# 

**Relatório Algoritmia Avançada**

2023 / 2024

**1180727 - Ruben Martins**

**1211171 – Pedro Mesquita**

**1210816 - João Castro**

**1191831 - Rui Gonçalves**

**1210913 - Pedro Mendes**



**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 Exemplo matriz Edifício D Piso 2

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

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 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, Tipo de letra, captura de ecrã

Descrição gerada automaticamente

Figura 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, Tipo de letra, captura de ecrã, algebra

Descrição gerada automaticamente

Figura 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, documento, Tipo de letra

Descrição gerada automaticamente**

Figura 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, número

Descrição gerada automaticamente

Figura 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.

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

Descrição gerada automaticamente

Figura 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.

Figura 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

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

Descrição gerada automaticamente

Figura : 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.

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

Descrição gerada automaticamente

Figura 10: Algoritmo A\*

**Conclusões**

Ao explorar os algoritmos de planeamento de trajetória para a navegação dos robôs foram abordadas várias implementações: a Pesquisa em Profundidade (DFS), o DFS melhorado, a Pesquisa em Largura (BFS) e o A\*.

Destes algoritmos, devido ao tempo necessário para encontrar o melhor caminho, a reposta óbvia é o algoritmo A\*, que os encontra visivelmente muito mais rápido do que os outros, sendo portanto, mais eficiente.

**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\*:

?- 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 .

# Sprint C

Foi-nos pedido para gerar um plano de atendimento de tarefas que indicasse uma execução dessas mesmas tarefas criada através da geração de todas as sequências e escolhas de modo a demorar menos tempo.

De acordo com o que nos foi fornecido as tarefas especificadas foram as seguintes:

* Limpeza de salas;
* Vigilância de pisos;
* Pegar e entregar objetos;

Para cada tarefa foi criada uma função em Swi-Prolog utilizando as matrizes de pontos onde o robot pode andar, os grafos criados através dessas matrizes e o algoritmo mais eficiente o aStar.

Foi necessário também indicar as Salas de cada piso de cada edifício:

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

Descrição gerada automaticamente

Para cada sala o foi definido um **m(edifício, piso, x, y, valor, sala).**

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

Descrição gerada automaticamentePara cada função de cada tarefa foi necessário fazer uma identificação da sala, por isso foi criada uma função que as identificasse:

**Limpeza de salas**

A função limparSala/3 foi concebida para determinar o caminho ótimo e o custo total da limpeza de uma sala específica.

**Identificação da localização:** Identifica as coordenadas do destino através da invocação do predicado identificar\_localizacao\_sala/2, que devolve as coordenadas da célula (edifício, piso, X, Y) correspondente à sala indicada.

**Cálculo de trajectórias:** Utiliza o algoritmo aStar/4 para calcular o caminho (Caminho) e o custo base (CustoBase) desde a localização atual do robot (Origem) até ao destino (Destino).

**Cálculo do custo:** Adiciona um custo fixo de limpeza (20 unidades) ao custo base para representar o custo total (CustoTotal) da tarefa.

**Output**: Emite o caminho percorrido para a tarefa de limpeza e o seu custo.

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

Descrição gerada automaticamente

**Exemplo**: Limpar a sala c101 partindo da célula x=7, y=6 do piso 1 do edifício C:

**Vigilância de pisos**

A função vigilância/4 tem como objetivo efetuar a vigilância de um determinado piso de um edifício, calculando o caminho e o custo total para visitar todas as células transitáveis.

**Identificação de Células Transitáveis:** Reúne todas as células transitáveis (CelulasTransitaveis) no piso especificado do edifício utilizando findall/3 com os parâmetros do edifício e do piso.

**Cálculo do trajeto e do custo:** Calcula iterativamente o caminho ótimo e o custo acumulado para visitar cada célula seguinte na lista utilizando o algoritmo aStar/4.

**Acumulação de trajetória e custo:** A função anexa o caminho parcial ao caminho acumulado (CaminhoAcumulado) e adiciona o custo parcial ao custo acumulado (CustoAcumulado).

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

Descrição gerada automaticamente**Output:** Quando todas as células são visitadas, a função devolve o custo total da tarefa de vigilância. Não achamos necessário retornar o caminho pelo qual o robot percorre, visto que ele percorre todas as células desse mesmo piso ao fazer a vigilância.

**Exemplo**: Vigilância do piso 2 do edifício C partindo da célula com x=12, y=4:



**Pegar e entregar objetos**

A função pegar\_entregar\_objeto/4 tem como objetivo modelar uma tarefa onde o robô precisa pegar um objeto de uma sala e entregá-lo em outra, calculando os caminhos e custos para ambos os segmentos.

**Identificação da Localização:** Identifica as coordenadas das salas de recolha (SalaPegar) e de entrega (SalaEntregar) utilizando o predicado identificar\_localizacao\_sala/2.

**Cálculo do Caminho e Custo de Recolha:** Calcula o caminho e o custo para se deslocar da localização atual do robô para a localização de recolha, utilizando o algoritmo aStar/4.

**Cálculo do caminho e custo de entrega:** Depois de recolher o objeto, calcula o caminho e o custo para se deslocar do local de recolha para o local de entrega utilizando novamente o algoritmo aStar/4.

**Cálculo do custo total:** Soma os custos de ambos os segmentos para determinar o custo total da tarefa.

**Output:** Emite os caminhos e custos para ambos os segmentos da tarefa.

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

Descrição gerada automaticamente

**Exemplo**: Pegar no objeto que está na sala b101, partindo da célula com x=12, y=4 do piso 2 do edifício C, para entregar na sala c101:

Uma imagem com texto, captura de ecrã, Tipo de letra, preto e branco

Descrição gerada automaticamente

**Plano de atendimento de tarefas**

O objetivo principal é gerar um plano para a realização de um conjunto de tarefas (limpeza de uma sala, vigilância ou recolha e entrega de um objeto), encontrando a sequência de execução de tarefas que minimiza o custo total em termos de tempo ou outros recursos.

O programa começa por pedir ao utilizador que introduza a localização inicial do robô e depois continua a pedir as tarefas a executar. Uma vez fornecidas todas as tarefas e os respetivos detalhes, o programa gera todas as sequências possíveis dessas tarefas e avalia-as para encontrar a mais eficiente. Segue-se uma descrição do processo:

1. Recolha de dados de entrada da tarefa (pedir\_tarefas/1 e processar\_tarefas/3):

**Solicitação de tarefas:** O sistema pede ao utilizador que introduza as tarefas uma a uma até o utilizador introduzir "n" para assinalar a conclusão.

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

Descrição gerada automaticamente**Processamento de tarefas:** À medida que cada tarefa é introduzida, é adicionada a uma lista acumulada de tarefas, juntamente com os detalhes necessários, como o identificador da sala para limpeza ou do edifício e do piso para vigilância.

1. Localização inicial do robô:

É pedido ao utilizador que forneça a localização inicial do robô no formato cel(Edificio, Piso, X, Y).

1. Geração de todas as sequências possíveis (findall/3):

Utiliza o predicado findall/3 combinado com permutation/2 para gerar todas as permutações possíveis (sequências) da lista de tarefas, considerando cada permutação como um plano potencial.

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

Descrição gerada automaticamente

1. Avaliação de cada sequência (calcular\_melhor\_sequencia/4):

**Cálculo do custo:** Para cada sequência, calcula o custo total utilizando calcular\_custo\_sequencia/4, que calcula iterativamente o custo de execução de cada tarefa na sequência a partir do ponto de partida dado.

**Execução de tarefas:** A execução de cada tarefa envolve o cálculo do caminho e do custo utilizando o algoritmo aStar/4 e, em seguida, o ajuste da posição do robô para as tarefas subsequentes.

**Otimização:** Depois de calcular o custo de todas as sequências, compara-as para encontrar a sequência com o custo total mais baixo.

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

Descrição gerada automaticamente

1. Atualizar a localização do robô (atualizar\_origem/3):

Após cada tarefa, a localização do robô é atualizada para a última posição do percurso da tarefa. Esta posição torna-se o ponto de partida para a próxima tarefa.

1. Saída da melhor sequência:

Uma vez determinada a sequência mais rentável, o sistema produz esta sequência juntamente com o seu custo total, fornecendo um plano claro e otimizado para a execução da tarefa.

1. Flexibilidade e escalabilidade:

O sistema foi concebido para lidar com um número arbitrário de tarefas e tipos, tornando-o flexível e escalável para vários cenários e tipos de tarefas adicionais.

Exemplo de um plano de atendimento de tarefas:

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

Descrição gerada automaticamenteSão definidas 3 tarefas diferentes, limpar a sala b202, fazer a vigilância do piso b1 e pegar em um objeto da sala b101 e entregar na sala b202.

**Conclusão**

Este plano de atendimento a um conjunto de tarefas baseia-se em gerar todas as sequências possíveis de tarefas e selecionar a que minimiza o custo total. Trata-se de uma abordagem exaustiva e abrangente que garante que a sequência escolhida é de facto a mais eficiente, tendo em conta as métricas de custo e as restrições dadas. Este método, embora computacionalmente intensivo, garante a melhor solução de todas as ordens de tarefas possíveis e é, portanto, adequado para cenários em que a precisão e a otimização são críticas.

**Estudo da complexidade para o plano de atendimento de tarefas**

**Análise Temporal**

Para melhor entender análise de complexidade no nosso algoritmo de atendimento de tarefas precisamos de avaliar a geração de permutações e o cálculo de custo de cada sequência de tarefas.

**Geração de permutações:**

No nosso algoritmo usamos a função ***permutation/2*** para gerar todas as sequências possíveis das tarefas inseridas pelo utilizador. A complexidade desta etapa depende no número de tarefas a serem permutadas.

A função ***permutation/2*** tem uma complexidade temporal fatorial, **O(n!)**, onde **n** é o número de tarefas. Isso significa que para um pequeno aumento no número de tarefas, o tempo de execução aumentará drasticamente.

Por exemplo, se o número de tarefas passar de 5 para 6, o número de permutações passa de 120 para 720.

Para um pequeno número de tarefas, esta abordagem pode ser viável, mas à medida que o número aumenta, torna-se rapidamente impraticável.

**Cálculo do Custo de Cada Sequência:**

Depois de gerar todas a permutações possíveis, o algoritmo procede para calcular o custo de cada sequência de tarefas para determinar a melhor sequência com o menor custo total.

Para cada sequência de tarefas, o algoritmo deve calcular o caminho e o custo total, o que envolve chamar o algoritmo A\* várias vezes (uma para cada tarefa na sequência).

O custo temporal desta etapa é multiplicativo em relação ao número de tarefas na sequência e à complexidade do algoritmo A\*. O A\* tem uma complexidade temporal de O(b^d), onde b é o fator de ramificação e d é a profundidade do menor caminho.

Portanto, o tempo total para calcular o custo de todas as sequências é **O(n! \* k \* t)**, onde **n** é o número de tarefas, **k** é o tempo médio para calcular o custo de uma tarefa individual, e **t** é o número de tarefas em uma sequência.

O custo de calcular todas as sequências aumenta com o número de tarefas e a complexidade do ambiente.

**Casos de teste**

Para iniciarmos a criação de cenários de teste precisamos de definir o que será medido:

**Número de tarefas:** Quantas tarefas diferentes estão a ser consideradas.

**Tempo de execução:** Quanto tempo o algoritmo leva para executar em função do número de tarefas.

Para observarmos como o nosso algoritmo de comporta decidimos definir 3 cenários diferentes:

* **Pequeno (2-3 tarefas):** Para entender o comportamento em uma escala menor.
* **Médio (4-7 tarefas):** Para identificar tendências ou problemas que começam a surgir.
* **Grande (8+ tarefas):** Para testar os limites da aplicabilidade do algoritmo.

**Considerações:**

Para cada cenário decidiu-se definir um ponto de origem próximo das tarefas e um ponto de origem distante.

Também decidiu-se fazer um estudo para tarefas mais ou menos próximas umas das outras e outro cenário em que as tarefas estão completamente dispersas pelos edifícios.

Para cada cenário terá sempre pelo menos uma tarefa diferente.

**Cenário pequeno (3 tarefas):**

**Ponto de origem próximo das tarefas && tarefas próximas umas das outras**

Ponto de origem:

Edifício C piso 1 X=2 Y=2

Tarefas:

Limpar sala C106

Vigilância piso 1 do edifício C

Pegar objeto da sala C103, entregar na sala C110

****

**Tempo de execução < 1s**

**Ponto de origem afastado das tarefas && tarefas próximas umas das outras**

Ponto de origem:

Edifício B piso 3 X=3 Y=2

Tarefas:

Limpar sala C106

Vigilância piso 1 do edifício C

Pegar objeto da sala C103, entregar na sala C110



**Tempo de execução =~ 14s**

**Tarefas afastadas umas das outras**

Ponto de origem:

Edifício C piso 1 X=2 Y=2

Tarefas:

Limpar sala D302

Vigilância piso 1 do edifício B

Pegar objeto da sala C401, entregar na sala C110

**Tempo de execução = 4min e 23s**

**Cenário médio (5 tarefas):**

**Ponto de origem próximo das tarefas && tarefas próximas umas das outras**

Ponto de origem:

Edifício C piso 1 X=2 Y=2

Tarefas:

Limpar sala C106

Limpar sala C104

Vigilância piso 1 do edifício C

Vigilância piso 1 do edifício B

Pegar objeto da sala C103, entregar na sala C110

**... muito tempo para obter o custo e a melhor sequência de tarefas ...**

**Tempo de execução > 10min**

**Ponto de origem afastado das tarefas && tarefas próximas umas das outras**

Ponto de origem:

Edifício B piso 3 X=3 Y=2

Tarefas:

Limpar sala C106

Limpar sala C104

Vigilância piso 1 do edifício C

Vigilância piso 1 do edifício B

Pegar objeto da sala C103, entregar na sala C110

**... muito tempo para obter o custo e a melhor sequência de tarefas ...**

**Tempo de execução > 10min**

**Tarefas afastadas umas das outras**

Ponto de origem:

Edifício C piso 1 X=2 Y=2

Tarefas:

Limpar sala C106

Limpar sala D302

Vigilância piso 1 do edifício B

Vigilância piso 4 do edifício C

Pegar objeto da sala D301, entregar na sala B202

**... muito tempo para obter o custo e a melhor sequência de tarefas ...**

**Tempo de execução > 10min**

**Cenário Grande (7 tarefas):**

**Ponto de origem próximo das tarefas && tarefas próximas umas das outras**

Ponto de origem:

Edifício C piso 1 X=2 Y=2

Tarefas:

Limpar sala C106

Limpar sala C104

Limpar sala B101

Vigilância piso 1 do edifício C

Vigilância piso 1 do edifício B

Pegar objeto da sala C103, entregar na sala C110

Pegar objeto da sala C110, entregar na sala C103

**... muito tempo para obter o custo e a melhor sequência de tarefas ...**

**Tempo de execução > 10min**

**Ponto de origem afastado das tarefas && tarefas próximas umas das outras**

Ponto de origem:

Edifício B piso 3 X=3 Y=2

Tarefas:

Limpar sala C106

Limpar sala C104

Limpar sala B101

Vigilância piso 1 do edifício C

Vigilância piso 1 do edifício B

Pegar objeto da sala C103, entregar na sala C110

Pegar objeto da sala C110, entregar na sala C103

**... muito tempo para obter o custo e a melhor sequência de tarefas ...**

**Tempo de execução > 10min**

**Tarefas afastadas umas das outras**

Ponto de origem:

Edifício C piso 1 X=2 Y=2

Tarefas:

Limpar sala C106

Limpar sala B203

Limpar sala C401

Vigilância piso 2 do edifício D

Vigilância piso 3 do edifício B

Pegar objeto da sala C103, entregar na sala C403

Pegar objeto da sala B302, entregar na sala D102

**... muito tempo para obter o custo e a melhor sequência de tarefas ...**

**Tempo de execução > 10min**

**Conclusões sobre os Cenários de Teste**

**Impacto da proximidade inicial:**

A proximidade do ponto de origem às primeiras tarefas tem um impacto significativo no tempo de execução e no custo total. Quando o robot começa mais perto das tarefas, o tempo de execução e o custo tendem a ser menores, uma vez que o caminho inicial é mais curto.

**Complexidade das tarefas de vigilância:**

As tarefas de vigilância, dada a sua natureza de percorrer um piso inteiro, são mais dispendiosas em termos de tempo e recursos computacionais. Isto é especialmente verdade em cenários maiores, onde são monitorizados mais pisos, o que pode aumentar exponencialmente o tempo e o custo de execução.

**Implicações das distâncias nas tarefas de recolha e entrega:**

Quando as distâncias entre as salas de recolha e de entrega são grandes, o custo e o tempo de execução aumentam significativamente. Isto deve-se ao aumento do caminho que o robot tem de percorrer, o que se torna ainda mais impactante em cenários maiores com mais tarefas.

**Escala de tarefas e tempo de execução:**

À medida que o número de tarefas aumenta, o tempo de execução e o custo total aumentam. Isto é particularmente evidente quando se passa de cenários pequenos para médios e grandes. Aumentar o número de permutações possíveis para a sequência de tarefas aumenta drasticamente o tempo necessário para calcular a melhor sequência, indicando que a escalabilidade é um desafio significativo para o algoritmo atual.

**Considerações Finais**

**A vigilância como uma tarefa crítica:**

A vigilância é identificada como a tarefa mais intensiva em termos de custo e tempo. Em cenários de média e grande dimensão, a inclusão de tarefas de vigilância aumenta drasticamente o tempo de execução, o que sugere a necessidade de otimizar ou limitar o número destas tarefas em cenários práticos ou de procurar algoritmos mais eficientes para a sua inclusão.

**Limitações de escalabilidade:**

A abordagem atual apresenta limitações significativas em termos de escalabilidade. À medida que o número de tarefas aumenta, o tempo de execução para encontrar a melhor sequência torna-se proibitivo, o que limita a utilidade prática do sistema para um grande número de tarefas.

**Equilíbrio dos tipos de tarefas:**

É crucial equilibrar os tipos de tarefas atribuídas ao robot. Diversificar demasiado ou concentrar demasiadas tarefas de alto custo (como a monitorização de vários pisos) pode não ser prático. A estratégia deve considerar uma combinação equilibrada que maximize a eficiência operacional do robot.

**Plano de atendimento de tarefas através de um algoritmo genético**

**Introdução**

Os algoritmos genéticos representam uma metodologia inspirada na evolução biológica e na genética para resolver problemas de otimização e pesquisa. São particularmente úteis em cenários complexos em que as soluções possíveis são numerosas e as estratégias convencionais não são eficientes. No contexto do planeamento de tarefas, os AG podem ser uma ferramenta poderosa para conceber planos de serviço eficientes, adaptáveis e otimizados.

A programação de tarefas em muitos ambientes operacionais e de produção é um desafio complexo devido à variedade de tarefas, restrições de tempo, recursos e interdependências. A utilização de um AG para este fim envolve a criação de uma população de soluções possíveis (indivíduos), cada uma representando uma sequência possível de tarefas. Estas soluções são depois melhoradas iterativamente através de processos inspirados na seleção natural, como a seleção, o cruzamento e a mutação.

No contexto deste projeto, a implementação de um algoritmo genético para planear o cumprimento de tarefas traz várias vantagens. Em primeiro lugar, ao modelar as tarefas o AG pode ajudar a identificar sequências de tarefas que otimizem a utilização dos recursos e minimizem o tempo total ou a penalização por atrasos. Além disso, com a capacidade de se ajustar a mudanças e restrições dinâmicas, o AG oferece uma solução robusta e flexível, o que é crucial em ambientes dinâmicos onde podem surgir novas tarefas ou as prioridades podem mudar rapidamente.

**Algoritmo genético vs Algoritmo por permutações**

**Exploração eficiente do espaço de soluções:**

Ao contrário da abordagem que consiste em gerar todas as sequências possíveis de tarefas e selecionar a melhor, que rapidamente se torna impraticável à medida que o número de tarefas aumenta (problema fatorial), os AG são capazes de explorar o espaço de soluções de forma mais eficiente, concentrando-se em áreas promissoras e ajustando as soluções de forma adaptativa.

**Adaptabilidade e flexibilidade:**

Os AG podem adaptar-se a alterações nas condições ou requisitos do problema. Por exemplo, se surgirem novas tarefas ou se as prioridades mudarem, o algoritmo pode gerar novas soluções sem a necessidade de começar do zero.

**Otimização contínua:**

Enquanto a abordagem de permutação procura uma solução ótima num conjunto finito e definido de soluções, os AGs trabalham com a ideia de melhoria contínua, encontrando frequentemente soluções satisfatórias mais rapidamente e continuando a melhorá-las ao longo do tempo.

**Estudo do Estado da Arte de métodos de planeamento de trajetória e movimento de robots que se movimentam numa superfície plana com obstáculos**

# **Introdução**

A geração da trajetória de robôs é uma tarefa com muitos usos, como, por exemplo, movimentação, montagem, robótica cirúrgica, serviços de limpeza, entrega e atendimento ao cliente e é cada vez mais usado em ambientes industriais e militares. O objetivo dessa geração é encontrar um caminho em que o robô se vá deslocar de um ponto ao outro de forma eficiente.

A pesquisa bibliográfica foi feita a partir do Google Scholar, como aconselhado, explorando em mais profundidade as primeiras pesquisas pois serem as mais citadas e, portanto, melhor desenvolvidas, excluindo as abordagens com braços robóticos a efetuarem movimentos 3D no espaço e só incluindo robots que se movimentam numa superfície plana com obstáculos.

# **Categorias de algoritmos de planeamento de trajetória**

De acordo com P. Raja e S. Puganzhenthi, o planeamento do caminho de movimento de um robot envolve determinar um caminho sem colisões de um ponto a outro enquanto otimizamos critérios como tempo, energia ou distância e existem duas principais categorias de algoritmos de planeamento de trajetória: off-line e on-line, e para cada uma delas, duas abordagens: clássica e evolucionária.

**Algoritmo Off-Line (ou global)**

Num algoritmo Off-Line, este cria o plano de trajetória antes que o robot se começa a mover. Todas as informações do trajeto são conhecidas antes que o robot comece o seu percurso, este tipo de algoritmo é mais adequado para ambientes onde o mapa é estático e não ocorrem mudanças com frequência.

Abordagem Clássica: uma abordagem onde o robot é representado como um ponto e são utilizados métodos como abordagem de mapa de caminho e de decomposição de células, construindo redes de percursos sem colisões.

Abordagem Evolucionária: Algoritmo Genético (**GA**: geração aleatória de percursos), como fazemos no nosso projeto, Otimização por Enxame de Partículas (**PSO**: inspirado no comportamento social), Otimização por Colónia de Formigas (**ACO**: baseado na forragem das formigas, promovendo convergência para o melhor trajeto, embora com um custo computacional alto) e Simulated Annealing (**SA**: imita o processo de arrefecimento de metais fundidos) são todos métodos utilizados.

**Algoritmo On-Line (ou local)**

Num algoritmo On-Line, ao contrário do que acontece no off-line, a geração de trajetórias é feita à medida que o robot se move no ambiente, feito com base nas informações sensoriais em tempo real, ajustando o seu percurso a qualquer obstáculo no seu caminho. Este tipo de planeamento é mais adaptável a ambientes de natureza mais dinâmica.

Abordagem Clássica: Potencial de Campo Artificial (APF), que envolve um robot pontual que se move a partir de cargas repulsivas e atrativas e o cone de colisão, que evade os obstáculos em tempo real.

Abordagem Evolucionária: Potencial de Campo Artificial, que usa Algoritmos Genéticos para derivar funções de campo potencial, PSO e ACO são integrados para otimizar o tempo computacional: PSO para conceitos de cone de colisão e ACO para planeamento global de percursos, enquanto que o APF é usado só para rotas locais.

Han-ye Zhang (<https://www.mdpi.com/2073-8994/10/10/450>), entra mais em detalhe nas abordagens escolhidas e caracteriza os algoritmos globais da seguinte forma:

**Abordagem Heurística**

Esta abordagem é uma forma de resolução de problemas baseada na experiência e no conhecimento, é uma maneira rápida de arranjar um problema menor ou de reduzir as opções. Han-ye Zhang apresenta os seguintes algoritmos:

Algoritmo de Djistrka: algoritmo de caminho mais curto num grafo direcionado, o ponto de partida é usado como centro. Mantém sempre dois conjuntos de vértices, movendo-os de um conjunto para outro para minimizar a soma dos pesos das arestas. Não é muito eficiente.

Algoritmo A\*: o algoritmo implementado no sprint anterior é uma extensão do algoritmo de Dijstrka e com uma função de avaliação *f(n) = g(n) + h(n)*, prioriza os nós com menor custo total e é então mais eficiente que o anterior. Muito usado em jogos eletrónicos e simulações de tráfico.

Algoritmo D\*: baseado também no Dijstrka, usa variáveis como Field D\* e Tetha\* para otimizar a sua eficiência e é pensado para ambientes mais complexos. Uma boa possibilidade para veículos de condução autónoma, onde o ambiente pode rapidamente mudar.

**Abordagem Inteligência Artificial**

Artificial Neural Network (ANN): este algoritmo mapeia o espaço percetual para espaço comportamental através de funções de energia. Apresenta desafios ao descrever ambientes complexos e aleatórios matematicamente, no entanto, oferece adaptabilidade e eficiência nos percursos com um nível de energia mínimo, mesmo que nem sempre o mais curto.

Algoritmo Genético (AG): utiliza codificação binária ou cromossómica para representar soluções, formando uma população inicial. É uma função destacada pela sua simplicidade, robustez e eficácia, embora suscetível a uma convergência prematura.

Particle Swarm Optimization (PSO): algoritmo inspirado no comportamento dos grupos de aves, este encontra o percurso através de iterações e destaca-se pela facilidade de implementação, alta precisão e rápida convergência.

**Conclusão**

A revisão identifica vários desafios e áreas que melhorar no planeamento de trajetória: objetivos de otimização, incertezas em sensores, previsão e controlo de movimento, desenvolvimento de modelos matemáticos para o planeamento on-line, eficiência computacional, falta de penalização no percurso off-line, entre outros. Ainda que existam algoritmos eficientes, a complexidade e natureza dinâmica dos percursos apresentam diversos desafios.

**Referências**

1. P. Raja, S.Pugazhenthi, “Optimal path planning of mobile robots: a review”<https://academicjournals.org/journal/IJPS/article-full-text-pdf/93D9A3918801.pdf>)
2. Han-ye Zhang, “Path Planning for the Mobile Robot: A Review” <https://www.mdpi.com/2073-8994/10/10/450>
3. Alaa Khamis, “Motion Planning” <https://pt.slideshare.net/AlaaKhamis/motion-planning>
4. Thi Thoa Mac, “Heuristic approaches in robot path planning: A survey” <https://www.sciencedirect.com/science/article/abs/pii/S0921889015300671>
5. Jianping Tu, Simon X. Yang, “Genetic Algorithm Based Path Planning for a Mobile Robot” <https://ieeexplore.ieee.org/abstract/document/1241759>