Universidade do Minho
Escola de Engenharia

Licenciatura em Engenharia Informática

Inteligência Artificial

Green Distribution
Logística de Distribuição de Encomendas

Grupo 25:
João Machado, A89510
Carolina Vila Chã, A89495
David Duarte, A93253
Daniel Faria, A81997

Novembro 2021

Resumo

Neste relatório encontra-se detalhada a resolução do sistema desenvolvido para a segunda fase do projeto.

O objetivo principal do projeto *Green Distribution* é Com a utilização 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.

Conteúdo

1 Introdução

Vivemos em tempos onde a ecologia é um tópico cada vez mais relevante no nosso dia-a-dia. A empresa *Green Distribution* é um serviço de entrega de encomendas e tem como objetivo principal privilegiar sempre o meio de entrega mais ecológico. Assim, foi desenvolvido um sistema de representação de conhecimento e raciocínio que demonstra as funcionalidades subjacentes à utilização da linguagem de programação em lógica PROLOG, no âmbito da representação de conhecimento e construção de mecanismos de raciocínio para a resolução de problemas.

Neste sistema, existem várias entidades: veículo, estafeta, encomenda, freguesia, rua, cliente e entrega. A principal diferença entre encomenda e entrega, é que a encomenda é o pedido do cliente, enquanto que o estafeta faz a entrega desse pedido, ou seja, faz e entrega da encomenda.

Para esta fase, o sistema deverá ser capaz de responder às seguintes questões:

- Gerar os circuitos de entrega, caso existam, que cubram um determinado território (e.g. rua ou freguesia)
- Representação dos diversos pontos de entrega em forma de grafo, tendo em conta que apenas se devem ter localizações (rua e/ou freguesia) disponíveis
- Identificar quais os circuitos com maior número de entregas (por volume e peso)
- Comparar circuitos de entrega tendo em conta os indicadores de produtividade
- Escolher o circuito mais rápido (usando o critério da distância)
- Escolher o circuito mais ecológico (usando um critério de tempo)

2 Formulação do Problema Como Problema de Pesquisa

O problema para resolver optado neste projeto foi adaptado para que os veículos sejam limitados pela velocidade máxima, peso máximo e tempo de deslocação.

Estado Inicial	Centro
Estado Final	Centro
Predicado	a Predicado aresta/3 e predicado estima/2
Custo	Distância Euclidiana entre as arestas

3 Estratégias de Procura

Variável	Função
b	Máximo fator de ramificação - o número máximo de nodos-filhos que um nodo pode ter.
d	Profundidade da melhor solução.
m	Máxima profundidade do espaço de estados.

Tabela 1: Variáveis a ter em conta para esta secção do relatório.

3.1 Pesquisa não informada

Estratégias de pesquisa não informada usam apenas as informações disponíveis na definição do problema.

Pesquisa em Profundidade - DFS

Funcionamento É sempre expandido o nó mais profundo da árvore primeiro. Depois é feito *backtracking* e são experimentados outros caminhos. Necessita de estruturas de dados auxiliares: uma lista dos nodos já visitados, e o caminho percorrido. Não garante a melhor solução uma vez que não devolve o caminho ótimo à primeira.

Aplicação Necessita de muito pouca memória, e é uma estratégia boa para problemas com muitas soluções. No entanto, não pode ser usada em árvores de profundidade infinita, pois pode ficar presa em ramos errados.

Complexidade Temporal $O(b^m)$

Complexidade Espacial $O(bm)$

Pesquisa em Largura - BFS

Funcionamento Todos os nós de menor profundidade são expandidos primeiro. Devolve a solução ótima à primeira.

Aplicação Pesquisa sistemática, no entanto demorada e ocupa bastante espaço em memória uma vez que tem de guardar todos os caminhos ainda não expandidos. Apenas para pequenos problemas.

Complexidade Temporal $O(b^d)$

Complexidade Espacial $O(b^d)$

Pesquisa Iterativa Limitada em Profundidade

Funcionamento Pesquisa DFS quando é definida uma profundidade limite.

Aplicação Tal como DFS, necessita de pouca memória, e é uma estratégia boa para problemas com muitas soluções. No entanto, já pode ser utilizada em árvores de profundidade infinita, uma vez que lhe é definido um limite de profundidade, evitando o risco de ficar presa em ramos errados.

Complexidade Temporal $O(b^d)$

Complexidade Espacial $O(bd)$

3.2 Pesquisa Informada

Utiliza informação do problema para evitar que o algoritmo de pesquisa fique perdido. As estratégias são definidas pela ordem de expansão dos nós, com o melhor nodo primeiro. Neste projeto, a heurística utilizada foi a distância em linha reta entre dois pontos de recolha. Este valor é obtido através dos cálculos das distâncias euclidianas dos pontos de entrega.

Gulosa

Funcionamento Expandir o nó que parece estar mais perto da solução. Nem sempre encontra a solução ótima, e é necessário detetar estados repetidos, caso contrário poderá entrar em ciclos. Mantém todos os nós em memória.

Aplicação O Algoritmo *Greedy* utiliza a heurística fornecida e escolhe o ponto com a heurística menor a cada iteração. Em termos de resultados de execução, o Algoritmo *Greedy* revelou ser o algoritmo mais lento a executar dos algoritmos de pesquisa informada e também revelou ser o menos eficiente em termos de utilização de memória.

Complexidade Temporal $O(b^m)$

Complexidade Espacial $O(b^m)$

A*

Funcionamento Evitar expandir caminhos que são mais caros. Combina Gulosa com Uniforme, minimizando a soma do caminho já efetuado com o mínimo previsto que falta até à solução.

Aplicação O Algoritmo $A*$ utiliza a heurística fornecia e conta também a distância real para determinar qual ponto a escolher. Em termos de resultados de execução, o Algoritmo $A*$ revelou ser o algoritmo mais rápido a executar dos algoritmos de pesquisa informada e também revelou ser o mais eficiente em termos de utilização de memória.

Complexidade Temporal Dependente do número de nodos com $g(n) + h(n) <= C*$

Complexidade Espacial Dependente do número de nodos com $g(n) + h(n) <= C*$

Onde $g(n)$ representa o custo do percurso até ao momento e h(n) o custo estimado para chegar ao objetivo.

4 Objetivos

4.1 Circuitos

Para este trabalho foram criados circuitos, entidades compostas pelo algoritmo que seguem, a rua destino, o caminho do centro para essa rua, o custo (neste caso, a distância), o peso total e o volume total.

O peso total e o volume total são valores que se vão acumulando, de cada vez que é feita uma entrega que utilize esse circuito. O peso e o volume da encomenda levada é acrescentado ao valor correspondente no circuito.

Essencialmente, os circuitos são, para cada algoritmo de pesquisa, mapas para todas as localizações da cidade (as ruas). Por exemplo, um circuito que tem de ser *hardcoded* no início do programa uma vez que deve já pré-existir aquando do início do programa.

```
% circuito(algoritmo, rua_entrega, [caminho], custo, peso, volume).
circuito(gulosa, 9, 2, 24).
```

No início da fase 2 do programa, é importante gerar os circuitos primeiro (`gerar_circuitos.`), uma vez que são necessários para vários dos outro objetivos principais.

4.2 Representação em Grafo

Para representar os pontos de entrega o grupo utilizou como estrutura de dados um grafo. O conteúdo deste pode ser consultado no ficheiro *grafo.pl* disponibilizado no código.
Esse grafo está aqui representado de forma mais intuitiva com os valores de distância associados às ligações entre pontos de entrega:

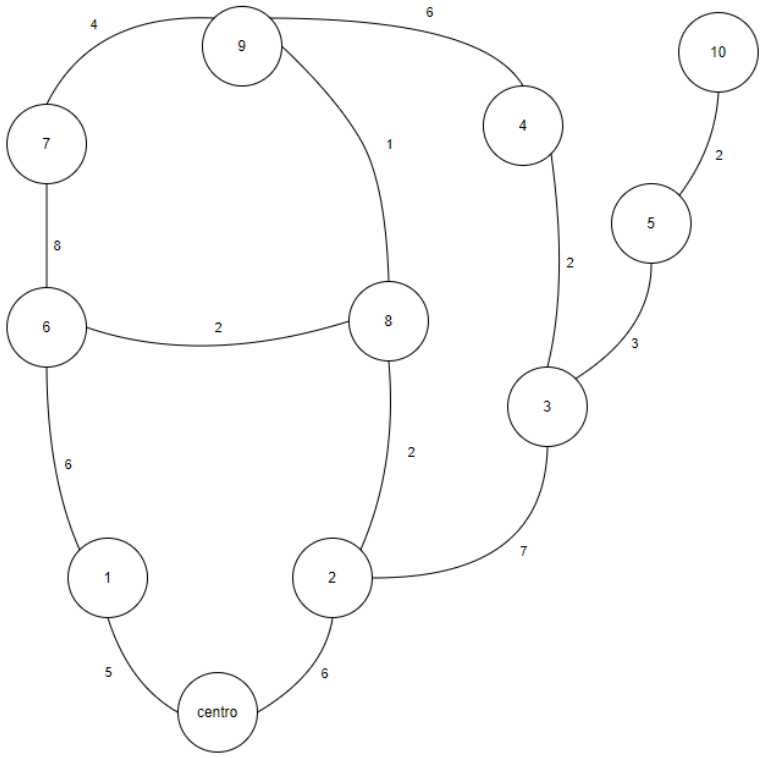

Figura 1: Grafo com os pontos de entrega.

4.3 MultiEntrega

Com o objetivo de avaliar o impacto resultante de cada estafeta poder passar a efetuar mais de uma entrega em cada viagem o grupo decidiu testar os tempos de viagem que o estafeta regista ao fazer duas entregas, uma na rua 3 e outra na rua 8. Para testar esta hipótese o veículo a usar será a mota em ambos os casos, isto é, **caso 1:** entrega na rua 3, volta ao centro, entrega na rua 8 e volta novamente ao centro ou **caso 2:** entrega na rua 3, depois entrega na rua 8 e volta ao centro.

Para aprofundar esta análise fizeram-se 3 testes para cada um dos casos acima assinalados. Um com o algoritmo de procura primeiro em profundidade, outro para o algoritmo de procura primeiro em largura e finalmente para o algoritmo de procura em profundidade iterativo.

4.3.1 Algoritmo de Procura Primeiro em Profundidade

Entregas Individuais

Primeiramente testou-se o resultado de entregar as duas encomendas de forma individual seguindo um circuito que é gerado pelo algoritmo de procura primeiro em profundidade.

```
?- multi_circuito(profundidade,[centro,3],Caminho,TempoViagem,5).
Caminho = [centro, 2, 3, 4, 9, 7, 6, 8, 2|...],
TempoViagem = 1.323076923076923 .
```

Figura 2: Entrega individual de encomenda em rua 3 - DFS.

Como é possível observar na Figura 1, o tempo que levou ao estafeta a entregar a encomenda e voltar ao centro foi 1.323 horas, que é o equivalente a 1h19.

De seguida faz-se o mesmo processo para a rua 8 e eis os resultados:

```
?- multi_circuito(profundidade,[centro,8],Caminho,TempoViagem,4).
Caminho = [centro, 2, 3, 4, 9, 7, 6, 8, 9|...],
TempoViagem = 1.7878787878787878 .
```

Figura 3: Entrega individual de encomenda em rua 3 - DFS.

Múltiplas Entregas

Se agora se entregarem as duas encomendas numa só viagem os resultados serão os seguintes:

```
?- multi_circuito(profundidade,[centro,3,8],Caminho,TempoViagem,9).
Caminho = [centro, 2, 3, 4, 9, 7, 6, 8, 9|...],
TempoViagem = 1.9344262295081966 .
```

Figura 4: Entrega múltipla de encomenda em ruas 3 e 8 - DFS.

Como se pode observar, as somas dos resultados das entregas individuais é superior ao tempo obtido com a entrega múltipla e assim sendo conclui-se que as entregas múltiplas têm o benefício de que o tempo será melhor aproveitado trazendo mais eficiência ao funcionamento do centro.

4.3.2 Algoritmo de Procura Primeiro em Largura

Os mesmos testes executados para a pesquisa em profundidade foram executados para a pesquisa em largura e aqui estão os resultados: **Entregas Individuais**

```
?- multi_circuito(largura,[centro,3],Caminho,TempoViagem,5).
Caminho = [centro, 2, 3, 2, centro],
TempoViagem = 0.8 .
```

Figura 5: Entrega individual de encomenda em rua 3 - BFS.

```
?- multi_circuito(largura,[centro,8],Caminho,TempoViagem,4).
Caminho = [centro, 2, 8, 2, centro],
TempoViagem = 0.48484848484848486 .
```

Figura 6: Entrega individual de encomenda em rua 8 - BFS.

Entregas Múltiplas

```
?- multi_circuito(largura,[centro,3,8],Caminho,TempoViagem,9).
Caminho = [centro, 2, 3, 2, 8, 2, centro],
TempoViagem = 0.9836065573770492 .
```

Figura 7: Entrega múltipla de encomendas nas ruas 3 e 8 - BFS.

4.3.3 Algoritmo de Procura Primeiro em Profundidade Iterativa

Repetem-se os cenários de teste para este algoritmo. **Entregas Individuais**

```
?- multi_circuito(iterativa,[centro,3],Caminho,TempoViagem,5).
Caminho = [centro, 2, 3, 2, centro],
TempoViagem = 0.8
```

Figura 8: Entrega individual de encomenda em rua 3 - IDDFS.

```
?- multi_circuito(iterativa,[centro,8],Caminho,TempoViagem,4).
Caminho = [centro, 2, 8, 2, centro],
TempoViagem = 0.48484848484848486 .
```

Figura 9: Entrega individual de encomenda em rua 8 - IDDFS.

Entregas Múltiplas

```
?- multi_circuito(iterativa,[centro,3,8],Caminho,TempoViagem,9).
Caminho = [centro, 2, 3, 2, 8, 2, centro],
TempoViagem = 0.9836065573770492 .
```

Figura 10: Entrega múltipla de encomenda nas ruas 3 e 8 - IDDFS.

Todas as informações obtidas até aqui estão agora representadas na seguinte tabela e nela pode concluir-se que na maioria dos casos as entregas múltiplas são mais eficientes. Caso dos algoritmo IDDSF e BFS. De seguida na lista de eficiência temos os mesmos algoritmo mas agora associados a entregas individuais. Em penúltimo o algorítmo BSF em entrega múltipla e para terminar o algoritmo DSF em entregas individuais.

	IIterativa	MIterativa	IProfundidade	MProfundidade	ILargura	MLargura
Encomenda 3	48min		1h19min		48min	
Encomenda 8	29min		1h47min		29min	
Total	1h17min	59min	3h06min	1h56min	1h17min	59min

Tabela 2: Resultados de entregas individuais e múltiplas.

4.3.4 Predicados utilizados

Para a realização dos testes apresentados acima foram implementados os seguintes predicados:

```
multi_circuito(Algoritmo,Ruas,Caminho,TempoViagem,Peso).
```

este predicado irá receber um *Algoritmo*, uma lista de *Ruas* e o *Peso* total da ou das encomendas que o estafeta levar. Calcúla o *Caminho* que o estafeta deve percorrer de acordo com o *Algoritmo* passado como argumento e para além disto devolve ainda o *TempoViagem* que é calculado com o auxílio de outro predicado.

```
get_caminho_profundidade(Ruas,Caminho,Custo).
```

assumindo que o *Algoritmo* passado ao predicado *multi_circuito* foi o de pesquisa em profundidade primeiro, o predicado *get_caminho_profundidade* será o próximo a ser chamado e a sua função é, utilizando o DFS, devolver o caminho que passe por todas as ruas referidas e ainda o custo (distância) desse percurso.

Note-se que para cada um dos restantes algorítmos existe um predicado *get_caminho* com o mesmo comportamento que o apresentado.

```
tempo_de_entrega(Velocidade,PerdaDeVelocidadePorKg,Peso,Custo,TempoViagem).
```

finalmente tem-se o predicado `tempo_de_entrega` que baseado na distância obtida com `get_caminho_profundidade`, no peso da/s encomenda/s, na velocidade do veículo e no fator perda de velocidade por kg calcula o tempo total do percurso a ser feito.

4.4 Circuito com maior número de entregas (por volume e por peso)

Por forma a descobrir qual o circuito com o maior número de entregas feitas o grupo decidiu, na execução do programa, criar todos os circuitos possíveis com os algorítmos disponíveis e identificá-los corretamente. Desta forma, sempre que um circuito seja utilizado para uma entrega será incrementado o o valor do volume e do peso das entregas utilizando aquele circuito em específico.

```
incrementa_circuito(Algoritmo, Rua, Volume, Peso) :-
    circuito(Algoritmo, Rua, Ca, C, V, P),
    X is Volume + V,
    Y is Peso + P,
    retract(circuito(Algoritmo, Rua, Ca, C, V, P)),
    assert(circuito(Algoritmo, Rua, Ca, C, X,Y)).
```

Assim sendo resta agora calcular qual o circuito com maior número de entregas tanto a nível de volume como de peso e para isso utiliza-se a função `findall` para obter a lista de pesos e volumes dos circuitos e a partir dessas listas calcula-se o máximo identificando assim o circuito com maior afluência.

4.5 Comparar circuitos por peso e por volume

Tal como no ponto anterior, aqui também é são utilizados os circuitos criados na execução do programa para obter, para cada tipo de circuito, aquele que tem maior volume acumulado e ainda o que possui melhores valores de tempo de entrega

Para este objetivo, implementou-se o predicado `call_comparar_circuitos_indicadores`. Segue um excerto do código responsável por indicar ao utilizador os circuitos, ordenados

por distância, para o algoritmo aestrela.

```
call_comparar_circuitos_indicadores:-
    % comparar distancia
    write('Distância:'),nl,
    write('------ Aestrela ------'), nl,
    % Encontrar todas as distâncias calculadas pelo aestrela
    findall(Distancia1,
        circuito(aestrela,_, _, Distancia1, _, _),
        ListaAE),
    % ordenar essas distâncias para que estejam por ordem crescente
    sort(ListaAE, SortedAE),
    comparar_aux(SortedAE, aestrela, distancia),
```

O algoritmo começa por encontrar todas as distâncias calculadas pelo aestrela, de seguida ordena-as, e fornece-as ao predicado auxiliar.

Para imprimir os circuitos por ordem, é utilizado um predicado auxiliar.

```
comparar_aux([H|T], Algoritmo, distancia):-
    circuito(Algoritmo, Rua, Caminho, H, Peso, Volume),
    write('circuito('),
        write(Algoritmo), write(','), write(Rua), write(','),
        write(Caminho), write(','), write(H), write(','),
        write(Peso), write(','), write(Volume), write(')'),
    nl,!,
    comparar_aux(T, Algoritmo, distancia).
```

Este predicado auxiliar apenas escreve para o terminal os circuitos ordenadamente.

O mesmo raciocínio pode ser aplicado para a ordenação dos circuitos por tempo, embora a complexidade seja maior do que para a distância. Uma vez que os circuitos não são relativos a certas encomendas, mas sim "mapas"gerais; e uma vez que este facto foi apenas identificado como um problema pelo grupo demasiado tarde na resolução do projeto, a ordenação dos circuitos foi feita de modo ligeiramente diferente.

Segue um excerto de código.

```
write('Tempo de Viagem:'),nl,
```

```
    write('------ Aestrela ------'), nl,
    % extrair os tempos de entrega das encomendas
    findall((Enc6, Veiculo6, Rua6, Tempo6, Distancia6), (
        entrega(Ent6, Enc6, Est6, Class6, Veiculo6),
        encomenda(Enc6, Data6, Prazo6, Peso6, Volume6, Preco6, Rua6,
        ↪  _),
        veiculo(Veiculo6, Carga6, Velocidade6, Decrescimo6),
        distancia_por_algoritmo(aestrela, Rua6, Distancia6),
        tempo_de_entrega(Velocidade6, Decrescimo6, Peso6, Distancia6,
        ↪  Tempo6)
        ), TemposAE),
    % ordenar os tempos de entregas
    sort(TemposAE, STemposAE),
    comparar_aux(STemposAE, aestrela, tempo),
```

Os circuitos são apresentados comforme quais das entregas que, hipoteticamente, seriam entregues neles mais rapidamente, e os seus algoritmos.

Para ordenar os circuitos, é necessário verificar as entidades que estão ligadas entre si: encomenda, veiculo, e entrega. É de seguida necessário calcular o veículo, e a distância para o objetivo segundo o algoritmo. É depois calculado o tempo de entrega, e a lista de tempos é impressa com ajuda da função auxiliar.

```
comparar_aux([(Enc, Veiculo, Rua, Tempo, Distancia)|T], Algoritmo,
↪  tempo):-
    % ver cada circuito que utilizam
    % ordenar esses circuitos
    circuito(Algoritmo, Rua, Caminho, Custo, Peso, Volume),
    write('Encomenda: '), write(Enc), write(' - '),
    write('circuito('),
        write(Algoritmo), write(','), write(Rua), write(','),
        write(Caminho), write(','), write(Distancia), write(','),
        write(Peso), write(','), write(Volume), write(')'),
    write(' - Tempo:'), write(Tempo),
    nl,!,
    comparar_aux(T, Algoritmo, tempo).
```

4.6 Escolher o circuito mais rápido (distância)

Para a satisfação deste objetivo, desenvolveu-se o predicado `criar_entrega_rapida`. Este predicado passa primeiro por verificar se as seguintes entidades existem: encomenda a entregar, estafeta a realizar a entrega, e se a classificação está entre 0 e 5.

```
% Escolhe o algoritmo, tentando sempre que tenha o custo mínimo
criar_entrega_rapida(EntId,EncId,EstId,Class):-
    encomenda(Enc,_,_,Peso,Volume,_,Rua,_),
    estafeta(Est,_),
    Class =< 5, Class >= 0,
    veiculo_encomenda_rapida(EncId, Algoritmo, Veiculo),
    incrementa_circuito(Algoritmo, Rua, Volume, Peso),
    evolucao(entrega(EntId, EncId, EstId, Class, Veiculo)).
```

Depois, é chamado o predicado `veiculo_encomenda_rapida`, que irá determinar qual o veiculo a utilizar para esta entrega, certificando-se que é uma entrega o mais rápida possível.

```
% Determina o algoritmo e o veículo que dão o resultado com menos
↪  custo (mais rápido)
veiculo_encomenda_rapida(EncId, Algoritmo, Veiculo):-
    encomenda(EncId, _,_,Peso, _,_,Rua,_),

    % escolhe o algoritmo com menor custo
    findall(Custo, circuito(_, Rua, _, Custo, _, _), Circuitos),
    min_list(Circuitos, MinCusto),
    circuito(Algoritmo, Rua, _, MinCusto, _, _),

    % escolhe o veiculo que consegue ir mais rápido
    findall(Velocidade, veiculo(Vx, _, Velocidade,_), Velocidades),
    max_list(Velocidades, MaxVelocidade),
    veiculo(Veiculo, Carga, MaxVelocidade, Decrescimo).
```

Infelizmente, este predicado não tem em conta se o veículo pode realizar a entrega. Uma vez que temos a opção do fogetão, que é praticamente instantânea (uma vez que a sua velocidade está adaptada a descolagens para órbita, e não para "pequenas"entregas dentro de uma cidade), este facto não apresenta grande risco, no entanto,

num sistema mais realista, esta função de considerar o peso deveria ser implementada. Consideramos isso como possível trabalho futuro.

4.7 Escolher o circuito mais ecológico

À semelhança do objetivo anteior, desenvolveu-se o predicado `criar_entrega_ecologica`.

Este predicado passa primeiro por verificar se as seguintes entidades existem: encomenda a entregar, estafeta a realizar a entrega, e se a classificação está entre 0 e 5.

```
% Escolhe o algoritmo, tentando que tenha o veiculo mais ecológico
criar_entrega_ecologica(EntId,EncId,EstId,Class):-
    encomenda(Enc,_,_,Peso,Volume,_,Rua,_),
    estafeta(Est,_),
    Class =< 5, Class >= 0,
    veiculo_encomenda_ecologica(EncId, Algoritmo, Veiculo),
    incrementa_circuito(Algoritmo, Rua, Volume, Peso),
    evolucao(entrega(EntId, EncId, EstId, Class, Veiculo)).
```

Depois, é chamado o predicado `veiculo_encomenda_ecologica`, que irá determinar qual o veiculo a utilizar para esta entrega, certificando-se que é uma entrega com o veículo o mais ecológico possível mas que entregue a tempo.

```
% Devolve o algoritmo e o veículo que dão o resultado de menor tempo de entrega (
veiculo_encomenda_ecologica(EncId, Algoritmo, Veiculo):-
    encomenda(EncId, DataEncomenda, Prazo, Peso, _,_, Rua, _),

    % tirar o custo mínimo
    findall(Custo, circuito(_, Rua, _, Custo, _, _), Custos),
    min_list(Custos, MinCusto),
    circuito(Algoritmo, Rua, _, MinCusto, _, _),

    % ver o primeiro veiculo que consegue entregar a tempo
    veiculo_encomenda_aux(DataEncomenda, Prazo, Peso, Rua, Veiculo, Algoritmo).
```

O algoritmo passa então por calcular a distância mais curta e extrai o algoritmo que fornece essa distância (através do circuito, que une, entre outros, a distância ao algoritmo que a calcula).

Depois, uma vez que entregar com o veículo mais ecológico, respeitando sempre o tempo de entrega pedido, já é a funcionalidade base de entrega de encomendas do programa, anteriormente desenvolvida na fase 1, apenas é necessário chamar o predicado `veiculo_encomenda_ecologica`.

4.8 Resultados

Estratégia	Tempo (s)	Espaço	Indicador/Custo	Encontrou melhor solução?
DFS	0.000s	21	Distância	Não
BFS	0.000s	27	Distância	Não
Pesquisa Iterativa Limitada em Profundidade	0.000s	9	Distância	Não
Gulosa	0.000s	2187	Distância	Não
A*	0.01s	9	Distância	Sim

Tabela 3: Resultados obtidos por estratégia de pesquisa.

5 Comentários Finais e Conclusão

Concluindo, com este trabalho foram aplicados os conhecimentos de *Prolog* lecionados ao longo deste semestre na unidade curricular de Inteligência Artificial. Nomeadamente conhecimentos relativos a Pesquisa, Grafos, Invariantes e Representação do conhecimento.

Como foi demonstrado anteriormente, foram implementadas todas as funcionalidades pedidas para tal e algum extra, embora achamos que podíamos ter feito a função origem /destino do algoritmo *A** e *gulaso* melhor pois não permitem mudar a origem do centro para outro lugar como acontece nas outras funções com os outros algoritmos também pensamos em implementar uma função que imprimisse o grafo usado, outra que devolvesse o útlimo estafeta online, implementação do volume da encomenda nos veículos e uma função de remoção de entidades.