Problema de Transporte - Método Dual

Henrique Oliveira da Cunha Franco

1 Introdução

Este documento apresenta uma análise experimental de uma implementação que calcula caminhos disjuntos em arestas em grafos direcionados utilizando o método de fluxo máximo. O objetivo é avaliar a eficiência da implementação, medida pelo tempo de execução para instâncias de diferentes tamanhos de grafos.

2 Descrição da Implementação

O código foi desenvolvido em Java e consiste de duas classes principais, e uma auxiliar: `TransportationProblem`, `DualMatrixTransportation`, e `TransportationProblemGenerator`, respectivamente.
A classe `TransportationProblem` é responsável pela combinação entre problemas aleatórios gerados em `TransportationProblemGenerator` e a resolução dos mesmos, que é feita utilizando a função `.solve()` da classe `DualMatrixTransportation`. A seguir está uma explicação mais detalhada do código da implementação utilizada para o problema em questão.

3 Metodologia

A classe `DualMatrixTransportation` possui uma ampla variedade de variáveis e métodos organizados em torno dos seguintes objetivos principais:

- Armazenar os parâmetros do problema, como a matriz de custos (`costMatrix`), os vetores de oferta e demanda (`supply` e `demand`), e outras variáveis auxiliares.
- Inicializar as estruturas necessárias para a solução, como vetores `u` e `v` e a matriz `dMatrix`.
- Implementar métodos iterativos para encontrar células de entrada e saída da base (`leavingCell` e `enteringCell`).
- Atualizar a matriz dual e recalcular os valores relevantes a cada iteração.
- Verificar a condição de otimalidade e encerrar o processo quando necessário.

3.1 Principais Métodos

3.1.1 `initializeA()`

Inicializa o vetor `A`, que combina os valores de oferta e demanda. O vetor `A` é composto pelos valores de demanda (`demand`) e pelos valores negativos da oferta (`supply`), refletindo a conservação de fluxo no problema.

3.1.2 `initializeUAndComputeV()`

Esse método inicializa o vetor `u` com zeros e calcula os valores de `v`. A escolha de `v` é baseada no menor custo em cada coluna da matriz de custos. Essa etapa prepara os vetores para uso nos cálculos da matriz dual.

3.1.3 `defineBasicCellSet()`

Determina o conjunto de células básicas que formarão a base inicial do problema. A seleção das células é feita com base nos menores custos encontrados em cada coluna da matriz de custos. Células virtuais também são definidas para atender as restrições do problema.

3.1.4 `computeObjectiveFunction()`

Calcula o valor da função objetivo ψ com base nos vetores `u` e `v` e nos conjuntos de células básicas. Esse valor representa o custo total associado à solução atual.

3.1.5 `determineLeavingCell()`

Identifica a célula que deixará a base no próximo passo. Para isso, calcula-se o vetor `Y` como o produto de `A` pela matriz dual `dMatrix`. A célula correspondente ao menor valor em `Y` é escolhida como a célula de saída.

3.1.6 `determineEnteringCell()`

Após determinar a célula de saída, este método identifica a célula de entrada, utilizando a matriz `theta`, que calcula os custos reduzidos para cada célula não básica.

3.1.7 `updateDMatrix()`

Atualiza a matriz dual `dMatrix` com base na célula de entrada e saída determinadas. A matriz `T`, que mantém o histórico das células básicas, também é atualizada.

3.1.8 `solve()`

Esse é o método principal que coordena o fluxo do algoritmo. Ele inicia as variáveis, executa os passos iterativos do método dual, e verifica a condição de otimalidade até que a solução ótima seja alcançada.

3.2 Geração de problema aleatórios

A classe `TransportationProblemGenerator` código Java implementa um gerador de problemas de transporte. Ele cria dados aleatórios para problemas balanceados ou desbalanceados, salvando-os em arquivos de texto. A seguir, descrevemos os principais elementos do programa:

1. **Importações Necessárias:**
   - `java.io.*`: Para manipulação de arquivos.
   - `java.util.Random`: Para geração de valores aleatórios.
2. **Método `generateRandomProblem`**:
   - Gera um problema de transporte com os seguintes parâmetros:
     - `filePath`: Caminho do arquivo para salvar os dados gerados.
     - `m`: Número de nós de oferta.
     - `n`: Número de nós de demanda.
     - `supplyMax`: Valor máximo de oferta.
     - `costMax`: Valor máximo de custo.
     - `isBalanced`: Indica se o problema deve ser balanceado ou não.
   - Gera valores aleatórios para oferta (`supply`) e demanda (`demand`).
   - Calcula os totais de oferta e demanda e, se necessário, ajusta os valores para balanceamento.
   - Cria uma matriz de custos (`costMatrix`) com valores aleatórios.
   - Escreve os dados gerados em um arquivo de texto com o seguinte formato:
     (a) Cabeçalho: Número de nós de oferta (`m`) e demanda (`n`).
     (b) Lista de ofertas.
     (c) Lista de demandas.
     (d) Matriz de custos.

3.3 Resolução dos problemas gerados

Por fim, O arquivo `TransportationProblem` implementa uma solução para o problema de transporte utilizando uma abordagem baseada em leitura de arquivos, manipulação de dados, e escrita de resultados. Ele contém métodos para:

- Leitura de arquivos contendo os dados do problema.
- Impressão e validação dos dados.
- Solução do problema de transporte utilizando a matriz de custos e as ofertas e demandas fornecidas.
- Escrita dos resultados em um arquivo de saída.
- `readFile(String filePath)`: Lê um arquivo contendo o número de nós de oferta e demanda, as listas de oferta e demanda, e a matriz de custos. Os dados são armazenados em variáveis de instância.
- `printData()`: Imprime os dados lidos no console para verificação.
- `writeDataToFile(String outputPath, DualMatrixTransportation s)`: Escreve os dados do problema e os resultados da solução no arquivo de saída.
- `getBalancedFile()`, `getBalancedFile2()`, `getUnbalancedFile1()`, `getUnbalancedFile2()`: Métodos utilitários que procuram por arquivos específicos no diretório atual, baseando-se em palavras-chave no nome do arquivo.
- `main(String[] args)`: Função principal que:
  1. Gera e/ou lê problemas de transporte balanceados e desbalanceados.
  2. Resolve o problema utilizando a classe `DualMatrixTransportation`.
  3. Escreve os resultados em arquivos de saída.

4 Funcionamento

Inicialmente, na função main(String[] args) da classe `TransportationProblem`, 4 problemas são gerados:

```
TransportationProblem tp = new TransportationProblem();
TransportationProblemGenerator tpGen = new TransportationProblemGenerator();

try {
  tpGen.generateRandomProblem(
    filePath:"1-B_transport_problem.txt",
    m:3,
    n:2,
    supplyMax:50,
    costMax:100,
    isBalanced:true
  );

  tpGen.generateRandomProblem(
    filePath:"2-B_transport_problem.txt",
    m:10,
    n:10,
    supplyMax:50,
    costMax:100,
    isBalanced:true
  );
```

```
tpGen.generateRandomProblem(
  filePath:"1-U_transport_problem.txt",
  m:3,
  n:2,
  supplyMax:50,
  costMax:100,
  isBalanced:false
);

tpGen.generateRandomProblem(
  filePath:"2-U_transport_problem.txt",
  m:10,
  n:10,
  supplyMax:50,
  costMax:100,
  isBalanced:false
);
```

A partir daí, itera-se sobre os arquivos, e então eles são lidos, resolvidos, e seus resultados são salvos em arquivos de saída (output).

```java
for (String filePath : balancedFiles) {
  tp.readFile(filePath);
  DualMatrixTransportation solver = new DualMatrixTransportation(
    tp.m,
    tp.n,
    tp.costMatrix,
    tp.supply,
    tp.demand
  );
  solver.solve();

  String outputFileName = "output_" + new File(filePath).getName();
  tp.writeDataToFile(outputFileName, solver);
  System.out.println("Solved and saved results for: " + filePath);
}
```

```java
for (String filePath : unbalancedFiles) {
  tp.readFile(filePath);
  DualMatrixTransportation solver = new DualMatrixTransportation(
    tp.m,
    tp.n,
    tp.costMatrix,
    tp.supply,
    tp.demand
  );
  solver.solve();

  String outputFileName = "output_" + new File(filePath).getName();
  tp.writeDataToFile(outputFileName, solver);
  System.out.println("Solved and saved results for: " + filePath);
}
```

Após a execução, o programa imprime os dados no console para validação e gera arquivos de saída com os resultados de cada um dos problemas resolvidos, sendo eles o custo total do problema e os valores transportados dos pontos de oferta para seus respectivos pontos de demanda. Aqui está um exemplo:

```
Supply Nodes (m): 3
Demand Nodes (n): 2
Supply Array: [400, 300, 400]
Demand Array: [450, 350]
Cost Matrix:
[3, 6]
[4, 5]
[7, 3]
Total cost of the given problem: 2450.0
Supply point 1.0 and demand point 1.0: 400.0
Supply point 3.0 and demand point 2.0: 350.0
Supply point 1.0 and demand point 1.0: 50.0
Supply point 2.0 and demand point 0.0: 300.0
Supply point 3.0 and demand point 0.0: 50.0
```

Listing 1: Arquivo .txt de saída do problema de transporte apresentado no artigo.

5 Problemas gerados

Aqui estão os arquivos de output gerados com a execução de cada um dos problemas (também contendo seus inputs):

5.1 Problemas balanceados

```
Supply Nodes (m): 3
Demand Nodes (n): 2
Supply Array: [15, 5, 28]
Demand Array: [38, 10]
Cost Matrix:
[92, 61]
[39, 89]
[36, 35]
Total cost of the given problem: 1778.0
Supply point 3.0 and demand point 1.0: 18.0
Supply point 3.0 and demand point 2.0: 10.0
Supply point 1.0 and demand point 1.0: 5.0
Supply point 2.0 and demand point 0.0: 5.0
Supply point 1.0 and demand point 1.0: 15.0
Total execution time: 0 ms
```

Listing 2: Problema balanceado 1

```
Supply Nodes (m): 6
Demand Nodes (n): 5
Supply Array: [77, 41, 63, 99, 98, 41]
Demand Array: [104, 7, 110, 81, 117]
Cost Matrix:
[20, 63, 94, 94, 33]
[152, 96, 7, 61, 75]
[47, 192, 123, 193, 45]
[156, 171, 129, 126, 44]
[163, 158, 2, 24, 149]
[131, 199, 199, 81, 95]
Total cost of the given problem: 10672.0
Supply point 1.0 and demand point 1.0: 140.0
Supply point 1.0 and demand point 2.0: 7.0
Supply point 5.0 and demand point 3.0: 110.0
Supply point 5.0 and demand point 4.0: 81.0
Supply point 1.0 and demand point 5.0: 117.0
Supply point 1.0 and demand point 1.0: 151.0
Supply point 2.0 and demand point 0.0: 41.0
Supply point 3.0 and demand point 0.0: 63.0
Supply point 4.0 and demand point 0.0: 99.0
Supply point 1.0 and demand point 1.0: 93.0
Supply point 6.0 and demand point 0.0: 41.0
Total execution time: 0 ms
```

Listing 3: Problema balanceado 2

```
Supply Nodes (m): 10
Demand Nodes (n): 10
Supply Array: [403, 17, 334, 251, 17, 178, 217, 438, 13, 967]
Demand Array: [112, 13, 449, 229, 229, 311, 376, 398, 489, 229]
Cost Matrix:
[276, 178, 244, 161, 20, 70, 225, 250, 244, 185]
[196, 187, 115, 145, 223, 262, 91, 177, 260, 117]
[34, 57, 221, 9, 94, 102, 143, 164, 168, 48]
[289, 233, 222, 102, 181, 182, 85, 90, 293, 266]
[93, 62, 127, 261, 292, 39, 23, 45, 142, 136]
[235, 227, 28, 15, 250, 75, 269, 40, 294, 292]
[270, 83, 171, 48, 51, 181, 300, 273, 157, 75]
[288, 186, 115, 167, 175, 44, 265, 195, 248, 129]
[199, 132, 187, 275, 284, 52, 268, 277, 178, 48]
[246, 172, 251, 206, 39, 205, 291, 143, 41, 300]
Total cost of the given problem: 101540.0
Supply point 1.0 and demand point 1.0: 3619.0
Supply point 3.0 and demand point 2.0: 13.0
Supply point 6.0 and demand point 3.0: 449.0
Supply point 3.0 and demand point 4.0: 229.0
Supply point 1.0 and demand point 5.0: 229.0
Supply point 5.0 and demand point 6.0: 311.0
Supply point 5.0 and demand point 7.0: 376.0
Supply point 6.0 and demand point 8.0: 398.0
Supply point 10.0 and demand point 9.0: 489.0
Supply point 3.0 and demand point 10.0: 229.0
Supply point 1.0 and demand point 1.0: 2392.0
Supply point 2.0 and demand point 0.0: 17.0
Supply point 3.0 and demand point 0.0: 7101.0
Supply point 4.0 and demand point 0.0: 251.0
Supply point 1.0 and demand point 1.0: 670.0
Supply point 1.0 and demand point 1.0: 669.0
Supply point 7.0 and demand point 0.0: 217.0
Supply point 8.0 and demand point 0.0: 438.0
Supply point 9.0 and demand point 0.0: 13.0
Supply point 10.0 and demand point 0.0: 478.0
Total execution time: 1 ms
```

Listing 4: Problema balanceado 3

5.2 Problemas desbalanceados

```
Supply Nodes (m): 3
Demand Nodes (n): 2
Supply Array: [48, 10, 39]
Demand Array: [13, 30]
Cost Matrix:
[72, 37]
[90, 35]
[46, 9]
Total cost of the given problem: 972.0
Supply point 3.0 and demand point 1.0: 9.0
Supply point 3.0 and demand point 2.0: 30.0
Supply point 1.0 and demand point 0.0: 44.0
Supply point 2.0 and demand point 0.0: 10.0
Supply point 1.0 and demand point 1.0: 4.0
Total execution time: 0 ms
```

Listing 5: Problema desbalanceado 1

```
Supply Nodes (m): 6
Demand Nodes (n): 5
Supply Array: [108, 10, 137, 32, 37, 95]
Demand Array: [45, 47, 55, 113, 118]
Cost Matrix:
[17, 155, 28, 34, 107]
[58, 117, 193, 78, 63]
[110, 169, 33, 20, 147]
[33, 9, 28, 196, 16]
[114, 98, 177, 18, 79]
[74, 164, 164, 103, 129]
Total cost of the given problem: 13053.0
Supply point 1.0 and demand point 1.0: 164.0
Supply point 4.0 and demand point 2.0: 47.0
Supply point 1.0 and demand point 3.0: 55.0
Supply point 5.0 and demand point 4.0: 113.0
Supply point 4.0 and demand point 5.0: 118.0
Supply point 1.0 and demand point 0.0: 8.0
Supply point 2.0 and demand point 0.0: 10.0
Supply point 3.0 and demand point 0.0: 137.0
Supply point 1.0 and demand point 1.0: 133.0
Supply point 1.0 and demand point 1.0: 76.0
Supply point 6.0 and demand point 0.0: 95.0
Total execution time: 0 ms
```

Listing 6: Problema desbalanceado 2

```
Supply Nodes (m): 10
Demand Nodes (n): 10
Supply Array: [260, 435, 391, 70, 387, 70, 497, 472, 125, 9]
Demand Array: [268, 103, 210, 13, 492, 311, 117, 273, 201, 288]
Cost Matrix:
[132, 279, 256, 206, 143, 12, 93, 103, 197, 123]
[253, 84, 11, 174, 153, 193, 31, 202, 81, 129]
[165, 121, 255, 223, 80, 217, 264, 274, 43, 50]
[122, 68, 107, 229, 4, 186, 81, 265, 151, 64]
[158, 196, 181, 281, 176, 112, 192, 136, 5, 43]
[154, 26, 106, 108, 16, 221, 5, 23, 198, 234]
[120, 285, 300, 52, 117, 90, 35, 32, 236, 88]
[24, 36, 32, 49, 230, 196, 2, 44, 132, 13]
[101, 61, 145, 263, 63, 15, 295, 121, 261, 136]
[291, 175, 216, 205, 154, 251, 156, 190, 114, 80]
Total cost of the given problem: 36837.0
Supply point 1.0 and demand point 1.0: 1239.0
Supply point 6.0 and demand point 2.0: 103.0
Supply point 2.0 and demand point 3.0: 210.0
Supply point 8.0 and demand point 4.0: 13.0
Supply point 4.0 and demand point 5.0: 492.0
Supply point 1.0 and demand point 6.0: 311.0
Supply point 8.0 and demand point 7.0: 117.0
Supply point 6.0 and demand point 8.0: 273.0
Supply point 5.0 and demand point 9.0: 201.0
```

```
Supply point 8.0 and demand point 10.0: 288.0
Supply point 1.0 and demand point 1.0: 2018.0
Supply point 2.0 and demand point 0.0: 225.0
Supply point 3.0 and demand point 0.0: 391.0
Supply point 1.0 and demand point 1.0: 422.0
Supply point 5.0 and demand point 0.0: 186.0
Supply point 1.0 and demand point 1.0: 306.0
Supply point 7.0 and demand point 0.0: 497.0
Supply point 8.0 and demand point 0.0: 3771.0
Supply point 9.0 and demand point 0.0: 125.0
Supply point 10.0 and demand point 0.0: 9.0
Total execution time: 1 ms
```

Listing 7: Problema desbalanceado 3

6 Resultados

Os resultados estão apresentados nas Tabelas 1 e 2, que mostram o tempo de execução em milissegundos para cada instância de problemas balanceados e desbalanceados, respectivamente.

Table 1: Tempo de Execução para Problemas Balanceados

Custo Máximo	Oferta máxima	Pontos de Oferta	Pontos de demanda	Tempo de Execução
100	50	3	2	<1ms
200	150	6	5	<1ms
500	300	10	10	1ms

Table 2: Tempo de Execução para Problemas Desbalanceados

Custo Máximo	Oferta máxima	Pontos de Oferta	Pontos de demanda	Tempo de Execução
100	50	3	2	<1ms
200	150	6	5	<1ms
500	300	10	10	1ms

7 Conclusão

Com base nos dados apresentados nas Tabelas 1 e 2, é possível concluir que o tempo de execução do algoritmo para resolver problemas de transporte, sejam eles balanceados ou desbalanceados, é extremamente eficiente para os cenários testados.

Observa-se que, mesmo com o aumento no número de pontos de oferta e demanda, bem como com o crescimento dos valores de custo e oferta máxima, o tempo de execução manteve-se estável, variando entre menos de 1ms e 1ms. Isso indica que o algoritmo implementado apresenta excelente desempenho computacional, sendo capaz de lidar eficientemente com problemas de pequeno a médio porte.

Essa estabilidade no tempo de execução pode ser atribuída à estrutura do algoritmo, que por sua vez foi otimizada para operações matriciais e levou em conta um gerenciamento eficiente dos recursos de processamento.