PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

Graduação em Engenharia de Software – Instituto de Ciências Exatas e Informática

Bruno Pontes Duarte, Davi Fernandes Ferreira Silva, Eduardo Augusto Brito,

Samuel Marques Sousa Leal

Fundamentos de Projeto e Análise de Algoritmos

Trabalho Prático

Belo Horizonte

2023

1. Funcionamento dos algoritmos e decisões

1.1. Programação dinâmica

Para resolver o problema utilizando a técnica de Programação Dinâmica, deve-se entender o problema e dividi-lo em subproblemas para decidir quais serão os limites das colunas, linhas e valores da tabela.

Primeiramente, resolve-se os limites da tabela. O limite superior da tabela é calculado pela soma de todas as rotas possíveis dividida pelo número de caminhões. Esse resultado é multiplicado por uma porcentagem de tolerância, neste trabalho a porcentagem foi de 0,05%. Finalmente, ocorre a conversão desse resultado flutuante para inteiro e cria-se as colunas das tabelas a partir desse resultado inteiro que varia do valor 0 até o valor resultante do cálculo do limite.

As linhas são cada rota possível e os valores da tabela são as somas das rotas possíveis para aquele limite superior após a criação das colunas com os limites. A cada linha que adiciona-se uma rota, os sub resultados são aperfeiçoados até a reconstrução da solução final desejada. Com a solução final da tabela, utiliza-se o algoritmo de recuperação das rotas que usa o resultado ótimo e verifica se a linha anterior ao valor é a mesma. Caso não seja, sabe-se que a rota da linha atual faz parte do resultado ótimo, então subtrai-se o valor da rota do valor limite e repete-se a verificação para a linha anterior até que descubra-se todos os valores que compõem o resultado ótimo.

A tabela descrita acima vale para um caminhão. Para resolver o problema para qualquer número “n” de caminhões, o algoritmo usa o método de calcular a tabela passando a quantidade de rotas possíveis para cada caminhão. Sendo assim, cada caminhão tem a sua tabela própria. Após o cálculo de sua tabela, ocorre a remoção das rotas utilizadas no caminho ótimo desse caminhão da lista de rotas possíveis, para que os próximos caminhões não usem essas rotas.

1.2. Algoritmo Guloso

O problema da distribuição de rotas entre caminhões de uma frota pode ser resolvido por meio de algoritmos gulosos, que tomam decisões locais com o objetivo de atingir uma solução globalmente eficiente. Implementa-se dois algoritmos gulosos distintos para abordar esse problema:

- **1.2.1. Iteração:** O primeiro critério de algoritmo guloso implementado consiste na ordenação das rotas do problema a ser resolvido de maneira crescente e posterior designação a um caminhão por vez, retornando ao primeiro quando o último é alcançado. O processo se repete de maneira iterativa até o fim das rotas. Dessa maneira, o algoritmo prioriza efetivamente a escolha da menor rota para cada veículo e busca (ainda que de maneira não muito eficiente quando comparada à algoritmos como o backtracking e programação dinâmica) uma distribuição equitativa das rotas.
- **1.2.2. Média com tolerância:** O segundo algoritmo guloso adota um critério mais complexo, considerando a média de rotas por caminhão associada a uma tolerância. As rotas são ordenadas de maneira decrescente para processar as rotas maiores primeiro para evitar a ocupação desproporcional de um caminhão enquanto outro

permanece vazio. Assim, ocorre uma iteração pelas rotas considerando se a carga atual do caminhão sendo analisado consegue comportar a rota índice sem ultrapassar a média estipulada por meio da divisão entre o valor total das rotas e a quantidade de caminhões, caso consiga, ela é adicionada ao caminhão. Já para o caso em que a rota exceda o limite médio do caminhão, ela é adicionada ao próximo veículo, que será analisado em vez do atual.

Portanto, embora os algoritmos gulosos no geral ofereçam eficiência na resolução de desafios que envolvem otimização e repartição de recursos, é crucial reconhecer que essas soluções não garantem sempre o resultado ótimo. A natureza local das decisões, tomadas a cada iteração, pode levar a soluções subótimas de maneira que é possível (dependendo do conjunto analisado) que um caminhão fique desproporcionalmente mais ocupado do que outros, especialmente em cenários complexos. Contudo, a simplicidade e eficiência computacional desses algoritmos os tornam práticos para problemas de grande escala.

1.3. Divisão e conquista

O método de Divisão e Conquista desenvolvido aborda o desafio de otimizar a distribuição de rotas entre caminhões de uma frota por meio de divisões recursivas das rotas.

Primeiramente, a base da recursão - isto é, a função executada quando chega-se ao fim da recursividade - é feita de maneira que a última rota será atribuída ao caminhão com menor carga da frota, visando equalizar a diferença entre eles.

Em seguida, a parte recursiva do algoritmo é feita de maneira que as rotas são divididas ao meio e apenas metade é processada e dividida entre os caminhões. Esse processamento é feito semelhante ao segundo algoritmo guloso, de forma que as rotas são ordenadas e inseridas aos caminhões até atingir a média de rotas por veículo do problema analisado. Essa iteração se repete até todas as rotas da metade analisada serem processadas. Em seguida, o método chama a si mesmo, porém com o conjunto que ainda não foi processado, visando a repetição recursiva do processo até que no final as rotas sejam todas distribuídas.

A decisão estratégica de ordenar as rotas de maneira decrescente antes da distribuição desempenha um papel fundamental na melhoria do resultado do algoritmo de Divisão e Conquista. Ao priorizar a distribuição das rotas maiores antes das menores, a estratégia busca otimizar a alocação dos recursos da frota de caminhões.

Essa abordagem visa equilibrar a carga dos caminhões desde o início da recursão, permitindo que rotas maiores, que geralmente têm um impacto mais significativo na capacidade do veículo, sejam alocadas de maneira eficiente. Ao distribuir as rotas maiores primeiro, o algoritmo busca minimizar a disparidade entre os caminhões, evitando a superlotação de um veículo ao final da execução do método de Divisão e Conquista.

1.4. Backtracking

O método de Backtracking desenvolvido para abordar o desafio da distribuição de rotas entre caminhões utiliza estratégias de otimização baseadas em técnicas de poda, aprimorando a abordagem de força bruta.

No caso do problema apresentado, a solução implementada roda o algoritmo de Backtracking para cada caminhão, buscando criar a melhor rota para cada um. Vale ressaltar que para o algoritmo funcionar corretamente é necessário remover as rotas já utilizadas por outros caminhões ao chamar o algoritmo de Backtracking novamente.

Destaca-se a importância crucial da técnica de poda no aprimoramento do processo de busca. Essa abordagem efetiva elimina caminhos desnecessários na árvore de decisão, resultando em uma significativa melhoria na eficiência do algoritmo - que é o que o diferencia de algoritmos força-bruta. Neste contexto, o nosso backtracking emprega a técnica de poda de acordo com duas condições:

Valor atual da rota + elemento para se adicionar não ultrapassa a média desejada
OU A distância ultrapassada é menor do que a última melhor distância ultrapassada*.

*Caso ultrapasse o limite, se armazena o quanto o limite foi ultrapassado para, caso não seja possível alcançar a melhor média entre os caminhões, se alcance o melhor dos piores casos.

2. Resultados

2.1. Backtracking

A busca do conjunto que não consegue ser executado em menos de 30 segundos pelo backtracking foi iniciada em rotas de 6 elementos. Essas executaram em uma média de 0.00003 segundo e trouxe o resultado ótimo. Em seguida, foram executadas rotas de 10 elementos, que por sua vez executaram em 0.00006 segundo. Seguindo o experimento, foram as rotas com 15 candidatos. Seu tempo médio de execução foi de 0.001 segundo, e assim como todos seus anteriores, trouxe o resultado ótimo para o conjunto. Em seguida, 20 rotas: 0.03 segundos de execução e resultado também ótimo. Mais uma vez pulando em múltiplos de 5, foi analisado o conjunto de 25 rotas: 0.62 segundos de execução. Como o tempo de execução beira a casa dos segundos, a partir daqui, será seguida uma estratégia aumentando de 2 em 2 o conjunto de rotas, até que os 30 segundos de execução sejam aproximados. 27 elementos trouxeram resultado em 2.8 segundos e 29 elementos em 9.6 segundos, demonstrado um aumento significativo com base no conjunto previamente analisado.

Aumentando agora a quantidade de rotas em 1 por iteração - em vez de 2, 30 elementos trouxeram um tempo de execução de 20 segundos, mais uma vez, com um aumento perceptível de seu anterior. Agora, 31 elementos já demoraram 51.7 segundos.

O conjunto analisado consiste em: 14, 23, 17, 19, 22, 15, 19, 23, 20, 15, 25, 13, 13, 15, 17, 15, 16, 16, 24, 19, 17, 18, 22, 21, 25, 17, 24, 17, 15, 25, 20

Esse mesmo conjunto será utilizado no resto do documento para fins de comparação de tempo de execução e resultado com os outros três tipos de algoritmos implementados.

E sua divisão para os caminhões, como consta na imagem abaixo fica:

- Caminhão 1: 193
- Caminhão 2: 193
- Caminhão 3: 195

É interessante saber que o aumento no tempo de execução observado com conjuntos maiores no algoritmo de backtracking pode ser atribuído à sua natureza exponencial. Nesse contexto, o backtracking, ao explorar todas as possíveis soluções, gera uma árvore de busca exponencial, onde o número de caminhos a serem explorados cresce exponencialmente com o tamanho do conjunto de rotas.

Conforme o conjunto de rotas aumenta, o número de ramificações na árvore de busca também aumenta exponencialmente, resultando em uma explosão combinatorial de cenários a serem explorados. Isso leva a um crescimento significativo no tempo de execução, tornando o algoritmo impraticável para conjuntos de dados maiores.

Essa característica exponencial do backtracking destaca a importância de considerar alternativas mais eficientes (no quesito de tempo de execução), como algoritmos gulosos ou técnicas de programação dinâmica e divisão e conquista, que serão analisados em seguida no relatório.

2.2. Algoritmo Guloso

Para a análise do algoritmo guloso, serão gerados 9 novos conjuntos, cada um para um tamanho diferente, de maneira que cada um sirva para a análise parelha entre as duas implementações diferentes do algoritmo.

Assim, será primeiramente utilizado o mesmo conjunto evidenciado anteriormente, para o caso de 1T (31 rotas). Em seguida, nos subtópicos de 2T(62) até 10T(310) serão utilizados novos conjuntos gerados a partir do gerador de problemas.

2.2.1. Primeiro método

1T: 31

A execução da primeira variação do algoritmo guloso ficou dividida da seguinte forma:

- Caminhão 1: 206
- Caminhão 2: 186

- Caminhão 3: 189

Outro número importante que vale ressaltar aqui é o tempo de execução. O número ficou em 0,009 segundos.

Analisando a implementação do algoritmo guloso em comparação com o backtracking, foi possível perceber grande diferença no tempo de execução, sendo bem mais performático. Entretanto, por se tratar de um critério consideravelmente mais simples para a resolução do problema, seu resultado obviamente não foi tão bom quanto o do backtracking.

Comparando o resultado ainda ao do backtracking, a diferença da quilometragem das rotas aumenta em 18, o que pode ser uma diferença não tolerável dependendo do contexto em que o algoritmo for executado. Por isso, é importante analisar o contexto ao se escolher o algoritmo ideal para a resolução de um problema

Nos casos em que é tolerável o resultado não ser ótimo podemos utilizar métodos como o guloso, mas em casos mais críticos nos quais não podemos ter o luxo de tolerâncias será necessário escolher outro tipo de algoritmo com um resultado mais eficiente. Ademais, também é importante analisar quanto tempo pode-se gastar para chegar ao resultado considerando que algoritmos com resultados mais eficientes podem gastar consideravelmente mais tempo ou mais memória que o guloso.

2T: 62

O tempo de execução foi de 0.0001 segundos com um conjunto de 62 rotas. a divisão ficou da seguinte forma:

- Caminhão 1: 397
- Caminhão 2: 398
- Caminhão 3: 377

3T: 93

O tempo de execução médio continuou em 0.0002 segundos com um conjunto de 93 rotas. A divisão ficou da seguinte forma:

- Caminhão 1: 599
- Caminhão 2: 605
- Caminhão 3: 608

4T: 124

O tempo de execução médio subiu para 0.002 segundos com um conjunto de 124 rotas. A divisão ficou da seguinte forma:

- Caminhão 1: 809
- Caminhão 2: 788

- Caminhão 3: 781

5T: 155

O tempo de execução médio subiu para 0.0006 segundos com um conjunto de **155** rotas. A divisão ficou da seguinte forma:

- Caminhão 1: 1028
- Caminhão 2: 1023
- Caminhão 3: 754

6T: 186

O tempo de execução médio subiu para 0.00006 segundos com um conjunto de **186** rotas. A divisão ficou da seguinte forma:

- Caminhão 1: 1364
- Caminhão 2: 1344
- Caminhão 3: 1347

7T: 217

O tempo de execução médio subiu para 0.001105 segundos com um conjunto de **217** rotas. A divisão ficou da seguinte forma:

- Caminhão 1: 1285
- Caminhão 2: 1268
- Caminhão 3: 953

8T: 248

O tempo de execução médio subiu para 0.0000825 segundos com um conjunto de **248** rotas. A divisão ficou da seguinte forma:

- Caminhão 1: 1561
- Caminhão 2: 1566
- Caminhão 3: 1547

9T: 279

O tempo de execução médio subiu para 0.0000720 segundos com um conjunto de **279** rotas. A divisão ficou da seguinte forma:

- Caminhão 1: 1794
- Caminhão 2: 1797
- Caminhão 3: 1802

10T: 310

O tempo de execução médio subiu para 0.0000865 segundos com um conjunto de **310** rotas. A divisão ficou da seguinte forma:

- Caminhão 1: 1928

- Caminhão 2: 1907
- Caminhão 3: 1911

2.2.2. Segundo método

1T: 31

O tempo de execução da primeira variação do algoritmo guloso ficou dividida da seguinte forma:

- Caminhão 1: 198
- Caminhão 2: 192
- Caminhão 3: 191

Levando em consideração também o tempo de execução do segundo método, temos o número de 0,0001 segundos.

Comparando o resultado da segunda implementação do algoritmo guloso com a primeira, é fato que o resultado se saiu bem melhor, saindo de um desvio padrão de 20 quilômetros, para apenas 7. O resultado é ainda mais impressionante quando é comparado também com o primeiro método, o tempo de execução, que foi consideravelmente menor no caso analisado. Por fim, a execução do algoritmo guloso pode parecer relativamente aleatória, visto que o resultado global depende das escolhas feitas a cada passo e, por isso, não é possível garantir que a vantagem vista neste exemplo não necessariamente será persistida ao analisar novos conjuntos com tamanhos múltiplos de T (31).

2T: 62

O tempo de execução também foi de 0.002 segundos com um conjunto de 62 rotas. Apesar do mesmo tempo de execução do primeiro método, seu resultado foi menos satisfatório: a divisão ficou da seguinte forma:

- Caminhão 1: 413
- Caminhão 2: 422
- Caminhão 3: 337

3T: 93

O tempo de execução continuou em 0.002 segundos com um conjunto de 93 rotas. A divisão ficou da seguinte forma:

- Caminhão 1: 653
- Caminhão 2: 653
- Caminhão 3: 506

4T: 124

O tempo de execução médio ficou em 0.0005 segundos com um conjunto de 124 rotas, que quando comparado com a implementação do primeiro método, é bem mais rápido. A divisão, entretanto, ficou da seguinte forma:

- Caminhão 1: 863
- Caminhão 2: 869
- Caminhão 3: 656

Apesar de mais veloz, seu resultado é pior.

5T: 155

O tempo de execução médio subiu para 0.001105 segundos com um conjunto de **155** rotas. A divisão ficou da seguinte forma:

- Caminhão 1: 1028
- Caminhão 2: 1023
- Caminhão 3: 754

6T: 186

O tempo de execução médio subiu para 0.001105 segundos com um conjunto de **186** rotas. A divisão ficou da seguinte forma:

- Caminhão 1: 1285
- Caminhão 2: 1268
- Caminhão 3: 953

7T: 217

O tempo de execução médio subiu para 00.00008 segundos com um conjunto de **217** rotas. A divisão ficou da seguinte forma:

- Caminhão 1: 1476
- Caminhão 2: 1487
- Caminhão 3: 1092

8T: 248

O tempo de execução médio subiu para 0.0000925 segundos com um conjunto de **248** rotas. A divisão ficou da seguinte forma:

- Caminhão 1: 1697
- Caminhão 2: 1706
- Caminhão 3: 1271

9T: 279

O tempo de execução médio subiu para 0.0001300 segundos com um conjunto de **279** rotas. A divisão ficou da seguinte forma:

- Caminhão 1: 1967

- Caminhão 2: 1959
- Caminhão 3: 1467

10T: 310

O tempo de execução médio subiu para 0.0001145 segundos com um conjunto de **310** rotas. A divisão ficou da seguinte forma:

- Caminhão 1: 2091
- Caminhão 2: 2097
- Caminhão 3: 1558

2.3. Divisão e Conquista

Executando o mesmo conjunto de tamanho T = 31 utilizado nos algoritmos anteriormente analisados, os resultados ficaram da seguinte maneira:

O tempo de execução ficou em 0,005 segundos, cujo tempo foi semelhante à implementação dos algoritmos gulosos

Quanto às divisões, seu resultado ficou:

- Caminhão 1: 202
- Caminhão 2: 190
- Caminhão 3: 189

Seu resultado, apesar de não ótimo, foi satisfatório, considerando seu rápido tempo de execução e relativamente pouca disparidade, quando comparado aos algoritmos gulosos implementados e descritos anteriormente. Comparando com o Backtracking, notamos uma evolução clara quanto a tempo de execução, essa característica pode ser de grande vantagem ao se executar conjuntos com um grande número de rotas, já que o Backtracking não consegue resolver esses problemas devido a sua natureza exponencial.

Sendo assim, concluímos que apesar do resultado não ser ótimo ele pode ser bom o suficiente para os casos nos quais o Backtracking não pode ser utilizado, garantindo uma resolução eficiente com um tempo de execução considerado baixo. Vale ressaltar que nem sempre suas respostas serão boas o suficientes, já que em alguns casos é necessário um resultado mais próximo do ótimo.

2.4. Programação Dinâmica

Por fim, o algoritmo de programação dinâmica executou o problema em T = 100 também, de maneira que o tempo ficou em 0.003 segundos. A divisão das rotas entre a frota ficou da seguinte maneira:

- Caminhão 1: 193

- Caminhão 2: 193
- Caminhão 3: 182

A implementação gulosa apesar de ter um tempo de execução ainda menor do que a programação dinâmica, com apenas 0.0001 segundos, conseguiu atingir uma distribuição mais equitativa das rotas entre os caminhões. A comparação desses resultados sugere que, em determinadas situações, o algoritmo guloso pode oferecer soluções mais balanceadas em termos de carga entre os caminhões, enquanto a programação dinâmica prioriza a eficiência computacional.

Deve-se levar em consideração que a implementação gulosa pode ser mais imprevisível ao garantir a otimização global do resultado, podendo ter seu resultado final subótimo dependendo das escolhas imediatas feitas durante sua execução. Enquanto isso, a programação dinâmica, diferentemente do guloso, leva em consideração o contexto global e pode levar a resultados gerais mais estáveis, dependendo do conjunto utilizado.