Universidade do Minho

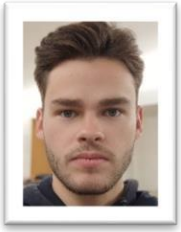
Licenciatura em Engenharia Informática

Relatório de Inteligência Artificial – 2 Fase

Grupo 31

Unidade Curricular: Inteligência Artificial [J305N3]

Coordenador: Professor Doutor Paulo Jorge Freitas Oliveira Novais



Nome: Carlos Filipe Almeida Dias

Número: 93185

Contacto: a93185@alunos.uminho.pt

Curso: Licenciatura em Engenharia Informática, Universidade do Minho

Nome: José Pedro Martins Magalhães

Número: 93273

Contacto: a93273@alunos.uminho.pt

Curso: Licenciatura em Engenharia Informática, Universidade do Minho

Nome: Francisco Reis Izquierdo

Número: a93241

Contacto: a93241@alunos.uminho.pt

Curso: : Licenciatura em Engenharia Informática, Universidade do Minho

Nome: Duarte Augusto Rodrigues Lucas

Número: a89526

Contacto: a89526@alunos.uminho.pt

Curso: : Licenciatura em Engenharia Informática

Ano Letivo 2021/2022

Índice

Índice de Figuras

Índice de Tabelas

1 Introdução

No âmbito da Unidade Curricular de Inteligência Artificial foi-nos proposto o desenvolvimento de um sistema de recomendação de circuitos de entrega de encomendas.

Este relatório pretende apresentar o raciocínio usado para a resolução do problema proposto no instrumento de avaliação da segunda fase. Este instrumento pretende estimular o uso de técnicas de formulação de problemas, a aplicação de diversas estratégias para a resolução de problema com o uso de algoritmos de procura e o desenvolvimento de mecanismos de raciocínio adequados a esta problemática.

No presente relatório é descrita as diversas ponderações tomadas pelo grupo de trabalho, bem como toda a análise, implementação e conclusões que o mesmo efetuou ao longo de todo o percurso.

Posto isto, será necessário através de procura não informada e informada, a seleção de circuitos de entrega de encomendas que tenham em conta as seguintes considerações:

- Estado Inicial – ponte de recolha dos estafetas, em particular no centro de distribuição da Green Distribution;
- Pontos de Entrega – local de entrega das encomendas;
- Estado Final – Regresso ao centro de distribuição da Green Distribution.

É também de realçar que, dada a informação presente no enunciado acerca da linguagem de programação a ser escolhida no âmbito do desenvolvimento do projeto, o grupo optou pela linguagem de programação Java, dada a experiência, simplicidade e conformismo do grupo.

2 Abordagem ao Problema

2.1 Organização da solução

Dado o enunciado proposto, uma vez analisada toda a informação relevante do mesmo, conseguimos identificar vários aspetos importantes que devem ser analisados de forma a resolver o problema apresentado. Assim, de entre os vários aspetos importantes, o primeiro a considerar e a ter como ponto de partida é o que o problema pretende abordar, isto é, qual a problemática que deve ser trabalhada e pensada de forma a obter a solução pretendida.

Com isto, a problemática supramencionada consiste em resolver o problema de pesquisa para circuitos de forma a criar um sistema de recomendação de circuitos de entrega de encomendas.

Posto isto, a abordagem ao problema passa por formular a representação do mundo real, mais concretamente a "cidade", de uma forma abstrata e que permita expandir a partir disto, a resolução do problema. Desta forma, tal como mencionado no enunciado, decidimos representar a abstração previamente mencionada em forma de grafo uma vez que esta abordagem é típica para a problemática em questão, mas também dado que a informação a apresentar, é feita de forma precisa, concisa e acima de tudo simples.

Ora o grafo criado pelo grupo foi pensado e planificado de forma a ser fácil a sua representação e manipulação de informação que nele se encontra, bem como tendo em conta as possíveis heurísticas que podem ser abordadas e que são apresentadas. O grafo proposto tem as seguintes definições:

→ Lista de todos os vértices/nodos, sendo que cada vértice representam as várias localizações.

→ Lista de todas as arestas, sendo que cada aresta representa as possíveis ligações entre localizações.

→ Para cada vértice existe ainda o custo (distância) estimado à localização *GreenDistribution*.

Grafo Geral**:**

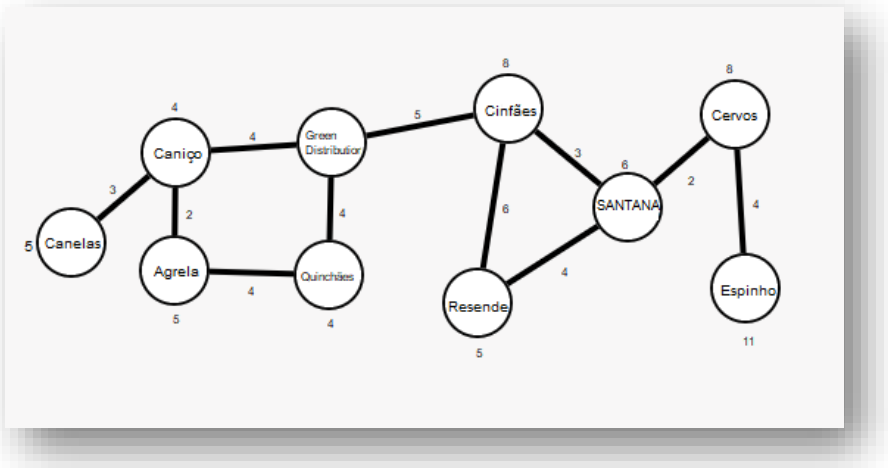


Figura 1: Grafo

3 Formulação do Problema

Dada a representação do grafo desenvolvida na abordagem do problema, o próximo passo passou por formular o problema no contexto em que se insere. Uma vez que o contexto se trata de pesquisa para circuitos de entrega de encomendas, é notável que o que se pretende desenvolver de uma forma mais genérica e crua é dada uma entrega a ser efetuada numa dada localização e partindo da origem *GreenDistribution*, obter um circuito que permita ir da origem ao destino.

Além disto e antes de detalhar a formulação do problema, há alguns aspetos a ter em consideração, de forma a obter uma melhor perceção e análise do problema. Com isto temos:

- Ambiente: O ambiente em que se insere o problema é determinístico uma vez que este é totalmente observável.
- Tipo de problema: Problema de estado único uma vez que o agente sabe exatamente em que localização estará.
- Solução: A solução gerada será uma sequência, isto é, será o caminho constituído por todas as localizações por onde se passou desde a origem até ao destino, sequencialmente.

Posto isto, podemos então formular o problema nas várias vertentes, de forma a formalizar o mesmo como um problema de pesquisa.

Estado inicial: *Green Distribution;*

Estado Objetivo: Localização do destino da encomenda;

Estados: As várias localizações;

Operações: Conduzir entre as localizações em diferentes transportes.

Solução:

- O custo da solução é determinado pelo custo de cada caminho, que reflete uma medida de desempenho da solução e transporte (neste caso o custo será a distância entre localizações, bem como o tempo entre as mesmas).
- Um caminho: Uma ou mais localizações até ao destino.

3.1 Algoritmos implementados

Após a formulação do problema como um problema de pesquisa, este pode ser resolvido tendo em conta diversos métodos e logísticas, havendo diferenças consoante a abordagem tomada. Com isto, a solução (que é o caminho pretendido a encontrar) do problema apresentado e formalizado, passa por tomar a decisão de que tipo de estratégia de pesquisa queremos abordar bem como qual o algoritmo a implementar que terá como base o tipo de estratégia de pesquisa abordado.

Dado isto, a solução depende acima de tudo das especificações do algoritmo de pesquisa escolhido que foi implementado, havendo diferenças em vários aspetos para diferentes algoritmos de pesquisa.

Com a formalização do problema apresentada, são propostos dois tipos de estratégias de pesquisa, para os quais desenvolvemos e implementámos algoritmos já conhecidos:

- **Pesquisa Informada: Pesquisa Primeiro em Profundidade, Pesquisa Primeiro em Largura e Busca Iterativa Limitada em Profundidade.**
- **Pesquisa Não Informada. A* e Pesquisa Gulosa.**

Apresentamos de seguida uma breve comparação dos dois tipos de estratégias de pesquisa supramencionados:

Base para comparação	**Pesquisa Informada**	**Pesquisa Não Informada**
Conceito	Usa conhecimento para encontrar as etapas para a solução.	Nenhum uso de conhecimento
Eficiência	Altamente eficiente, consome menos tempo e custo.	A eficiência é mediadora
Custo	Baixo	Comparativamente alto
atuação	Encontra a solução mais rapidamente	A velocidade é mais lenta que a pesquisa informada
Algoritmos	Pesquisa A* e Pesquisa Gulosa	Pesquisa primeiro em profundidade, Pesquisa primeiro em largura e Busca Iterativa Limitada em Profundidade.

Tabela 1: Pesquisa Informada vs Pesquisa Não Informada

Desta forma e como pedido, descrevemos o funcionamento e a nossa implementação de cada algoritmo de procura supramencionado. É também importante realçar o facto de que a explicação do funcionamento de cada algoritmo terá por base a árvore de procura do mesmo. Esta árvore de procura é nada mais que uma forma de simplificar o grafo representado em formato de árvore dado um vértice de origem como ponto de partida, com o intuito de simplificar o processo da iteração de pesquisa que o algoritmo realiza, sendo que a árvore de procura mencionada é constituída pelos seguintes aspetos:

- **Raiz**: A raiz da árvore representa o vértice de origem de onde o algoritmo parte para iterar sobre o grafo.
- **Nodo**: Cada nodo representa um outro vértice do grafo.
- **Ramos**: Cada ramo representa a aresta que une o vértice pai e o vértice filho (adjacente).

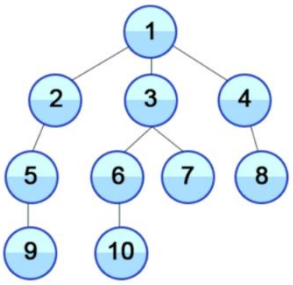


Figura 2: Exemplo de Árvre de Pesquisa

Pesquisa Primeiro em Largura:

Como funciona:

- Este algoritmo de pesquisa é considerado uma pesquisa não informada e tal como o nome indica, realiza uma pesquisa em largura, isto é, expande sempre todos os vértices de menor profundidade, pelo que dado um vértice, expande sempre primeiro os vértices adjacentes a este antes de avançar na profundidade da árvore de pesquisa.

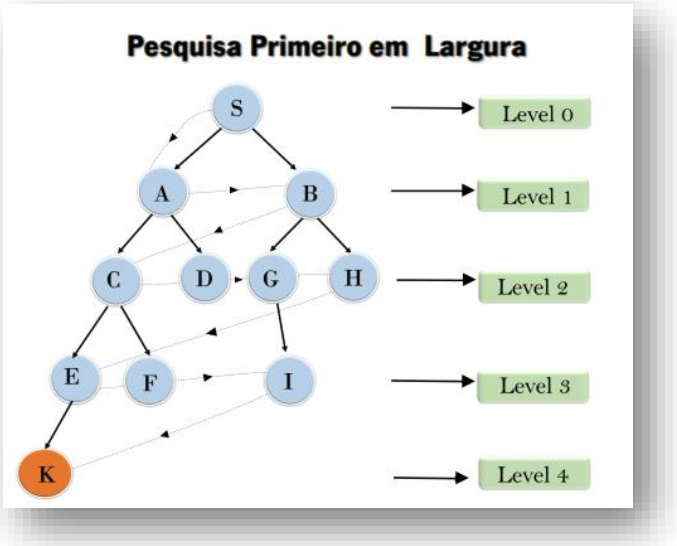


Figura 3: Pesquisa Primeiro em Largura

Considerações:

- **Complexidade Temporal/ Espacial**: É de realçar que quanto maior for o nível de ramificação (o número de vértices adjacentes), maior será o tempo da pesquisa uma vez que teremos várias arestas por pesquisar e vértices por expandir. Assim, o algoritmo tende a demorar bastante tempo para soluções cuja profundidade é elevada e além disso é bastante dispendioso em termo de memória, uma vez que armazena os vários vértices em memória.
- **Completude**: Com o ponto supramencionado, é de realçar que a solução só é obtida caso o número de vértices adjacentes seja finito.
- **Aplicabilidade**: Dado o ponto supramencionado, o algoritmo é útil para problemas de pequena dimensão, ou seja, problemas que não exijam ao algoritmo demorar muito tempo bem como ocupar muito espaço.

- **Otimalidade**: Caso o custo de cada aresta seja uniforme, o algoritmo ao encontrar a solução esta é garantidamente a melhor.
- **Nota**: O algoritmo deve garantir que, nenhum vértice ou aresta seja, visitado mais de uma vez, de forma a evitar possíveis ciclos e o algoritmo ficar "preso" nestes.

Implementação:

- A implementação do algoritmo acima descrito passou por, dado um vértice de origem, iterar sobre o primeiro vértice da lista dos vértices adjacentes. A cada vértice visitado, verificamos se é o destino, caso não seja então, adicionamos os seus vértices adjacentes a uma lista de adjacência, adicionamos o próprio vértice à lista de visitados, retiramos o vértice da lista de adjacência, guardamos numa lista de pais e filhos os seus vértices adjacentes associando-os ao vértice em questão, como se tratasse de um pai com vários filhos e por fim itera-se novamente sobre o primeiro vértice da lista de adjacentes. Quando chegamos ao vértice pretendido, então começo a construir o meu caminho, uma lista de vértices. Nela será adicionado primeiro o vértice final, depois através da lista de pais e filhos, verifica-se o seu vértice pai adicionando-o também ao caminho, assim sucessivamente até chegarmos à origem. Após chegarmos à origem e esta ter sido adicionada ao caminho, invertemos a ordem do caminho e este é devolvido.

Pesquisa Primeiro em Profundidade:

Como funciona:

- Este algoritmo de pesquisa é considerado uma pesquisa não informada que avança através da expansão do primeiro vértice adjacente da árvore de pesquisa, e incrementa a profundidade cada vez mais da forma supramencionada, até que o vértice pretendido seja alcançado ou até que se depare com um vértice que não possua vértice adjacentes. Assim, tende a expandir sempre um dos vértices mais profundos da árvore de pesquisa.

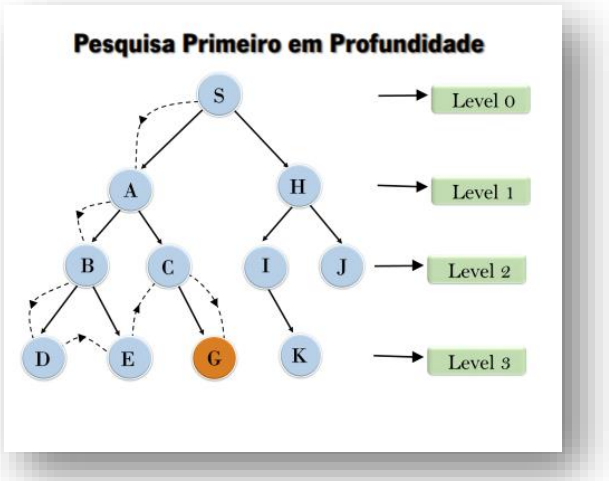


Figura 4: Pesquisa Primeiro em Profundidade

Considerações:

- **Complexidade Temporal/ Espacial**: Uma vez que apenas um dos vértices de maior profundidade é expandido a cada iteração até ser encontrada a solução, o algoritmo é relativamente rápido e ocupa pouca memória.
- **Completude**: Em contrapartida ao ponto acima referido, caso a árvore de pesquisa tenha profundidade infinita, a solução poderá recair numa profundidade dita "infinita" pelo que o algoritmo não consegue contornar esta problemática, podendo não encontrar a solução.
- **Aplicabilidade**: Com isto, o algoritmo é útil para problemas com várias soluções, dado que cada solução não requer muito tempo (caso a profundidade não seja infinita) ou memória.
- **Otimalidade**: Dado que o algoritmo ao realizar a pesquisa expande sempre um dos vértices mais profundos da árvore de pesquisa, devolve a primeira solução que encontra, podendo esta não ser a melhor.
- **Nota**: O algoritmo deve garantir que, nenhum vértice ou aresta seja, visitado mais de uma vez, de forma a evitar possíveis ciclos e o algoritmo ficar "preso" nestes.

Implementação: A implementação proposta pelo grupo consiste em, dado um vértice de origem, iterar sobre o primeiro vértice adjacente a este incrementando a profundidade e por cada vértice visitado, adicioná-lo a uma lista que contenha todos os vértices já visitados, de forma a evitar possíveis ciclos, sendo este processo feito recursivamente em cada vértice em que se encontra. No final, a solução será um caminho desde o vértice de origem até ao vértice de destino, que foi construído sequencialmente.

Busca Iterativa Limitada em Profundidade:

Como funciona:

- Este algoritmo de pesquisa é considerado uma pesquisa não informada e faz uso das propriedades descritas do algoritmo de Pesquisa Primeiro em Profundidade, pelo que além disso combina ainda um limite de profundidade que é incrementado de forma iterativa até encontrar o vértice pretendido. Assim, a profundidade limite inicia-se a zero e a pesquisa efetuada tem como base o algoritmo de Pesquisa Primeiro em Profundidade e caso não seja encontrada a solução, é incrementada a profundidade e repetindo o processo descrito, agora para os vértices desde a raiz até aos vértices da profundidade estabelecida.

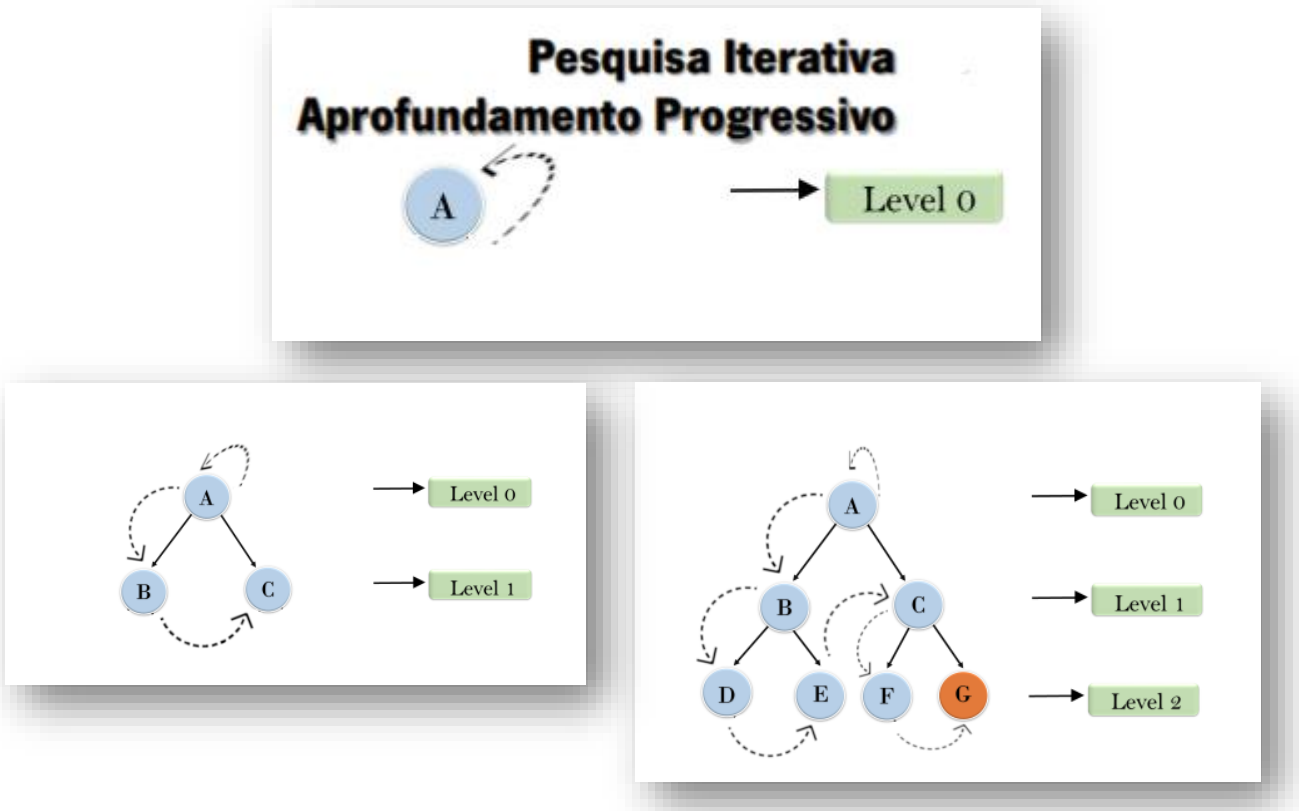


Figura 5: Busca Iterativa Limitada em Profundidade

Considerações:

- **Complexidade Temporal/ Espacial**: Dadas as especificações acima descritas sobre o funcionamento do algoritmo, é de realçar que o algoritmo tende a demorar bastante tempo para soluções cuja profundidade é elevada, pelo que a profundidade será incrementada várias vezes e além disso é bastante dispendioso em termos de memória, uma vez que armazena os vários vértices em memória.
- **Completude**: Dado que o algoritmo consegue controlar a profundidade da árvore de pesquisa a que está disposto a pesquisar e o faz de forma iterativa, este consegue sempre encontrar a solução, caso o fator de ramificação seja finito (número de vértices adjacentes).
- **Otimalidade**: Caso o custo de cada aresta seja uniforme, o algoritmo ao encontrar a solução esta é garantidamente a melhor.
- **Aplicabilidade**: Contrariamente ao que se poderia esperar, o algoritmo é útil para problemas de grande dimensão e não se sabe a profundidade da solução, ou seja, problemas que possam exigir ao algoritmo demorar muito tempo bem como ocupar muito espaço.
- **Nota**: O algoritmo deve garantir que, nenhum vértice ou aresta seja, visitado mais de uma vez, de forma a evitar possíveis ciclos e o algoritmo ficar "preso" nestes.

Implementação: A implementação proposta pelo grupo consiste em, dado um vértice de origem e um limite para a profundidade, gerar a lista de vértices de adjacentes e para cada um destes, aplicar o algoritmo de Pesquisa Primeiro em Profundidade até ao limite de profundidade imposto, aplicando este processo recursivamente e decrementado a cada chamada recursiva a profundidade que é passada como argumento. Caso este argumento chegue ao valor zero, a profundidade limite foi atingida, pelo

que deve ser efetuado o algoritmo Pesquisa Primeiro em Profundidade agora para o seguinte vértice adjacente onde o algoritmo se encontra. Caso todos os vértices até à profundidade limite estabelecido sejam aplicados ao algoritmo Pesquisa Primeiro em Profundidade, então a profundidade limite é incrementado e o algoritmo Busca Iterativa em Profundidade Limitada é efetuado novamente com o novo limite de profundidade. Caso o vértice de destino seja alcançado é devolvida a solução que é um caminho de profundidade igual à profundidade estabelecida aquando da pesquisa efetuado pelo algoritmo.

A*(A Estrela):

Como funciona:

- Este algoritmo de pesquisa é considerado uma pesquisa informada, que tem como estratégia a busca de melhor escolha, pois avalia os vértices através da combinação de duas heurísticas, o custo para alcançar o vértice visitado *g(n)*, e o custo para ir do vértice visitado ao vértice de destino *h(n)*. Assim é realizado a pesquisa de caminhos mínimos, evitando expandir caminhos que são caros utilizando funções heurísticas, ou seja, a seleção dos vértices é baseada na distância a partir do vértice de início mais a distância aproximada até ao vértice de destino. Essa estimativa de aproximação pode ser representada pela função:

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

f(n) = Custo do caminho total estimado.
g(n) = Custo do caminho do vértice inicial até ao vértice n.
h(n) = Custo do caminho estimado do vértice n até vértice de destino.

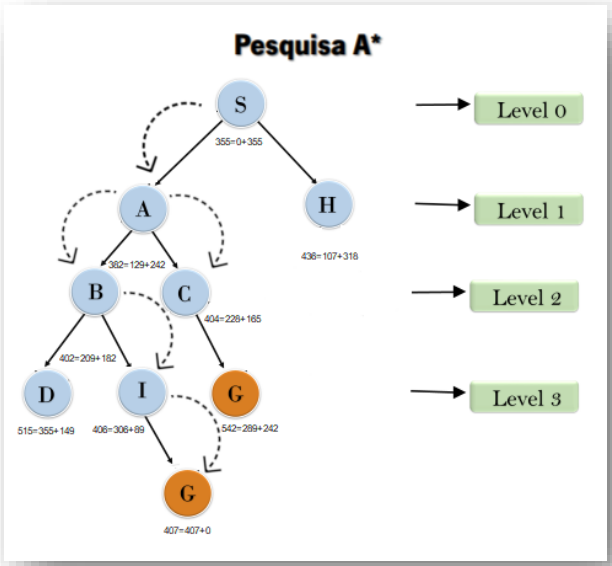


Figura 6: A Estrela

Considerações:

- **Complexidade Temporal/ Espacial**: É de realçar que quanto maior for o nível de ramificação (o número de vértices adjacentes), maior será o

tempo da pesquisa uma vez que teremos várias arestas por pesquisar e vértices por expandir. Assim, o algoritmo tende a demorar bastante tempo para soluções cuja profundidade é elevada e além disso é bastante dispendioso em termo de memória, uma vez que armazena os vários vértices em memória.

- **Completude**: Uma vez que o algoritmo combina as heurísticas acima descritas, encontra sempre uma solução.
- **Otimalidade**: Dadas as heurísticas descritas, a solução passa por ser o caminho com menor custo associado, uma vez que o desenvolvimento da solução é refinado pelas heurísticas, pelo que a solução obtida é sempre a melhor.
- **Aplicabilidade**: O algoritmo descrito é útil para problemas em que temos mais informação para além do que possa ser visível à primeira vista, tornando assim a pesquisa informada. Para além disso, dadas as complexidades descritas, o algoritmo consegue ser útil para problemas de pequenas dimensões e ser mais preciso dados o teor das heurísticas usadas.

Implementação: A implementação proposta pelo grupo consiste em iterar sobre os vértices do grafo, guardando os visitados numa lista de modo a evitar ciclos.

A partir do nodo atual, que inicialmente é o nosso destino, gera-se uma lista com os vértices que lhe são adjacentes sendo estes adicionados à lista de todos os nodos que faltam visitar, se estes já se encontrarem nessa lista, então compara-se o valor das heurísticas (apenas a distância estimada), o valor das estimativas até à Green Distribution, escolhendo-se o menor.

Depois, escolhe-se dessa lista o nodo com o menor valor da heurística e guardamos numa estrutura qual o seu predecessor e atualizamos o nodo atual.

Quando chegamos ao nodo da Green Destribution a iteração pára e cria-se um ciclo que percorre todos os predecessores da Green Distribution adicionando-os ao caminho.

Pesquisa Gulosa:

Como funciona:

- Este algoritmo de pesquisa é considerado uma pesquisa informada e tenta expandir o vértice que parece mais próximo do vértice de destino com base na estimativa feita pela heurística, que é o custo estimado para ir do vértice visitado ao vértice de destino *h(n)*.

h(n) = Custo estimado do caminho mais barato desde o vértice n ao vértice de destino.

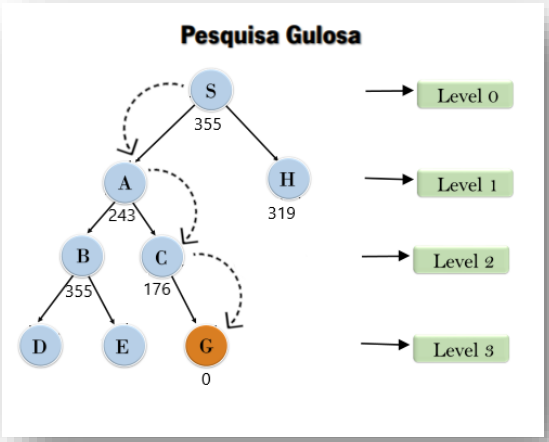


Figura 7: Gulosa

Considerações:

- **Complexidade Temporal/ Espacial**: É de realçar que quanto maior for o nível de ramificação (o número de vértices adjacentes), maior será o tempo da pesquisa uma vez que teremos várias arestas por pesquisar e vértices por expandir. Assim, o algoritmo tende a demorar bastante tempo para soluções cuja profundidade é elevada e além disso é bastante dispendioso em termo de memória, uma vez que armazena os vários vértices em memória.
- **Completude**: Caso a árvore de pesquisa tenha profundidade infinita, a solução poderá recair numa profundidade dita "infinita" pelo que o algoritmo não consegue contornar esta problemática, podendo não encontrar a solução, devido a ficar "preso" em ciclos, pelo que nem sempre encontra a solução.
- **Otimalidade**: Uma vez que a heurística abordada pelo algoritmo apenas visa a analisar o custo estimado para ir do vértice visitado ao vértice de destino, pode nem sempre obter a melhor solução, uma vez que o escolhe o caminho que é mais económico à primeira vista e podem existir caminho mais curtos.
- **Aplicabilidade**: O algoritmo descrito é útil para problemas em que temos mais informação para além do que possa ser visível à primeira vista, tornando assim a pesquisa informada. Para além disso, dadas as complexidades descritas, o algoritmo consegue ser útil para problemas de pequenas dimensões e é mais fácil de implementar dado o teor da heurística usada.
- **Nota**: O algoritmo deve garantir que, nenhum vértice ou aresta seja, visitado mais de uma vez, de forma a evitar possíveis ciclos e o algoritmo ficar "preso" nestes.

Implementação:

- A implementação proposta pelo grupo consiste em iterar sobre os vértices do grafo, guardando os visitados numa lista de modo a evitar ciclos. A partir do nodo atual, que inicialmente é o nosso destino, gera-se uma lista com os vértices que lhe são adjacentes sendo estes adicionados à lista de todos os nodos que faltam visitar, se estes já se encontrarem nessa lista, então compara-se o valor das heurísticas, o valor das estimativas até à Green Distribution, escolhendo-se o menor. Depois, escolhe-se dessa lista o nodo com o menor valor da heurística e guardamos numa estrutura qual o seu predecessor e atualizamos o nodo atual. Quando chegamos ao nodo da Green

Destribution a iteração pára e cria-se um ciclo que percorre todos os predecessores da Green Distribution adicionando-os ao caminho.

4 Resultados

Durante a elaboração do projeto foi procurado implementar as diversas formas de procura sugeridas no enunciado, assim explicando qual da procura tem a melhor solução.

De forma a simplificar a complexidade de cada algoritmo em diversas vertentes, temos de ter alguns aspetos relevantes a ter em conta:

- *b* é o fator de ramificação.
- *d* é a profundidade da solução.
- *m* é a máxima profundidade da árvore de pesquisa.
- *l* é a profundidade limite da pesquisa.

Estratégia	Tempo (Segundos)	Espaço	Encontra a melhor solução?
DFS	$O(b^m)$	$O(bm)$	Não, uma vez que devolve a primeira solução que encontra.
BFS	$O(b^d)$	$O(b^d)$	Sim, se o custo de cada passo forem todos iguais.
Busca Iterativa Limitada em Profundidade.	$O(b^d)$	$O(b^d)$	Sim, se o custo de cada passo forem todos iguais.
Gulosa	$O(b^m)$	$O(b^m)$	Não, por nem sempre encontrar a solução ótima.
A* (A estrela)	Exponencial	Mantém todos os nós em memória.	Sim.

Tabela 2: Análise Comparativa

5 Conclusão e análise de resultados

Após os vários algoritmos implementados e tendo em conta todas as especificações que foram detalhadas ao longo do relatório, criámos uma tabela que permite comparar os tempos de execução médios para os diversos casos de pesquisa a serem efetuados. É também de realçar que todos os testes efetuados foram realizados tendo por base o grafo descrito e desenvolvido pelo grupo durante a realização do projeto.

Estratégia	Tempos de Execução (nanossegundos)	Indicador/Custo
BFS	238700	3
DFS	295200	8
Busca Iterativa Limitada em Profundidade	1859000	3

Tabela 3: Encomenda para Santana

Estratégia	Tempos de Execução (nanossegundos)		Indicador/Custo	
	Mais Rápida	Mais ecológica		
Gulosa	100200	98800	3	3
A*	126800	162200	3	3

Tabela 4: Encomenda para Santana

Através dos resultados obtidos, conseguimos perceber a relação entre as especificações detalhadas para cada algoritmo e o resultado respetivo, pelo que é visível a coerência entre o que foi obtido e o que se esperava obter, em termos comparativos entre os vários algoritmos.

Tendo em conta os vários algoritmos implementados e os respetivos tempos de execução, apercebemo-nos do impacto das diversas estratégias em relação ao tempo, como por exemplo a BFS chega á solução em menos tempo em relação á DFS, uma vez que, a BFS expande primeiro os vértices adjacentes a este, antes de avançar na profundidade enquanto na DFS avança através da expansão do primeiro vértice adjacente da árvore de pesquisa, e incrementa cada vez mais a sua profundidade. Dependendo de onde se encontrar a solução, se a solução estive a uma profundidade menos elevada, a BFS encontrará primeiro em relação a DFS que procura sempre em profundidade. Tal como dito nas considerações a BFS é mais rápida em grafos de pequenas dimensões, o que explica os resultados que obtivemos nos nossos testes uma vez que o nosso grafo é relativamente pequeno aumentando ligeiramente a eficiência da BFS em relação à DFS.

O algoritmo de procura informada, a gulosa, foi mais rápida a encontrar a solução em relação ao algoritmo A estrela, uma vez que a gulosa apenas usa a heurística usando estimativa enquanto a A estrela usa a heurística que soma a estimativa com a distância que já percorreu.

De um modo geral esta segunda fase do projeto permitiu-nos aprofundar o conceito de pesquisa não informada e pesquisa informada e alado a isto o funcionamento de cada algoritmo que se insere no respetivo tipo de pesquisa. Além disso, conseguimos perceber quais os algoritmos que melhor se adequam dadas as circunstâncias impostas aquando da resolução do problema imposta ao longo do projeto. Este problema era subdivido em vários fatores, como qual o melhor veículo a ser escolhido tendo em conta o tipo e teor da encomenda ou conjunto de encomendas e com isto, gerar através do melhor algoritmo dadas as circunstâncias, que devolveria a melhor solução, solução esta que seria o caminho a percorrer de forma a entregar a encomenda ou um conjunto de encomendas.

Com isto, conseguimos também perceber o impacto que os algoritmos sofrem quando se tem de encontrar um conjunto de soluções, isto é para o caso em que se tem de entregar um conjunto de encomendas, pois cada algoritmo deve ser formalizado para puder lidar com esta situação, havendo por isso um impacto notório ao nível das várias especificações, nomeadamente da complexidade temporal/ espacial e também da otimalidade da solução.