**Faculdade de Engenharia da Universidade do Porto**



**Relatório Projeto CAL**

Parte 1

*RideSharing: partilha de viagens*

11 abril 2018

Grupo 10 – Turma 2

Gonçalo Santos 201603265 up201603265@fe.up.pt

João Vieira 201603190 up201603190@fe.up.pt

Susana Lima 201603634 up201603634@fe.up.pt

Conteúdo

[Introdução 3](#_Toc510974469)

[Explicação do problema 4](#_Toc510974470)

[Descrição do problema 5](#_Toc510974471)

[1ªiteração: percurso mais rápido 5](#_Toc510974472)

[2ªiteração: percurso mais rápido que maximize o número de passageiros sem restrições temporais e de capacidade 5](#_Toc510974473)

[3ªiteração: percurso mais rápido que maximize o número de passageiros com restrições temporais e de capacidade 6](#_Toc510974474)

[4ªiteração: alargamento do critério de escolha de passageiros 6](#_Toc510974475)

[Formalização do problema 7](#_Toc510974476)

[Dados de entrada 7](#_Toc510974477)

[Dados de saída 8](#_Toc510974478)

[Restrições 8](#_Toc510974479)

[Sobre os dados de entrada 8](#_Toc510974480)

[Sobre os dados de saída 9](#_Toc510974481)

[Funções objetivo 9](#_Toc510974482)

[Descrição da solução 10](#_Toc510974483)

## Introdução

No âmbito da unidade curricular de Conceção e Análise de Algoritmos do 2º ano do MIEIC foi proposta a resolução de um problema da forma mais eficiente possível usando os conhecimentos adquiridos nas aulas.

Neste relatório são descritos o problema e a melhor estratégia encontrada para a sua resolução.

## Explicação do problema

O tema do projeto incide sobre *RideSharing*, partilha de viagens cujo objetivo seria a otimização do recurso automóvel, evitando que este se desloque com apenas um ocupante.

A ideia básica por detrás deste conceito é a partilha de automóveis por diferentes pessoas cuja origem e destino de viagem coincida parcial ou totalmente com as de um condutor.

O programa desenvolvido permite o cálculo de percursos mais curtos que permitam maximizar o número de ocupantes de um veículo entre uma origem e um destino, tendo em consideração as restrições de tempo impostas tanto pelo condutor como pelos possíveis passageiros, bem como as horas de chegada aos pontos de encontro.

Foi considerada também a familiaridade entre passageiros, ou seja, pessoas que viagem em conjunto.

### Descrição do problema

O problema em questão foi decomposto em quatro iterações:

## 1ªiteração: percurso mais rápido

Numa primeira fase, o objetivo tratou-se simplesmente em calcular a rota mais curta entre a origem e o destino do condutor. Deste modo, os passageiros transportados seriam aqueles cujo caminho mais curto (entre as origens e destinos dos mesmos) estivesse contido de forma parcial ou total no trajeto mais rápido do condutor.

É importante notar que o percurso só pode ser calculado caso existam vias de ligação, diretas ou indiretas entre a origem e o destino considerados.

## 2ªiteração: percurso mais rápido que maximize o número de passageiros sem restrições temporais e de capacidade

Numa segunda fase, o cálculo do caminho teve em consideração o número de pessoas a transportar, sendo que o objetivo passou a ser o caminho mais curto que permitisse o transporte do máximo número de pessoas.

É de notar que a maximização do número de passageiros foi privilegiada sobre a minimização do tempo de viagem.

A capacidade do veículo do condutor bem como qualquer tipo de condicionamentos temporais foram desprezados.

## 3ªiteração: percurso mais rápido que maximize o número de passageiros com restrições temporais e de capacidade

Numa terceira fase, foram consideradas algumas restrições, nomeadamente a capacidade do veículo em cada ponto de recolha de passageiros, o limite de tempo, imposto pelo condutor e pelos passageiros, de chegada ao destino e as horas de comparência nos pontos de encontro.

Assim sendo a rota calculada teria de ter uma duração menor ou igual ao limite de tempo definido pelo motorista. Do mesmo modo, só seriam recolhidas pessoas cujo limite de tempo imposto não fosse excedido.

Relativamente ao limite de capacidade, caso o automóvel se encontrasse completamente ocupado ou não tivesse lugares disponíveis para acomodar todos os elementos de um grupo os passageiros não seriam transportados.

## 4ªiteração: alargamento do critério de escolha de passageiros

Numa quarta fase, a escolha de passageiros foi alargada de modo a que pessoas cujo trajeto mais curto não estivesse contido no itinerário do condutor passassem a ser transportada caso os pontos de origem e de destino estivessem contidos, tendo sempre em consideração as restrições impostas.

### Formalização do problema

O programa implementado permite, de uma forma geral, resolver o problema de *ride sharing* recebendo um ponto de partida, com a respetiva hora, e um ponto de chegada, um limite de tempo máximo para o percurso a efetuar, atendendo à capacidade do veículo, caso o utilizador se trate de um condutor ou o número de pessoas a viajar em grupo, caso o utilizador seja um passageiro.

## Dados de entrada

Pi – Sequência de passageiros que aguardam transporte, sendo Pi(i) o seu i-ésimo elemento. Cada um é caracterizado por:

* nump - Número de pessoas com quem viaja
* srcp - Ponto de partida
* destp - Ponto de chegada
* hip -Hora de partida
* tlp - Limite de tempo

Ci – Sequência de condutores disponíveis, sendo Ci(i) o seu i-ésimo elemento. Cada um é caracterizado por:

* cap - Capacidade do automóvel que conduz (excluindo o lugar do condutor)
* srcc - Ponto de partida
* destc - Ponto de chegada
* hic - Hora de partida
* tlc -Limite de tempo

Gi(Vi,Ei) – grafo dirigido pesado composto por:

* V – vértices – representam todos os pontos de encontro (temo de por aqui a cena do time e assim??)
* E – arestas – representam vias de ligação entre os pontos (estradas, ruas) com:
  + wt – corresponde ao tempo de viagem entre os dois vértices que a delimitam
  + wp – corresponde ao número de pessoas que aguardam transporte entre os dois vértices que a delimitam
  + w – peso da aresta - valor pesado entre o tempo de viagem e o numero de possíveis passageiros a aguardar transporte

As origens e destinos dos passageiros e condutores estão contidos em V.

## Dados de saída

Gf(Vf,Ef) – grafo dirigido em que Vf e Ef têm os mesmos atributos que Vi e Ei, embora os valores dos mesmos podem ser diferentes. No entanto, é de referir que Ef diferencia- se de Ei caso tenham sido transportados passageiros (remoção dos passageiros que aguardavam transporte).

Cf – sequência ordenada de todos os condutores utilizados sendo Cf(i) o seu i-ésimo elemento. Cada um com:

* I – sequência de vértices a visitar entre o vértice de partida e o de chegada, correspondente ao itinerário do condutor
* Tp - Número de pessoas transportadas ao longo do percurso
* Hfc - Hora de chegada

Pf – sequência de passageiros, sendo Pf(i) o seu i-ésimo elemento. Pode diferir de Pi caso parte (ou todos) dos passageiros que aguardavam transporte tenham sido transportados. Cada um com:

* I – sequência de vértices visitados, caso o passageiros tenha sido transportado, nula senão
* Pc – posição atual, entre srcp (a pessoa não foi transportada) e a destp (a pessoa foi transportada até ao seu destino) inclusive
* Hcp - hora atual, pode ou não ser igual a hip (a pessoa não foi transportada)

## Restrições

## Sobre os dados de entrada

* ∀ i ∈ [1 ; |Ci|] , cap(Ci(i)) > 0, visto a capacidade de um veiculo se tratar de um número inteiro positivo e , no contexto do problema, não interessar um automóvel que apenas possa transportar o condutor (capacidade = 0).
* ∀ i ∈ [1 ; |Pi|] , nump(Ci(i)) > 0, uma vez que o número de pessoas (representativo do número de elementos de um grupo) é um número inteiro positivo maior que zero, visto um grupo com zero elementos não fazer sentido no problema em questão.
* Seja P = Pi U Ci (não sei se é esta a notação correta), ∀ i ∈ [1 ; |P|], 18<= idade(P(i)) <= 70, intervalo considerado relativamente realista no contexto apresentado.
* ∀ i ∈ Ei, wt > 0, dado que representa o tempo entre dos pontos de um mapa
* ∀ i ∈ Ei, wp >= 0, dado que representa o número de pessoas que aguardam transporte entre dois pontos de um mapa. Pode não existir nenhuma pessoa a aguardar transporte entre dois determinados pontos mas o veículo nunca vai vazio (wp = 0).
* ∀ i ∈ Ei, w > 0, consequência das duas restrições anteriormente apresentadas, uma vez que é calculado em função de wt e wp.

## Sobre os dados de saída

* ∀ i ∈ [1 ; |Ci|] , i ∈ [1 ; |Cf|], ou seja, todos os condutores presentes inicialmente também estão presentes no fim, mas com diferentes valores nos atributos
* |Ci | = |Cf |, pelo motivo apresentado em cima
* ∀ i ∈ [1 ; |Pi|] , i ∈ [1 ; |Pf|], ou seja, todos os passageiros presentes inicialmente também estão presentes no fim, mas com diferentes valores nos atributos, caso sejam transportados
* |Pi | = |Pf |, pelo motivo apresentado em cima
* ∀ i ∈ [1 ; |Vi|] , i ∈ [1 ; |Vf|], o que significa que todos os vértices presentes no grafo inicial estão presentes no grafo final, embora possam diferir no valor de certos atributos
* |Vi| = |Vf| pelo motivo apresentado em cima
* ∀ i ∈ [1 ; |Ei|] , i ∈ [1 ; |Ef|], o que significa que todas as arestas presentes no grafo inicial estão presentes no grafo final, embora possam diferir no valor de certos atributos
* |Ei| = |Ef| pelo motivo apresentado em cima

## Funções objetivo

(não sei se é isto)

A solução ótima encontrada passa por minimizar o tempo de viagem e maximizar o número de passageiros transportados durante a mesma.

f - |wt|

g - |wp|

No programa implementado foi privilegiada a maximização do número de passageiros sobre a minimização do tempo. (explicar função reta 3d????)

## Descrição da solução

(Não esta completo)

Numa primeira abordagem do problema foram apenas considerados como possíveis passageiros as pessoas cujo caminho mais curto coincidisse parcial ou totalmente com o percurso do condutor, tendo sempre em consideração as restrições temporais impostas e as horas de chegada dos passageiros e dos condutores aos pontos de encontro. A capacidade do veículo também foi uma restrição imposta.

Convém salientar que a familiaridade entre pessoas foi considerada sendo que grupos de pessoas que viajassem em conjunto não poderiam ser separados, ou seja, um grupo só seria transportado caso no veículo existissem lugares disponíveis para acomodar todos os elementos do mesmo.

No entanto, pessoas cuja origem e destino estivessem contidas no percurso do condutor não seriam transportadas, embora todas as outras restrições fossem cumpridas, se o percurso mais rápido da mesma não estivesse contido (em parte ou na totalidade) no itinerário do motorista.

Numa segunda fase, de modo a evitar a situação descrita, foi acrescentado ao programa um pós-processamento que verificasse a existência de pessoas que cumprissem todas as restrições de tempo nos pontos de recolha contidos no percurso do condutor, tendo em consideração a lotação do carro nos sucessivos pontos, cuja origem e destino pertencessem ao caminho em questão. Se tal acontecesse essas pessoas seriam também transportadas.

Como consequência deste pós-processamento, pessoas cuja origem e destino estivessem contidas no percurso do condutor seriam transportadas se as condições impostas fossem cumpridas.

O algoritmo principal no cálculo dos passageiros que acompanham os condutores e os caminhos percorridos é uma versão modificada do algoritmo de Dijkstra, desenhada para além de minimizar o tempo gasto no percurso condutor, maximizar o número de viajantes no carro ao longo do percurso. Este algoritmo tem em conta a origem e o destino dos condutores e dos passageiros, e várias outras restrições anteriormente explicadas. Os vértices são guardados numa *min-priority quele* / fila de prioridade mínima ordenados pelo tempo à origem do condutor, por ordem crescente (será repetitivo?). Antes de visitar as arestas de cada vértice a capacidade do veículo naquele momento é recalculado, evitando assim a sua sobrelotação. Em cada aresta, calcula-se o número de passageiros que têm condições de entrar no carro. Este valor é usado no cálculo da distância ao próximo vértice: (sendo *u* o vértice atual e *v* o seguinte)

Caso este valor seja menor que a anterior distância de v, o limite de tempo do condutor não seja ultrapassado e o carro não esteja sobrelotado, os valores de *v* são atualizados, a chave de *v* *min-priority quele* / fila de prioridade mínima na é decrementada e os passageiros são acrescentados ao condutor.

Por fim, quando o algoritmo chega ao nó de destino, o caminho percorrido é visitado recursivamente, os passageiros transportados são acrescentados a uma lista de passageiros e contabilizados para serem retornados. O local em que cada passageiro foi deixado pelo condutor é também atualizado.

# Análise da solução

No pior caso, a complexidade do algoritmo descrito é de O(D\* (E + V + P)logV ). A solução apresentada não é ótima, uma vez que não é capaz de, por exemplo, deixar de apanhar um grupo que não lhe encha o carro para mais à frente, apanhar outro grupo que preencha a totalidade do carro.

# Apêndice A: Pseudocódigo do Algoritmo

1. function dijkstraPeopleDistancePath(Graph, source, destination, driver):
2. create vertex set Q
3. create vertex list Path
4. create passenger list Passengers
5. for each vertex v in Graph:
6. v[distance] <- INFINITY
7. v[previous] <- UNDEFINED
8. v[pickedUp].clear()
9. insert v in Q
11. source[distance] <- 0
12. source[time] <- 0
13. ending <- destination
15. while Q is not empty:
16. u <- Q.extractMin()
18. numberPeopleTransported <- 0
19. if (u == ending)
20. while (u[previous] != UNDEFINED)
21. add u to Path
23. for each Passenger p in u[pickedUp]:
24. add p to Passengers
25. add p to driver[passengers]
26. p[currentTime] <- temp[time]
28. if (!p[dropped])
29. add p to driver[droppedAt]
30. p[dropped] = TRUE
31. numberPeopleTransported <- numberPeopleTransported + p[numberPassengers]
33. add u to Path
34. driver[PeopleTransported] <- numberPeopleTransported + driver[PeopleTransported]
36. return Path, Passengers, numberPeopleTransported
37. alreadyPicked <- 0
38. for each Passenger p in driver[passengersPickedAt]:
39. if (p[destination] != p[position] && p[destination] != u)
40. alreadyPicked <- alreadyPicked + p[numberPassengers]
42. for each neighbor v of u:
43. lastAlreadyPicked <- alreadyPicked
44. driver[currentTime] <- u[time]
46. numberPassengersPicked <- 0
47. create Passenger list temporaryPassengersPicked
49. for each Passenger p in Edge(u, v):
50. if (p[position] == u
51. && (alreadyPicked + p[numberPassengers]) <= driver[capacity]
52. && p[currentTime] <= driver[currentTime]
53. && driver[currentTime] < p[startTime] + p[timeLimit])
55. if (!p[picked])
56. lastAlreadyPicked <-lastAlreadyPicked + p[numberPassengers]
58. numberPassengersPicked <- numberPassengersPicked + p[numberPassengers]
59. add p to temporaryPassengersPicked
61. alt <- u[distance] + length(u, v) / (pow(numberPassengersPicked, 2) + 1)
62. time <- u[time] + length(u, v)
63. if (lastAlreadyPicked <= driver[capacity] && time <= driver[timeLimit]
64. && alt < v[distance])
65. v[distance] <- alt
66. v[time] <- time
67. v[previous] <- u
68. v[pickedUp] <- temporaryPassengersPicked
70. Q.decreaseKey(v);
71. for each Passenger p in temporaryPassengersPicked:
72. p[previousPosition] <- p[position]
73. p[position] <- v
75. if (m[picked])
76. erase p from temporaryPassengersPicked
77. else
78. m[picked] <- TRUE
79. add temporaryPassengersPicked to driver[passengersPickedAt]