Smart Delivery

Empresa de Distribuição de Mercadorias

Autores

Diogo Silva up201706892 up201706892@fe.up.pt

Liliana Almeida up201706908 up201706908@fe.up.pt

Raul Viana up201208089 up201208089@fe.up.pt

Professor: Rosaldo Rossetti

Grupo: 2MIEIC04_C_4

Concepção e Análise de Algoritmos

Faculdade de Engenharia

Universidade do Porto

25 de abril de 2019

Índice

Lista de Figuras

1 Descrição do Tema

A entrega de mercadorias é, atualmente, uma atividade em crescimento exponencial devido, em parte, ao aumento da confiança dos consumidores nas compras online.

Uma dada empresa de distribuição tem a sua atividade na distribuição de encomendas. Para isso utiliza carrinhas que recolhem, diariamente, estas encomendas de um depósito e leva-os até aos seus respetivos destinatários. Posteriormente, confluem à garagem, onde terminam o serviço.

Neste trabalho, vai ser implementado um sistema que permita à empresa gerir a sua frota e os percursos de entregas, minimizando os pesos das distâncias percorridas pelas carrinhas.

As encomendas terão diferentes destinatários, volumes e valor de conteúdo, entre outras informações. As carrinhas terão um volume de carga e o constrangimento de ter a garagem no ponto do depósito ou noutro ponto diferente.

Serão, então, contemplados os casos de aplicação indicados de seguida.

1.1 Carrinha Única com Volume de Carga Ilimitado

Inicialmente será considerado que existe apenas uma carrinha com um volume de carga, sempre maior do que o de todas as encomendas. Um possível caso real desta situação poderá ser o de uma empresa que distribui apenas encomendas de reduzido volume, em que, garantidamente, a soma dos volumes de todas as encomendas daquele dia não ultrapassa o volume total de carga da carrinha.

1.2 Várias Carrinhas com Volume de Carga Limitado

1.2.1 Início e Final de Percurso Diferentes

Nesta situação inicia-se o percurso no depósito e acaba-se numa garagem (localizada num ponto diferente do depósito). As encomendas são distribuídas acrescentando-se carrinhas quando necessário, sendo posteriormente calculado o percurso ideal para cada carrinha. Um possível caso real seria o de uma empresa que dispõe de dois depósitos e cada carrinha inicia o dia carregando as encomendas num desses depósitos, fazendo, de seguida, as entregas e terminando no outro depósito. No dia seguinte faz o sentido inverso.

1.2.2 Início e Final de Percurso Correspondentes

Esta situação poderá modelar um caso real mais provável, no qual a empresa dispõe apenas de uma base, que serve simultaneamente de depósito e de garagem.

Assim, pretende-se que o sistema calcule o número mínimo de carrinhas a serem utilizados e ainda o percurso ótimo para a viagem de cada um, tendo em conta os pontos de entrega de cada encomenda e os pontos inicial e final, minimizando o peso da viagem.

2 Formalização do Problema

Neste trabalho vai ser implementado um grafo para representar o mapa, com as características seguidamente explicitadas.

2.1 Dados de Entrada

M, conjunto de pontos com as moradas (coordenadas) alcançáveis dentro da área de influência escolhida.

D $\in M$, localização do depósito com as encomendas a entregar.

A $\in M$, localização da garagem (poderá ser ou não igual a D).

P $\subset M$, conjunto dos pontos de entrega das encomendas, sendo necessário estar incluídos no percurso entre D e G.

E, conjunto de estradas que ligam as moradas ($E \in M$), com W (peso) calculado pelo custo da viagem (distância percorrida, tempo de viagem, custo do combustível, ...).

G(V,E), grafo dirigido e cíclico pesado, no qual **V** corresponde às várias moradas ($\in M$) e E às estradas que as ligam, podendo ter um ou dois sentidos.

2.2 Dados de Saída

F, conjunto dos vértices ordenados que representam o melhor caminho possível nos diversos percursos que existem, passando por todos os vértices (pontos de entrega), começando em D e terminando em A.

W, peso final resultante da soma dos pesos das estradas percorridas (custo final da viagem).

2.3 Restrições

2.3.1 Restrições aos Dados de Entrada

Qualquer que seja $e \in E$, $peso(e) \geqslant 0$ como o peso de uma estrada representa a distância, ou custo de a percorrer, esta grandeza não pode ser negativa.

2.3.2 Restrições aos Dados de Saída

- $W > 0$, peso terá de ser maior do que zero. Esta será uma consequência da restrição de entrada em que o $peso(e) \geqslant 0$.
- $D \in M$ e $D = M_0$ o depósito terá de ser o primeiro vértice do conjunto de vértices que representa o percurso ótimo.
- $G \in M$ e $G = M_f$ a garagem terá de ser o último vértice do conjunto de vértices que representa o percurso ótimo.
- Qualquer que seja $p \in P$, $p \in F$ todos os pontos de entrega devem pertencer ao percurso calculado.

2.4 Função Objetivo

O objetivo é minimizar o custo da viagem, sendo para isso necessário otimizar as somas dos pesos das arestas (estradas) que ligam D a G passando por todos os pontos de P.

A função h que descreve o que foi referido é:

$$h = \sum weight(e) \quad , \quad e \in F$$

3 Análise de Algoritmos

3.1 1.º Caso

Para começar, há que garantir que o nó correspondente ao ponto de entrega de cada encomenda existe dentro do mapa/raio de entrega da empresa e que está acessível, ou seja, que os seus acessos não estão a ser intervencionados por obras ou existam quaisquer outros constrangimentos que tornem esse nó inacessível, sendo estes removidos se tal acontecer.

Para isso, será utilizada a **pesquisa em largura**.

```
BFS(G, s):
1.      for each v ∈ V do discovered(v) ← false
2.      Q ← ∅

3.      ENQUEUE(Q, s)
4.      discovered(s) ← true

5.      while Q ≠ ∅ do
6.          v ← DEQUEUE(Q)
7.          pre-process(v)
8.          for each w ∈ Adj(v) do
9.              if not discovered(w) then
10.                 ENQUEUE(Q, w)
11.                 discovered(w) ← true
12.         post-process(v)
```

A segunda parte pode ser vista como um percurso num grafo. Este tipo de problema pode ser dividido em duas categorias: problemas do tipo **euleriano** e do tipo **hamiltoniano**. Os problemas do tipo euleriano (carteiro chinês) requerem que cada aresta seja percorrida pelo menos uma vez. Por outro lado, os problemas do tipo hamiltoniano (caixeiro viajante) requerem que cada vértice seja percorrido pelo menos uma vez. Assim, o problema da distribuição de encomendas é enquadrado no problema do **caixeiro viajante**.

A analogia do **caixeiro viajante** pode ser transportada para cada uma das carrinhas de distribuição de encomendas com a restrição de que o ponto inicial será diferente do final, que, neste caso, é uma carrinha única de carga ilimitada, que passa por cada ponto de entrega apenas uma vez. Foi, então, necessário calcular qual o percurso que otimiza o custo total do percurso.

Como referido acima, este caso trata-se de um problema com solução em tempo fatorial, ou seja, a melhor solução apenas pode ser obtida por *brute force*, calculando todas as possibilidades, tendo assim uma complexidade temporal **O(n!)**. É possível, ainda assim, encontrar algumas **heurísticas** que reduzam substancialmente este custo temporal para aproximadamente linear.

Algumas delas, apesar de aparentarem serem soluções muito boas, tanto pela sua fácil implementação como pela rapidez da resposta encontrada, apresentam alguns problemas e limitações. Por exemplo, poder-se-ia utilizar a heurística do **vizinho mais próximo**, mas este algoritmo escolhe sempre como vértice seguinte o vértice que se encontra mais próximo do atual, o que pode levar a que o percurso escolhido não seja o ótimo.

Para contrariar esta questão, pode-se utilizar ainda outra heurística - de *back-tracking* - para integrar na lista dos vértices qualquer vértice que tenha ficado "esquecido" pelo cálculo do vizinho mais próximo. Ao aliar as duas heurísticas é possível reduzir o custo temporal e minimizar eventuais erros inerentes às suas limitações.

Apesar das previsões, acabamos por utilizar a heurística **two-opt** para encontrar a ordem ótima de todas as entregas da carrinha, pois é a que apresenta melhor performance neste tipo de problema. Este algoritmo tem uma complexidade de $\mathbf{O(N^2)}$, sendo N o número de entregas a ordenar.

```
TwoOptAlgorithm(v):  // deliveries' vector with random order
1.      bestWeight ← calcTotalWeight(v)
2.      while noImprovementMade() do
3.          lastBestWeight ← bestWeight
4.          for (i = 1; i < number of nodes eligible to be swapped - 1; i++)
5.              for (k = i + 1; k < number of nodes eligible to be swapped; k++)
6.                  v ← twoOptSwap (newPath, i, k)
7.                  newPath ← v;
8.                  if isPathValid(v)
9.                      newWeight = calcTotalWeight(v)
10.                     if newWeight < bestWeight
11.                         bestWeight = newWeight
12.                         bestPath = v
```

```
TwoOptSwap(v, i, k):  // vector which elements will be swapped between the range
    [i,k]
1.      while i < k do
2.          swap (v[i],[k])
3.          i++
4.          k--
```

Para auxiliar o cálculo da ordem ótima de passagem pelos pontos de entrega, será necessário calcular o melhor percurso a efetuar entre cada ponto, ou seja, o percurso com menos peso total. Isto será realizado aplicando o algoritmo de **Dijkstra**.

```
DIJKSTRA(G, s):   // G = (V,E), s ∈ V
1.      for each v ∈ V do
2.          dist(v) ← ∞
3.          path(v) ← nil
4.      dist(s) ← 0
5.      Q ← ∅ // min-priority queue
6.      INSERT(Q, (s, 0)) // inserts s with key 0

7.      while Q ≠ ∅ do
8.          v ← EXTRACT-MIN(Q) // greedy
9.          for each w ∈ Adj(v) do
10.             if dist(w) > dist(v) + weight(v,w) then
11.                 dist(w) ← dist(v)+ weight(v,w)
12.                 path(w) ← v
13.                 if w ∉ Q then // old dist(w) was ∞
14.                     INSERT(Q, (w, dist(w)))
15.                 else
16.                     DECREASE-KEY(Q, (w, dist(w)))
```

Este algoritmo tem como objetivo o cálculo do melhor caminho entre dois vértices do grafo. Na nossa aplicação vai ser utilizado, expecificamente, para calcular o melhor caminho entre um ponto de entrega e o próximo ponto de entrega. Como se trata de um algoritmo **ganancioso**, garante sempre o melhor caminho, o que o torna bastante eficiente e, relativamente, fácil de implementar. Este algoritmo tem um comportamento semelhante ao usado anteriormente (Pesquisa em Largura), mas, em alternativa ao uso da fila para armazenar a ordem dos vértices a pesquisar, é utilizada uma **fila de prioridade variável**. Nesta fila de prioridade são guardados os vértices ordenados pelo menor peso total das suas arestas, pertencentes ao melhor caminho. Assim, este algoritmo apresenta-se com uma complexidade temporal de $\mathbf{O((|V| + |E|) \times log|V|)}$.

3.2 2.º Caso

Este caso trata-se de um problema do tipo "Problema da Mochila" e será solucionado aplicando **programação dinâmica** a cada carrinha, até não haver mais nenhuma entrega para distribuir por estas.

Este algoritmo tenta encontrar uma solução ótima, maximizando, para isso, o número de encomendas alocadas a cada carrinha, através das relações entre o volume total de uma carrinha e o volume e peso total das encomendas a serem entregues. No entanto, acabou-se por adaptar este algoritmo para maximizar apenas o volume das encomendas.

```
DistributeDeliveries(v): // vector of deliveries
1.    V[n+1][maxVol+1]
2.    for ( i = 0; i <= n; i++)
3.       for (w = 0; w <= maxVol; w++)
4.          if (i == 0 || w == 0)
5.             V[i][w] = 0;
6.          else if (volumes[i - 1] + V[i - 1][w - volumes.at[i - 1]] <= w).
7.             V[i][w] = max(volumes[i - 1] + V[i - 1][w - volumes.[i - 1] ], V[i -
     1][w]);
8.          else
9.             V[i][w] = V[i - 1][w];
```

Assim sendo, terá, portanto, uma complexidade temporal de $\mathbf{O(N \times M)}$ e espacial de $\mathbf{O(M)}$, sendo $\mathbf{M}$ o volume total de uma dada carrinha e $\mathbf{N}$ o número de encomendas a entregar.

O restante do caso será tratado da mesma forma que o primeiro caso.

3.2.1 Ponto Inicial Igual ao Ponto Final

Este caso particular trata-se de uma instância direta do problema do *caixeiro viajante*, em que o ponto inicial é o ponto final.

Será, assim, calculado o número necessário de carrinhas, como no 2.º caso, e, posteriormente, o percurso ótimo, também como no 2.º caso, com a única diferença do ponto inicial ser também o final.

4 Estruturas de Dados Utilizadas

As estruturas de dados utilizadas foram as seguintes:

- *Graph* desenvolvido ao longo das aulas práticas que possui vários algoritmos que operam sobre este, tal como o algortimo *Breadth First Search* (BFS) e *Dijkstra*. Esta estrutura recorre também à *MutablePriorityQueue*, da autoria de João Pascoal Faria.
- *GraphViewer* fornecido pelos docentes da Unidade Curricular que permite vizualizar e manipular o grafo.
- *Node* guarda a informação de cada vértice do grafo como o seu ID, a sua posição e a sua etiqueta, utizadas posteriormente no *GraphViewer*.

5 Conectividade do Grafo

O grafo utilizado para representar e operar sobre os mapas foi criado com base nas informações destes constantes nos ficheiros fornecidos pelos docentes e extraídos do *OpenStreetMaps*.

Após o seu *parsing*, constatamos que existiam alguns problemas de conetividade. Existiam muitos nós isolados, alguns com conexão apenas para si próprios e outros com apenas uma única aresta unidirecional a partir destes. À exceção destes últimos em que consideramos que ainda assim seria possível efetuar entregas dentro destes circuitos, procedemos à remoção dos primeiros casos, através da implementação simples de alguns algoritmos para o efeito.

De qualquer forma, tudo isto aumenta o risco de não ser possível encontrar um ponto válido entre os pontos escolhidos, passando pelos pontos de entrega. Para aumentar a taxa de sucesso, optamos por tornar todas as arestas de ligação bidirecionais.

Deste modo, o grafo torna-se conexo, diminuindo a complexidade das operações nele realizadas.

6 Casos de Utilização

A interação com o utilizador é intermediada por uma interface simples e intuitiva. Nesta, é possível escolher o mapa de uma determinada região que deverá ser a área de operação pretendida para a empresa de distribuição. Este mapa permite, portanto, a sua decomposição num grafo sobre o qual o programa vai operar e disponibilizar as variadas funcionalidades.

O programa inicia-se no menu principal.

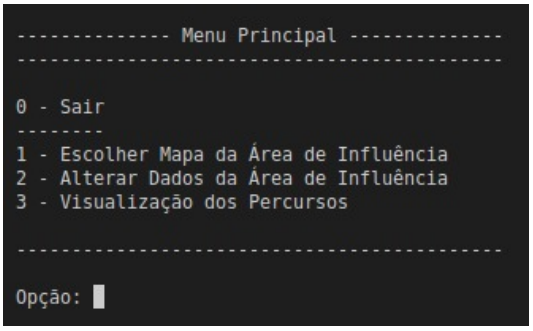

Figura 1: Menu principal.

6.1 Escolher Mapa da Área de Influência

Aqui é permitido selecionar um dos mapas disponíveis (que se encontre no diretório /Maps), e este será o mapa sobre o qual se vai operar até ao fim do programa.

Figura 2: Escolha do mapa da área de influência.

Depois de escolhido, procede-se ao carregamento das informações relacionadas com aquela região, tais como **nós**, **arestas**, **etiquetas** e **encomendas**.

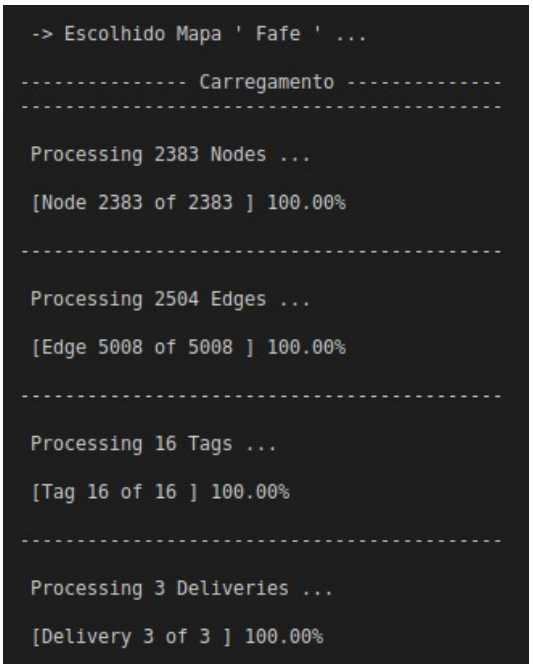

Figura 3: Carregamento do mapa da área de influência.

6.2 Alterar Dados da Área de Influência

Após a seleção e respetivo carregamento do mapa da área pretendida (só assim poder-se-á prosseguir à visualização de caminhos), é possível alterar alguns dos seus dados, entre os quais, **local de partida**, **local de chegada**, **pontos de entrega de encomendas** e **carrinhas**.

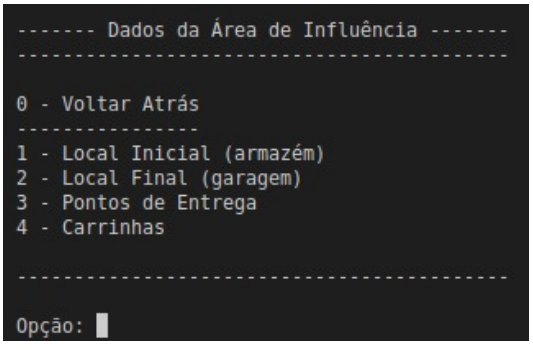

Figura 4: Menu dos dados da área de influência.

6.2.1 Local Inicial

Aqui é possível escolher pontos classificados como **logísticos**, i.e, pontos do mapa etiquetados para o efeito. É dada a oportunidade de **escolher um pontos específico** (1ª opção) (que terá de verificar a condição anteriormente referida para poder ser usado) ou escolher entre uma lista de cinco pontos gerada aleatoriamente.

Figura 5: Escolha de um ponto de partida.

6.2.2 Local Final

Este menu é em tudo semelhante ao anterior, com o detalhe de agora ser sugerido também o ponto inicial (2.ª opção) já escolhido. Aliás, sempre que necessário escolher um destes pontos, poderá ser apresentado nenhum, um ou ambos os pontos iniciais e finais de acordo com o que já tiver sido escolhido.

Figura 6: Escolha de um ponto de chegada.

6.2.3 Pontos de Entrega

Neste sub-menu pode-se proceder a uma listagem dos pontos de entrega existentes ou adicionar um novo ponto de entrega.

De referir que foram criados ficheiros com listas de encomendas, de forma a simular mais eficazmente um plano de trabalho diário, sendo que estas são, como referido anteriormente, carregadas assim que é carregado o mapa.

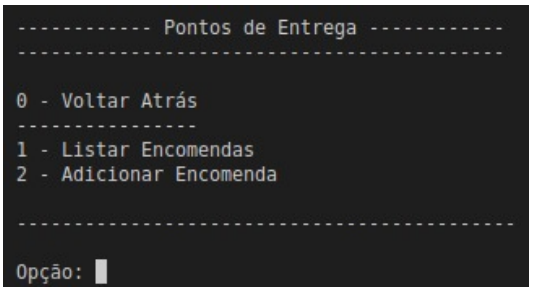

Figura 7: Menu de entregas.

Para adicionar uma encomenda faz-se uma pequena listagem das etiquetas dos locais indicados para entregas e posteriormente uma listagem com os locais assim etiquetados igualmente disponíveis.

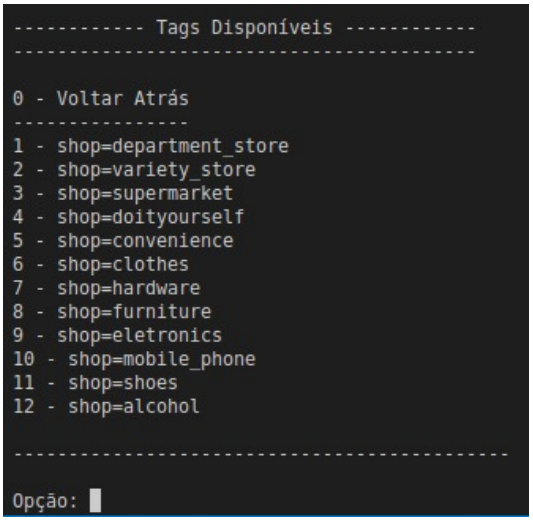

Figura 8: Menu de locais de entregas.

6.2.4 Carrinhas de Entregas

Tal como no anterior, é possível listar as carrinhas já disponíveis ou adiconar novas.

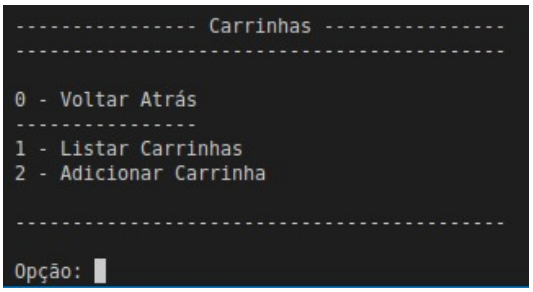

Figura 9: Menu de carrinhas.

6.3 Visualização de Percursos

Depois de se definirem os pontos inciais de finais e caso existam encomendas para se entregar, poder-se-á aceder ao menu de visualização de caminhos.

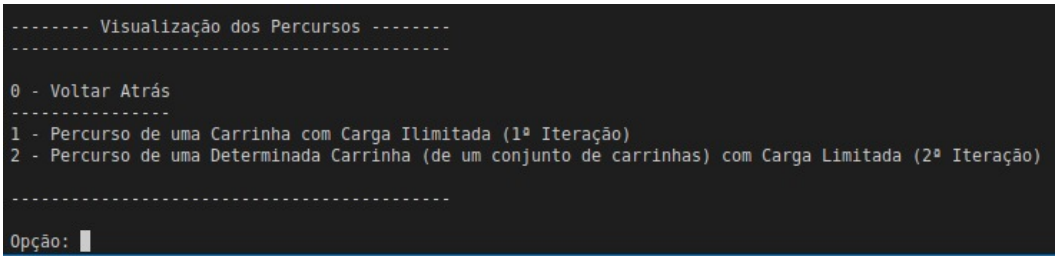

Figura 10: Menu dos caminhos.

6.3.1 Percurso da Carrinha com Carga Ilimitada

Esta é a primeira iteração do projeto, sendo possível visualizar o melhor percurso a ser efetuado por uma só carrinha de capacidade ilimitada para entregar todas as encomendas necessárias. Este caminho é apresentado tanto graficamente (no *GraphViewer*), como em texto, como pode ser visto na *Figura 11*.

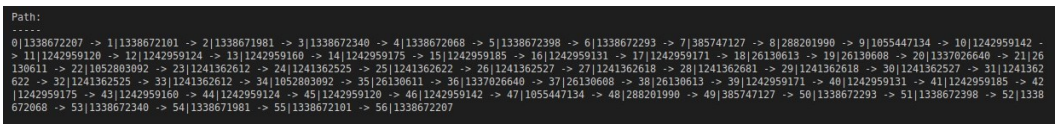

Figura 11: Percurso da carrinha com capacidade ilimitada.

6.3.2 Percurso das Carrinhas com Cargas Limitadas

Por fim, a segunda iteração do projeto. Aqui só é permitido avançar caso estejam disponíveis carrinhas. As carrinhas usadas para a distribuição são apresentadas para que o seu percurso possa ser visualizado. As carrinhas são identificáveis pela enumeração e respetiva capacidade.

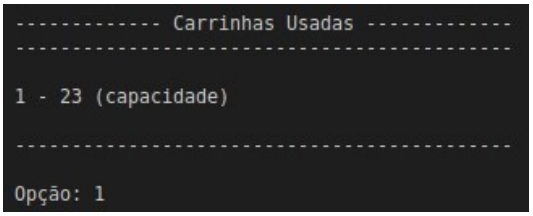

Figura 12: Menu das carrinhas utilizadas.

Sendo escolhida uma das carrinhas é apresentado o seu percurso (exatamente como acontece na 1ª iteração), seguido das informações acerca do seu conteúdo.

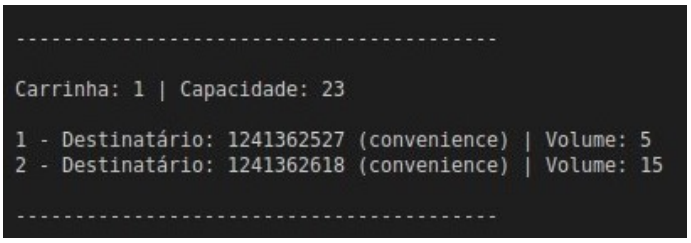

Figura 13: Percurso da carrinha com capacidade ilimitada.

7 Conclusão

Com este trabalho pusemos em prática os conhecimentos adquiridos ao longo do semestre de forma bastante mais prática e autónoma, o que nos permitiu não só a desenvolver as nossas capacidades de programação, como a utilizar corretamente estruturas de dados valiosas, nomeadamente os grafos.

Acabamos por cumprir praticamente todos os parâmetros que previmos, sendo apenas necessário modificar e/ou adaptar alguns dos algortimos que planificamos.

De notar que, a separação das encomendas pelas carrinhas tendo em conta apenas o seu volume, pode levar a erros consideráveis em termos de distância percorrida por cada carrinha, e.g., podem acontecer situações em que tendo apenas em consideração os seus volumes, uma carrinha poderá ter de efetuar entregas em pontos extremamente distantes entre si, enquanto uma outra faz um conjunto de entregas situados, de tal forma, que seria mais eficiente ser esta a entregar alguma dessas encomendas.

Para minimizar este problema foi ponderado acrescentar um algoritmo que dividisse as encomendas em áreas para que, posteriormente, a distribuição das encomendas pelas várias carrinhas tivesse esse fator em consideração e, desta forma, otimizado também o seu uso, mas por restrições de tempo não foi possível incluir este detalhe.

Os elementos do grupo trabalharam de forma equitativa, tendo as tarefas sido dividas igualmente por todos. Esta atitude de cooperação entre os membros revelou-se essencial para uma adequada planificação e desenvolvimento do projeto.

Bibliografia

Rossetti, R., Ferreira, L., Teófilo, L., Filgueiras, J., & Andrade, F. (2019). Slides das Aulas Teóricas. *Concepção e Análise de Algoritmos, Faculdade de Engenharia da Universidade do Porto*.

Wikipedia. (n.d.). *2-opt*. Retrieved from `https://en.wikipedia.org/wiki/2-opt`

(Rossetti, Ferreira, Teófilo, Filgueiras, & Andrade, 2019)
(Wikipedia, n.d.)