Gestão de caminhos de transporte de grupos

- João Ricardo Alves - up202007614
- Marco André Rocha - up202004891
- Ricardo de Matos - up202007962

Descrição do problema

Este trabalho tem como objetivo implementar um sistema de gestão de pedidos para transporte de grupos de pessoas entre diferentes locais. Para tal, recorrer-se-á a grafos e algoritmos de capacidade máxima , de fluxo e de caminhos críticos em grafos arco-atividade.

Os algoritmos desenvolvidos devem ser o mais eficientes possíveis temporal e espacialmente, adotando, para isso, vários dos métodos abordados em aula. Deve também ser feita uma avaliação empírica que justifique as opções pelo grupo tomadas.

O trabalho divide-se em **2 cenários** principais a serem explorados:

- **Cenário 1** - Grupos não se separam (capacidade máxima)
- **Cenário 2** - Grupos separam-se (problema de fluxo)

Formalização - Cen. 1

Seja G = (V, E, capacity) um grafo dirigido
Seja C, conjunto de arestas de caminhos em G de *s* para *t*.

Cenário 1.1:

- **Maximizar: Min(e_1 capacity, ... , e_n capacity)**

com e ∈ C

Cenário 1.2:

- **Maximizar: Min(e_1 capacity, ... , e_n capacity)**

com e ∈ C

- **(Λ) Minimizar: |X| - 1,** x ∈ C

cenário pode ser irrealista

Sujeito a:

Caminho de s para t

∀ e ∈ E, e.capacity >= 0

∀ x ∈ C, x[1].ori = s **Λ** x[n].dest = t

∀ x ∈ C, (**∀i** ∈ N, **0 <= i** <= |X| - 1, ∃ e ∈ E:
e = (X[i].dest , X[i+1].ori))

Caminho é válido

Nota: Todos os valores são inteiros positivos

Algoritmos - Cen. 1

No primeiro cenário procuramos caminhos de capacidade máxima e que sejam simultaneamente os mais curtos possíveis (minimizar transbordos). Na procura destes caminhos no grafo poderá ser dada prioridade a qualquer um destes critérios : capacidade / nº transbordos. Portanto, mais do que uma única solução ótima pode existir de acordo com os critérios usados.

Os casos de preferência supramencionados, sendo incomensuráveis, devem ser apresentados e uma solução que seja o “meio-termo” das anteriores deve também ser apresentada. À frente será explicado o conceito de “meio-termo”.

Algoritmos:

- **(1.1) :** Adaptação do algoritmo de Dijkstra com Max Heap
- **(1.2) :** Semelhante ao anterior com dualidade entre capacidade e nº transbordos.

Na 1.2 existe também procura de caminho balanceado dentro do espaço “ótimo”, definido pelas soluções pareto-ótimas. Este assunto será aprofundado mais à frente.

Complexidade - Cen. 1

Sejam:

E: Nº Edges | V: Nº Nodes

Complexidade Temporal:

- **Dijkstra adaptation:** O(E + V * Log V)
- **Dijkstra adaptation:** O(E + V * Log V)

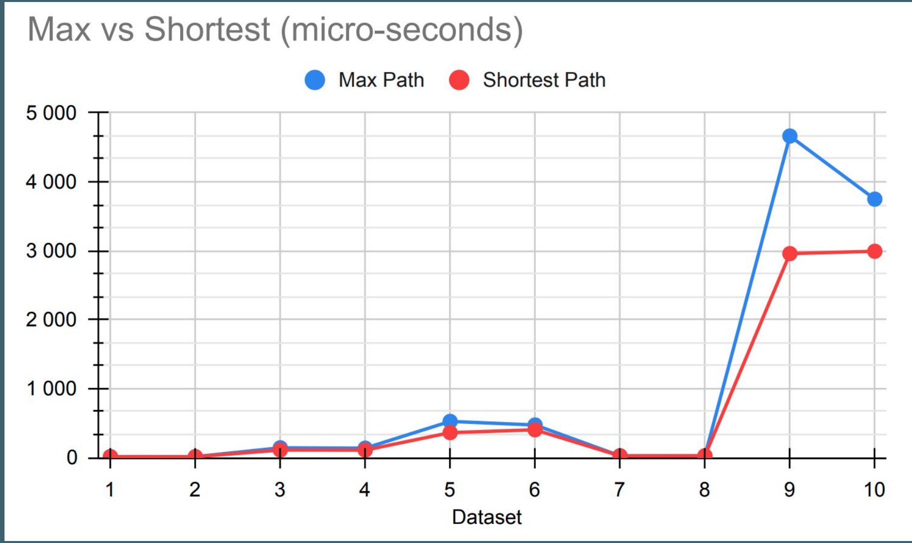
Complexidade Espacial:

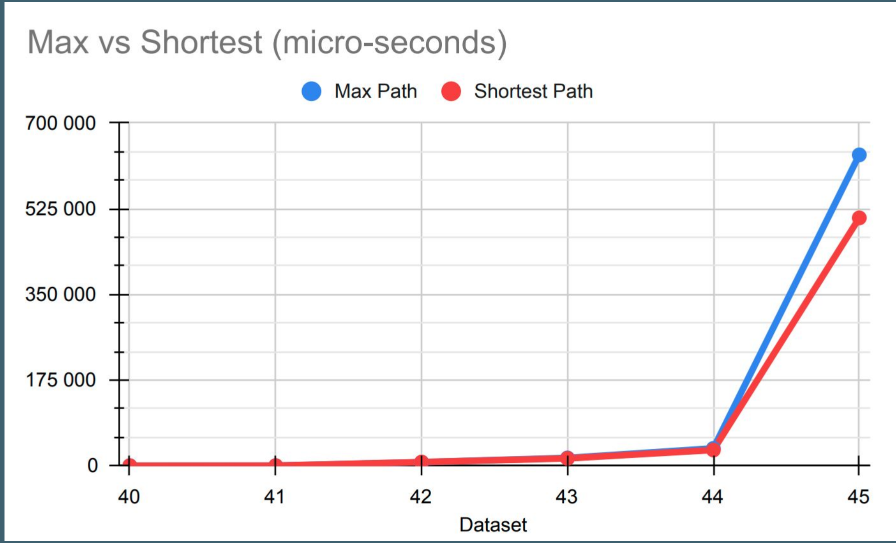
- **Dijkstra adaptation:** O(V + E)
- **Dijkstra adaptation:** O(V + E)

Nota:

No cálculo de soluções “meio termo” ou balanceadas, estes algoritmos são repetidos até mais
nenhuma solução ser encontrada

Resultados - Cen. 1





Como expectável, os algoritmos adaptados a partir do Dijkstra com uma Max Heap têm desempenho virtualmente idêntico (dentro da margem de erro).

É importante realçar que estes valores não incluem o cálculo do caminho intermédio entre os resultados pareto-ótimos. Esse valor dependerá unicamente do tamanho (número de edges) do caminho máximo de cada grafo.

Nota: Dataset(1-10) foi fornecido pelos professores. Dataset(40-45) tem grafos de grande dimensão

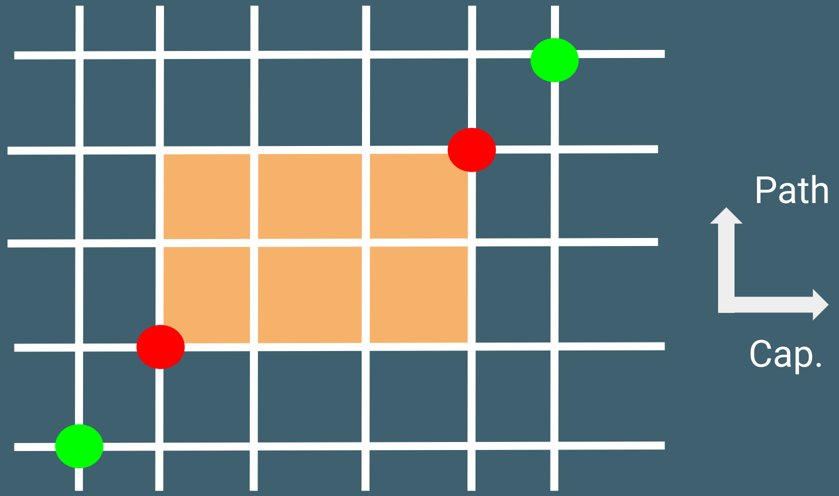
Resultados - Cen. 1

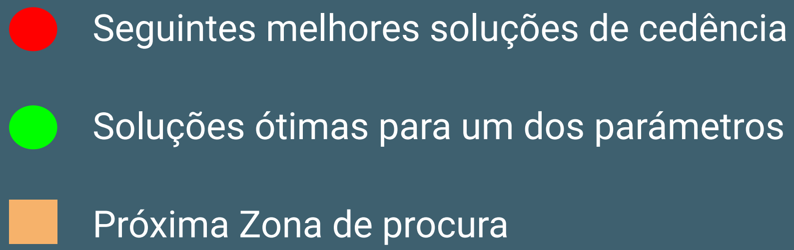
Note-se que uma solução X é melhor que outra Y se Y se situar no 2.º quadrante com centro em X. Ou seja, se Y tiver uma menor ou igual capacidade e/ou maior ou igual nº de transbordos.

Os dois algoritmos garantem a menor path com maior cap. e vice-versa, porém as soluções pareto-ótimas impedem que haja um aumento para uma delas sem prejudicar a outra. É preciso encontrar valores de cedência.

<u>Cedência</u>

Podem existir mais soluções no quadrado central.
Note-se que as duas soluções a vermelho também não são comparáveis.





Formalização - Cen. 2

Seja G = (V, A, {s, t}, c, d, f) um DAG com V nodes e A edges onde s é o source node e t o sink node e onde c,d e f representem a capacidade e duração das edges e o fluxo, respetivamente. Sendo dim, a dimensão do grupo pretendida

Cenario 2.1:

. $|f| = \Sigma$ (s.ori, t.dest).fluxo, **s,t** $\in$ A -
Σ (s.dest, t.ori).fluxo, **s,t** $\in$ A, **Λ |f| = dim**

Cenario 2.2:

. $|f| = \Sigma$ (s.ori, t.dest).fluxo, **s,t** $\in$ A -
Σ (s.dest, t.ori).fluxo, **s,t** $\in$ A **Λ |f| = dim +** X

Cenario 2.3:

. **$Max_s(|f|)$ Λ $|f| = \Sigma$** (s.ori, t.dest).fluxo, **s,t** $\in$ A -
Σ (s.dest, t.ori).fluxo, **s,t** $\in$ A)

Sujeito a:

$\forall$ **e** $\in$ A, $0 \le e.f \le e.c$

Σ (s.ori, t.dest).fluxo =Σ (s.dest, t.ori).fluxo, **s,t** $\in$ A

$\forall$ **x** $\in$ C, x[1].ori = s **Λ** x[n].dest = t

$\forall$ **x** $\in$ C, ($\forall$ **0 <= i** <= |X| - 1, $\exists$ e $\in$ E:
e = (X[i].dest , X[i+1].ori))

Nota: Fluxos inteiros e capacidades em inteiros positivos, como descrito na seção acima.

Seja G = (V, A) um DAG (arco-atividade) com atividades (ori,dest,dur)
Seja C, caminho crítico, composto por atividades, de s para t

Cenario 2.4:

. Σ e.ES, e $\in$ C

Cenario 2.5:

. Max_s(e_{jk}.ES - (e.ES + e.dur), e $\in$ A

Sujeito a:

$\forall$ **e** $\in$ A, e.dur >= 0

$\forall$ **X** $\in$ C, x[1].ori = s $\wedge$ x[n].dest = t

$\forall$ **X** $\in$ C, ($\forall$ **0 <= i** <= |X| - 1, $\exists$ e $\in$ E:
e = (X[i].dest , X[i+1].ori))

$\forall$ **X** $\in$ C, ($\forall$ **1 <= i** <= |X|, X[i].ES = X[i-1].ES + X[i-1].dur

Caminho é crítico

Algoritmos - Cen. 2

- **Edmonds-Karp :** Cálculo do caminho mais curto em cada iteração (através de uma BFS). Atualização do fluxo através desse caminho com base na capacidade mínima. Repetir até não existir caminho residual de S para T. Também foi **feito o algoritmo de Ford-Fulkerson.**

- **Dinic's algorithm :** Cálculo do nível em cada iteração (através de uma BFS), encaminhar fluxo apenas pelas arestas do tipo (ori.level, ori.level + 1), recursivamente, até esgotar a capacidade.

- **Critical path,** earliest start (arco-atividade): cálculo do inicio mais cedo, começando por calcular os graus de cada nó, e indo analisá-los sempre que o grau fica a 0 (usando uma queue onde são inserindo os de grau 0), atualizando os outro quando o cálculo do ES termina (dando push e pop na stack). Partiu-se do princípio que o encaminhamento vem das alíneas anteriores.

- **Max waited time:** Corresponde à folga livre máxima. Depois de recebido o encaminhamento, é computado o ES (acima) e com base neste, ocorre uma iteração nas arestas com fluxo de um dado nó e é calculado para cada a sua folga. É retornada a lista de nós de máxima folga livre.

- **Extras**: implementação da folga total, com uso de um grafo transposto como descrito nos slides

Complexidade - Cen. 2

Sejam:

E -> Nº Edges
V -> Nº Nodes

- Edmonds-Karp algorithm:

$O(N) = O(V E^2)$

$S(N) = O(V + E)$

- Dinic's algorithm:

$O(N) = O(E V^2)$

$S(N) = O(V + E)$

- Critical path(arco-atividade) :

$O(N) = O(V E)$

-> E.flow > 0 e V visitado

$S(N) = O(1)$

- Max waited Time (FL) :

$O(N) = O(V E)$

-> E.flow > 0 e V visitado

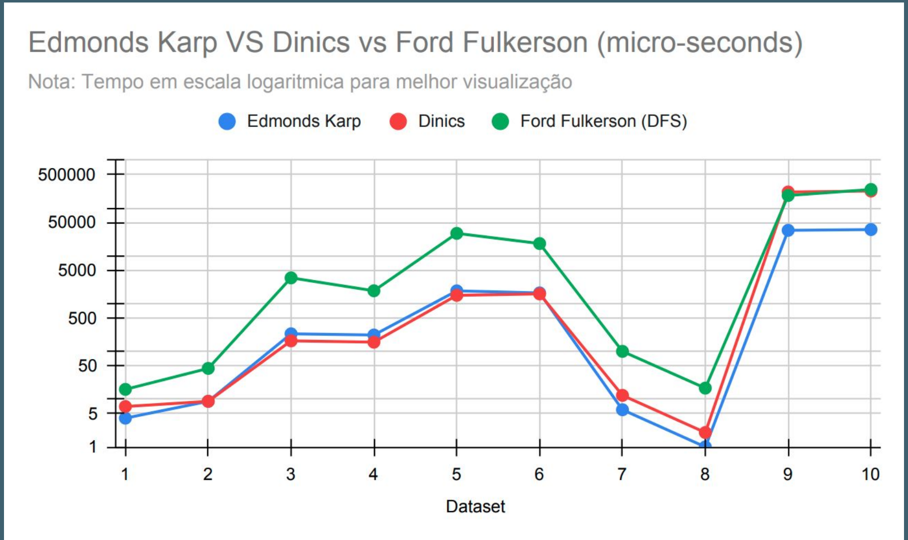
$S(N) = O(N)$

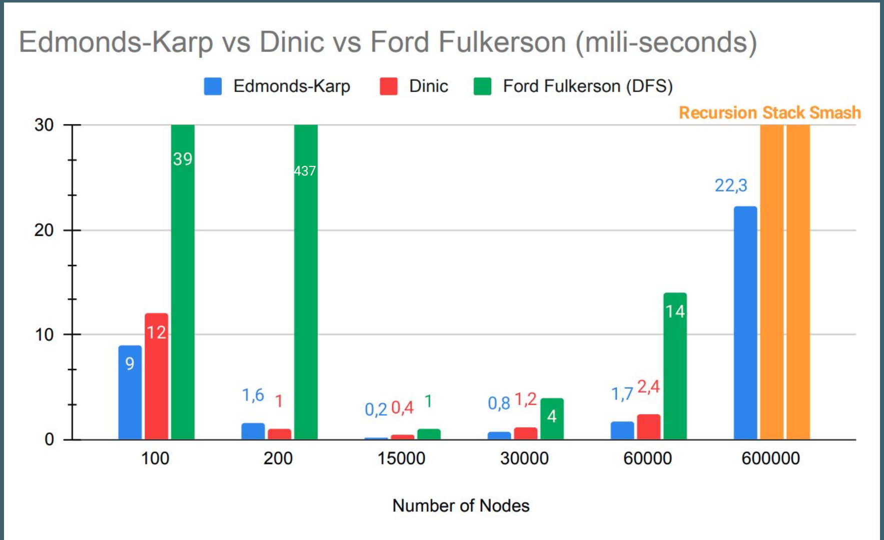
-> Nº de soluções

- Ford-Fulkerson:

$O(N) = O(m^2C)$ / $S(N) = O(n)$

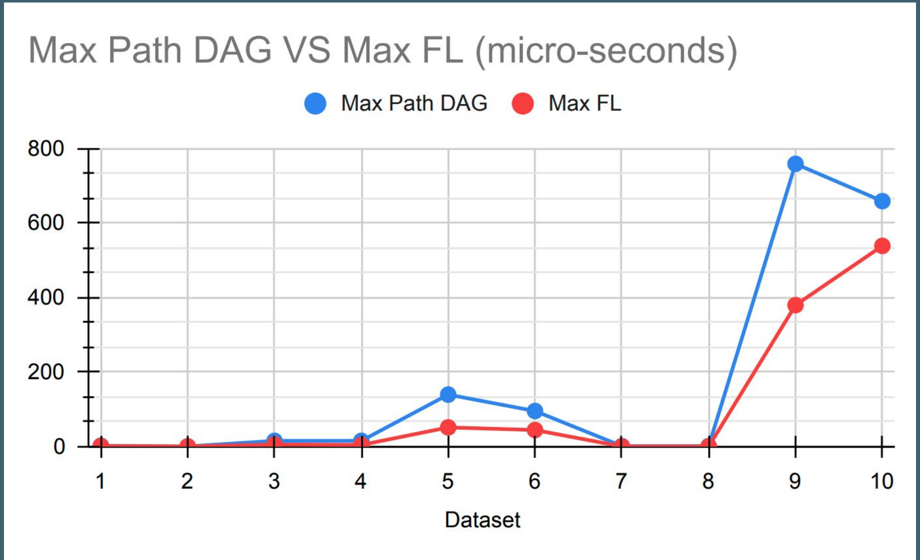
Resultados - Cen. 2

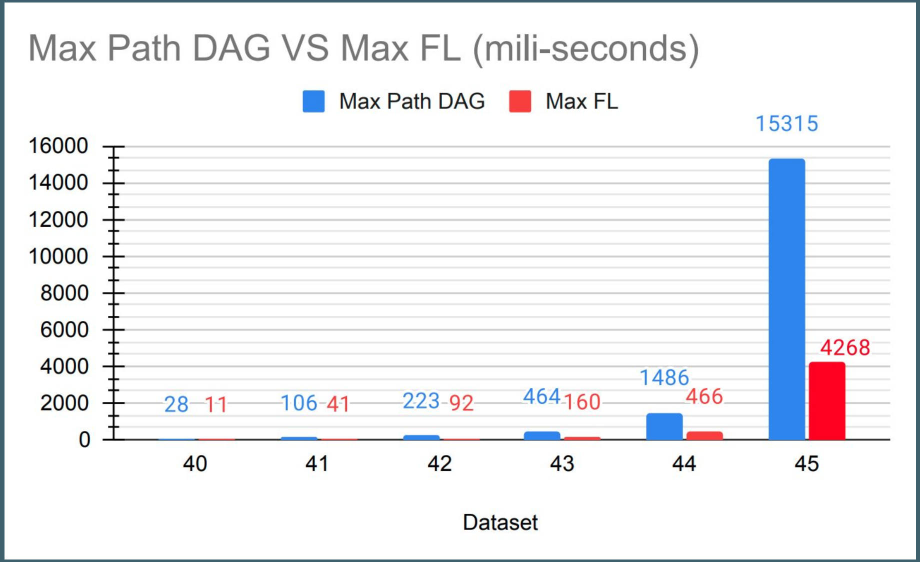




O algoritmo Edmonds-Karp **O(V E ^{2})** é melhor para grafos cujos nodes tenham muitas edges, enquanto que o algoritmo de Dinic, **O(E V ^{2})**, é melhor nas situações opostas. Além disso, a recursividade da nossa implementação do algoritmo de Dinic limita a sua utilização em grafos de grande dimensão por atingir o nº de chamadas recursivas. Também se nota que o algoritmo de Dinic é pior em situações em que temos muitas edges a sair de um nó.
Por fim, veja-se a performance consistentemente pior quando se usa Ford Fulkerson (DFS) e que também está sujeito a limitação de chamadas recursivas.

Resultados - Cen. 2





O algoritmo para saber a folga livre máxima apenas funciona se os Earliest starts já se encontrarem definidos, pelo que na realidade o algoritmo de folga livre máxima seria uma “acrescento” ao algoritmo do caminho crítico em termos de tempo.

Para mais detalhes sobre os gráficos usados consultar:
https://docs.google.com/spreadsheets/d/1UJBUA1irZuZ8NkQW5H9ExPWfttTCzsMqLZEjmcxUxzI/edit#gid=0

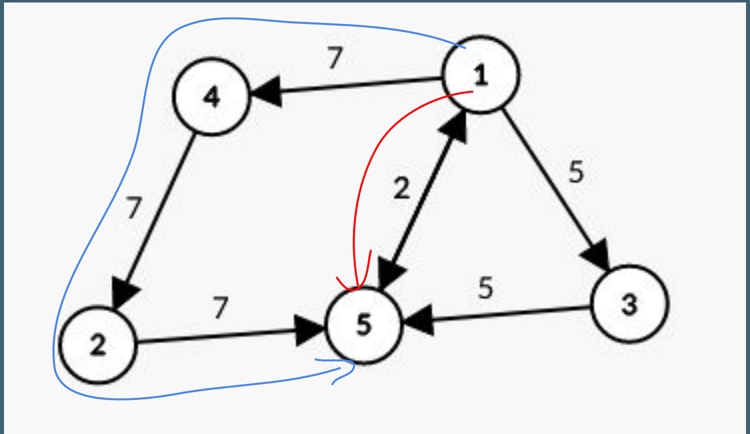
Funcionalidades Extra

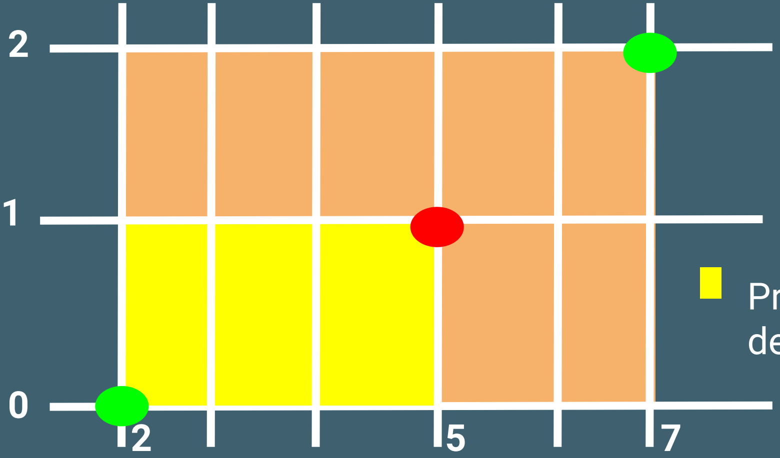
Foram implementadas 4 funcionalidades extra:

1. Cálculo do caminho de capacidade máxima para verificar se se justifica usar fluxo e separar os grupos.
2. Saber a folga total máxima disponível. Ótimo para saber quando um certo encaminhamento pode sair atrasado sem penalizar o fim da viagem. Em vez de poder fazer várias folgas pontuais ao longo do caminho.
3. Cálculo do latest-finish para cada uma das atividades / viagens .
4. Algoritmos de diferentes complexidades para atender a diferentes tipos de DAG e para fins comparativos. A saber:

Ford Fulkerson , Edmonds - Karp , Dinic

Algoritmo Destaque





Próxima Zona de procura

A vermelho encontra-se a solução de path mais pequena no grafo com a maior capacidade. A azul de maior capacidade com a menor path possível. Um algoritmo que apenas procura-se pela de maior capacidade e pela de menor path encontraria apenas estas duas soluções que não podem ser comparáveis.

Porém, existem outras soluções escondidas no grafo. Note o caminho 1-> 3 -> 5, com capacidade de 5 e 1 transbordo.

Achamos que esta propriedade é difícil de notar e, por isso, foi a nossa escolha para algoritmo de destaque.

Dificuldades do Grupo

A principal dificuldade enfrentada pelo grupo foi a conciliação deste trabalho com os de outras unidades curriculares. Tal inclui a gestão do tempo como também gestão dos elementos do grupo.

No final, todos os membros do grupo contribuíram de igual forma para o projeto.

No que concerne aos algoritmos, os de cálculo de caminhos de fluxo máximo foram os mais trabalhosos e difíceis de fazer debug.

Apesar das adversidades, consideramos termos atingidos resultados satisfatórios e que vão de encontro ao que nos foi proposto.

Exemplos de Execução

```
==============================================
                 Scenario 1
==============================================
   1)  Maximize Group Size
   2)  Maximize Group Size VS Shortest Path
   0)  Go Back
==============================================
 > 1

Enter Data set id: 5

The path will be:
1    464  891  587  437  18   564  132  50   431
962  627  741  1000

Path size: 14
The max group size of this path is: 16
```

Cenário 1.1

```
==============================================
                 Scenario 1
==============================================
   1)  Maximize Group Size
   2)  Maximize Group Size VS Shortest Path
   0)  Go Back
==============================================
 > 2

Enter Data set id: 5

Two solutions were computed:

(Max Capacity with shortest path)

The path will be:
1  464  891  587  437  18  564  132  50  431  962  627  741  1000

Path size: 14
The max group size of this path is: 16

(Shortest path with max Capacity)

The path will be:
1  657  171  66  755  1000

Path size: 6
The max group size of this path is: 10

However, there are solutions not represented in the scope.
Those solutions are not parameter optimal but rather balanced solutions
Here are the best balance solutions ...


The path will be:
1  464  547  529  627  741  1000

Path size: 7
The max group size of this path is: 11

The path will be:
1  802  573  587  618  992  616  1000

Path size: 8
The max group size of this path is: 12
```

Cenário 1.2

Exemplos de Execução

```
========================================
               Scenario 2
========================================
   1)  User Given Group Size
   2)  Max Group Size
   3)  Dinic's solution
   4)  Ford Fulkerson
   0)  Go Back
========================================
 > 1

Enter Data set id: 25

Group size: 5

Flow Paths:
6 <-- 5 <-- 2 <-- 1 <-- 0 <-- New flow = 3
6 <-- 5 <-- 3 <-- 1 <-- 0 <-- New flow = 2

Found flow is: 5

------------------ Times ------------------

MIN Duration of the trip: 2
If schedule does not have any delay:
No Waiting breaks

'Folga Total': 3
```

Cenário 2.1

```
=====================================
              Scenario 2
=====================================
   1)  User Given Group Size
   2)  Max Group Size
   3)  Dinic's solution
   4)  Ford Fulkerson
   0)  Go Back
=====================================
 > 2

Enter Data set id: 26


Flow Paths:
16 <-- 8 <-- 3 <-- 2 <-- 1 <-- New flow = 5
16 <-- 8 <-- 7 <-- 5 <-- 1 <-- New flow = 3
16 <-- 15 <-- 14 <-- 13 <-- 1 <-- New flow = 4
16 <-- 12 <-- 11 <-- 10 <-- 9 <-- 1 <-- New flow = 9

Found flow is: 21

------------------ Times ------------------

MIN Duration of the trip: 5
If schedule does not have any delay:
Max Wait at stop:  16 is 1

'Folga Total': 4
```

Cenário 2.2

Nota: Os casos **2.4** e **2.5** são incluídos em todas as instâncias de **2.1** a **2.3**

Exemplos de Execução

```
=====================================
              Scenario 2
=====================================
   1)  User Given Group Size
   2)  Max Group Size
   3)  Dinic's solution
   4)  Ford Fulkerson
   0)  Go Back
=====================================
 > 2

Enter Data set id: 1


Flow Paths:
50 <-- 12 <-- 46 <-- 8 <-- 1 <-- New flow = 1
50 <-- 41 <-- 33 <-- 6 <-- 1 <-- New flow = 1
50 <-- 31 <-- 40 <-- 48 <-- 8 <-- 1 <-- New flow = 2
50 <-- 39 <-- 19 <-- 7 <-- 37 <-- 38 <-- 1 <-- New flow = 2
50 <-- 39 <-- 44 <-- 22 <-- 35 <-- 38 <-- 1 <-- New flow = 1
50 <-- 39 <-- 17 <-- 36 <-- 10 <-- 6 <-- 1 <-- New flow = 3

Found flow is: 10

------------------ Times ------------------

MIN Duration of the trip: 135
If schedule does not have any delay:
Max Wait at stop/node 50 is of time: 99

'Folga Total': 133
```

Cenário 2.3 (Edmonds-karp)

```
=====================================
              Scenario 2
=====================================
   1)  User Given Group Size
   2)  Max Group Size
   3)  Dinic's solution
   4)  Ford Fulkerson
   0)  Go Back
=====================================
 > 3

Enter Data set id: 25


Flow Paths:
1 --> 2 --> 5 --> 6
1 --> 2 --> 3 --> 5 --> 6
1 --> 2 --> 3 --> 5 --> 4 --> 6
1 --> 2 --> 1 --> 2 --> 3 --> 5 --> 4 --> 6
1 --> 2 --> 3 --> 5 -->
Found flow is: 8

------------------ Times ------------------

MIN Duration of the trip: 4
If schedule does not have any delay:
Max Wait at stop/node 3 is of time: 1
Max Wait at stop/node 5 is of time: 1

'Folga Total': 3
```

Cenário 2.3 (Dinic’s algorithom)

Exemplos de Execução

```
=====================================
           Scenario 2
=====================================
   1)  User Given Group Size
   2)  Max Group Size
   3)  Dinic's solution
   4)  Ford Fulkerson
   0)  Go Back
=====================================
 > 4

Enter Data set id: 26


Flow Paths:
16 <-- 8 <-- 4 <-- 3 <-- 2 <-- 1 <-- New flow = 5
16 <-- 8 <-- 7 <-- 5 <-- 1 <-- New flow = 3
16 <-- 12 <-- 11 <-- 10 <-- 9 <-- 1 <-- New flow = 9
16 <-- 15 <-- 14 <-- 13 <-- 1 <-- New flow = 4
16 <--
Found flow is: 21

------------------ Times ------------------

MIN Duration of the trip: 5
If schedule does not have any delay:
Max Wait at stop/node 8 is of time: 1
Max Wait at stop/node 16 is of time: 1

'Folga Total': 4
```

Cenário 2.3 (Ford-Fulkerson)

```
=====================================
           Scenario 2
=====================================
   1)  User Given Group Size
   2)  Max Group Size
   3)  Dinic's solution
   4)  Ford Fulkerson
   0)  Go Back
=====================================
 > 1

Enter Data set id: 5

Group size: 5

A path was found where the group doesnt need to break apart:

The path will be:
1 464 891 587 437 18 564 132 50 431 962 627 741 1000

The max group size of this path is: 16

New group size (0 to skip): 18

Applying flow...

Flow Paths:
1000 <-- 300 <-- 987 <-- 534 <-- 802 <-- 1 <-- 0 <-- New flow = 2
1000 <-- 821 <-- 113 <-- 694 <-- 802 <-- 1 <-- 0 <-- New flow = 3
1000 <-- 14 <-- 701 <-- 27 <-- 802 <-- 1 <-- 0 <-- New flow = 1
1000 <-- 755 <-- 604 <-- 566 <-- 702 <-- 1 <-- 0 <-- New flow = 6
1000 <-- 936 <-- 719 <-- 118 <-- 702 <-- 1 <-- 0 <-- New flow = 1
1000 <-- 838 <-- 442 <-- 924 <-- 754 <-- 1 <-- 0 <-- New flow = 5

Found flow is: 18

------------------ Times ------------------

MIN Duration of the trip: 108
If schedule does not have any delay:
Max Wait at stop/node 1000 is of time: 96

'Folga Total': 108
```

Cenário 2.1 (Caminho Único)

Fim

- João Ricardo Alves - up202007614
- Marco André Rocha - up202004891
- Ricardo de Matos - up202007962