Relatório Técnico – Distribuição de Rotas entre Veículos

Igor Pinheiro dos Santos

Diogo Martins

1. INTRODUÇÃO

Esse relatório tem como principal objetivo descrever a implementação de alguns algoritmos desenvolvidos para um problema clássico de otimização combinatória conhecido como "Problema de Distribuição de Rotas entre Veículos" (PDRV). A solução do problema prevê a alocação mais eficiente das rotas entre os caminhões, de modo que a diferença entre a somatória das quilometragens percorridas por cada caminhão seja a menor possível.

Para propor possíveis soluções para esse problema, foram elaboradas soluções com base em algumas estratégias conhecidas na literatura, no âmbito de desenvolvimento de *software*, como *backtracking* e algoritmo guloso.

2. BACKTRACKING

O *backtracking* é uma técnica de busca recursiva utilizada para encontrar soluções para problemas computacionais complexos, especialmente aqueles que envolvem a exploração de todas as possíveis combinações de um conjunto de escolhas, como o apresentado nesse relatório. Com isso, no contexto do problema da distribuição de rotas em uma frota de caminhões de uma empresa de logística, o *backtracking* pode ser aplicado para encontrar a alocação mais eficiente das rotas entre os caminhões, assegurando que cada caminhão percorra uma quilometragem semelhante, minimizando a diferença total entre eles.

Durante a execução, a cada passo, o algoritmo avalia diferentes combinações de rotas para cada caminhão, utilizando podas para descartar opções que não levam a soluções ótimas, buscando encontrar uma distribuição que minimize a disparidade entre as quilometragens percorridas pelos caminhões, otimizando assim o desempenho logístico da empresa.

2.1 Gerar rotas

Após obter o conjunto de rotas através do método **‘ geracaoDeRotas(quantRotas, tamConjunto, dispersao)’**, o algoritmo irá encaminhar o conjunto de rotas para o método **distribuirRotas()**, que, por sua vez, irá chamar os métodos responsáveis por distribuir as rotas para cada caminhão **‘distribuir (int[] rotas, int maxTotalRotasPorCaminhao, ArrayList<Integer> caminhao, int indice, ArrayList<Integer> melhorRota)’**, utilizando *backtracking* com uma estratégia de poda, e atualizar o conjunto principal de rotas **‘atualizarRotas(rotas, caminhoes[i])’**, removendo as rotas já atribuídas à um caminhão, para o novo conjunto seja distribuído para o próximo caminhão.

2.2 Execução

Durante a execução do método, a cada chamada recursiva, o algoritmo explora diferentes combinações de rotas para um caminhão específico, considerando limites de capacidade. A função avalia a soma total das rotas no caminhão em questão, atualizando a melhor rota caso seja encontrada uma distribuição com menor diferença em relação à capacidade máxima permitida. O *backtracking* continua explorando todas as combinações possíveis, adicionando e removendo rotas, até que as condições de parada sejam atingidas (quando a soma total das rotas no caminhão ultrapassa a capacidade máxima permitida ou quando todas as rotas foram consideradas para distribuição). O resultado é a melhor combinação que minimiza a diferença total de distância entre os caminhões.

Com isso, a execução fica da seguinte forma:

- **Chamada:**

1. A função principal, 'distribuirRotas', é chamada para iniciar o processo de distribuição.
2. A função utiliza um loop para iterar sobre os caminhões e tentar diferentes combinações de distribuição.

```
// Distribui as rotas entre os caminhões
for (int i = 0; i < caminhoes.length; i++) {
    caminhoes[i] = distribuir(rotas, media, caminhoes[i], indice:0, new ArrayList<>());
    melhorDiferencaAtualCaminhao = Integer.MAX_VALUE;
    rotas = atualizarRotas(rotas, caminhoes[i]);
}
```

- **Função recursiva de distribuição:**

1. A função 'distribuir' é chamada recursivamente para explorar diferentes combinações de rotas para cada caminhão.
2. A poda por diferença atual é aplicada, descartando ramos que não podem levar a uma solução ótima.

```
// Adiciona a rota atual ao caminhão e chama recursivamente para a próxima rota
caminhao.add(rotas[indice]);
distribuir(rotas, maxTotalRotasPorCaminhao, caminhao, indice + 1, melhorRota);

// Remove a última rota adicionada e chama recursivamente para a próxima rota
caminhao.remove(caminhao.size() - 1);
distribuir(rotas, maxTotalRotasPorCaminhao, caminhao, indice + 1, melhorRota);
```

- **Atualização da Rota Atual:**

1. A cada passo, a melhor rota é atualizada se uma distribuição mais otimizada é encontrada.

```
// Atualiza a melhor rota caso a diferença seja menor
if (diferenca > 0 && diferenca < melhorDiferencaAtualCaminhao) {
    melhorDiferencaAtualCaminhao = diferenca;
    melhorRota.clear();
    melhorRota.addAll(caminhao);
}
```

- **Condição de Parada/Poda:**

1. O processo de *backtracking* continua até que todas as rotas sejam distribuídas ou até que as condições de parada sejam alcançadas.

```
// Verifica se a soma total ultrapassa o limite ou se todas as rotas foram distribuídas
if (somaTotalCaminhao > maxTotalRotasPorCaminhao || indice == rotas.length) {
    return null;
}
```

Caso a condição do *if* seja verdadeira, o *backtracking* retorna removendo a última rota inserida, dado que nesse momento, a somatória das rotas inseridas no subconjunto, até o momento, ultrapassa a condição.

O algoritmo continua iterando sobre os caminhões, explorando diferentes combinações até que todas as rotas sejam distribuídas ou até que a condição de parada seja alcançada. O resultado é a distribuição que minimiza a diferença total de distância entre os caminhões.

- **Melhor subconjunto de rotas para um caminhão:**

Dado conjunto de rotas definido por ‘S = [18 16 14 14 15 15 19]’;

Média = 37;

O subconjunto S1, teria uma distribuição “melhor” que S2.

S1: rotas 16, 19 - total 35km

S2: rotas 16, 14 - total 33km

Com isso, o caminhão seria atribuído com o subconjunto de rotas S1.

2.3 Resultados obtidos:

Após executar o algoritmo de *backtracking* implementado, os seguintes resultados foram observados em relação ao tempo:

```
Tamanho 6 foi resolvido em média em 2 ms
Tamanho 7 foi resolvido em média em 0 ms
Tamanho 8 foi resolvido em média em 0 ms
Tamanho 9 foi resolvido em média em 0 ms
Tamanho 10 foi resolvido em média em 0 ms
Tamanho 11 foi resolvido em média em 1 ms
Tamanho 12 foi resolvido em média em 1 ms
Tamanho 13 foi resolvido em média em 3 ms
Tamanho 14 foi resolvido em média em 1 ms
Tamanho 15 foi resolvido em média em 3 ms
Tamanho 16 foi resolvido em média em 6 ms
Tamanho 17 foi resolvido em média em 7 ms
Tamanho 18 foi resolvido em média em 13 ms
Tamanho 19 foi resolvido em média em 18 ms
Tamanho 20 foi resolvido em média em 22 ms
Tamanho 21 foi resolvido em média em 46 ms
Tamanho 22 foi resolvido em média em 94 ms
Tamanho 23 foi resolvido em média em 159 ms
Tamanho 24 foi resolvido em média em 244 ms
Tamanho 25 foi resolvido em média em 495 ms
Tamanho 26 foi resolvido em média em 881 ms
Tamanho 27 foi resolvido em média em 1715 ms
Tamanho 28 foi resolvido em média em 3310 ms
Tamanho 28 não pôde ser resolvido em até 30 segundos.
```

Com base nos resultados obtidos, foi observado que, para vetores menores, o algoritmo demonstra eficiência para vetores pequenos, resolvendo rapidamente casos com até 10 rotas em média em menos de 2 ms. No entanto, a eficiência começa a diminuir à medida que o tamanho do vetor aumenta, deteriora-se significativamente à medida que o tamanho do vetor ultrapassa 20 rotas. O tempo de execução aumenta exponencialmente, tornando-se impraticável para tamanhos maiores, e não conseguindo resolver casos de 28 rotas dentro do limite de 30 segundos.

2.4 Solução ótima:

O algoritmo implementa uma abordagem heurística para encontrar uma solução que minimize a diferença total entre as quilometragens dos caminhões. No entanto, não há garantia de que a solução encontrada seja ótima. O algoritmo pode encontrar uma solução próxima à ótima, mas não necessariamente a melhor solução possível.

2.5 Considerações finais:

Em resumo, enquanto o algoritmo oferece soluções aceitáveis para casos menores, sua eficiência diminui rapidamente à medida que o tamanho do problema aumenta, e não garante soluções ótimas devido à sua abordagem heurística exponencial. Considerar abordagens alternativas pode ser necessário para lidar eficientemente com instâncias maiores do problema.

3. ALGORITMO GULOSO

O algoritmo guloso implementado para a distribuição de rotas em uma frota de caminhões busca otimizar a quilometragem percorrida por cada veículo sem recorrer a uma solução exaustiva. Duas estratégias são oferecidas: a primeira ordena as rotas em ordem crescente e as distribui sequencialmente aos caminhões, enquanto a segunda atribui cada rota ao caminhão com a menor quilometragem acumulada até o momento, utilizando um comparador para tomar essa decisão. Este algoritmo é uma abordagem eficiente para o problema de distribuição logística, pois procura minimizar a diferença

de quilometragem entre os caminhões, evitando ociosidade e garantindo uma distribuição equitativa das rotas.

3.1 Primeira estratégia: ordenar um conjunto de rotas

Ao ordenar um conjunto de rotas do menor para o maior e inseri-lo sucessivamente, caminhão por caminhão, é possível garantir uma diferença mínima entre o total de quilometragem percorrida por cada caminhão. Com isso, ao executar o algoritmo guloso, seguindo essa estratégia, para um conjunto de rotas [12, 16, 10, 14, 17, 11] e 3 caminhões, o primeiro passo será ordenar o vetor de forma decrescente:

Vetor Ordenado: [10, 11, 12, 14, 16, 17]

Após isso, inserir um para cada caminhão, ficando:

Caminhão 1: Rota 10

Caminhão 2: Rota 11

Caminhão 3: Rota 12

Caminhão 1: Rota 10, 14

Caminhão 2: Rota 11, 16

Caminhão 3: Rota 12, 17

Com isso, garantiremos uma diferença mínima para cada caminhão

Resultados:

Após executar o algoritmo, os resultados foram:

DADOS OBTIDOS AO EXECUTAR A PRIMEIRA ESTRATÉGIA (ORDENANDO O ARRAY DE ROTAS).

Quantidade de Rotas: 27, Tempo (em ms) por execução: [3, 1, 0, 1, 1, 0, 0, 1, 0, 1] Média (em ms): 0.8

Quantidade de Rotas: 54, Tempo (em ms) por execução: [1, 0, 1, 0, 0, 0, 1, 0, 1, 0] Média (em ms): 0.4

Quantidade de Rotas: 81, Tempo (em ms) por execução: [1, 1, 1, 0, 1, 0, 1, 0, 1, 0] Média (em ms): 0.6

Quantidade de Rotas: 108, Tempo (em ms) por execução: [1, 1, 0, 1, 2, 1, 1, 1, 1, 1] Média (em ms): 1.0

Quantidade de Rotas: 135, Tempo (em ms) por execução: [1, 1, 1, 2, 1, 1, 3, 2, 2, 1] Média (em ms): 1.5

Quantidade de Rotas: 162, Tempo (em ms) por execução: [1, 1, 1, 1, 0, 1, 1, 1, 0, 1] Média (em ms): 0.8

Quantidade de Rotas: 189, Tempo (em ms) por execução: [3, 2, 2, 1, 1, 2, 1, 1, 1, 1] Média (em ms): 1.5

Quantidade de Rotas: 216, Tempo (em ms) por execução: [1, 3, 1, 1, 1, 1, 1, 1, 1, 1] Média (em ms): 1.2

Quantidade de Rotas: 243, Tempo (em ms) por execução: [1, 0, 2, 1, 1, 1, 1, 2, 1, 1] Média (em ms): 1.1

Quantidade de Rotas: 270, Tempo (em ms) por execução: [2, 1, 1, 1, 1, 1, 1, 1, 1, 1] Média (em ms): 1.1

3.2 Segunda estratégia: atribuir rota ao menor caminhão

A segunda estratégia do algoritmo guloso consiste em distribuir as rotas para os caminhões, iterativamente, atribuindo cada rota ao caminhão com o menor total de quilometragem percorrida até o momento. O processo é repetido até que todas as rotas sejam alocadas. Essa abordagem busca minimizar a discrepância na quilometragem entre os caminhões, priorizando aqueles com menor carga de trabalho acumulada. A estratégia envolve a utilização de um comparador para selecionar o caminhão apropriado em cada iteração, resultando em uma distribuição mais equitativa das rotas entre os veículos.

Com isso, ao executar o algoritmo guloso, seguindo essa estratégia, para um conjunto de rotas [12, 16, 10, 14, 17, 11] e 3 caminhões, a execução ficará

Caminhão 1: Rota 12

Caminhão 2: Rota 16

Caminhão 3: Rota 10

Caminhão 3: Rota 10, 14

Caminhão 1: Rota 12, 17

Caminhão 2: Rota 16, 11

Garantindo, uma diferença mínima para cada caminhão.

Resultados:

Após executar o algoritmo, os resultados foram:

DADOS OBTIDOS AO EXECUTAR A SEGUNDA ESTRATÉGIA (PROCURANDO O CAMINHÃO COM MENOR TOTAL DE ROTAS)...

Quantidade de Rotas: 27, Tempo (em ms) por execução: [3, 1, 1, 2, 1, 1, 0, 1, 1, 1] Média (em ms): 1.2

Quantidade de Rotas: 54, Tempo (em ms) por execução: [1, 1, 1, 1, 2, 0, 1, 2, 1, 1] Média (em ms): 1.1

Quantidade de Rotas: 81, Tempo (em ms) por execução: [1, 2, 1, 1, 1, 1, 2, 1, 1, 1] Média (em ms): 1.2

Quantidade de Rotas: 108, Tempo (em ms) por execução: [1, 1, 1, 1, 1, 1, 1, 1, 0, 1] Média (em ms): 0.9

Quantidade de Rotas: 135, Tempo (em ms) por execução: [2, 2, 1, 1, 1, 3, 0, 1, 1, 0] Média (em ms): 1.2

Quantidade de Rotas: 162, Tempo (em ms) por execução: [1, 1, 0, 1, 1, 0, 1, 1, 0, 1] Média (em ms): 0.7

Quantidade de Rotas: 189, Tempo (em ms) por execução: [1, 1, 0, 1, 1, 1, 0, 1, 1, 1] Média (em ms): 0.8

Quantidade de Rotas: 216, Tempo (em ms) por execução: [1, 1, 1, 0, 1, 1, 1, 1, 1, 1] Média (em ms): 0.9

Quantidade de Rotas: 243, Tempo (em ms) por execução: [1, 1, 1, 2, 1, 1, 2, 1, 2, 2] Média (em ms): 1.4

Quantidade de Rotas: 270, Tempo (em ms) por execução: [3, 2, 2, 1, 2, 1, 2, 1, 2, 1] Média (em ms): 1.7

3.3 Comparação de resultados – primeira e segunda estratégia

Após executarmos o algoritmo guloso seguindo as duas estratégias, temos que:

Tempo Total da primeira estratégia gulosa: 100 ms

Tempo Total da segunda estratégia gulosa: 111 ms

Com isso, podemos concluir que, O algoritmo guloso implementado apresenta duas estratégias para a distribuição de rotas em uma frota de caminhões. A primeira estratégia consiste em ordenar as rotas em ordem crescente e distribuí-las sequencialmente entre os caminhões, enquanto a segunda estratégia seleciona, a cada iteração, o caminhão com o menor total de rotas acumulado até o momento. Os resultados obtidos indicam que a segunda estratégia, apesar de ter um tempo total de execução ligeiramente maior (111 ms contra 100 ms da primeira estratégia), demonstra ser mais estável em termos de tempo por execução, com média de 1.2 ms, enquanto a primeira estratégia tem média de 0.8 ms.

Em relação à complexidade, ambas as estratégias apresentam complexidade linear, sendo O(n), onde n é o número de rotas. A primeira estratégia tem uma abordagem mais simples, ordenando as rotas e distribuindo-as, enquanto a segunda estratégia envolve um processo de busca e seleção do caminhão com menor total de rotas.

Quanto à eficiência do algoritmo, ele se mostra adequado para vetores tanto pequenos quanto grandes, mantendo um desempenho razoável. No entanto, é importante destacar que a solução obtida não garante uma solução ótima, pois os algoritmos gulosos geralmente buscam uma solução localmente ótima em cada etapa, podendo não resultar na solução globalmente ótima. No contexto desse problema de distribuição logística, a garantia de uma solução ótima é desafiadora, dada a complexidade do problema, e a abordagem gulosa busca uma solução satisfatória, minimizando a diferença de quilometragem entre os caminhões.

4. COMPARAÇÃO DE RESULTADOS*: BACKTRACKING* E ALGORITMO GULOSO

Os resultados obtidos com as estratégias gulosam e de *backtracking* revelam diferenças significativas em termos de desempenho e escalabilidade para o problema de distribuição de rotas entre caminhões. No contexto do algoritmo guloso, as duas estratégias apresentaram tempos médios por execução relativamente baixos, com a primeira estratégia (ordenando o array de rotas) mantendo uma média mais consistente de 0.8 ms em comparação com a média de 1.2 ms da segunda estratégia (procurando o caminhão com menor total de rotas). O tempo total de execução para a primeira estratégia foi de 100 ms, enquanto a segunda estratégia levou 111 ms. Ambas as abordagens gulosas demonstraram eficiência em resolver o problema para conjuntos de rotas pequenos e médios.

Por outro lado, a abordagem de *backtracking* mostrou-se menos eficiente em termos de tempo de execução, particularmente quando o tamanho do conjunto de rotas aumenta. Enquanto para conjuntos menores o tempo é praticamente negligenciável, a estratégia de *backtracking* começa a exibir um aumento exponencial no tempo de execução à medida que o número de rotas cresce. Isso é evidenciado pelos tempos médios crescentes, culminando na incapacidade de resolver o conjunto de tamanho 28 em até 30 segundos. Em contraste, as estratégias gulosas mantiveram uma eficiência notável mesmo para conjuntos maiores.

Em relação à garantia de encontrar a melhor solução, é importante destacar que os algoritmos gulosos não garantem que a solução encontrada seja a melhor em todos os casos, enquanto o *backtracking*, em teoria, pode explorar todas as possíveis soluções. No entanto, o *backtracking* enfrenta um problema sério de demorar muito para conjuntos de dados maiores, tornando-se impraticável em termos de tempo. Mesmo que o *backtracking* tenha o potencial de encontrar a solução ideal, sua eficiência na prática é bastante limitada. Por outro lado, os algoritmos gulosos oferecem soluções rápidas e razoáveis, embora não possam garantir que a solução encontrada seja a melhor possível em todos os casos. Assim, a escolha entre essas abordagens vai depender das necessidades específicas do problema, considerando se é mais crucial ter uma solução rápida ou se busca a melhor solução possível.

MÉTODO DE AVALIAÇÃO ESCOLHIDO: AVALIAÇÃO INDIVIDUAL

***Backtracking* – Igor Pinheiro**

Algoritmo Guloso – Diogo Martins