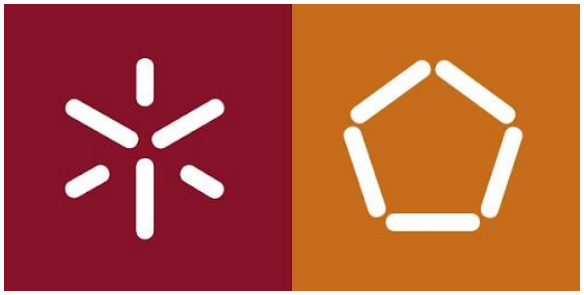

Universidade do Minho

Relatório do Projeto Fase 2

Grupo 7

Ano letivo 2021/2022

Dezembro 2021

Licenciatura em Engenharia Informática

Unidade Curricular de Inteligência Artificial

Ana Gonçalves a93259

Bruno Pereira a93298

Francisco Toldy a93226

João Delgado a93240

Índice

Introdução

No âmbito da unidade curricular da Inteligência Artificial atualizamos o sistema anteriormente desenvolvido e criamos um sistema de recomendação de circuitos de entrega de encomendas. Esta atualização consistiu na criação de um cenário minimamente realista, onde serão efetuadas as entregas.

Com este objetivo vamos apresentar neste documento a estruturação da nossa base de conhecimento, as pesquisas que implementamos e para cada uma destas os seus parâmetros de performance.

Base de Conhecimento

Perante o novo desafio de criar um sistema de recomendação de rotas baseamo-nos na base de conhecimento que foi elaborada na fase anterior e a expandimos de modo a manter-se equilibrada com o nível de complexidade das funcionalidades que foram requisitadas. Assim, adicionamos os seguintes predicados:

- ***velocidadeMedia(TipoDeVeiculo,VelocidadeMedia)***

Dado um veículo (TipoDeVeiculo) permite saber a velocidade média associada a esse veículo (VelocidadeMedia).

- ***veiculoDescrescimo(TipoDeVeiculo,Decrescimo)***

Dado um veículo (TipoDeVeiculo) permite saber o decréscimo médio por quilo relativamente à encomenda a entregar, a esse veículo (Decrescimo).

- ***inicial(Inicio)***

Inicio vai corresponder sempre ao armazém, que é o nodo inicial da procura em todos os grafos.

- ***coord(Rua, Latitude, Longitude)***

Onde estão associadas coordenadas de uma determinada rua.

- ***grafoCeleiros(grafo(ListaArestas))***

Contém uma lista das conexões (arestas) entre 2 ruas para todas as ruas relativas à freguesia de Celeirós e a correspondente distância entre essas 2.

- ***grafoMartim(grafo(ListaArestas))***

Contém uma lista das conexões (arestas) entre 2 ruas para todas as ruas relativas à freguesia de Martim e a correspondente distância entre essas 2.

- ***encomendaNE(Id,Peso,Volume,Classificacao,Rua,Freguesia,Preço,idCliente,idEstafeta)***

Corresponde às encomendas que ainda não foram entregues com as correspondentes informações

- ***grausParaRads(Graus,Radianos)***

Dado um valor em graus, converte-o para radianos. Predicado auxiliar ao predicado *distancia/3.*

- ***distancia(PontoA, PontoB, Distancia)***

Dados dois pontos A e B, este predicado irá calcular a distância em linha reta entre eles.

- ***dataNE(IdEncomenda, DataEncomenda, Prazo)***

Em comparação com a fase 1, este predicado foi alvo de uma simples alteração nas datas, tendo agora estas um elemento adicional, a Hora, sendo agora constituídas por Ano, Mês, Dia e Hora.

Por último, para maior versatilidade e completude na realização de testes, foram alterados predicados elaborados na fase anterior, o predicado **encomenda/10**, que não fazia qualquer discriminação entre encomendas já entregues e encomendas por entregar. Foi então feita a separação em dois predicados diferentes, encomenda e **encomendaNE/8** com exatamente os mesmos parâmetros, exceto a classificação e veículo utilizado na entrega. Além dessa separação, a quantidade de instâncias do predicado **encomendaNE/8** foi expandida.

Foram feitas as alterações necessárias às instâncias já existentes dos predicados cliente e estafeta para acomodar as alterações nos predicados anteriormente referidos.

Formulação do Problema

Perante então o novo desafio concordamos que o problema em questão se identifica como um problema de estado único. Além disto, apresentamos a seguir, como a representação é feita dos seguintes conceitos:

- **Conjunto de estados**: Vários nodos que constituem o grafo, ou seja, as várias ruas da freguesia. Representado pelo predicado grafoCeleiros, por exemplo, cuja lista de arestas, é constituída por triplos cujos 2 primeiros valores são 2 ruas e o último, a distância entre estas últimas.

- **Estado inicial**: O estado inicial é conhecido para cada grafo, o armazém, representado pelo predicado “inicial”.

- **Estado final**: Local da entrega da encomenda em questão, mais especificamente a rua destino.

- **Operadores**: **adjacente/3**. Este predicado permite identificar todas as ruas vizinhas da rua atual.

- **Custo**: O custo de cada passo será igual à distância da rua atual para a rua que vamos visitar na próxima iteração. Assim, o custo final da solução será a soma das distâncias entre todas as ruas visitadas.

Requisitos

Perante os requisitos explicitados no enunciado, apresentaremos a seguir a descrição da nossa implementação para o cumprimento destes mesmos requisitos:

- **Representação dos diversos pontos de entrega em forma de grafo, tendo em conta que apenas se devem ter localizações (rua e/ou freguesia) disponíveis;**

Para esta representação, elaboramos grafos representantes de uma freguesia, onde cada nodo simboliza uma rua. Como exemplos, utilizamos 2 grafos, um relativamente à freguesia de Martim e outro relativamente à freguesia de Celeirós.

- **Gerar os circuitos de entrega, caso existam, que cubram um determinado território (e.g. rua ou freguesia)**

Para gerar circuitos de entrega, (já com grafos de freguesias disponibilizados) criamos apenas algoritmos de pesquisa que devolverão 1 caminho (senão muitos) para uma rua destino.

- **Identificar quais os circuitos com maior número de entregas (por volume e peso);**

Para resolver este requisito foi criado o predicado **melhorCircuito/5** com o propósito de calcular, em simultâneo, qual o melhor circuito (para uma dada encomenda) para os 3 parâmetros referidos: número de entregas, volume e peso. Para tal começou-se por, dado o Id da encomenda, identificar o destino da mesma e , recorrendo a um findall , unificar em *Caminhos* a lista de todas as soluções da procura breadth first (ou seja todos circuitos possíveis para esse destino).

Com a lista de *Caminhos* elaborada, o raciocínio para os 3 parâmetros foi análogo. Usando o número de entregas como exemplo desse raciocínio, recorreu-se ao predicado **calculaNumeroEnc/2** (correspondido por **calculaPesos/2** para o parâmetro peso e **calculaVolumes/2** para o parâmetro volume) para transformar a lista de caminhos numa lista de pares caminho/número de encomendas. Para fazer esse cálculo do número de encomendas para cada caminho recorreu-se ao predicado **calculaNumeroEncAux/2** que, para cada nodo do caminho, executa um findAll para conseguir a lista de encomendas não entregues que têm como destino esse mesmo nodo. Por fim, é usado o predicado **calculaMelhor/2** que, para uma lista de pares caminho/número devolve qual o par com maior número, seja este peso,volume,entregas, consoante o parâmetro que estiver a ser avaliado.

- **Comparar circuitos de entrega tendo em conta os indicadores de produtividade;**

Com este objetivo, criamos 2 predicados, **maisEco/6** e **maisCurto/4.** Estes 2 predicados vão devolver caminhos, de tal forma que o primeiro predicado devolva o caminho que durará menos tempo, e o segundo devolva o caminho que percorrerá menos quilometros. Para realizar as comparações entre ruas de modo a escolher o melhor caminho, é utilizado o predicado **melhorEstEstrela/2** que calculará o melhor caminho até ao momento dependendo do seu custo.

- **Escolher o circuito mais rápido (usando o critério da distância);**

De modo a escolher o circuito mais rápido, utilizamos o algoritmo de pesquisa A*, de modo a escolher o caminho cuja distância a percorrer será a menor. E assim sendo, criamos o predicado **maisCurto/4**.

- **Escolher o circuito mais ecológico (usando um critério de tempo);**

De modo a escolher o circuito mais ecológico, ou seja, cuja duração fosse a menor possível, criamos o predicado **maisEco/6** que calculará o circuito através duma pesquisa A*, no entanto, recorrendo a uma versão que calculará o tempo que demoraria efetuar o circuito bem como o veículo mais ecológico

- **Avalie as alterações e impacto resultantes de cada estafeta poder passar a efetuar mais de uma entrega (levar mais de uma encomenda) em cada viagem (circuito) que realiza. Utilize exemplos concretos, exemplificando através da linguagem de programação escolhida**

Para que, ao escolher um circuito para entregar uma encomenda, pudessem ser entregues mais do que uma encomenda, era necessário que o estafeta introduzisse a encomenda que queria realizar. Com isso, o sistema calcularia o caminho mais curto até ao destino dessa encomenda e, analisá-lo-ia de forma a ver se existiriam mais encomendas para serem entregues nesse mesmo circuito. Para isso, teria de ser feita também a verificação de peso máximo, para que este não fosse excedido.

Em termos de código, isto seria efetuado, por exemplo, executando a procura A* para a encomenda desejada. Obtendo o melhor caminho, seria necessário analisá-lo e verificar quais as encomendas que podiam ser entregues nesse circuito e, por fim, ver quantas dessas encomendas poderiam ser efetuadas, de forma que o peso não fosse excedido.

Estas alterações poderiam ter um impacto grande no tempo que demoraria a efetuar a entrega e possivelmente alteraria também o veículo de eleição, uma vez que ambos são determinados através do peso da encomenda a ser levada.

Exemplificando, ao calcularmos o caminho para a entrega na rua sub carreira ([armazem,rua_monte_carrinhos, avenida_de_sao_loureno , rua_sub_carreria]) verificamos que o caminho inclui a rua monte carrinhos, a qual tem uma encomenda por entregar (Id 17). Logo, no fim de calcularmos o circuito a percorrer, confirmaria que também entregaria a encomenda cujo Id é 17.

Em termos de pesquisa envolvendo a duração da viagem, verificamos que ao calcular o caminho o veículo selecionado é o carro. Sendo a soma do peso das 2 encomendas menor do que 100 quilogramas, e tendo em consideração o decréscimo de velocidade acrescido provocado pelo peso da 2º encomenda, confirmamos que a encomenda é entregue dentro do prazo. Concluindo, seria possível enviar as 2 encomendas na mesma viagem

Estratégias de Procura Utilizadas

Procura Não Informada

❖ DepthFirst

O algoritmo Depth First (profundidade primeiro) tem como estratégia a expansão contínua do primeiro nodo. Tem como vantagem o facto de utilizar pouca memória mas no entanto tem como fraqueza o facto de ser inadequada para árvores de tamanho elevado ou infinito. Não é uma estratégia completa uma vez que pode falhar em espaços de profundidade contínua.

A nível de código em Prolog, o algoritmo começa por procurar no grafo o primeiro nodo adjacente ao nodoAtual, unificando com Novo. Este Novo é acrescentado a Solucao, depois de verificado que este não é um nodo visitado. Por último é feita a chamada recursiva e é calculado o C (custo) somando o custo da aresta encontrada (C1) e o custo anterior (C2).

❖ Breadth First

A pesquisa Breadth first tem como principal estratégia a expansão de todos os nós do grafo, expandindo sempre os todos os nós de nível n antes de começar a expandir os nós do nível n+1. No pior caso esta pesquisa acaba por expandir todos os nós do grafo.

Esta pesquisa é utilizada para encontrar o percurso de menor tempo desde o armazém até ao destino da encomenda a realizar. Para além disso também é utilizada para encontrar o percurso de menor distância desde o armazém até ao destino da encomenda.

Por fim, a pesquisa de Breadth First é ainda utilizada para gerar os circuitos com maior número de entregas, percurso com maior volume de encomendas e percurso com maior peso de encomendas.

❖ Procura Iterativa com profundidade limitada

A procura iterativa com profundidade limitada consiste em efetuar uma pesquisa em profundidade limitada sucessivamente até ser encontrada uma solução. Inicialmente, é invocada a pesquisa em profundidade com uma profundidade de 1. Caso seja encontrada a solução, então não será preciso fazer mais nada. Caso a solução não seja encontrada com uma pesquisa limitada de profundidade 1, será necessário incrementar a profundidade e efetuar a pesquisa com profundidade 2. Este processo será efetuado sucessivamente até ser encontrada a solução para o problema.

Procura Informada

❖ Pesquisa gulosa

A pesquisa gulosa (**greedyDistancia/3**) baseia-se unicamente nas estimativas de cada rua até à rua destino. Com este objetivo, calculamos a estimativa inicial (**distancia/3** em linha reta) do armazem à rua destino e iniciamos a pesquisa. Esta pesquisa vai ter na sua constituição dois valores associados ao caminho (Caminho/Custo/Estimativa), dessa lista vai escolher expandir (**expande/4**) aquele caminho cuja estimativa seja menor. Para expandir este caminho, vai criar outros X caminhos quantas X ligações a última rua adicionada tiver. Finalmente, vai refazer este processo recursivo, até que nessa lista, existia um caminho, cujo nodo final, é a rua destino, devolvendo esse caminho como solução.

❖ Pesquisa A*

Tem como estratégia base evitar a expansão de caminhos que são caros. Para tal, combina a pesquisa gulosa (explicada no ponto anterior) e a uniforme. Assim é feita uma expansão de todos os nodos a partir do nodo atual e esses são colocados numa lista. Dessa lista, o nodo selecionado será aquele cujo custo desde o nodo origem somado à estimativa (distância em linha reta ao nodo final) é menor . É, à semelhança da estratégia breadth first, uma estratégia completa.

A nível de código, o predicado criado foi o **resolveAEstrelaDistancia/3** que começa por calcular a distância em linha reta do *NodoInicial* até ao *NodoFinal*, conseguindo assim a primeira estimativa. Depois disso é invocado o predicado **aEstrelaAux/4.**

O predicado **aEstrelaAux/4** começa por, no caso recursivo, calcular o melhor caminho de entre os caminhos disponíveis, recorrendo ao predicado **melhorEstEstrela/2**.

Depois disso, recorrendo ao predicado **seleciona_caminho/3**, é retirado o melhor caminho da lista de caminhos, sendo a lista resultante denominada por *OutrosCaminhos*. O passo seguinte é recorrer ao predicado **expande/4** para fazer a expansão do melhor caminho , criando uma lista de todos os caminhos possíveis com adjacentes do nodo mais recente do *MelhorCaminho*. Por fim, é feito o append da lista *OutrosCaminhos* e da lista *ExpandeCaminhos* (solução do predicado **expande/4**) para uma lista *NovosCaminhos*, que será usada na chamada recursiva do **aEstrelaAux/4.**

Adicionalmente, incluímos também a elas todas a sua versão de duração temporal, que recorre sempre ao predicado **escolheVeiculo/5**. Este predicado tem como objetivo principal escolher o veículo que será usado na entrega com base em 3 requisitos: o veículo escolhido suporta o peso da encomenda, consegue entregá-la dentro do prazo estabelecido pelo cliente e constitui-se como o veículo cuja duração de viagem será a menor.

Com objetivo de confirmar os resultados das outras pesquisas, criamos a pesquisa que encontra o melhor caminho (com menor custo de distância) encontrado de todas as pesquisas Breadth first, **encontraMelhorBf/5.** Com este objetivo realizamos um findall que reunirá todos os caminhos encontrados pela BreathFirst e recorrerá ao predicado **encontraMelhor/2** que devolverá como solução o caminho com menor duração possível.

Testes

Para efetuar os testes a cada uma das pesquisas, com o objetivo de obter o tempo que demoraria a efetuar a entrega, escolhendo o veículo mais rápido, é necessário que a data do prazo da encomenda esteja ainda depois da data atual somada com o tempo que demora a percorrer o circuito. Deste modo, é preciso construir os testes de forma que isso se verifique. Caso se queira testar o contrário e gerar um caso em que o prazo já foi vencido, então a pesquisa deverá dar *false*, uma vez que não encontrará nenhuma solução cuja solução do predicado **entregaDentroPrazo/2** seja verdadeiro.

Assim, para efeitos de testes em que a encomenda poderá ser entregue dentro do prazo, deve ser utilizada a encomenda de Id 16. Para testes em que a encomenda estará fora do prazo poderá ser usada, por exemplo, a encomenda de Id 8.

Segue-se a lista de exemplos de comandos para teste de cada uma das pesquisas:

- **Breath First**

grafoCeleiros(Grafo), bfDistancia(Grafo, rua_do_outeiro, Solucao/CustoSolucao)

grafoCeleiros(Grafo), bfTempo(Grafo, 16, Solucao, CTempoSolucao, CDistanciaSolucao, VeiculoMaisEcologico)

- **Depth First**

grafoCeleiros(Grafo), dfDistancia(Grafo, rua_do_outeiro, Solucao, CustoSolucao)

grafoCeleiros(Grafo),dfTempo(Grafo, 16, Solucao, CTempoSolucao, CDistanciaSolucao, VeiculoMaisEcologico)

- **Pesquisa com Aprofundamento Iterativo**

grafoCeleiros(Grafo),iterativeDeepeningDistancia(Grafo,rua_do_outeiro, Caminho, CustoDistancia, ProfundidadeSolução).

grafoCeleiros(Grafo),iterativeDeepeningTempo(Grafo, 16, Caminho, CTempo, CDistancia, VeiculoMaisEcologico, ProfundidadeSolução)

- **Greedy**

grafoCeleiros(Grafo), greedyDistancia(Grafo, rua_do_outeiro , Caminho/Custo)

grafoCeleiros(Grafo), greedyTempo(Grafo, 16, Solucao, CustoTempoSolucao, CustoDistSolucao, VeiculoMaisEcologico)

- **A***

grafoCeleiros(Grafo), resolveAEstrelaDistancia(Grafo, rua_do_outeiro, Caminho/Custo)

grafoCeleiros(Grafo), resolveAEstrelaTempo(Grafo,16, Solucao, CustoDistSolucao, CustoTempoSolucao, VeiculoMaisEcologico)

- **Encontra Melhor Breath First**

grafoCeleiros(Grafo), encontraMelhorBF(Grafo, 16, carro, MelhorCusto, MelhorCaminho)

- **Mais Ecológico**

grafoCeleiros(Grafo), maisEco(Grafo, 16, Caminho, CDistancia, CTempo, VeiculoMaisEco)

- **Mais Curto**

grafoCeleiros(Grafo), maisCurto(Grafo, rua_do_outeiro , Caminho, Custo)

- **Melhor Circuito**

grafoCeleiros(Grafo), melhorCircuito(Grafo, 16, CamMaisEntregas/NrEntregas , CamMaiorPeso/Pesos, CaminhoMaiorVolume/Volumes)

Resultados

Os resultados apresentados na tabela seguinte foram obtidos ao testar as várias pesquisas no mesmo grafo(grafoCeleiros) e com o mesmo ponto de origem (armazém) e ponto de destino (rua_sub_carreira).

Estratégia	Tempo (segundos)	Espaço	Distância do Caminho Solução (kilometros)	Encontrou a melhor solução?
DFS	0.004	O(bm)	8.9	Não
DFS limitada	0.006	O(bl)	8.9	Não
BFS	0.001	O(b^d)	5.0	Não
Procura Iterativa limitada em profundidade	0.006	O(b^d)	5.0	Não
Greedy	0.006	O(b^m)	2.90	Não
A*	0.001	O(b^d)	2.65	Sim

b -> fator de ramificação

d -> profundidade da solução menos profunda

m -> máxima profundidade da árvore

l -> limite de profundidade

Comentários Finais e Conclusão

A nível de resultados obtidos no final da elaboração deste trabalho, o primeiro ponto a mencionar é a completude da solução apresentada, já que foram elaboradas resoluções para cada uma das estratégias requeridas no enunciado. O segundo ponto a mencionar será a discussão de resultados.

A nível de melhor solução obtida pudemos verificar que apenas a pesquisa A* conseguiu encontrá-la. É digno de notar também, que existe uma discrepância bastante grande entre os resultados obtidos com as pesquisas não informadas e as pesquisas informadas, sendo as segundas as mais eficazes. Essa discrepância é explicada por terem acesso a informação privilegiada relativamente à distância em linha reta entre 2 ruas.

A nível de tempo de execução foram obtidos resultados semelhantes. A causa disto será o facto do grafo utilizado nos testes, apesar de já com um tamanho considerável (12 nodos e 27 arestas) este não será suficientemente grande para haver uma diferença considerável entre as várias estratégias de pesquisa a nível de tempo de execução.

Com o objetivo de melhorar as nossas pesquisas por soluções, confirmamos que a implementação de grafos mais extensos em conjunto com estimativas mais numerosas e precisas levará ao seguimento de pesquisas também mais precisas.

Anexos

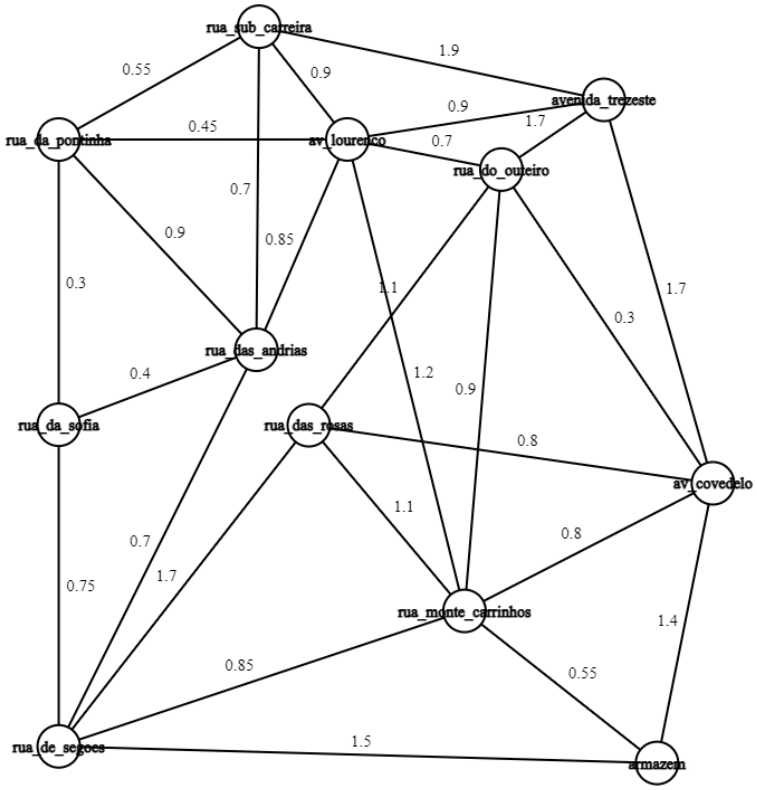

Figura 1 Grafo da freguesia de Celeirós

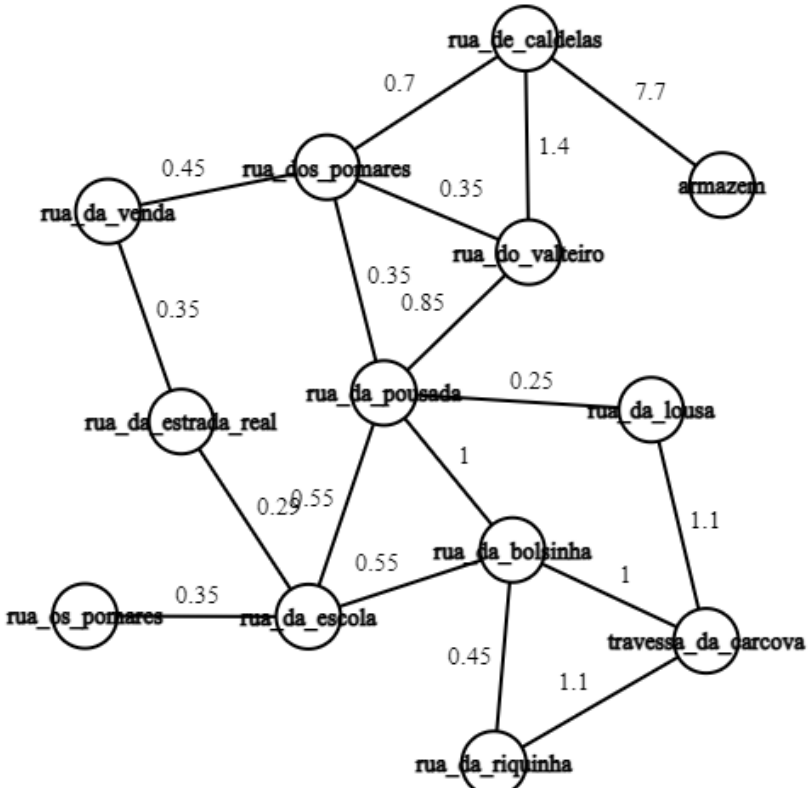

Figura 2 Grafo da freguesia de Martim