BosHBus: Transporte de trabalhadores

CAL 2019/2020

MIEIC FEUP

João de Jesus Costa
up201806560
joao.jcosta@fe.up.pt

João Lucas Silva Martins
up201806436
up201806436@fe.up.pt

Tiago Duarte da Silva
up201806516
up201806516@fe.up.pt

April 24, 2020

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Contents

Chapter 1

Descrição do tema

1.1 Descrição sucinta do problema

Uma empresa de transportes especializa-se em transportes de trabalhadores no percurso *casa-trabalho* (e vice-versa) de forma eficiente. Para isso, dispõe de veículos que precisam de fazer o percurso de uma garagem até à empresa cliente, passando por diversos pontos de encontro onde estarão os trabalhadores a transportar. No final do dia de trabalho, o transporte fará o percurso inverso. Podem existir diversos pontos de encontro de modo a que os trabalhadores consigam facilmente chegar desde suas casas ao seu designado ponto.
Inicialmente, a empresa de transporte possui apenas um veículo e fornece os seus serviços a apenas uma outra empresa. No entanto, o número de veículos e empresas clientes poderá vir a aumentar ao longo do tempo, acompanhando a expansão da empresa.

Este problema pode ser dividido em quatro partes.

1.2 Escolha de pontos de encontro e seu número

É necessário definir uma distância máxima a que a casa de um trabalhador pode estar em relação ao ponto de encontro mais próximo. Esta distância tem de ser pequena o suficiente para que qualquer trabalhador se consiga deslocar a pé até esse ponto de encontro (e posteriormente até casa) em pouco tempo.
Com base nessa distância, é possível definir a quantidade mínima de pontos e as suas posições para recolher todos os trabalhadores.

Com isto, podemos ver que o grafo terá quatro tipos de nós: o nó de origem, os nós de destino, os nós marcados como ponto de encontro e os restantes nós (sem nenhuma nomenclatura especial).

1.3 Cálculo da rota mais curta

Depois de todos os pontos de encontro serem conhecidos, é necessário calcular a melhor rota para o transporte seguir.
A melhor rota é definida como: a rota mais curta que parte da garagem do autocarro e termina na empresa cliente, passando por todos os pontos de encontro pelo menos uma vez.
Esta rota será seguida pelo autocarro em ambos os sentidos, ou seja, da garagem à empresa cliente e da empresa cliente à garagem.
É necessário ter em atenção que obras podem tornar certas vias inacessíveis.
Nesta fase do problema, assumimos que o autocarro tem capacidade para transportar todos os trabalhadores na sua rota.

1.4 Determinação de rotas para vários autocarros (um só destino)

É possível que um autocarro não tenha capacidade para transportar todos os trabalhadores de uma empresa cliente de uma só vez.
Para isto, será necessário calcular o número mínimo de veículos necessários para transportar todos os trabalhadores e a melhor rota para cada um desses veículos.

É de notar que, mesmo que um autocarro tenha capacidade para fazer uma rota completa sozinho, usar mais autocarros (caso estejam disponíveis) pode ser mais eficiente.

1.5 Determinação de rotas para vários autocarros (vários clientes)

Serão precisos vários autocarros para conseguir satisfazer os requerimentos de diversas empresas clientes ao mesmo tempo, visto que cada autocarro só tem uma empresa como destino.
Este problema pode ser resolvido determinando uma rota para cada empresa de forma individual.

Chapter 2

Formalização do problema

2.1 Definições base

- V é o conjunto de todos os nós do mapa. Cada nó representa uma intersecção entre duas ou mais vias.
- i representa o índice de um nó no mapa.
- E é o conjunto de todas as arestas do mapa. Cada aresta representa uma via no mapa.
- $G = (V, E)$ é um grafo dirigido, pesado e conexo, constituído por vértices e suas arestas. Cada aresta do grafo representa uma via e cada vértice representa uma intersecção entre duas ou mais vias.
- C é a lista de empresas clientes e sua informação.
- n representa o índice de uma empresa cliente na lista *clientes*.

2.2 Dados de entrada

- d_{max} representa a distância máxima a que um ponto de encontro pode estar de um local de residência.
- b_{num} representa o número de autocarros disponíveis.
- b_{cap} representa capacidade (em número de pessoas) de cada autocarro disponível.
- Todas as interseções, V_i, de duas ou mais vias no mapa: $V_i \in V$ é o nó de índice i no grafo.
- O nó onde se localiza a garagem dos autocarros, $V_0 \in V$.

- Todas as vias, $i \to j$, no mapa: $i \to j \in E$ representa uma aresta dirigida (pesada) desde o nó de índice i até ao nó de índice j.
- O comprimento, w_{ij}, de todas as vias de comutação $i \to j$. w_{ij} é o peso da aresta $i \to j$.
- A localização do local de trabalho dessa empresa, ou seja, o local onde os trabalhadores devem ser deixados, D_{C_n}, para cada empresa.
- A lista dos locais de residência, H_{C_n}, de cada trabalhador de cada empresa cliente.

2.3 Dados de saída

- A lista dos autocarros, B, a usar.
- b é o índice de um autocarro na lista B.
- O comprimento, L_{B_b}, do caminho a ser percorrido por cada autocarro.
- A sequência ordenada, $P_{B_b} \in E$, de arestas a visitar por cada autocarro, terminando no ponto de destino.
- A lista de pontos de encontro, $M_{B_b} \in V$, que o autocarro vai visitar.
- A lista de passageiros, T_{B_b}, atribuídos a cada autocarro.
- A lista dos pontos de encontro, M, a usar (nós do grafo).
- u representa o índice de um certo ponto de encontro na lista M.
- A lista dos trabalhadores, W_{M_u}, atribuídos ao ponto de encontro de índice u.

2.4 Função objectivo

Minimizar o comprimento total do caminho percorrido:
$min(\sum_{b:B_b \in B} L_{B_b})$
Minimizar o número total de autocarros usados:
$min(|B|)$

Todos os pontos de encontro são atingidos:
$\forall m \in M, \exists p \in M_{B_b} : m = p$

Todos os autocarros têm de chegar ao destino correto.
Sendo *dest* o último elemento da sequência B_{bm} e D o destino da empresa cliente a que B_b se refere, tem de se verificar:
$\forall b : B_b \in B : dest = D$

Todos os trabalhadores têm de estar atribuídos a apenas um ponto de encontro.
Todos os trabalhadores têm de entrar no seu devido autocarro:
$\forall worker \in W_{M_y}, worker \in T_{B_b}$

Chapter 3

Casos de utilização e funcionalidades

3.1 Casos de utilização

Existem quatro casos possíveis de operação para a empresa **BosHBus**, nos quais a empresa fornece os seus serviços a:

- um só cliente usando apenas um autocarro.
- um só cliente usando vários autocarros.
- a vários clientes usando apenas um autocarro para cada um deles.
- a vários clientes podendo usar um ou mais autocarros para cada um deles.

3.2 Funcionalidades a implementar

Para a empresa **BosHBus** deverão ser implementadas as seguintes funcionalidades:

- Registo de um novo cliente (podem existir vários clientes registados).
- Remoção do registo de um cliente atual.
- Adicionar/remover autocarros da garagem (implica um possível atualização de todas as atuais rotas). É de notar que remover um autocarro pode, em certos casos, não ser possível dadas as necessidades dos atuais clientes.

Para as empresas clientes deverão ser implementadas as seguintes funcionalidades:

- Remover/Adicionar novos trabalhadores à sua empresa cliente.
- Atualizar a informação relativa aos trabalhadores da empresa.
- Atualizar o local de destino dos seus trabalhadores.

Chapter 4

Formalização da Solução

4.1 Prespetiva de solução

4.1.1 Escolha de pontos de encontro

Uma solução alternativa à maximização do número de pontos de encontro também foi equacionada pelo grupo. Nesta estratégia, uma fila de prioridade auxiliar com o máximo à cabeça é usada para armazenar os pontos de encontro. Os elementos são inseridos nesta fila de prioridade com base no número de trabalhadores que se encontram a uma distância aceitável desses (distância do trabalhador ao ponto $<= d_{max}$).

- $\forall T \in H_{C_n}$ de uma dada C_n, fazer uma pesquisa em profundidade pelas suas arestas adjacentes. A pesquisa num vértice pára quando a sua distância a T ultrapassar d_{max}. Cada vértice alcançado que ainda não tenha sido visitado anteriormente deverá ser processado.

- Se o ponto de encontro referente ao vértice encontrado estiver na fila de prioridade então é lhe adicionado uma referência a T.
  Caso contrário, um novo ponto de encontro é adicionado à fila com uma referência ao trabalhador T.

- Quando todos os trabalhadores forem processados, a fila de prioridade deverá ser esvaziada, seguindo os seguintes passos:

  - Extrair o ponto de encontro do topo da fila, adicionando-o à lista de resultados.
  - Apagar todas as referências a trabalhadores que podem deslocar-se ao ponto retirado (no passo anterior).
  - Atualizar a fila.
  - Repetir até não haverem mais pontos de encontro na fila de prioridade.

Uma estratégia descartada pelo grupo consistia em:

- Encontrar um grupo de trabalhadores com grande chance de poderem ter um ponto de encontro em comum, usando o primeiro método nesta secção.
- Usar o algoritmo de **dijsktra bidirecional** entre dois trabalhadores do grupo, escolhidos aleatoriamente.
- Obter o ponto de encontro dos dois algoritmos de **dijsktra** a correr em simultâneo.
- Executar os dois passos anteriores entre um outro trabalhador e o novo ponto obtido, até não haverem mais trabalhadores a processar.

Esta estratégia foi descartada pois, em alguns casos, a solução obtida ultrapassa a distância máxima d_{max} à qual um trabalhador pode estar de um ponto de encontro.

Diminuição da amostra a analisar

Com o propósito de diminuir o tempo de processamento total, iremos tentar dividir o número total de pessoas a recolher em vários *clusters*. Cada *cluster* será composto por, idealmente, só trabalhadores que poderão ter pontos de encontro em comum.
Para isso, iremos usar conjuntos disjuntos, unindo dois grupos sempre que a área que um trabalhador pode alcançar a pé (um círculo com raio d_{max} e centro no trabalhador) se intersetar com a área de outro.
Para simplificar esta fase de "triagem", iremos ter em conta apenas um círculo geométrico à volta do trabalhador. Assim, dois trabalhadores irão potencialmente ter pontos de encontro em comum sempre que a distância geométrica entre esses for menor ou igual a d_{max}.
Esta estratégia pode ser descrita da seguinte forma:

- Criar um *container* com todos os trabalhadores e inicializar cada um deles com **NULL** na raiz (todos estão em conjuntos disjuntos).
- Iterar por todos os trabalhadores e, caso não pertençam ao mesmo conjunto, verificar se a distância entre ele é inferior a d_{max}. Caso a distância entre eles seja inferior a d_{max}, unimos os trabalhadores num mesmo conjunto.
- Durante as pesquisas, comprimir sempre que possível a união para reduzir o tempo das futuras iterações.
- Para cada conjunto resultante, utilizar o método descrito no capítulo acima.

4.1.2 Cálculo da rota mais curta

Para o cálculo da rota mais curta de cada autocarro, a lista de pontos de encontro a visitar tem de ser previamente obtida (por exemplo, com uma das estratégias mencionadas anteriormente).
De seguida, deve-se calcular todas as distâncias entre D_{C_n}, onde C_n é a empresa cliente pela qual B_b é responsável, V_0 e cada um dos M_{B_b}. Este cálculo deve ser feito através do **algoritmo de Dijkstra**[1] ou do **algoritmo de Floyd Warshall** [2]. Com esta informação, é possível construir um novo grafo, G', que nos permitirá reduzir drasticamente as dimensões de $|V|$ e $|E|$.

Neste caso, o cálculo do melhor caminho passando por cada M_{B_b} corresponde a uma versão alternativa do 'Travelling Salesman Problem'[3] (daqui em diante referido como **TSP**). As diferenças entre o nosso problema e a solução original centram-se em intencionarmos que o **'tour'** termine num ponto previamente especificado (local de trabalho da empresa em questão) em vez do ponto inicial.
É possível converter esta versão alternativa do **TSP** na versão *original* deste adicionando um vértice e duas arestas auxiliares. Para isso, adicionamos a G' um vértice V_{n+1} com 2 novas arestas, $V_{n+1} \rightarrow V_0$ e $V_{n+1} \rightarrow D_{C_n}$, ambas com peso 0.
O caminho posteriormente obtido, utilizando um algoritmo de resolução do **TSP** (como por exemplo, o **'Nearest Neighbour Algorithm'**) partindo de V_0, irá ser uma aproximação à rota mais curta desde D_{C_n} a V_0.
Depois de calculado, este caminho deverá ser invertido de modo a obtermos o caminho que o autocarro deve seguir. Em alternativa, se D_{C_n} for escolhido como vértice inicial para os cálculos, o caminho não terá de ser invertido no fim do processo.
No caso de haver obras numa via, os pontos de encontro e as rotas são recalculadas retirando as vias afetadas do grafo.

4.1.3 Determinação de rotas para vários autocarros (vários clientes)

Tal como mencionado durante a descrição deste trabalho, iremos sempre assumir que múltiplas empresas não poderão partilhar um mesmo autocarro. Assim, cada empresa terá sempre no mínimo um autocarro exclusivamente para si.

Para resolver a necessidade de múltiplos autocarros para uma só empresa, se um autocarro ficar cheio, ele é enviado diretamente para a empresa e é chamado um novo para recomeçar a rota a partir do ponto em que outro ficou.
Este processo é repetido até todos os trabalhadores tiverem sido levados à respetiva empresa.

4.2 Principais algoritmos a ser considerados

Abaixo encontra-se o pseudo código de alguns algoritmos a ser considerados para o projeto.

Algoritmo de Dijkstra[1]:

```
1 function Dijkstra(Graph, source):
2   // Initialization
3   dist[source] <- 0
4   create vertex priority queue Q
5
6   for each vertex v in Graph:
7     if v != source
8       dist[v] <- INFINITY      // Unknown distance from source to v
9       prev[v] <- UNDEFINED     // Predecessor of v
10    Q.add_with_priority(v, dist[v])
11
12  while Q is not empty:
13    u <- Q.extract_min()       // Remove and return best vertex
14    for each neighbor v of u:  // only v that are still in Q
15      alt <- dist[u] + length(u, v)
16      if alt < dist[v]
17        dist[v] <- alt
18        prev[v] <- u
19        Q.decrease_priority(v, alt)
20
21  return dist, prev
```

'Nearest Neighbour Algorithm'[4]:

```
1 function NearestNeighbour(D[1..n, 1..n], s):
2   // Input: A nxn distance matrix D[1..n, 1..n] and
3   // an index s of the starting city.
4   // Output: A list Path of the vertices containing
5   // the tour is obtained.
6
7   for i <- 1 to n:
8     Visited[i] <- false
9   Initialize the list Path with s.
10  Visited[s] <- true
11  Current[s] <- s
12
13  for i <- 2 to n:
14    Find the lowest element in row current and unmarked
15    column j containing the element.
16    Current <- j
17    Visited[j] <- true
18    Add j to the end of list Path.
19
20  Add s to the end of list Path.
21  return Path
```

Chapter 5

Conclusão

Este estudo tinha como foco a resolução do problema da escolha das rotas mais eficientes para transporte de trabalhadores desde suas casas até aos seus locais de trabalho. Para isto, definimos diversas etapas a alcançar para resolver os subproblemas: determinação de rotas para vários autocarros (vários clientes), determinação de rotas para vários autocarros (um só destino), cálculo da rota mais curta e escolha de pontos de encontro e seu número.
Infelizmente, não foi possível encontrar uma solução ótima e eficiente para todos os subproblemas. Nomeadamente, a determinação dos pontos de encontro e a solução que escolhemos implementar para a determinação das rotas mais curtas ('Nearest Neighbour Algorithm').

De qualquer modo, acreditamos que as soluções a implementar conduzirão a resultados próximos o suficiente do ótimo.

O trabalho foi distribuído uniformemente por todos os elementos do grupo.

Bibliography

[1] W. contributors, "Dijkstra's algorithm." https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm, April 2020. Accessed on 2020-04-02.

[2] W. contributors, "Floyd–warshall algorithm." https://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm, April 2020. Accessed on 2020-04-02.

[3] W. contributors, "Travelling salesman problem." https://en.wikipedia.org/wiki/Travelling_salesman_problem, April 2020. Accessed on 2020-04-03.

[4] Unknown, "Introduction to the design and analysis of algorithms, chapter 12.3, problem 2e." https://www.chegg.com/homework-help/write-pseudocode-nearest-neighbor-algorithm-assume-input-gi-chapter-12.3-problem-2e-solution-9780132316811-exc. Accessed on 2020-04-08.