

**Universidade do Minho**

Escola de engenharia

**Inteligência artificial**

**Trabalho Prático**

Beatriz Salgado Fernandes, a100602;

Diogo Fontes Oliveira Silva Santos, a100714;

João Silva Loureiro, a100832;

Luís de Sá Ribeiro, a100608;

Martim Quintelas Pinto Félix, a100647;

Novembro de 2023

**Avaliação por pares:**

a100602: temos de dar uma nota a cada um

a100714:

a100832:

a100608:

a100647:

**Resumo**

Com este relatório, é pretendido ilucidar e descrever todo o processo feito relativamente ao projeto prático da unidade curricular de Inteligência Artifical.

Ao longo deste, vão ser mostrados os diferentes algoritmos de procura utilizados, tanto os de pesquisa informada, como os de pesquisa não informada, para que se possa verificar como diferentes variáveis e condições afetam os algoritmos, que podem dar um melhor, ou pior, resultado, consoante o custo e a heuristica (nos casos em que esta é utilizada). Será feita então uma analise detalhada de cada algoritmo, verificando a sua abordagem e a forma como se adapta ao problema em questão, para que seja possível notar quais as grandes diferenças entre os diferentes algoritmos e a sua eficiência dependo da situação onde se encontrem.

O relatório irá consolidar tudo o que foi aprendido ao longo do semestre, analisando futuras melhorias. O objetivo é proporcionar uma visão completa do desenvolvimento do projeto, dando a conhecer as suas funcionalidades e possíveis limitações.

**Índice**

Introdução4

Contextualização4

Descrição do problema4

Formulação do problema6

Estado inicial6

Estado/teste objetivo6 Operadores6

Custo da solução6

Implementação7

**2 Índice das figuras**

[Figura 1: Diferentes freguesias de Famalicão 5](file:///C:\Users\joaos\Downloads\IA-relatorio.docx#_Toc155200311)

[Figura 2: Dicionário do grafo 7](file:///C:\Users\joaos\Downloads\IA-relatorio.docx#_Toc155200312)

[Figura 3: Cálculo da heurística, tendo em conta Calendário como nó inicial 8](file:///C:\Users\joaos\Downloads\IA-relatorio.docx#_Toc155200313)

[Figura 4: Código para a representação gráfica do grafo 8](file:///C:\Users\joaos\Downloads\IA-relatorio.docx#_Toc155200314)

[Figura 5: Representação do grafo 9](file:///C:\Users\joaos\Downloads\IA-relatorio.docx#_Toc155200315)

[Figura 6: Criação da classe Estafetas 9](file:///C:\Users\joaos\Downloads\IA-relatorio.docx#_Toc155200316)

[Figura 7: Criação da classe Entrega 10](file:///C:\Users\joaos\Downloads\IA-relatorio.docx#_Toc155200317)

[Figura 8: Inicialização da procura em largura 10](file:///C:\Users\joaos\Downloads\IA-relatorio.docx#_Toc155200318)

[Figura 9: Loop que percorre os nodos 11](file:///C:\Users\joaos\Downloads\IA-relatorio.docx#_Toc155200319)

[Figura 10: Processo final, para reconstruir o caminho e calcular o custo 11](file:///C:\Users\joaos\Downloads\IA-relatorio.docx#_Toc155200320)

[Figura 11: conjunto e lista definido para resolução do algoritmo 11](file:///C:\Users\joaos\Downloads\IA-relatorio.docx#_Toc155200321)

[Figura 12: Loop principal do algoritmo 11](file:///C:\Users\joaos\Downloads\IA-relatorio.docx#_Toc155200322)

[Figura 13: Listas para resolução do algoritmo 12](file:///C:\Users\joaos\Downloads\IA-relatorio.docx#_Toc155200323)

[Figura 14: Função DFS 12](file:///C:\Users\joaos\Downloads\IA-relatorio.docx#_Toc155200324)

[Figura 15: Incremento da profundidade até dfs retornar True 12](file:///C:\Users\joaos\Downloads\IA-relatorio.docx#_Toc155200325)

[Figura 16: Inicialização do algoritmo 13](file:///C:\Users\joaos\Downloads\IA-relatorio.docx#_Toc155200326)

[Figura 17: Loop principal 13](file:///C:\Users\joaos\Downloads\IA-relatorio.docx#_Toc155200327)

[Figura 18: Caso em que não encontra um caminho até ao destino 13](file:///C:\Users\joaos\Downloads\IA-relatorio.docx#_Toc155200328)

[Figura 19: Inicialização do algoritmo 14](file:///C:\Users\joaos\Downloads\IA-relatorio.docx#_Toc155200329)

[Figura 20: Caso em que o estado atual é o estado objetivo 14](file:///C:\Users\joaos\Downloads\IA-relatorio.docx#_Toc155200330)

[Figura 21: Caso em que o nodo explorado ainda não é o estado objetivo 14](file:///C:\Users\joaos\Downloads\IA-relatorio.docx#_Toc155200331)

[Figura 22: Inicialização do algoritmo 15](file:///C:\Users\joaos\Downloads\IA-relatorio.docx#_Toc155200332)

[Figura 23: Ciclo do Algoritmo 15](file:///C:\Users\joaos\Downloads\IA-relatorio.docx#_Toc155200333)

# **Introdução**

**Contextualização**

Cada vez mais, e devido à situação que o mundo se encontra, é necessário procurar maneiras de tornar o nosso dia a dia o mais ecologicamente responsável, de forma a conseguir preservar o planeta durante muito mais tempo.

Tendo isto em conta, a indústria de distribuição, nos dias de hoje, ainda é um mercado que muitas das vezes não tem em consideração a escolha de meios mais amigos do ambiente. Por conseguinte, é proposto desenvolver um projeto que consiga, utilizando os diversos algoritmos aprendidos nas aulas, distribuir encomendas da maneira mais eficiente possível, tendo em conta os diversos caminhos que pode seguir e os diferentes veículos que pode utilizar, mas também, tendo em conta a satisfação do cliente, proporcionando o melhor tempo de entrega possível.

**Descrição do problema**

Para este projeto ser executado e para ser possível verificar a sua atuação num contexto concreto, foi nos dada a liberdade de utilizarmos um mapa escolhido pelo próprio grupo. Decidimos então utilizar algumas das freguesias da cidade de Vila Nova de Famalicão.

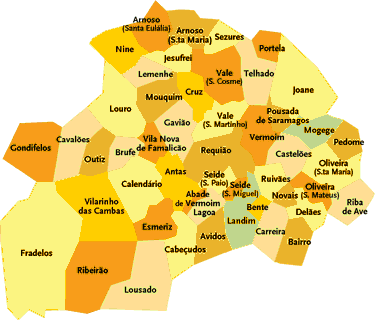


Figura 1: Diferentes freguesias de Famalicão

Neste mapa, cada nodo representa uma freguesia, onde, posteriormente, cada posição, relativa à respetiva freguesia, no grafo, vai servir para calcular a heurística. Os estafetas vão se poder mover entre estes nodos para chegar ao destino pretendido da entrega.

Para a movimentação dos estafetas, e para estes conseguirem utilizar os caminho mais benéficos, vamos utilizar diferentes algoritmos de procura, nomeadamente no que toca à pesquisa não informada, vamos utilizar os algoritmos BFS, DFS, Bidirecional, Custo Uniforme e Iterative Deepening, no que toca à pesquisa informada, vamos utilizar os algoritmos Greedy e A\*.

É de notar ainda que para os diferentes estafetas e para as entregas, utilizamos dois CSV com essas informações, para assim facilitar a sua manipulação e o seu uso.

# **Formulação do problema**

**Estado inicial**

O estado inicial representa a localização do estafeta ao começar as entregas, especificando o tipo de veículo utilizado, bem como o conjunto de encomendas que ele vai entregar.

**Estado/teste objetivo**

O estado objetivo consiste em entregar todas as encomendas, dentro dos prazos estabelecidos (minimizando as penalizações no ranking dos diferentes estafetas e maximizando a satisfação do cliente).

**Operadores**

***Mover Estafeta->*** Deslocação do estafeta para a próxima entrega.

***Entregar Encomenda->*** Encomenda é entregue e classificação do estafeta pode ser feita pelo cliente.

***Atualizar Classificação do Estafeta->*** Atualizar a classificação do estafeta, com base na avaliação feita pelo cliente e pela respetiva penalização, caso seja aplicada.

**Custo da solução**

O custo da solução, resulta na soma de todos os custos associados entre dois nodos, desde o nodo inicial, até ao nodo destino.

1. **Implementação**

**Criação do grafo**

Começando por abordar a raiz do programa. O grafo é criado a partir do dicionário “graph\_dict”. Tal como podemos ver na figura abaixo, neste dicionário, cada chave representa um nó, ou seja, uma freguesia, tendo associado a este nó, as diferentes freguesias que são vizinhas desta, com o respetivo custo da deslocação.

Uma imagem com texto, captura de ecrã, software, Software de multimédia

Descrição gerada automaticamente

Figura 2: Dicionário do grafo

Para a representação do grafo, e para esta ficar completa torna-se imprescindível associar a cada nó uma heurística, para resolução dos algoritmos de procura informada. A heurística de cada freguesia é calcula através da distância euclidiana entra a posição de cada nó e a posição específica que queremos considerar como nó inicial.

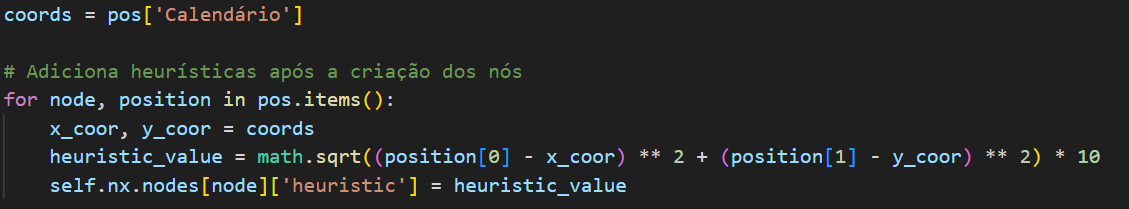


Figura 3: Cálculo da heurística, tendo em conta Calendário como nó inicial

Estando agora já definidas as diferentes freguesias e os seus respetivos custos e heurísticas, podemos passar para a visualização do grafo. Para esta parte foram usadas as bibliotecas NetworkX e a biblioteca matplotlib. Utilizando a ajuda da função “nx.spring\_layout” são calculadas as posições dos nós do grafo para a sua visualização. No que toca às arestas do grafo, são desenhadas usando “nx.draw” e o seu custo é adicionado utilizando o “nx.draw\_networkx\_edge\_labels”

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

Descrição gerada automaticamente

Figura 4: Código para a representação gráfica do grafo

Uma imagem com file, diagrama, círculo

Descrição gerada automaticamenteTendo isto tudo definido obtemos a seguinte representação do grafo.

Figura 5: Representação do grafo

**Estafetas**

Para os estafetas, a informação destes é obtida a partir do dataset Estafetas, onde se encontra informações acerca do id, do ponto de partida e da lista de encomendas de cada um deles. Esta informação acaba por ser processada no Estafetas.py, que lê linha a linha o respetivo csv, e cria a classe Estafetas.

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

Descrição gerada automaticamente

Figura 6: Criação da classe Estafetas

**Entregas**

As entregas funcionam de forma semelhante aos estafetas, sendo a informação destas obtida através do dataset Entregas, onde é apresentada informações do identificador, o destino, o peso (em Kg), o volume (em cm^3) e ainda o tempo (em min). Da mesma maneira que os estafetas, o csv das entregas é manipulado em Entregas.py, onde é definida a classe Entrega, com os seus atributos.

Uma imagem com texto, Software de multimédia, software, captura de ecrã

Descrição gerada automaticamente

Figura 7: Criação da classe Entrega

**Algoritmos**

**Procura em largura**

O algoritmo de procura em largura corresponde à função “procura\_BFS”. Esta função inicialmente define o conjunto “visited”, que vai permitir ao algoritmo perceber quais são os nós que já foram visitados, define também, uma queue que permite ter os atuais nós que estão a ser expandidos, define a lista “expansão” que regista a ordem de expansão, uma vez visitadas, do nodo inicial até ao nodo final e ainda um dicionário “parent”, que em cada posição do nó coloca o seu respetivo pai.

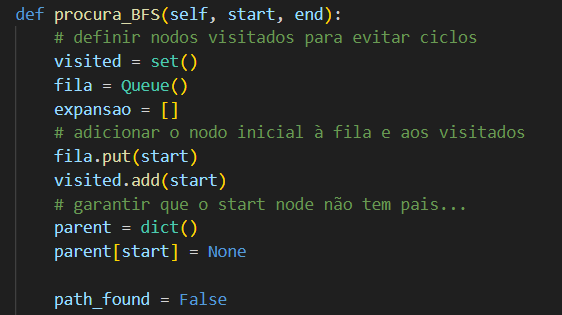


Figura 8: Inicialização da procura em largura

Estando os conjuntos e listas auxiliares definidas, o processo começa colocando o nó inicial na queue e atualizando no conjunto visited, o valor do nó para visitado. Durante cada iteração, o nó atual, que neste primeiro caso é o nó inicial, é retirado da fila e é adicionado à lista de expansão, sendo os seus nodos adjacentes adicionados à queue. Este loop vai ocorrer enquanto a fila não estiver vazia e o caminho ainda não tenha sido encontrado.

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

Descrição gerada automaticamente

Figura 9: Loop que percorre os nodos

No final deste processo, o caminho é construído utilizando o dicionário já referido. Calculando-se o custo deste caminho através da função auxiliar calcula\_custo, que soma o custo de todos os nós até ao atual. É retornado um tuplo contendo o caminho, o custo total e ainda a ordem de expansão dos nós.

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

Descrição gerada automaticamente

Figura 10: Processo final, para reconstruir o caminho e calcular o custo

**Procura em profundidade**

A função procura\_DFS implementa o algoritmo de busca em profundidade (DFS). De forma semelhante à procura em largura, neste algoritmo é utilizado um conjunto, designado de "visitados", que acompanha os nós explorados e uma lista "ordem\_expansao" para registar a ordem de expansão.

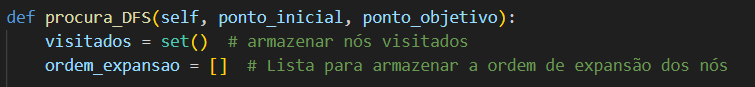


Figura 11: conjunto e lista definido para resolução do algoritmo

A função DFS explora os nós em profundidade, adicionando o nó atual a "visitados" e a "ordem\_expansao". Se o nó atual é o ponto objetivo, a função retorna o caminho e o custo. Caso contrário, ela chama recursivamente a DFS para explorar vizinhos não visitados.

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

Descrição gerada automaticamente

Figura 12: Loop principal do algoritmo

O Algoritmo é terminado quando o caminho é encontrado. O resultado é um tuplo com o caminho, a ordem única de expansão e o custo total do caminho.

**Aprofundamento iterativo**

A função iterative\_deepening\_dfs implementa o algoritmo de Aprofundamento Iterativo. Inicialmente, são definidas listas para armazenar o caminho da solução “path” e a ordem de expansão dos nós “order\_of\_expansion”.

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

Descrição gerada automaticamente

Figura 13: Listas para resolução do algoritmo

A função dfs realiza a busca em profundidade limitada até à depth, passada como argumento. Cada chamada adiciona o nó atual à lista order\_of\_expansion. Se o nó atual é o destino, o caminho é registado e a função retorna “True”. Caso não seja o nó destino, se a profundidade permitir, a função é chamada recursivamente para os nós filhos, dando como resultado “False”, se o destino não for encontrado até a profundidade específica.

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

Descrição gerada automaticamente

Figura 14: Função DFS

O loop do Aprofundamento Iterativo inicia a profundidade como 0 e incrementa gradualmente até encontrar o destino.

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

Descrição gerada automaticamente

Figura 15: Incremento da profundidade até dfs retornar True

No final, é gerada uma lista única de nós expandidos “unique\_list”. O resultado consiste num tuplo com o caminho da solução (invertido), a lista única de expansão e a profundidade em que a solução foi encontrada.

**Custo Uniforme**

A função custoUniforme implementa o algoritmo de custo uniforme (UCS). Para implementar o algoritmo é utilizado auxiliarmente uma fila de prioridade, um conjunto de nodos designado por “visitado” que guardas o valor dos nodos já percorridos e uma lista de expansão, que no final irá mostrar o caminho resultado.

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

Descrição gerada automaticamente

Figura 16: Inicialização do algoritmo

Uma imagem com texto, captura de ecrã, software, Software de multimédia

Descrição gerada automaticamenteEste algoritmo, em cada iteração remove o nó com o menor custo acumulado, até ao momento e é marcado como visitado e adicionado ao caminho percorrido. Se o nó removido for o nó de destino, o algoritmo retorna o tuplo com o custo acumulado, o caminho percorrido até o nó de destino e a ordem de expansão dos nós. Caso este nó não seja o destino, o algoritmo expande os seus vizinhos, adicionando-os à fila de prioridade e realiza novamente o mesmo processo, até encontrar o nó destino, ou até explorar todos os caminhos possíveis.

Figura 17: Loop principal

Se no final do algoritmo, ocorrer que nenhum caminho até ao destino seja descoberto, a função irá retornar um tuplo com custo infinito, uma lista vazia para o caminho e uma lista vazia para a ordem de expansão.

Figura 18: Caso em que não encontra um caminho até ao destino

**Gulosa**

Inicialmente, é criada a lista fronteira, que vai ser usada para colocar os nós que estão a ser visitados, sendo ordenada pela heurística associada a cada nó, para além desta lista, definimos ainda o conjunto “explorados”, que guarda os nós já visitados.

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

Descrição gerada automaticamente

Figura 19: Inicialização do algoritmo

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

Descrição gerada automaticamenteDurante o processo, o algoritmo seleciona o nó mais promissor da fronteira, ou seja, aquele com a menor heurística. Caso este nó seja o destino, a função auxiliar “reconstruir\_caminho” e chamada e retorna o caminho percorrido.

Figura 20: Caso em que o estado atual é o estado objetivo

Caso o nó atual não seja o destino, ele é adicionado aos explorados, e os seus vizinhos não explorados são inseridos na fronteira. Este ciclo continua até que o destino seja alcançado ou todos os caminhos possíveis tenham sido explorados.

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

Descrição gerada automaticamente**

Figura 21: Caso em que o nodo explorado ainda não é o estado objetivo

**A\***

Para finalizar, no que toca a algoritmos, temos o algoritmo A\*. Que acaba por ser um algoritmo muito semelhante à gulosa, com a única diferença de este utilizar também o custo, para além da heurística. O algoritmo mantém uma fila de prioridade (fila\_prioridade) ordenada pelo custo total acumulado até o nó atual, que inclui o custo real e uma heurística. O conjunto “visitados” acaba por controlar os nós já explorados, e a lista “expansão” guarda a ordem de expansão dos nós.

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

Descrição gerada automaticamente

Figura 22: Inicialização do algoritmo

Enquanto existirem nós na fila de prioridade, mesmo já tendo encontrado o destino, o ciclo continua. A cada iteração, o nó de menor custo é retirado da fila e caso ainda não tenha sido visitado, é adicionado ao conjunto, ao caminho e à expansão. Se o nó for o destino, a função retorna o custo total, o caminho e a ordem de expansão. Posteriormente, os vizinhos do nó atual são explorados, e os seus custos totais, que representa o custo até ele e a sua heurística, são adicionados à fila de prioridade.

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

Descrição gerada automaticamente**

Figura 23: Ciclo do Algoritmo

**Conclusão**

Com este trabalho, conseguimos perceber como é que seria possível aplicar todo o conhecimento adquirido ao longo do semestre, num contexto mais concreto e como pode ser implementado para criar um futuro mais sustentável e ecológico. Conseguimos perceber melhor a importância que os algoritmos estudados têm no dia a dia, sem darmos contas dele.

No que toca às funcionalidade presentes no nosso trabalho, conseguimos implementar tudo o que era previsto, acabando por ficar um trabalho sólido.

Tenho de acabar.