célula a outra porém quando aumentamos as possibilidades já não é possível em tempo normal.

```
better_dfs(Orig, Dest, Cam):-
    all_dfs(Orig, Dest, LCam),
    shortlist(LCam, Cam, _).

%Encontra todos os caminhos possíveis
all_dfs(Orig, Dest, LCam):-
    findall(Cam, dfs(Orig, Dest, Cam), LCam).

%Auxiliar para encontrar o caminho mais curto
shortlist([L], L, N):-
    !, length(L, N).
shortlist([L|LL], Lm, Nm):-
    shortlist(LL, Lm1, Nm1),
    length(L, NL),
    ((NL < Nm1, !, Lm = L, Nm is NL); (Lm = Lm1, Nm is Nm1)).
```

Figura 8 Algoritmo de better_dfs

Primeiro em largura (BFS)

O BFS é conhecido pela sua eficiência em encontrar o caminho mais curto em termos de número de passos. A implementação inicia com bfs, que chama a função auxiliar bfs_aux para explorar sistematicamente cada nível de profundidade antes de passar para o próximo. Esse método é particularmente útil para encontrar a solução mais rápidas em ambientes com muitas rotas paralelas ou em espaços mais abertos.

Porém mais uma vez o algoritmo tal como o DFS retorna o caminho de alguns caminhos possíveis mas não todos, devido também á sua complexidade escalável quando a matriz aumenta.

```
% BFS
bfs(Inicio, Fim, Caminho) :-
    bfs_aux(Fim, [[Inicio]], [], CaminhoReverso),
    reverse(CaminhoReverso, Caminho).

% BFS auxiliar
bfs_aux(Fim, [[Fim|T]|_], _, [Fim|T]).
bfs_aux(Fim, [Atual|Resto], Visitados, Caminho) :-
    Atual = [Ultimo|_],
    findall(X, (ligacel(Ultimo, X), \+ member(X, Visitados)), Vizinhos),
    maplist(cria_novo_caminho(Atual), Vizinhos, NovosCaminhos),
    append(Resto, NovosCaminhos, NovaFila),
    bfs_aux(Fim, NovaFila, [Ultimo|Visitados], Caminho).
bfs_aux(_, [], _, []).

cria_novo_caminho(Caminho, Prox, [Prox|Caminho]).
```

Figura 9 Algoritmo de bfs

A*

De modo a encontrar o caminho mais curto, o algoritmo A* junta o que há de bom dos algoritmos Primeiro o Melhor (BFS), ou seja, o uso de funções que estimam a distância à solução, com o do Branch & Bound, um algoritmo de pesquisa com avaliação de transições locais mas com a possibilidade de alterar a qualquer momento o próximo nó, ou seja, o uso de custos acumulados conhecidos e a possibilidade de comutar de um ponto para outro na árvore de pesquisa sem que o novo ponto seja um descendente do primeiro.

Este algoritmo retorna caminhos de maior extensão mas mesmo assim demora muito tempo quando em caminhos muito grandes, devido à extensão do nosso grafo.

```
aStar(Orig, Dest, Cam, Custo):-
    aStar2(Dest, [(_, 0, [Orig])], [], Cam, Custo).

aStar2(Dest, [(_, Custo, [Dest|T])|_], _, Cam, Custo):-
    reverse([Dest|T], Cam).

aStar2(Dest, [(_, Ca, LA)|Outros], Visitados, Cam, Custo):-
    LA = [Act|_],
    findall((CEX, CaX, [X|LA]),
            (Dest \== Act,
             ligacel(Act, X),
             \+ member(X, Visitados),
             \+ member(X, LA),
             CaX is Ca + 1,  % Cost from start to current node
             heuristic(X, Dest, EstX),  % node to goal estimation
             CEX is CaX + EstX),  % Total cost including heuristic
            Novos),
    append(Outros, Novos, Todos),
    append(Visitados, [Act], VisitadosAtualizados),
    sort(Todos, TodosOrd),
    aStar2(Dest, TodosOrd, VisitadosAtualizados, Cam, Custo).

heuristic(cel(_, _, X1, Y1), cel(_, _, X2, Y2), Estimativa) :-
    Estimativa is abs(X2 - X1) + abs(Y2 - Y1).
```

*Figura 10: Algoritmo A**

A* com minimização no uso de elevadores

De maneira a ser possível a utilização de um algoritmo que ditasse uma menor utilização de elevadores através do caminho Origem ao caminho destino, pegamos no

algoritmo mais eficiente(A*) e fizemos umas alterações para que o algoritmo cumprisse essa ordem.

A nossa abordagem foi penalizar o custo no uso do elevador no algoritmo de maneira a que o elevador tente procurar outras soluções que não o uso do elevador. Introduziu-se um custo adicional de 1.1 (em vez de 1) para transitar por um elevador. Além disso, o algoritmo mantém um registo dos elevadores usados ao longo do caminho. Isso permite uma análise detalhada do percurso e a avaliação do impacto do uso de elevadores na rota final.

```
aStar2E(Dest, [(_, Ca, Elevadores, LA)|Outros], Visitados, Cam, Custo, ElevadoresUs
    LA = [Act|_],
    findall((CEX, CaX, ElevadoresAtualizados, [X|LA]),
            (Dest \== Act,
             ligacel(Act, X),
             \+ member(X, Visitados),
             \+ member(X, LA),
             custoAdicional(Act, X, CustoAdicional, Elevadores, ElevadoresAtualizac
             CaX is Ca + 1 + CustoAdicional,
             heuristicE(X, Dest, EstX),
             CEX is CaX + EstX),
            Novos),
    append(Outros, Novos, Todos),
    append(Visitados, [Act], VisitadosAtualizados),
    sort(Todos, TodosOrd),
    aStar2E(Dest, TodosOrd, VisitadosAtualizados, Cam, Custo, ElevadoresUsados).

custoAdicional(Act, Next, CustoAdicional, Elevadores, ElevadoresAtualizados) :-
    (elevador(Act), elevador(Next) ->
        CustoAdicional = 0.1, % Custo mais alto para desencorajar uso do elevador
        (Act \== Next -> ElevadoresAtualizados = [Next|Elevadores] ; ElevadoresAtua
    ;
        CustoAdicional = 0, ElevadoresAtualizados = Elevadores).

heuristicE(cel(_, _, X1, Y1), cel(_, _, X2, Y2), Estimativa) :-
    Estimativa is abs(X2 - X1) + abs(Y2 - Y1).

elevador(cel(Edificio, Piso, Col, Lin)) :-
    m(Edificio, Piso, Col, Lin, _, e).
```

Conclusões

No nosso estudo sobre o planeamento de trajectórias para a navegação de robôs autónomos, abordámos a implementação e a melhoria de algoritmos clássicos como o DFS (Depth-First Search), o BFS (Breadth-First Search) e o A*. Durante as fases de desenvolvimento e teste, identificámos que o algoritmo A* é notavelmente mais rápido e mais eficiente na localização do caminho ótimo em comparação com o DFS e o BFS, especialmente em matrizes de grande escala e ambientes complexos.

Este projeto representa um avanço significativo no domínio da robótica e da inteligência artificial, introduzindo abordagens inovadoras à representação espacial e à navegação em ambientes multidimensionais. A inclusão de movimentos diagonais e a capacidade de otimizar a utilização de elevadores são exemplos de como a nossa solução transcende as técnicas tradicionais, oferecendo uma modelação mais realista e aplicável em cenários reais.

Para além disso, o estudo detalhado da complexidade dos algoritmos e da sua viabilidade em diferentes cenários reforça a robustez da nossa abordagem. Com este trabalho, não só desenvolvemos uma solução técnica avançada para o planeamento do movimento de robôs, como também alargámos a nossa compreensão teórica e prática dos algoritmos de pesquisa e otimização, estabelecendo uma base sólida para futuras investigações e aplicações nesta área.

Análise de Complexidade

Matriz 4x4

Nº nós: 16 Nº ligações:

1º solução de BFS:

?-
bfs_timed(cel(1,1),cel(4,4),Cam,Tempo).
Cam =
[cel(1,1),cel(2,1),cel(3,1),cel(4,1),cel(4,2),cel(4,3),cel(4,4)],
Tempo = 0.0020329952239990234;

1º solução de BFS C/Mov.Diagonal:

?-
bfs_timed(cel(1,1),cel(4,4),Cam,Temp).
Cam =
[cel(1,1),cel(2,2),cel(3,3),cel(4,4)],
Tempo = 0.007363796234130859

1º solução de DFS:

?-
dfs_timed(cel(1,1),cel(4,4),Cam,Tempo).
Cam=[cel(1,1),cel(2,1),cel(3,1),cel(4,1),cel(4,2),cel(3,2),cel(2,2),cel(1,2),cel(1,3),cel(2,3),cel(3,3),cel(4,3),cel(4,4)],
Tempo = 4.792213439941406e-5

1º solução de DFS C/Mov.Diagonal:

?-
dfs_timed(cel(1,1),cel(4,4),Cam,Tempo).
Cam =
[cel(1,1),cel(2,1),cel(3,1),cel(4,1),cel(4,2),cel(3,2),cel(2,2),cel(1,2),cel(1,3),cel(2,3),cel(3,3),cel(4,3),cel(4,4)],
Tempo = 4.792213439941406e-5

1º solução de Better_DFS:

?- better_dfs(cel(1,1),cel(4,4),Cam).
Cam =
[cel(1,1),cel(2,1),cel(3,1),cel(4,1),cel(4,2),cel(4,3),cel(4,4)].
Tempo: 0.004775047302246094

1º solução de Better_DFS C/Mov.Diagonal:

?- better_dfs(cel(1,1),cel(4,4),Cam).
Cam =
[cel(1,1),cel(2,2),cel(3,3),cel(4,4)].
Tempo:2.3600540161132812

A*:

?- aStar_timed(cel(1,1),cel(4,4),Cam, Custo, Tempo).
Cam =
[cel(1,1),cel(1,2),cel(2,2),cel(2,3),cel(3,3),cel(3,4),cel(4,4)],
Custo = 17.084259940083065,
Tempo = 0.011364936828613281

A* C/Mov.Diagonal:

?- aStar_timed(cel(1,1),cel(4,4),Cam, Custo, Tempo).
Cam =
[cel(1,1),cel(2,2),cel(3,3),cel(4,4)],
Custo = 7.242640687119285,
Tempo = 0.0004200935363769531

Matriz 5x5

Nº nós: 25 Nº ligações:

1º solução de BFS:

?-
bfs_timed(cel(1,1),cel(5,5),Cam,Tempo).
Cam =
[cel(1,1),cel(2,1),cel(3,1),cel(4,1),cel(5,1),cel(5,2),cel(5,3),cel(5,4),cel(5,5)],
Tempo = 0.01694798469543457

1º solução de BFS C/Mov.Diagonal:

?-
bfs_timed(cel(1,1),cel(5,5),Cam,Tempo).
Cam =
[cel(1,1),cel(2,2),cel(3,3),cel(4,4),cel(5,5)],
Tempo = 0.03982901573181152

1º solução de DFS:

?-
dfs_timed(cel(1,1),cel(5,5),Cam,Tempo).
Cam=[cel(1,1),cel(2,1),cel(3,1),cel(4,1),cel(5,1),cel(5,2),cel(4,2),cel(3,2),cel(2,2),cel(1,2),cel(1,3),cel(2,3),cel(3,3),cel(4,3),cel(5,3),cel(5,4),cel(4,4),cel(3,4),cel(2,4),cel(1,4),cel(1,5),cel(2,5),cel(3,5),cel(4,5),cel(5,5)],
Tempo = 0.00010609626770019531

1º solução de DFS C/Mov.Diagonal:

?-
dfs_timed(cel(1,1),cel(5,5),Cam,Tempo).
Cam =
[cel(1,1),cel(2,1),cel(3,1),cel(4,1),cel(5,1),cel(5,2),cel(4,2),cel(3,2),cel(2,2),cel(1,2),cel(1,3),cel(2,3),cel(3,3),cel(4,3),cel(5,3),cel(5,4),cel(4,4),cel(3,4),cel(2,4),cel(1,4),cel(1,5),cel(2,5),cel(3,5),cel(4,5),cel(5,5)],
Tempo = 0.0061419010162353516

1º solução de Better_DFS:

?- better_dfs(cel(1,1),cel(5,5),Cam).
Cam =
[cel(1,1),cel(2,1),cel(3,1),cel(4,1),cel(5,1),cel(5,2),cel(5,3),cel(5,4),cel(5,5)].
Tempo: 0.31765103340148926

1º solução de Better_DFS C/Mov.Diagonal:

?- better_dfs(cel(1,1),cel(5,5),Cam).
Cam =?
Tempo: +5min

A*:

?- aStar_timed(cel(1,1),cel(5,5),Cam, Custo, Tempo).

Cam =
[cel(1,1),cel(1,2),cel(2,2),cel(2,3),cel(3,3),cel(3,4),cel(4,4),cel(4,5),cel(5,5)],
Custo = 28.32690062720235,
Tempo = 0.025428056716918945

A* C/Mov.Diagonal:

?- aStar_timed(cel(1,1),cel(5,5),Cam, Custo, Tempo).

Cam = [cel(1,1),cel(2,2),cel(3,3),cel(4,4),cel(5,5)],
Custo = 12.485281374238571,
Tempo = 0.0021569728851

Matriz 6x6

Nº nós: 36 Nº ligações:

1º solução de BFS:

?- bfs_timed(cel(1,1),cel(6,6),Cam,Tempo).
Cam=[cel(1,1),cel(2,1),cel(3,1),cel(4,1),cel(5,1),cel(6,1),cel(6,2),cel(6,3),cel(6,4),cel(6,5),cel(6,6)],
Tempo = 0.2745649814605713

1º solução de BFS C/Mov.Diagonal:

?- bfs_timed(cel(1,1),cel(6,6),Cam,Tempo).
Cam = [cel(1,1),cel(2,2),cel(3,3),cel(4,4),cel(5,5),cel(6,6)],
Tempo = 0.4512150287628174

1º solução de DFS:

?- dfs_timed(cel(1,1),cel(6,6),Cam,Tempo).
Cam=[cel(1,1),cel(2,1),cel(3,1),cel(4,1),cel(5,1),cel(6,1),cel(6,2),cel(5,2),cel(4,2),cel(3,2),cel(2,2),cel(1,2),cel(1,3),cel(2,3),cel(3,3),cel(4,3),cel(5,3),cel(6,3),cel(6,4),cel(5,4),cel(4,4),cel(3,4),cel(2,4),cel(1,4),cel(1,5),cel(2,5),cel(3,5),cel(4,5),cel(5,5),cel(6,5),cel(6,6)],
Tempo = 6.29425048828125e-5

1º solução de DFS C/Mov.Diagonal:

?- dfs_timed(cel(1,1),cel(6,6),Cam,Tempo).
Cam = [cel(1,1),cel(2,1),cel(3,1),cel(4,1),cel(5,1),cel(6,1),cel(6,2),cel(5,2),cel(4,2),cel(3,2),cel(2,2),cel(1,2),cel(1,3),cel(2,3),cel(3,3),cel(4,3),cel(5,3),cel(6,3),cel(6,4),cel(5,4),cel(4,4),cel(3,4),cel(2,4),cel(1,4),cel(1,5),cel(2,5),cel(3,5),cel(4,5),cel(5,5),cel(6,5),cel(6,6)],
Tempo = 0.0001168251037597656

1º solução de Better_DFS:

?- better_dfs(cel(1,1),cel(6,6),Cam).
Cam=[cel(1,1),cel(2,1),cel(3,1),cel(4,1),cel(5,1),cel(6,1),cel(6,2),cel(6,3),cel(6,4),cel(6,5),cel(6,6)].
Tempo: 82.11509084701538

1º solução de Better_DFS C/Mov.Diagonal:

?- better_dfs(cel(1,1),cel(6,6),Cam).
Cam=?
Tempo: +5min

A*:

?- aStar_timed(cel(1,1),cel(6,6),Cam, Custo, Tempo).
Cam = [cel(1,1),cel(1,2),cel(2,2),cel(2,3),cel(3,3),cel(3,4),cel(4,4),cel(4,5),cel(5,5),cel(5,6),cel(6,6)],
Custo = 42.38687911412757,
Tempo = 0.18146610260009766

A* C/Mov.Diagonal:

?- aStar_timed(cel(1,1),cel(6,6),Cam, Custo, Tempo).
Cam = [cel(1,1),cel(2,2),cel(3,3),cel(4,4),cel(5,5),cel(6,6)],
Custo = 19.14213562373095,
Tempo = 0.005396127700805664 .

Matriz 7x7

Nº nós: 49 Nº ligações:

1º solução de BFS:

?- bfs_timed(cel(1,1),cel(7,7),Cam,Tempo).
Cam=[cel(1,1),cel(2,1),cel(3,1),cel(4,1),cel(5,1),cel(6,1),cel(7,1),cel(7,2),cel(7,3),cel(7,4),cel(7,5),cel(7,6),cel(7,7)],
Tempo = 10.14252495765686

1º solução de BFS C/Mov.Diagonal:

?- bfs_timed(cel(1,1),cel(7,7),Cam,Tempo).
Cam = [cel(1,1),cel(2,2),cel(3,3),cel(4,4),cel(5,5),cel(6,6),cel(7,7)],
Tempo = 12.740067958831787

1º solução de DFS:

?- dfs_timed(cel(1,1),cel(7,7),Cam,Tempo).
Cam=[cel(1,1),cel(2,1),cel(3,1),cel(4,1),cel(5,1),cel(6,1),cel(7,1),cel(7,2),cel(6,2),cel(5,2),cel(4,2),cel(3,2),cel(2,2),cel(1,2),cel(1,3),cel(2,3),cel(3,3),cel(4,3),cel(5,3),cel(6,3),cel(7,3),cel(7,4),cel(6,4),cel(5,4),cel(4,4),cel(3,4),cel(2,4),cel(1,4),cel(1,5),cel(2,5),cel(3,5),cel(4,5),cel(5,5),cel(6,5),cel(7,5),cel(7,6),cel(6,6),cel(5,6),cel(4,6),cel(3,6),cel(2,6),cel(1,6),cel(1,7),cel(2,7),cel(3,7),cel(4,7),cel(5,7),cel(6,7),cel(7,7)],
Tempo = 0.01235818862915039

1º solução de DFS C/Mov.Diagonal:

?- dfs_timed(cel(1,1),cel(7,7),Cam,Tempo).
Cam = [cel(1,1),cel(2,1),cel(3,1),cel(4,1),cel(5,1),cel(6,1),cel(7,1),cel(7,2),cel(6,2),cel(5,2),cel(4,2),cel(3,2),cel(2,2),cel(1,2),cel(1,3),cel(2,3),cel(3,3),cel(4,3),cel(5,3),cel(6,3),cel(7,3),cel(7,4),cel(6,4),cel(5,4),cel(4,4),cel(3,4),cel(2,4),cel(1,4),cel(1,5),cel(2,5),cel(3,5),cel(4,5),cel(5,5),cel(6,5),cel(7,5),cel(7,6),cel(6,6),cel(5,6),cel(4,6),cel(3,6),cel(2,6),cel(1,6),cel(1,7),cel(2,7),cel(3,7),cel(4,7),cel(5,7),cel(6,7),cel(7,7)],
Tempo = 0.02221393585205078

1º solução de Better_DFS:

?- better_dfs(cel(1,1),cel(7,7),Cam).
Cam=?
Tempo:+5min

1º solução de Better_DFS C/Mov.Diagonal:

?- better_dfs(cel(1,1),cel(7,7),Cam).
Cam=?
Tempo:+5min

A*:

aStar_timed(cel(1,1),cel(7,7),Cam,
Custo, Tempo).
Cam =
[cel(1,1),cel(1,2),cel(2,2),cel(2,3),cel(3,3),cel(3,4),cel(4,4),cel(4,5),cel(5,5),cel(5,6),cel(6,6),cel(6,7),cel(7,7)],
Tempo = 2.445850133895874

A* C/Mov.Diagonal:

?- aStar_timed(cel(1,1),cel(7,7),Cam,
Custo, Tempo).
Cam =
[cel(1,1),cel(2,2),cel(3,3),cel(4,4),cel(5,5),cel(6,6),cel(7,7)],
Custo = 27.213203435596427,
Tempo = 0.022696971893310547 .

Matriz 4x4:

- BFS sem movimentos diagonais encontrou um caminho com 7 passos em 0.002 segundos, enquanto com movimentos diagonais, um caminho mais curto de 4 passos foi encontrado em 0.007 segundos.
- DFS mostrou o mesmo tempo (aproximadamente 0.00005 segundos) para ambas as abordagens, porém os caminhos são mais longos.
- Better_DFS teve desempenho similar ao BFS.
- A* sem movimentos diagonais encontrou um caminho com custo 17.08 em 0.011 segundos, e com movimentos diagonais, um caminho mais eficiente com custo 7.24 em um tempo significativamente menor (0.00042 segundos).

Matriz 5x5:

- O tempo de resposta do BFS aumentou significativamente, especialmente com movimentos diagonais (0.039 segundos).
- DFS e Better_DFS mostraram um aumento no tempo de resposta com a complexidade crescente, com Better_DFS levando mais de 5 minutos para resolver com movimentos diagonais.
- A* continua a ser eficiente, encontra caminhos com custos razoáveis em tempos aceitáveis (0.025 e 0.002 segundos para as duas abordagens).

Matriz 6x6:

- O tempo de resposta do BFS continua a crescer, chegando a 0.451 segundos para movimentos diagonais.
- DFS e Better_DFS tornam-se menos práticos, com Better_DFS levando mais de 5 minutos em movimentos diagonais.
- A* ainda se destaca em eficiência, mantendo tempos de resposta gerenciáveis (0.181 e 0.005 segundos).

Matriz 7x7:

- O BFS torna-se significativamente mais lento, especialmente para movimentos diagonais (mais de 12 segundos).
- DFS também mostra um aumento no tempo de resposta.
- Better_DFS torna-se impraticável, ultrapassando 5 minutos para ambas as abordagens.
- A* mantém a eficiência, especialmente com movimentos diagonais, encontrando caminhos com custos razoáveis em tempos aceitáveis (2.446 e 0.023 segundos).

Conclusão:

A eficiência do A* é consistente em matrizes de diferentes tamanhos, especialmente com movimentos diagonais, onde consegue encontrar soluções ótimas em tempos consideravelmente menores.

O BFS é eficiente em matrizes menores, mas o seu tempo de resposta cresce rapidamente com o aumento da complexidade da matriz.

O DFS e Better_DFS tornam-se impraticáveis em matrizes maiores, especialmente com movimentos diagonais, indicando que estes métodos não são adequados para problemas de maior escala.

Em geral, a adição de movimentos diagonais melhora a eficiência dos caminhos encontrados, mas também pode aumentar o tempo de resposta, dependendo do algoritmo.