# 

**Relatório Algoritmia Avançada**

2023 / 2024

**1180727- Ruben Martins**



**Introdução**

No âmbito da disciplina de Algoritmia Avançada (ALGAV), este relatório apresenta o desenvolvimento do Sprint B, focando-se no planeamento da 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, navegando por corredores de ligação, acedendo a elevadores e transitando 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 busca 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.

**Uma imagem com file, Gráfico, diagrama, captura de ecrã

Descrição gerada automaticamente**

Figura 1 Exemplo matriz Edifício D Piso 2

Uma imagem com texto, captura de ecrã, design

Descrição gerada automaticamente

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).

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

**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.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

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'.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

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. Este modelo constitui a base sobre a qual os algoritmos de procura e planeamento de trajetórias foram aplicados, permitindo uma simulação realista da movimentação autônoma de um robô em um ambiente urbano complexo.

**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**.**

Uma imagem com texto, captura de ecrã, menu, Tipo de letra

Descrição gerada automaticamente

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.

Uma imagem com texto, captura de ecrã, Tipo de letra, design

Descrição gerada automaticamente

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.

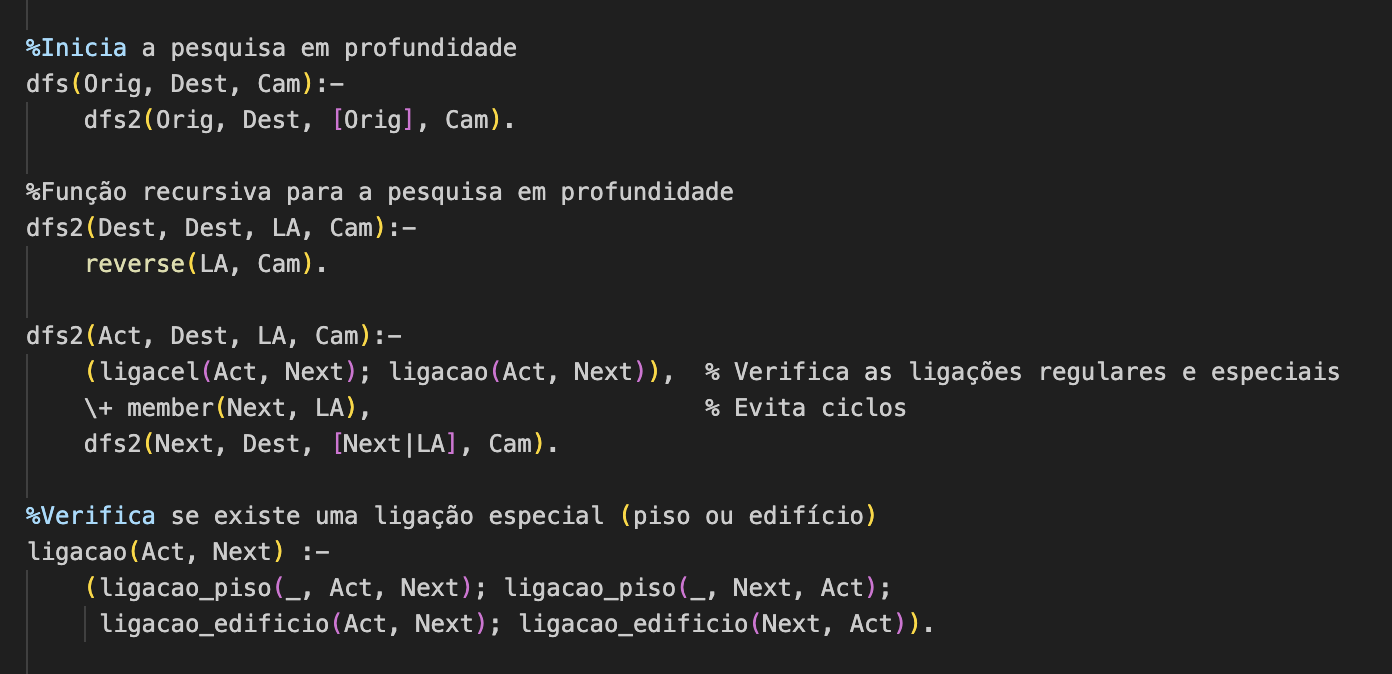
****

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.

Uma imagem com texto, captura de ecrã, Tipo de letra, software

Descrição gerada automaticamente

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 rápida 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.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura 9 Algoritmo de BFS

**A\***

\*\* Escrever sobre o algoritmo de A\* \*\*

**Conclusões**

\*\* Escrever conclusão sobre os 4 algoritmos. Os 3 primeiros são insuficientes, o A\* deverá ser o mais apropriado para usar no nosso projeto \*\*

**Melhor solução para o planeamento (ponto 4 no ppw de apoio 4)**

\*\* Nesta secção explicar o porque usamos o A\* e fazer uma demonstração de uma trajetória de uma célula a outra \*\*

# 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\*:

## A\* C/Mov.Diagonal:

# 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\*:

## A\* C/Mov.Diagonal:

# 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 = 0.00011682510375976562

## 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\*:

## A\* C/Mov.Diagonal:

# 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\*:

## A\* C/Mov.Diagonal: