Relatório Final de Iniciação Científica
Modalidade PDPD – Edital 02/2018

Algoritmos para Problemas de Roteamento de Veículos

Resumo

Na versão mais simples do problema de Roteamento de Veículos, temos um depósito de onde saem veículos carregados com produtos e que devem passar por algumas cidades. O objetivo do problema é minimizar o custo total (por exemplo, distância percorrida) para entregar todos os produtos a todas as cidades respeitando a capacidade dos veículos. Esse é um problema clássico na área de otimização combinatória, com diversas aplicações práticas.

Este relatório descreve as atividades desenvolvidas pela aluna Letícia Almeida Santos durante sua iniciação científica. Elas envolveram o estudo de algumas variações do problema mencionado, bem como outros problemas relacionados.

Palavras-chave: Algoritmos; Roteamento de Veículos; Otimização Combinatória; Programação Linear; Programação Linear Inteira.

Área do conhecimento: Teoria da Computação.

Autora:
Letícia Almeida Santos (voluntária)

Orientadora:
Profa. Dra. Carla Negri Lintzmayer

12 de dezembro de 2023

Sumário

1 Resumo das atividades desenvolvidas

O principal objetivo desse projeto foi introduzir o aluno à pesquisa na área de otimização combinatória por meio do estudo de problemas de roteamento de veículos e variações do mesmo. Nossa ênfase se deu no estudo de formulações dos problemas e métodos exatos baseados em programação linear inteira.

Ao começo do projeto revisamos as práticas de computação básica como algoritmos recursivos, estruturas, listas, pilhas, filas e árvores.

Passamos então para a análise de algoritmos em sua eficiência e corretude com as notações $O(f(n))$, $\Theta(f(n))$ e $\Omega(f(n))$ com $f(n)$ sendo uma função qualquer no tamanho da entrada. Também revisamos os conceitos de algoritmos gulosos e de programação dinâmica, para problemas em geral, e de algoritmos heurísticos, algoritmos de aproximação e algoritmos exatos, para problemas de otimização combinatória. Para isso vimos os algoritmos de divisão e conquista (*MergeSort*), a mochila fracionária e o problema do escalonamento de tarefas. Por fim vimos ainda alguns conceitos de complexidade computacional, com as classes **P**, **NP** e **NP**-difícil.

Estudamos também alguns conceitos de teoria dos grafos e algoritmos clássicos que trabalham com grafos. Nesse sentido estudamos os conceitos de vértices, arestas, arcos, pesos, grafos completos, grafos direcionados e não direcionados, o algoritmo de Kruskal para árvores geradoras mínimas, Dijkstra para caminhos mínimos em grafos, o problema da cobertura por vértices e o problema do caixeiro viajante.

Por fim revisamos os conceitos e estruturas de Programação Linear *(PL)*, Programação Linear Inteira *(PLI)* e o conceito de *Problemas de Otimização*, tais como os problemas de roteamento cujos resultados se encontram nas seções seguintes.

2 Introdução

Um problema de otimização é um problema onde desejamos maximizar ou minimizar uma função real objetivo f, da forma $f\colon S \to \mathbb{R}$, onde S é um conjunto finito de soluções viáveis [8], isto é, soluções que respeitam as restrições do problema.

Quando o conjunto S vem a partir de um conjunto 2^E (conjunto de todos os subconjuntos de E), para algum conjunto E, temos um problema de *Otimização Combinatória* e nos cabe encontrar $s^* \in S$ que otimize a função objetivo. Um exemplo de problema de otimização combinatória é, dados possíveis investimentos, encontrar quais investimentos gerariam mais lucro em uma determinada janela de tempo.

Para que encontremos s^* podemos enumerar o conjunto S, porém isso não é eficiente quanto maior o tamanho de S. As soluções viáveis costumam poder ser

descritas de uma maneira concisa, então o desafio é encontrar algoritmos que sejam provavelmente ou comprovadamente mais eficientes do que enumerar todas as soluções e as testar.

Para alguns dos problemas de otimização combinatória, os melhores algoritmos têm tempo de execução que crescem de maneira polinomial dado o tamanho da entrada, mas há outros para os quais não há esperança de que existam algoritmos eficientes que resolvam qualquer entrada. Duas das técnicas que existem para tratar problemas desse tipo são programação linear e algoritmos de aproximação, que serão descritos nas Seções 2.1 e 2.2, respectivamente. Serão mostradas aplicações dessas técnicas sobre o clássico problema *vertex cover*, descrito na Seção 2.3. Na Seção 2.4 apresentamos alguns resultados para o também clássico problema do caixeiro viajante, que é bem relacionado com problemas de roteamento. Os problemas principais que foram estudados durante o período da iniciação científica são descritos na Seção 3.

2.1 Programação Linear

Programação linear é uma forma de descrever um problema de otimização combinatória por meio de de equações e/ou inequações lineares [2]. Desta maneira, todo programa linear apresenta uma expressão que queremos otimizar (a função objetivo), isto é, maximizar ou minimizar, e várias equações e inequações que indicam restrições do problema. Todas essas expressões são descritas utilizando variáveis de decisão, que indicam as escolhas feitas para se chegar a uma solução.

Em geral um programa linear com n variáveis e m restrições é descrito da seguinte forma:

$$\begin{aligned} \text{minimizar} \quad & \sum_{j=1}^{n} c_j x_j & \\ \text{sujeito a} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq b_i & \forall i \in \{1, \ldots, m\} \\ & x_j \geq 0 & \forall j \in \{1, \ldots, n\} \end{aligned}$$

Onde:

- a_{ij}, b_i e c_j são constantes
- x_j são as variáveis de decisão
- $\sum_{j=1}^{n} a_{ij} x_j \geq b_i$ são as restrições
- $\sum_{j=1}^{n} c_j x_j$ é a função objetivo

Considere uma empresa de produção de computadores mono-processados que fabrica máquinas cujas diferenças estão na quantidade de placas de memória de

256KB que elas possuem e na quantidade de discos para expansão [2].

As especificações de cada máquina são dadas a seguir:

Máquina	Preço	Disco	Placas
GP1	60	0	4
GP2	40	1	2
GP3	30	0	2
WS1	30	1	2
WS2	15	0	1

Esta empresa quer garantir lucro máximo de venda. Suponha que todo computador produzido seja vendido e sejam a, b, c, d e e a quantidade produzida das máquinas GP1, GP2, GP3, WS1 e WS2, respectivamente. Gostaríamos então de maximizar:

$$\ell = 60a + 40b + 30c + 30d + 15e$$

Precisamos considerar ainda que a empresa tem as seguintes restrições:

1. o suprimento de processadores é limitado a 7000 unidades;
2. o suprimento de discos é incerto, variando de 3000 a 7000 unidades;
3. o suprimento de placas também é incerto e varia de 8000 a 16000 unidades;
4. uma placa de 512KB pode substituir as placas usadas no modelo GP1 e tem-se um suprimento de até 4000 unidades dessas ;
5. a demanda esperada para as máquinas é de 1800 unidades da GP1, 200 da GP2, 3800 para a família GP e 3200 para a família WS;
6. dentro da demanda esperada existem pedidos de 500 unidades do GP2, 500 do WS1 e 400 do WS2.

Formalmente, podemos descrever essas restrições da seguinte maneira:

Restrição 1. O total de processadores utilizados na fabricação tem que ser no máximo o total de processadores em estoque:

$$a + b + c + d + e \leq 7000$$

Restrição 2. O total de discos utilizados deve ser no máximo o valor em estoque. Como o estoque de discos é incerto, podemos ou considerar o menor valor

possível ou podemos criar uma nova variável que represente o valor real em estoque. Neste caso, seja D o número de discos que de fato está em estoque:

$$3000 \leq D \leq 7000$$
$$0a + b + 0c + d + 0e \leq D$$

Restrições 3 e 4. O total de placas utilizadas deve ser no máximo o valor em estoque. Como o estoque de placas de memória também é incerto, usaremos a variável P para identificar o valor. O computador GP1 pode ser produzido com duas placas diferentes, então usaremos as variáveis a_1 para indicar as unidades que foram produzidas com as placas de 256KB e a variável a_2 para as que utilizam a placa de 512KB. Assim:

$$8000 \leq P \leq 16000$$
$$4a_1 + 2b + 2c + 2d + e \leq P$$
$$2a_2 \leq 4000$$
$$a_1 + a_2 = a$$

Restrição 5. Não podemos produzir mais do que a demanda esperada, já que não teremos expectativa de vender unidades excedentes:

$$a \leq 1800$$
$$b \leq 200$$
$$a + b + c \leq 3800$$
$$d + e \leq 3200$$

Restrição 6. Não podemos deixar de atender os pedidos que já foram feitos:

$$b \geq 500$$
$$d \geq 500$$
$$e \geq 400$$

Assim, o programa linear referente ao problema dessa empresa é descrito da

seguinte maneira:

$$
\begin{aligned}
\text{maximizar} \quad & 60a + 40b + 30c + 30d + 15e \\
\text{sujeito a} \quad & a + b + c + d + e \leq 7000 \\
& 3000 \leq D \leq 7000 \\
& b + d \leq D \\
& 8000 \leq P \leq 16000 \\
& 4a_1 + 2b + 2c + 2d + e \leq P \\
& 2a_2 \leq 4000 \\
& a_1 + a_2 = a \\
& a \leq 1800 \\
& b \leq 200 \\
& a + b + c \leq 3800 \\
& d + e \leq 3200 \\
& b \geq 500 \\
& d \geq 500 \\
& e \geq 400 \\
& a, b, c, d, e, a_1, a_2, D, P \geq 0
\end{aligned}
$$

2.1.1 Programação Linear e Programação Inteira

Um *Programa Linear Inteiro* é estruturado da mesma forma que um Programa Linear, porém as variáveis podem apenas assumir valores inteiros. No entanto, algoritmos que resolvam este tipo de problema em tempo polinomial são desconhecidos, ao contrário dos programas lineares [11]. Infelizmente, muitos dos problemas de otimização combinatória interessantes podem apenas ser descritos por meio de programas lineares inteiros.

2.1.2 Programação Linear e Programação Não Linear

Uma problema é considerado um problema de Programação Não Linear (*PNL*) se a função objetivo ou qualquer uma das restrições for uma expressão não linear sobre as variáveis de decisão [4].

Por exemplo,

$$\begin{aligned} \text{Maximizar} \quad & \sum_{i=1}^{n} c_i x_i \\ \text{sujeito a} \quad & \sum_{i=1}^{n} a_i (x_i)^2 = b_1 \\ & \sin(x_i) \geq b_2 \\ & x_i \geq 0 \qquad\qquad\qquad\qquad \forall i \in 1, \ldots, n \end{aligned}$$

é um programa onde temos duas restrições que usam de funções não lineares (as funções quadrática e seno), sendo portanto um programa não linear.

2.2 Algoritmos de Aproximação

Algoritmos de aproximação são algoritmos rápidos, geralmente de tempo polinomial, que devolvem uma solução para um problema de otimização cujo custo está a um fator α de distância do custo da solução ótima, sendo α muito bom quando próximo de 1 [9].

Dada uma instância de um problema, seja ℓ o custo da solução calculada pelo algoritmo e seja ℓ^* o custo de uma solução ótima. Se estivermos falando de um problema de minimização e a expressão

$$\ell \leq \alpha \ell^*$$

vale para qualquer instância do problema, então dizemos que o algoritmo é uma α-aproximação. Se o problema for de maximização e a expressão

$$\ell \geq \alpha \ell^*$$

vale para qualquer instância, então dizemos que o algoritmo é uma α-aproximação.

Por exemplo, suponha que queremos minimizar o tempo que impressoras em uma gráfica ficam ligadas [5], sabendo que:

- Temos 3 impressoras exatamente iguais em funções e tempo para executar tarefas;
- Precisamos copiar 1 apostila, o que leva 1 hora;
- Precisamos imprimir 2 banners, levando 2 horas cada e podendo ser impressos separadamente;
- Precisamos imprimir 1 álbum de fotos, o que leva 5 horas;

- Precisamos fazer a tiragem de 1 lote de anuários, o que leva 6 horas;
- Precisamos fazer a tiragem de 1 revista, o que leva 7 horas.

Sem entrar em detalhes da resolução deste problema, sabemos que o menor tempo que vamos usar para completar todas as tarefas é de 9 horas, isto é, $\ell^* = 9$.

Digamos agora que foi utilizado um algoritmo para organizar as tarefas e que ele indicou que são necessárias 10 horas para executá-las. Nesse caso, $\ell = 10$ e podemos escrever a seguinte informação sobre o algoritmo:

$$\ell = 10 \leq \alpha \times 9 = \alpha\ell^* \qquad \Rightarrow \qquad \alpha \geq \frac{10}{9} \approx 1.1 \ .$$

Isto é, o fator de aproximação dele é pelo menos 1.1. Mas dependendo da lista de tarefas a serem realizadas, o algoritmo poderia entregar uma solução ainda pior, com um tempo sendo o dobro do ótimo, por exemplo. Por isso a definição de fator de aproximação é feita sobre todas as entradas possíveis. Esse fator de aproximação serve, portanto, para termos ideia de quão ruim pode ser a solução feita pelo algoritmo.

Assim, se o pior valor que o algoritmo devolver para qualquer entrada for $\ell = 10$, então temos que

$$\ell = 10 \geq \alpha \times 9 = \alpha\ell^* \qquad \Rightarrow \qquad \alpha \leq \frac{10}{9} \approx 1.1$$

e nesse caso podemos dizer que o algoritmo é uma 1.1-aproximação, pois esse é o maior valor de α.

2.3 Cobertura por vértices

O problema de cobertura por vértices (*Vertex Cover*) consiste em, dado um grafo qualquer, escolher o menor número possível de vértices tal que toda aresta pertencente ao grafo seja incidente a pelo menos um dos vértices escolhidos. Este problema é NP-difícil [9], isto é,um dos problemas para os quais não se conhece nenhum algoritmo eficiente que possa resolvê-lo para qualquer instância.

Por exemplo, imagine que você precisa decidir onde serão instaladas câmeras de segurança em um prédio:

- existe um número n de locais possíveis para instalar as câmeras;
- cada local cobre uma área do prédio;
- não podem haver pontos cegos, isto é, áreas que não tenham uma câmera que as cubra; e

- cada câmera tem um mesmo custo de instalação que você precisa minimizar, isto é, instalar o menor número possível de câmeras.

Podemos modelar esse problema por um grafo da seguinte forma:

- Cada vértice é um possível ponto para instalar uma câmera
- Dois vértices são ligados por uma aresta quando as duas câmeras que eles representam podem cobrir a mesma área

Claramente, uma cobertura por vértices desse grafo resolve o problema da instalação de câmeras.

Seja x_u uma variável que tem valor 1 se o vértice u for escolhido para a cobertura ou o valor 0, caso contrário. O sistema em programação linear para o problema de cobertura por vértices é da seguinte forma:

$$\text{minimizar} \quad \sum_{u \in V} x_u \tag{1}$$

$$\text{sujeito a} \quad x_u + x_v \geq 1 \qquad \forall uv \in E \tag{2}$$

$$x_u \in \{0, 1\} \qquad \forall u \in V \tag{3}$$

Note que ele representa o problema pois (1) indica que queremos minimizar a quantidade de vértices escolhidos e (2) indica que, para cada aresta do grafo, pelo menos um dos seus extremos precisa ser escolhido para a solução.

Resolver o programa linear acima nem sempre é possível em tempo razoável, uma vez que ele é um programa linear inteiro. O Algoritmo 1 apresenta outra forma de resolver esse problema.

Algoritmo 1 Algoritmo para o vertex cover.

```
1: função VertexCover(G = (V, E))
2:     C ← ∅
3:     enquanto E ≠ ∅ faça
4:         Seja (u, v) ∈ E uma aresta qualquer
5:         C ← C ∪ {u, v}
6:         E ← E\{(u, v)}
7:         E ← E\{(u, x) : ∀(u, x) ∈ E}
8:         E ← E\{(v, x) : ∀(v, x) ∈ E}
9:     devolve C
```

Este algoritmo retorna um conjunto de vértices C do grafo G em que os vértices pertencentes a C fazem parte de um emparelhamento maximal[1] M em G, pois quando uma aresta é escolhida, ambos seus extremos são colocados em C e todas

[1] Um emparelhamento é um conjunto de arestas sem vértices em comum. Ele é maximal quando tem o maior número possível de arestas de G.

as arestas incidentes a esses extremos são removidas (linhas 6, 7 e 8). Note ainda que C cobre todas as arestas do grafo, pois todas as arestas incidentes a vértices colocados em C são removidas de E e o algoritmo só para quando E fica vazio. Não é difícil perceber que esse algoritmo executa em tempo polinomial.

O algoritmo é também uma 2-aproximação conforme demonstrado no seguinte teorema:

Teorema 1. *O algoritmo* VertexCover *é uma 2-aproximação para o problema de cobertura por vértices.*

Demonstração. Seja $C = V(M) = \textsc{VertexCover}(G = (V, E))$, onde $V(M)$ é o conjunto de vértices do emparelhamento M e seja $OPT_{VC}(G)$ uma solução ótima do vertex cover para o mesmo grafo G.

Note que $OPT_{VC}(G)$ deve cobrir as arestas de M, isto é, para cada aresta de M, existe pelo menos um vértice na solução ótima que a cobriu. Por isso, $|OPT_{VC}(G)| \geq |M|$.

Como o algoritmo retorna C, cujo tamanho é $2|M|$, temos que o custo do algoritmo é

$$|C| = |V(M)| = 2|M| \leq 2|OPT_{VC}(G)|$$

Assim, esse algoritmo é uma 2-aproximação. □

2.4 Caixeiro viajante

O problema do caixeiro viajante consiste em, dado um grafo com custos nas arestas, partir de um desses vértices, passar por todos os vértices uma única vez e retornar ao vértice inicial fazendo um percurso cuja soma das arestas utilizadas seja mínima. Formalmente, dado um grafo $G = (V, E)$ onde a aresta ij tem custo c_{ij}, queremos encontrar um ciclo hamiltoniano (ciclo que passa por todos os vértices uma única vez) de custo mínimo.

Este problema costuma aparecer em várias situações reais onde deve-se encontrar rotas de entrega e os trechos a serem percorridos têm custos, como pedágio, tempo de duração, distância percorrida, etc. Muitas vezes o grafo de entrada é orientado, para representar, por exemplo, ruas de mão única.

Suponha um caminhão de entregas saindo do centro fornecedor. Ele precisa ir a vários endereços diferentes deixar a sua mercadoria e depois retornar à garagem. Passar mais de uma vez por um endereço não é interessante, sendo inclusive um desperdício de recursos. No caso, a melhor estratégia para o caminhão é passar uma única vez em cada endereço e, só depois de visitar todos, retornar à garagem. Podemos representar os endereços como vértices de um grafo e as ruas que levam de

um local para o outro como suas arestas. Nesse caso temos o problema do caixeiro viajante.

Seja x_{ij} uma variável de decisão que tem valor 1 caso a aresta ij seja escolhida para o percurso e tem valor 0 caso contrário. A representação em programação linear deste problema é feita da seguinte maneira:

$$\text{minimizar} \quad \sum_{j=1}^{n}\sum_{i=1}^{n} c_{ij}x_{ij} \tag{4}$$

$$\text{sujeito a} \quad \sum_{i=1}^{n} x_{ij} = 2 \qquad \forall j \in V \tag{5}$$

$$\sum_{i,j\in S} x_{ij} \leq |S| - 1 \qquad \forall S \subset V \tag{6}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j \in V, i < j \tag{7}$$

Note que ele formaliza o problema do caixeiro viajante:

- (4), a função objetivo, indica que queremos minimizar o custo das arestas escolhidas,
- (5) garante que cada vértice será visitado uma vez pelo ciclo (pois duas arestas devem sair desse vértice),
- (6) garante que não serão formados ciclos que não sejam hamiltonianos (todo subconjunto de vértices de G que não seja o próprio V deve ter no máximo $|S| - 1$ arestas ligando seus vértices (dessa forma, não é possível S conter um ciclo, já que um ciclo com t vértices possui t arestas).

Este também é um problema da classe *NP-difícil* na computação [9]. Infelizmente, o problema geral do caixeiro viajante não admite um algoritmo que aproxime a solução com taxa de aproximação constante [10]. No entanto, uma versão particular do problema que tem muita relevância prática e contorna essa dificuldade é o **caixeiro viajante métrico**. Para esta versão temos de entrada um grafo G com custos c nas arestas é dito **métrico** se, para todo trio de vértices u, v, w temos que $c_{uw} \leq c_{uv} + c_{vw}$. Um dos algoritmos de aproximação que trata deste problema é dado no Algoritmo 2.

Algoritmo 2 Algoritmo para o caixeiro viajante.

```
1: função DoubleTree(G = (V, E), c)
2:     T ← ArvoreGeradoraMinima(G, c)
3:     T² ← T com as arestas duplicadas
4:     F ← TrilhaEuleriana(T²)
5:     C ← CicloHamiltoniano(G, F)
6:     devolve C
```

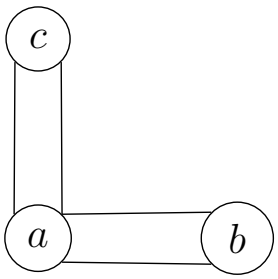

Figura 1: Representação de passeio (b, a, c, a, b) em um grafo completo.

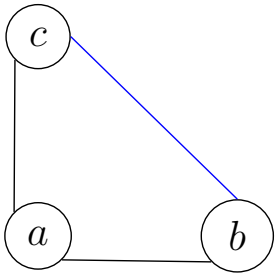

Figura 2: Atalho feito no passeio da Figura 1.

Note que o algoritmo 2 de fato retorna um ciclo hamiltoniano para o grafo G. O funcionamento do algoritmo se dá da seguinte maneira: Primeiro, ele encontra uma árvore geradora T de G (ela passa, portanto, por todos os vértices do grafo). Em seguida, ele duplica as arestas de T, formando T^2. Claramente, todos os vértices de T^2 possuem grau par, de forma que é possível encontrar uma trilha euleriana F sobre T^2, que é uma trilha que passar por todas as arestas de T^2 uma única vez. Essa trilha F também passa por todos os vértices do grafo G, mas ela certamente irá repetir vértices. Como o grafo é completo, podemos fazer atalhos em F para criar um ciclo hamiltoniano C.

Um **atalho** é dado, por exemplo da seguinte maneira. Seja L um grafo completo. Um passeio $P = (b, a, c, a, b)$ por L está representado na Figura 1 para um exemplo. Esse passeio leva b para c passando por a: $b \to a \to c$. Um atalho seria pegarmos uma nova aresta que faça o caminho diretamente entre b e c. Como F é um grafo completo, essa aresta existe e o atalho é viável, de forma que basta substituirmos o trecho de P pela nova aresta, como mostrado na Figura 2.

O algoritmo 2 é uma 2-aproximação conforme demonstrado no teorema 2:

Teorema 2. *O algoritmo* DoubleTree *é uma 2-aproximação para o problema do caixeiro viajante métrico.*

Demonstração. Seja $OPT_{TSP}(G)$ uma solução ótima para o caixeiro viajante no grafo G. Note que se removermos uma aresta de $OPT_{TSP}(G)$, teremos uma árvore geradora T' para o mesmo grafo. Claramente, a árvore geradora mínima T encontrada pelo algoritmo tem custo no máximo o custo de T', então $c(OPT_{TSP}(G)) \geq c(T') \geq c(T)$.

Temos também que $c(F) = c(T^2) = 2c(T)$ e que $c(C) \leq c(F)$, pois C foi obtido de F por meio de atalhos e o grafo é métrico.

Portanto, o custo da solução dada pelo algoritmo é

$$c(C) \leq 2c(T) \leq 2c(OPT_{TSP}(G)) \ ,$$

e assim vemos que o algoritmo é uma 2-aproximação. □

Note que o DoubleTree usa outros algoritmos que têm tempo de execução polinomial para calcular sua resposta:

- ArvoreGeradoraMinima(G, c): dado um grafo G, busca uma árvore geradora mínima em G, isto é, um grafo conexo que contém todos os vértices de G e não possui ciclos, logo, não necessariamente possui todas as arestas de G. O algoritmo de Prim ou Kruskal pode ser utilizado aqui.
- TrilhaEuleriana(G): dado um grafo G, retorna um ciclo que passa por todas as arestas de G exatamente uma vez, podendo repetir vértices. O algoritmo de Fleury pode ser utilizado aqui.
- CicloHamiltoniano(G, F): dados um grafo completo G e um ciclo euleriano F sobre G, percorre o ciclo F sem repetir vértices para poder devolver um ciclo hamiltoniano – no momento que um vértice de F se repetir, faz substituições por arestas diretas já que o grafo é completo.

Sendo assim, o algoritmo também executa em tempo polinomial.

Há outro algoritmo que aproxima a solução para o problema do caixeiro viajante métrico em tempo polinomial, que é uma alteração do algoritmo anterior [3]. Ele é dado no Algoritmo 3.

Algoritmo 3 Algoritmo para o caixeiro viajante alterado.

```
1: função Christofides(G = (V, E), c)
2:     T ← ArvoreGeradoraMinima(G, c)
3:     M ← EmparelhamentoPerfeitoMínimoNosVerticesImpares(T)
4:     F ← TrilhaEuleriana(T + M)
5:     C ← CicloHamiltoniano(G, F)
6:     devolve C
```

Note que o algoritmo acima também retorna um ciclo hamiltoniano para o grafo G. Primeiro, ele encontra uma árvore geradora T de G, e então ele encontra um emparelhamento na árvore geradora T que cobre todos os vértices de grau ímpar de T. Assim, $T + M$ terá todos os vértices com grau par. Podemos então encontrar uma trilha euleriana F sobre $T + M$. O ciclo hamiltoniano é gerado percorrendo a trilha euleriana e fazendo atalhos da mesma forma que no algoritmo anterior.

Encontrar um emparelhamento perfeito de menor custo nos vértices de grau ímpar de G, que é um grafo completo, é um passo que pode ser feito em tempo polinomial [3]. Os outros passos, como mencionado anteriormente, também podem ser feitos em tempo polinomial. Logo, este algoritmo também funciona em tempo polinomial.

Teorema 3. *O algoritmo* CHRISTOFIDES *é uma* $(3/2)$*-aproximação para o problema do caixeiro viajante métrico.*

Demonstração. Pela construção da solução C temos que

$$c(C) \leq c(F) = c(M + T) = c(M) + c(T) \ . \tag{8}$$

O custo do ciclo hamiltoniano é menor que o da trilha euleriana pois o grafo é métrico e fizemos atalhos em F para criar C. O custo de F por sua vez é igual ao custo do grafo $M + T$ que é o mesmo custo que dos grafos T e M.

Sabemos, pela prova do Teorema 2, que:

$$c(T) \leq c(OPT_{TSP}(G)) \ . \tag{9}$$

Seja H um ciclo sobre os vértices ímpares da árvore geradora mínima T encontrada no algoritmo, construído a partir de atalhos feitos no ciclo hamiltoniano da solução ótima $OPT_{TSP}(G)$. Para este ciclo H, podemos tomar dois emparelhamentos M_1 e M_2. Temos então que

$$c(M_1) + c(M_2) = c(H) \leq c(OPT_{TSP}(G)) \ . \tag{10}$$

Considerando o emparelhamento M gerado no algoritmo, que é um emparelhamento mínimo sobre os vértices de grau ímpar de T, então vale que

$$c(M) \leq c(M_1) \tag{11}$$
$$c(M) \leq c(M_2) \tag{12}$$

Somando as três equações acima temos que:

$$2c(M) \leq c(M_1) + c(M_2) \leq c(OPT_{TSP}(G)) \Rightarrow c(M) \leq \frac{c(OPT_{TSP}(G))}{2} \ . \tag{13}$$

Então, juntando (8) com (9) e (13), temos que

$$c(C) \leq c(M) + c(T) \leq \frac{c(OPT_{TSP}(G))}{2} + c(OPT_{TSP}(G)) = \frac{3}{2} c(OPT_{TSP}(G)) \ , \tag{14}$$

de onde vemos que o algoritmo é uma $\frac{3}{2}$-aproximação. □

3 Problemas de Roteamento

O problema do roteamento de veículos é um problema que resolve a seguinte tarefa:

> Determine um conjunto de *rotas* para fazer todos (ou alguns) *transportes requisitados* com a *frota de veículos* dada, com o *menor custo*; em particular, decida qual veículo lida com qual requisição e em qual sequência, de maneira que todas as rotas sejam *viáveis* [12].

Os problemas do tipo *VRP* (Problema do Roteamento de Veículos) aparecem em grande variedade, por se tratarem de problemas de aplicação muito específica que exigem, diversas vezes, uma modelagem especial para otimizar diferentes relações. Os problemas podem ser, assim, classificados de acordo com algumas características:

1. **Capacidade do Veículo.** Problemas que consideram a demanda dos clientes e a capacidade de as atender de acordo com a capacidade de carga transportada por um veículo (a entrega de jornais em uma rota, por exemplo).
2. **Estrutura da Rede.** Consideram as características da rede de movimentação e suas restrições (se a rede representa grandes ou pequenas distâncias, por exemplo).
3. **Tipo de Requisição de Transporte.** Consideram o tipo de serviço (coleta, entrega, coleta e entrega, etc.) e suas restrições.
4. **Restrições Intra-Rotas.** Problemas que consideram se um solução é viável a partir de outras restrições internas de uma rota (agendamento de serviços, tipo de veículo necessário, tamanho da rota, capacidade de carga, etc.).
5. **Características da Frota.** Considera o problema para frotas não homogêneas podendo variar em capacidade de carga (ex.: motos, carros e fiorinos), o tipo de carga que pode ser transportada (ex.: caminhão de carga seca, caminhões pipa, veículos refrigerados), velocidade e habilidade de acessar certas áreas.
6. **Restrições Inter-Rotas.** Considera se uma solução é viável com base nas relações das rotas e no agendamento do serviço (evitar que a rota de um veículo seja muito mais longa que a de outro ou movimentos que precisam ser sincronizados ou executados em uma determinada ordem).
7. **Objetivos Alternativos.** Problemas interessados não em apenas modelar o menor comprimento das rotas, mas em atender outros objetivos, como tempo de duração ou satisfação dos clientes.

Nas seções a seguir descreveremos melhor algumas variações do VRP.

3.1 Problema de Roteamento de Veículos Capacitados

A variação mais comum dos problemas de roteamento é o *problema de roteamento de veículos capacitados* (CVRP), onde buscamos encontrar as rotas de menor custo para uma frota de veículos com uma mesma capacidade de carga para atender uma demanda de clientes. Definimos o *CVRP* formalmente a seguir.

Sejam

- $N = \{1, \ldots, n\}$ um conjunto de clientes,
- q_i a solicitação de transporte de $i \in N$, isto é, o peso de entrega relacionado com o cliente i (a quantidade de materiais a ser carregada, por exemplo),
- K um conjunto de veículos ou frota de veículos,
- Q a capacidade de carga de cada veículo,
- $S \subseteq N$ o conjunto de clientes cujas solicitações de transporte serão atendidas,
- $i_0 = 0$ a garagem da frota,
- $c_{i,j}$ o custo de ir de um ponto i a um ponto j (distância, pedágio existente, etc.),
- $r = (i_0, i_1, \ldots, i_m, i_0)$, uma rota, que é uma sequência tal que $i_1, \ldots, i_m \in S$ e ela será executada por um único veículo.
  Uma rota r é **viável** se $\sum_{1 \leq \ell \leq m} q_{i_\ell} \leq Q$, isto é, se a demanda atendida pela rota for menor ou igual a capacidade de carga do veículo,
- R o conjunto de todas as rotas viáveis.

O objetivo do *CVRP* é determinar os elementos $r \in R$ de menor custo que atendam a maior quantidade de requisições de transportes.

O problema pode ser representado por um grafo $G = (V, E)$ (não orientado) ou um digrafo $G = (V, A)$ (orientado), ambos grafos completos[2], com:

- $V(G) = \{i_0\} \cup N$,
- pesos $c_{i,j}$ nas arestas,
- pesos q_i nos vértices.

[2]Um grafo onde cada vértice é ligado a todos os demais.

Ele é comumente descrito em um problema de programação linear da seguinte maneira. Considerando sua representação em um digrafo e sendo $x_{i,j}$ uma variável que indica se um veículo passou pela aresta $(i, j) \in A$:

$$\text{minimizar} \quad \sum_{(ij)\in A} c_{ij}x_{ij} \tag{15}$$

$$\text{sujeito a} \quad \sum_{(nj)\in A} x_{nj} = 1 \qquad \forall n \in N \tag{16}$$

$$\sum_{(in)\in A} x_{in} = 1 \qquad \forall n \in N \tag{17}$$

$$\sum_{(0j)\in A} x_{0j} = |K| \tag{18}$$

$$\sum_{s\in S,(sj)\in A} x_{sj} \geq \left\lfloor \frac{\sum_{s\in S} q_s}{Q} \right\rfloor \qquad \forall S \subseteq N, S \neq \emptyset \tag{19}$$

$$x_{ij} \in \{01\} \qquad \forall (i\ j) \in A \tag{20}$$

Neste formato do problema, minimizamos o custo das arestas escolhidas (15), onde se garante que:

- (16) cada vértice $n \in N$ tem uma única aresta saindo dele (um veículo sai de um cliente);
- (17) cada vértice $n \in N$ tem uma única aresta entrando nele (um veículo chega a um cliente);
- (18) do vértice inicial i_0, isto é, da garagem, temos $|K|$ arestas saindo, uma para cada veículo da frota;
- (19) a quantidade de veículos que atendem cada subconjunto S de clientes deve ser maior ou igual à quantidade mínima de veículos necessários para atender a demanda q_i destes clientes.

Esta formulação ainda não é a ideal pois tem uma quantidade exponencial de restrições do tipo (19), onde buscamos $\forall S \subseteq N$ e verificamos se a demanda atendida respeita a capacidade de carga. Isto porque para $|N|$ elementos temos $2^{|N|}$ possíveis conjuntos $S \subseteq N$. A formulação *CVRP-MTZ* [12] mostrada a seguir substitui esta restrição para contornar este problema. Introduzimos as variáveis u_i, que indicam a demanda acumulada do veículo até encontrar o cliente i, isto é, a capacidade de

carga utilizada pelo veículo até atender i, o incluindo. Então, temos:

$$\text{minimizar} \quad \sum_{(ij)\in A} c_{ij}x_{ij} \tag{21}$$

$$\text{sujeito a} \quad \sum_{(nj)\in A} x_{nj} = 1 \qquad \forall n \in N \tag{22}$$

$$\sum_{(in)\in A} x_{in} = 1 \qquad \forall n \in N \tag{23}$$

$$\sum_{(0j)\in A} x_{0j} = |K| \tag{24}$$

$$u_i - u_j + Qx_{ij} \leq Q - q_j \qquad \forall (ij) \in A \tag{25}$$

$$q_i \leq u_i \leq Q \qquad \forall i \in N \tag{26}$$

$$x_{ij} \in \{0, 1\} \qquad \forall (ij) \in A \tag{27}$$

- (26) garante que u_i é de fato a capacidade de acumulada até o cliente i ($q_i \leq u_i$) e garante que a carga do veículo nunca seja excedida ($u_i \leq Q$);

- (25) garante que a variável u_i será preenchida com o valor correto, isto é, que se um veículo atende os clientes i e j um em sequência do outro ($x_{ij} = 1$), então a diferença entre u_i e u_j é justamente q_j:

  se $x_{ij} = 1$**:** neste caso a restrição equivale a $u_i - u_j \leq -q_j$
  Sabemos que se j foi atendido em sequência de i, então $u_j = u_i + q_j$, logo $u_i - u_j \leq -q_j \Leftrightarrow Q + u_i - u_j \leq Q - q_j$ e caso o contrário a restrição não será válida.

  se $x_{ij} = 0$**:** neste caso a restrição é equivalente a $u_i - u_j \leq Q - q_j$
  Sabemos que $u_i \leq Q$ e que $u_j \geq q_j \Leftrightarrow -u_j \leq -q_j$, ambas por (26).

  Somando ambas as equações acima, chegamos na restrição (25).

Quando garantimos que a capacidade de carga do veículo seja respeitada, garantimos que há um número de veículos maior ou igual o necessário para atender as demandas dos clientes, o que torna as restrições (25) e (19) equivalentes.

A formulação *CVRP-MTZ* produz cerca de $|N|^2$ restrições (uma para cada aresta) e $|N|$ variáveis (uma u_i para cada $i \in N$), porém o relaxamento linear[3] feito ao inserir u produz um limite inferior mais fraco que a *CVRP-1*.

O *CVRP* é uma generalização do *TSP* (Problema do Caixeiro Viajante,ver Seção 2.4), pois o *TSP* é uma versão do problema com um único veículo (o caixeiro) que tem capacidade de carga infinita e todos os clientes têm demandas iguais a 1.

[3]Relaxamento Linear é a aproximação usada ao resolvermos um PLI como se fosse um PL.

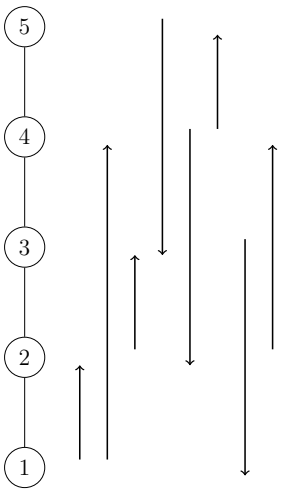

Figura 3: Exemplo de entrada para o CDARP onde temos 5 clientes e a conexão entre eles é um caminho simples. Os arcos orientados de R estão representados de forma vertical à direita, para melhor visualização.

Por se tratar de uma classe muito extensa, o *CVRP* também tem subdivisão de classes para as quais se buscam algoritmos eficientes para os resolver.

3.2 Problema Capacitado Dial-a-Ride - CDARP

O Problema *Capacitado Dial-a-Ride* (CDARP) é um problema da família *CVRP* onde cada demanda é um item de carga que deve ser retirado de um ponto e entregue em outro ponto da rede, isto é, retirado em um vértice e entregue em outro vértice do grafo [7].

Formalmente, para o problema *Capacitado Dial-a-Ride* temos como entrada um grafo misto $G = (V, E, R)$ (veja a Figura 3 para um exemplo), uma função w e um vértice i_0, onde

- V é um conjunto de vértices onde $i_0 \in V$ é a garagem da frota e $i_1, \ldots, i_n \in V$ são os clientes,
- E é um conjunto de arestas (conexões que os veículos podem seguir),
- R é um conjunto de arcos orientados (representam as requisições de transporte entre vértices),
- $w\colon E \to \mathbb{R}$ é uma função de peso nas arestas tal que $w(ij)$ é a distância entre os vértices i e j.

Na versão do *CDARP* que tratamos, temos um único veículo com capacidade Q de carga, que deve servir as requisições de transporte (arcos de R) da sua origem ao seu destino, sem nunca ultrapassar sua capacidade de carga.

Definimos um *movimento* do veículo como sendo sua ida de um vértice i para outro j carregando um subconjunto $R' \subseteq R$ de requisições. Ele é válido se $|R'| \leq Q$ e $ij \in E$, isto é, se as requisições sendo atendidas por ele não ultrapassarem a capacidade de carga do veículo e houver caminho entre os pontos.

O objetivo nesse problema é encontrar uma sequência T de movimentos que começa e termina em i_0 tal que se uma requisição $r \in R$ é atendida, com $r = ij$, então uma existe uma subsequência contínua de movimentos em T que começa em i e termina em j carregando r, no máximo Q requisições estão em cada movimento e o custo das arestas de seguidas pelo veículo (dado por w) é mínimo [6].

O problema CDARP é NP-difícil [6]. Krumke, Rambay e Weider [7] desenvolveram um algoritmo de 3-aproximação para quando o grafo formado por V e E é um caminho (como no exemplo da Figura 3), a capacidade de carga é Q e o vértice inicial desse caminho é a garagem i_0. Esse algoritmo funciona da seguinte maneira:

1. Encontra o conjunto $R^{\uparrow}$ com todas as requisições de R que vão de um vértice i para um vértice j com $j > i$ (arcos que sobem).

2. Cria subconjuntos M de $R^{\uparrow}$ onde as requisições de M nunca estão simultaneamente no veículo, isto é, se $ij \in M$ e $k\ell \in M$, então $j < k$.

3. Encontra o conjunto $R^{\downarrow}$ com todas as requisições que vão de um vértice j para um vértice i com $j > i$ (arcos que descem).

4. Cria subconjuntos N de $R^{\downarrow}$ onde as requisições nunca estão simultaneamente no veículo.

5. Cria um novo conjunto de arestas direcionadas R' tal que $r' \in R'$ corresponde a no máximo Q conjuntos de $M \veebar N$, isto é, r' é um único arco direcionado que começa e termina nos mesmos vértices de seu conjunto correspondente. Dessa forma, cada $r' \in R'$ corresponde a uma quantidade de arcos de R que pode ser carregados ultrapassar a capacidade de carga Q do veículo.

   Por exemplo, para o grafo na Figura 3, se a capacidade é $Q = 2$, temos um $r'_1 = \{1, 4\}$ que corresponde ao primeiro e último arcos e um $r'_2 = \{1, 5\}$ correspondente ao segundo e antepenúltimo arcos.

6. Busca o menor percurso s' que atenda todos os $r' \in R'$, isto é, encontra o menor passeio que passe pelos arcos r' que comece e termine na garagem.
   Para isso, usamos o algoritmo proposto por Atallah e Kosaraju [1], que encontra este passeio para grafos caminho com frotas de apenas um veículo de capacidade unitária.

7. Reescrevemos o s' em para gerar um s que faça sentido no conjunto R, isto é, para cada r' de s', transformamos de volta em algum elemento de $M \cup N$.

Como a solução s' é valida para um veículo com capacidade de carga 1, então podemos usar um veículo de capacidade Q para atender as requisições originais equivalentes de s.
A solução proposta pelo algoritmo é s.

Este algoritmo foi aprimorado por Hu [6] de maneira que se tornasse uma 2.5-aproximação.

3.3 Problema Dial-a-Ride - DARP

Problemas de Agendar um Transporte (*Dial a Ride*) são problemas da família do Tipo Requisição de Transporte 3. Estes problemas modelam visitas a uma determinada localização, sem necessariamente coletar ou entregar algo. Por exemplo, visitas de uma empresa de manutenção aos seus clientes, planejamento de transporte público, etc [6].

Este problema pode ser modelado de forma estática ou dinâmica, dependendo do momento em que as requisições de transporte chegam para o algoritmo. A variação mais simples do *DARP* é a situação estática, que é considerada uma generalização do *TSP com Precedências*, também conhecido como *Problema de Ordenamento Sequencial*, onde a origem de cada requisição deve ser atendida antes do seu destino. Na ocasião de haver também restrições de capacidade de carga, o problema é chamado *Problema Capacitado do Caixeiro Viajante com Coleta e Entrega* [12].

O que o diferencia de sua versão capacitada para transporte de *commodities* (ver Seção 3.2) é que não buscamos otimizar apenas o problema para a demanda de carga, mas também minimizar os custos com relação a ordem e horário estabelecidos para as rotas considerando uma inconveniência para o cliente. A inconveniência pode ser o tempo que um passageiro fica dentro do carro ou quanto de atraso ele tem para chegar no destino [12].

Para todas as variações do *DARP* devemos considerar a conveniência para o cliente com as janelas de tempo em que as requisições de transporte devem ser atendidas, isto é, considerar se tentamos minimizar o tempo que leva para o cliente ir da sua origem ao destino levando o tempo mínimo possível para esse trajeto [6].

Um exemplo atual é o *CarPool*, onde um motorista busca diversas pessoas, sendo que cada uma é buscada em um local e deixada em outro. Este problema pode ser considerado um problema capacitado quando consideramos a quantidade máxima de pessoas no carro, mas é mais completo quando pensamos que as pessoas não só devem ir de um ponto ao outro e sim devem ser buscadas em um determinado horário. Além disso, ao contrário de cargas simples, elas não podem passar o dia inteiro no veículo, mas tem um tempo em que devem ser deixadas em seu destino e quais as tolerâncias de atraso do motorista e do passageiro.

Considere a notação para os parâmetros que recebemos no problema:

1. n é o número de clientes;
2. i é a origem de uma solicitação de transporte (um cliente);
3. $i + n$ é o destino de uma solicitação de transporte;
4. $P = \{1, 2, \ldots, n\}$ é o conjunto de todas as origens;
5. $D = \{n + 1, n + 2, \ldots, 2n\}$ é o conjunto de todos os destinos;
6. 0 e $2n + 1$ são as garagens da frota;
7. $N = P \cup D \cup \{0, 2n + 1\}$ é o conjunto de vértices que representam as origens e destinos das solicitações de transporte e a garagem da frota;
8. A o conjunto de arcos entre todos os pares de vértices $i, j \in N$, salvo:
   - $i = 0$ e $j = 2n + 1$ (não há arco entre as garagens),
   - $i = 0$ e $j \in D$ (não há arco entre a garagem inicial e um destino),
   - $i \in P$ e $j = 2n + 1$ (não há arco entre uma origem e a garagem final);
9. $[e_i;\ l_i]$ é a janela de tempo em que podemos visitar o vértice $i \in N$ onde e_i é o mais cedo e l_i o mais tarde que podemos atender o vértice i. Por exemplo, se uma solicitação precisa ser retirada na origem (ou deixada no destino) i às 9 horas com tolerância de 15 minutos, um carro deve estar em i entre as 8h45 e 9h15;
10. L_i é o tempo máximo que a solicitação de transporte do cliente $i \in P$ pode estar dentro do veículo;
11. K é o conjunto de veículos da frota;
12. Q_k é a capacidade de carga do veículo $k \in K$;
13. q_i é a quantidade de espaço que a solicitação i toma em um veículo, onde $i \in P \Rightarrow q_i = -q_{n+i}$ e $q_0 = q_{2n+1} = 0$;
14. $c^k_{i,j}$ é o custo de ir de i para j, com $ij \in A$, usando o carro $k \in K$;
15. T_k é o tempo máximo que a rota do carro k pode durar. Por exemplo, para um carro dirigido por um motorista que trabalha 8 horas por dias ele deve sair e voltar para a garagem em no máximo $T_k = 8$ horas;
16. $t_{i,j}$ é o tempo que leva para ir de i para j, com $ij \in A$.

Podemos considerar o $DARP$ como um problema onde recebemos um grafo $G = (N, A)$ formado pelo subgrafo $O = (P \cup D, A)$ completo que conecta todos os clientes e destinos, a garagem de saída 0 conectada às origens de todos os serviços e a garagem de chegada $2n + 1$ conectada a todos os destinos.

Considere as variáveis de decisão:

- $x_{i,j}^k$ é uma variável de decisão tal que $x_{i,j}^k = 1$ se o carro k passou pela aresta i, j e 0 caso contrário;
- L_i^k indica o tempo que a solicitação $i \in P$ de fato esteve dentro do veículo k;
- B_i^k indica o tempo em que o veiculo k chega no vértice $i \in N$;
- Q_i^k indica a quantidade de carga acumulada do veículo k ao atender o vértice $i \in N$, que deve ser igual a soma de toda a carga no veículo mais a carga q_i, da solicitação $i \in N$.

Temos então o seguinte modelo para o $DARP$:

$$\text{minimizar} \quad \sum_{k \in K} \sum_{i \in P} \sum_{j \in D} c_{i,j}^k x_{i,j}^k \tag{28}$$

$$\text{sujeito a} \quad \sum_{k \in K} \sum_{j \in D} x_{i,j}^k = 1 \qquad \forall i \in P \tag{29}$$

$$\sum_{j \in N} x_{i,j}^k - \sum_{j \in N} x_{n+i,j}^k = 0 \qquad \forall i \in P,\ \forall k \in K \tag{30}$$

$$\sum_{j \in N} x_{0,j}^k = 1 \qquad \forall k \in K \tag{31}$$

$$\sum_{j \in N} x_{j,i}^k - \sum_{j \in N} x_{i,j}^k = 0 \qquad \forall i \in P \cup D,\ \forall k \in K \tag{32}$$

$$\sum_{i \in N} x_{i,2n+1} = 1 \qquad \forall k \in K \tag{33}$$

$$B_j^k \geq (B_i^k + d_i + t_{i,j}) x_{i,j}^k \qquad \forall i \in N,\ \forall j \in N,\ \forall k \in K \tag{34}$$

$$L_i^k \geq B_{n+i}^k - (B_i^k + d_i) \qquad \forall i \in P,\ \forall k \in K \tag{35}$$

$$B_{2n+1}^k - B_0^k \leq T_k \qquad \forall k \in K \tag{36}$$

$$e_i \leq B_i^k \leq l_i \qquad \forall i \in N,\ \forall k \in K \tag{37}$$

$$t_{i,n+i} \leq L_i^k \leq L_i \qquad \forall i \in P,\ \forall k \in K \tag{38}$$

$$Q_j^k \geq (Q_i^k + q_j) x_{i,j}^k \qquad \forall i \in N,\ \forall j \in N,\ \forall k \in K \tag{39}$$

$$\max\{0, q_i\} \leq Q_i^k \leq \min\{Q_k, Q_k + q_i\} \qquad \forall i \in N,\ \forall k \in K \tag{40}$$

$$x_{i,j}^k \in \{0, 1\} \qquad \forall i \in N,\ \forall j \in N,\ \forall k \in K \tag{41}$$

Note que o modelo descreve o problema pois:

- (28) Minimizamos o custo total das rotas dos carros, que vão de vértices i para vértices j

- (29) Garante que todos os vértices de origem sejam visitados por algum carro k

- (30) Garante que o carro k atenda um serviço $i, n+i$ em sua rota. Isto é, que se ele passar por uma origem $i \in P$, independente do caminho que ele faça pelo grafo, ele deve chegar e sair do destino $n+i$

- (31) Garante que todos os veículos k sejam utilizados, ou seja, partam da garagem

- (32) Garante que o mesmo veiculo k chegue em um vértice i e saia do mesmo, se i não for nenhuma garagem.

- (33) Garante que todo carro k vá para a garagem de destino $2n+1$ apenas uma vez

- (34) Se o carro k vai do nó i para o j, então o tempo B_j^k em que ele chegou em j é maior ou igual à soma do tempo em que ele chegou em i, B_i^k, mais o tempo que ele passou parado em i, d_i, mais o tempo do trajeto de i para j, $t_{i,j}$. Caso o carro não tenha passado por i, j, o tempo precisa apenas ser maior ou igual a zero.

- (35) Garante que L_i^k armazene o tempo que o cliente $i \in P$ passou no carro k, que é a diferença entre o momento de chegada em $i+n$, B_{i+n}^k, e o momento em que i saiu da origem somado ao tempo que ele passou na origem, $B_i^k + d_i$.

- (36) Garante que nenhum carro k passe mais que o seu limite de operação T_k em rota.

- (37) Garante que um carro atenda o vértice $i \in N$ dentro da janela de tolerância de tempo $[e_i; l_i]$.

- (38) Garante que o tempo que o cliente i passa dentro do veículo k, L_i^k, seja pelo menos o tempo mínimo de atendimento $t_{i,n+i}$ (se o serviço for atendido imediatamente) e no máximo o tempo limite que o cliente pode passar dentro do veículo, L_i.

- (39) Se o carro k vai do nó i para o j, garante a quantidade de carga acumulada em j, Q_j^k, seja pelo menos a quantidade de carga acumulada em i mais a quantidade de carga a ser pega ou deixada em j. Caso o carro não tenha passado por i, j, a carga precisa apenas ser maior ou igual a zero.

- (40) Garante que a quantidade de carga acumulada pelo veículo k ao passar em i, Q_i^k, seja pelo menos a quantidade de carga a ser retirada em q_i (note que se i for um destino q_i é negativo, por isso no mínimo é 0) e no máximo a capacidade de carga do veículo Q_k

Por causa das restrições (34) e (39), o programa acima é *Não-Linear*. Para torná-lo um *PL* precisamos considerar os novos parâmetros:

- $M_{i,j}^k = \max\{0,\ l_i + d_i - e_j\}$
- $W_{i,j}^k = \min\{Q_k,\ Q_k + q_i\}$

E substituímos as restrições (34) e (39) por, respectivamente:

$$B_j^k \geq B_i^k + d_i + t_{i,j} - M_{i,j}^k(1 - x_{i,j}^k) \qquad \forall i \in N,\ \forall j \in N,\ \forall k \in K \qquad (42)$$
$$Q_j^k \geq Q_i^k + q_j - W_{i,j}^k(1 - x_{i,j}^k) \qquad \forall i \in N,\ \forall j \in N,\ \forall k \in K \qquad (43)$$

Dessa maneira, o *PL* do *DARP* fica:

$$\begin{aligned}
\text{minimizar} \quad & \sum_{k\in K}\sum_{i\in P}\sum_{j\in D} c_{i,j}^{(k)} x_{i,j}^{(k)} & \\
\text{sujeito a} \quad & \sum_{k\in K}\sum_{j\in D} x_{i,j}^k = 1 & \forall i \in P \\
& \sum_{j\in N} x_{i,j}^k - \sum_{j\in N} x_{n+i,j}^k = 0 & \forall i \in P,\ \forall k \in K \\
& \sum_{j\in N} x_{0,j}^k = 1 & \forall k \in K \\
& \sum_{j\in N} x_{j,i}^k - \sum_{j\in N} x_{i,j}^k = 0 & \forall i \in P \cup D,\ \forall k \in K \\
& \sum_{i\in N} x_{i,2n+1} = 1 & \forall k \in K \\
(42)\ & B_j^k \geq B_i^k + d_i + t_{i,j} - M_{i,j}^k(1 - x_{i,j}^k) & \forall i \in N,\ \forall j \in N,\ \forall k \in K \\
& L_i^k \geq B_{n+i}^k + (B_i^k + d_i) & \forall i \in P,\ \forall k \in K \\
& B_{2n+1}^k - B_0^k \leq T_k & \forall k \in K \\
& e_i \leq B_i^k \leq l_i & \forall i \in N,\ \forall k \in K \\
& t_{i,n+i} \leq L_i^k \leq L_i & \forall i \in P,\ \forall k \in K \\
(43)\ & Q_j^k \geq Q_i^k + q_j - W_{i,j}^k(1 - x_{i,j}^k) & \forall i \in N,\ \forall j \in N,\ \forall k \in K \\
& \max\{0, q_i\} \leq Q_i^k \leq \min\{Q_k, Q_k + q_i\} & \forall i \in N,\ \forall k \in K \\
& x_{i,j}^k \in \{0, 1\} & \forall i \in N,\ \forall j \in N,\ \forall k \in K
\end{aligned}$$

4 Conclusão

Apresentamos alguns resultados sobre três variações de problemas de roteamento. Eles se encaixam em duas das sete classes mencionadas na Seção 3, de forma que não cobrem, de forma alguma, todas as particularidades que podem surgir em problemas de roteamento. Mesmo assim, pudemos estudar várias delas e cumprir o objetivo da iniciação científica. Notamos ainda que os três problemas escolhidos são bem relacionados, no sentido que um generaliza o outro.

Referências

[1] Mikhail J Atallah and S Rao Kosaraju. Efficient solutions to some transportation problems with applications to minimizing robot arm travel. *SIAM Journal on Computing*, 17(5):849–869, 1988.

[2] D. Bertsimas and J. N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific series in optimization and neural computation. Athena Scientific, 1997.

[3] Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, 1976.

[4] Britannica Encyclopædia. Optimization. https://www.britannica.com/science/optimization.

[5] Prof. Dra. Cristina Gomes Fernandes. https://www.youtube.com/watch?v=sBbCyvCfzag.

[6] Yuzhuang Hu. *Approximation algorithms for the capacitated vehicle routing problem*. PhD thesis, School of Computing Science-Simon Fraser University, 2009.

[7] S. O. Krumke, J. Rambau, and S. Weider. *An Approximation Algorithm for the Non-preemptive Capacitated Dial-a-ride Problem*. Konrad-Zuse-Zentrum für Informationstechnik Berlin. ZIB, 2000.

[8] Jon Lee. *A First Course in Combinatorial Optimization*. Cambridge Texts in Applied Mathematics. Cambridge University Press, 2004.

[9] C. E. Leiserson, T. H. Cormen, R. L. Rivest, and C. Stein. *Algoritmos: teoria e prática*. Elservier, 2002.

[10] S. Sahni and T. Gonzalez. P-Complete Approximation Problems. *Journal of the ACM*, 23(3):555–565, 1976.

[11] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley Series in Discrete Mathematics & Optimization. John Wiley & Sons, 1998.

[12] P. Toth and D. Vigo. *Vehicle Routing: Problems, Methods, and Applications, Second Edition*. MOS-SIAM Series on Optimization. Society for Industrial and Applied Mathematics, 2014.