UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
FACULDADE DE CIÊNCIA E TECNOLOGIA (FCT)

IGOR J. RODRIGUES
LEONARDO S. CORADELI
LUCAS V. C. IKEDA
MARCO V. M. FARIA

ESTUDO SOBRE PROBLEMAS DE OTIMIZAÇÃO: TRANSPORTE, ATRIBUIÇÃO E MENOR CAMINHO

Presidente Prudente
2025

IGOR J. RODRIGUES
LEONARDO S. CORADELI
LUCAS V. C. IKEDA
MARCO V. M. FARIA

ESTUDO SOBRE PROBLEMAS DE OTIMIZAÇÃO: TRANSPORTE, ATRIBUIÇÃO E MENOR CAMINHO

Trabalho apresentado à disciplina de Programação Linear do curso de Ciência da Computação da Universidade Estadual Paulista (UNESP), Campus de Presidente Prudente.

Presidente Prudente
2025

RESUMO

A Pesquisa Operacional (PO) oferece um arsenal de ferramentas matemáticas para a tomada de decisão e otimização de sistemas complexos. Este trabalho foca na análise de três problemas clássicos e fundamentais da PO: o Problema do Transporte, o Problema da Atribuição e o Problema do Menor Caminho. Cada um desses problemas é explorado desde sua definição conceitual e aplicações práticas até sua formulação matemática como um Problema de Programação Linear (PPL). O estudo detalha os métodos de resolução específicos para cada caso, comparando a eficiência de algoritmos especializados, como o Algoritmo Húngaro para atribuição, e os algoritmos de Dijkstra e Bellman-Ford para caminhos mínimos, com a abordagem genérica do método Simplex. O objetivo é consolidar o entendimento teórico e prático desses modelos, demonstrando suas particularidades, restrições e a importância da escolha do método de solução adequado para garantir a eficiência computacional e a otimalidade.

Palavras-chave: Pesquisa Operacional. Otimização. Problema do Transporte. Problema da Atribuição. Problema do Menor Caminho. Programação Linear.

Sumário

CAPÍTULO 1

INTRODUÇÃO

A Pesquisa Operacional (PO) é um ramo da matemática aplicada que se dedica ao desenvolvimento e aplicação de métodos analíticos para auxiliar na tomada de decisões. Seu objetivo primordial é a otimização, seja ela a maximização de lucros e eficiência ou a minimização de custos, tempo e riscos. Dentro do vasto campo da PO, os problemas de otimização em redes e alocação ocupam um lugar de destaque devido à sua ampla aplicabilidade em logística, finanças, engenharia e ciência da computação.

Este trabalho se propõe a analisar três dos problemas mais fundamentais e didáticos da Pesquisa Operacional: o Problema do Transporte, o Problema da Atribuição e o Problema do Menor Caminho.

O Problema do Transporte busca determinar a forma mais econômica de expedir quantidades de um bem de diversas origens para diversos destinos. O Problema da Atribuição, um caso particular do transporte, foca em designar agentes a tarefas de forma otimizada (um-para-um), minimizando o custo total. Por fim, o Problema do Menor Caminho, onipresente em sistemas de navegação e roteamento de redes, visa encontrar a rota de menor custo (distância, tempo ou latência) entre dois pontos em um grafo ponderado.

Para cada um desses problemas, este relatório apresentará a definição formal, a formulação matemática como um Problema de Programação Linear (PPL) e os algoritmos especializados desenvolvidos para sua solução, comparando suas características e eficiência.

CAPÍTULO 2

O PROBLEMA DO TRANSPORTE

- É uma classe especial dentro dos problemas de programação linear que trata da distribuição de mercadorias de várias origens para vários destinos;
- O objetivo é encontrar um plano de transporte que **minimize o custo total** de envio, respeitando as restrições de **oferta e demanda**.

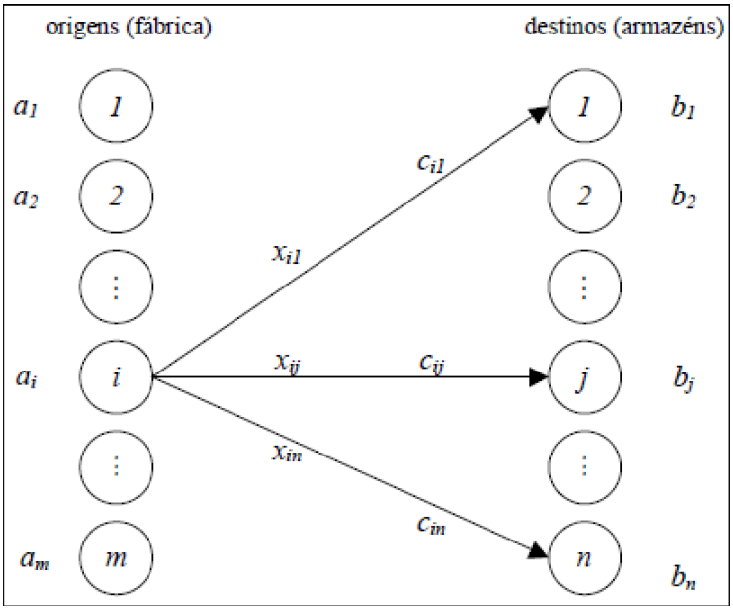

Figura 2.1: Diagrama esquemático do Problema do Transporte.

2.0.1 EXEMPLO PRÁTICO

- Uma empresa de calçados possui três fábricas e precisa distribuir sua produção para cinco lojas espalhadas pelo país. Cada fábrica possui uma capacidade máxima de produção semanal, e cada loja possui uma demanda específica a ser atendida.
- O custo de transporte de um par de sapatos entre cada fábrica e cada loja é conhecido e varia conforme a distância e o tipo de transporte utilizado.

- As capacidades de produção (oferta) das fábricas são: **200**, **300** e **250 pares**, respectivamente.
- As demandas das lojas são: **150**, **180**, **100**, **200** e **120 pares**, respectivamente.

Tabela 2.1: Tabela de Custos de Transporte (R$ por par)

	L1	L2	L3	L4	L5	Oferta
F1	8	6	10	9	7	200
F2	9	12	13	7	5	300
F3	14	9	16	5	8	250
Demanda	150	180	100	200	120	

2.0.2 FORMULAÇÃO MATEMÁTICA

- **Variáveis de decisão:**

$$x_{ij} = \text{quantidade transportada da origem } i \text{ para o destino } j$$

- **Função objetivo:**

$$\min Z = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij}$$

onde c_{ij} representa o custo de transporte por unidade.

- **Restrições:**

$$\sum_{j=1}^{n} x_{ij} = a_i, \qquad \forall i = 1, \ldots, m \quad \text{(oferta)}$$

$$\sum_{i=1}^{m} x_{ij} = b_j, \qquad \forall j = 1, \ldots, n \quad \text{(demanda)}$$

$$x_{ij} \geq 0, \qquad \forall i, j$$

2.0.3 MODELANDO O EXEMPLO

Formulação das Restrições do Exemplo

Restrições de Oferta (Capacidade de Produção):

$$x_{11} + x_{12} + x_{13} + x_{14} + x_{15} = 200$$

$$x_{21} + x_{22} + x_{23} + x_{24} + x_{25} = 300$$

$$x_{31} + x_{32} + x_{33} + x_{34} + x_{35} = 250$$

Restrições de Demanda (Necessidade das Lojas):

$$x_{11} + x_{21} + x_{31} = 150$$
$$x_{12} + x_{22} + x_{32} = 180$$
$$x_{13} + x_{23} + x_{33} = 100$$
$$x_{14} + x_{24} + x_{34} = 200$$
$$x_{15} + x_{25} + x_{35} = 120$$

Restrições de não negatividade:

$$x_{ij} \geq 0 \quad \forall i, j$$

Interpretação do Exemplo

- O objetivo é determinar quantos pares de sapatos cada fábrica deve enviar para cada loja de forma a **minimizar o custo total de transporte**.
- Todas as demandas devem ser atendidas, e nenhuma fábrica pode produzir mais do que sua capacidade máxima.
- Esse tipo de problema é conhecido como **Problema do Transporte Balanceado**, pois a soma das ofertas é igual à soma das demandas:

$$200 + 300 + 250 = 150 + 180 + 100 + 200 + 120 = 750$$

- Pode ser resolvido utilizando métodos como o **Canto Noroeste**, **Vogel**, **Menor Custo**

Equilíbrio entre Oferta e Demanda (Problemas Desbalanceados)

Os métodos de solução clássicos, como o Método de Vogel e o MODI, exigem que o problema esteja **balanceado**, ou seja, que a Oferta Total seja exatamente igual à Demanda Total.

$$\sum_{i=1}^{m} \text{Oferta}_i = \sum_{j=1}^{n} \text{Demanda}_j$$

Quando isso não ocorre, o problema é dito **desbalanceado** e precisa ser ajustado antes da resolução, criando-se rotas "fictícias"para equilibrar o modelo:

- **Caso 1: Oferta Total > Demanda Total** (Excesso de oferta)
  - **Solução:** Cria-se um **destino fictício** (uma coluna "Folga"na tabela).

- **Demanda Fictícia:** A demanda desse destino será a diferença (Oferta Total− Demanda Total).
- **Custo Fictício:** O custo de envio (c_{ij}) de todas as origens para esse destino fictício é **zero**. (Isso representa a mercadoria que "sobra"e não é enviada).

- **Caso 2: Oferta Total $<$ Demanda Total** (Excesso de demanda)
  - **Solução:** Cria-se uma **origem fictícia** (uma linha "Falta"na tabela).
  - **Oferta Fictícia:** A oferta dessa origem será a diferença (Demanda Total − Oferta Total).
  - **Custo Fictício:** O custo de envio (c_{ij}) dessa origem fictícia para todos os destinos é **zero**. (Isso representa a demanda que não será atendida).

Após a adição dessa origem ou destino fictício, o problema torna-se balanceado e pode ser resolvido pelos métodos padrões.

2.0.4 MÉTODOS DE SOLUÇÃO

- Método do Canto Noroeste;
- Método do Custo Mínimo;
- **Método de Vogel.**

2.0.5 MÉTODO DE VOGEL

- O **Método de Vogel** é uma técnica heurística usada para encontrar uma **solução inicial viável** para o problema do transporte.
- Ele busca equilibrar **custo e penalidade**, tentando minimizar o custo total logo na alocação inicial.
- O princípio é penalizar as escolhas caras: a cada etapa, calcula-se o quanto se "perde" ao não escolher a opção mais barata de cada linha e coluna.

Etapas do Método de Vogel

1. **Calcular as penalidades:** Para cada linha e coluna, determine a diferença entre os dois menores custos dessa linha/coluna. Essa diferença representa a **penalidade**.
2. **Identificar a maior penalidade:** Escolha a linha ou coluna com a penalidade mais alta (ou seja, onde o erro seria mais caro se não for escolhida a menor rota).

3. **Selecionar a menor célula de custo** nessa linha ou coluna e alocar:

$$x_{ij} = \min(\text{oferta}_i, \text{demanda}_j)$$

4. **Atualizar as ofertas e demandas:** Subtraia o valor alocado e risque a linha ou coluna esgotada.

5. **Repetir o processo** até que todas as ofertas e demandas sejam atendidas.

Exemplo de Aplicação do Método de Vogel

Tabela 2.2: Tabela de custos e demandas (Exemplo Vogel)

	D1	D2	D3	Oferta
O1	6	8	10	20
O2	7	11	11	15
O3	4	5	12	25
Demanda	10	10	40	

Vogel — Iteração 1

- Penalidades (diferença entre dois menores custos):
  - Linhas: $r_1 = 2,\ r_2 = 4,\ r_3 = 1$
  - Colunas: $c_1 = 2,\ c_2 = 3,\ c_3 = 1$
- Maior penalidade: **linha O2** ($r_2 = 4$). Menor custo nessa linha: $c_{2,1} = 7$.
- Alocação: $x_{2,1} = \min(15, 10) = 10$.

Tabela 2.3: Vogel - Iteração 1

	D1	D2	D3	Oferta
O1	6	8	10	20
O2	**7 (10)**	11	11	5
O3	4	5	12	25
Demanda	0	10	40	

Vogel — Iteração 2

- Penalidades (após atualização):
  - Linhas: $r_1 = 2,\ r_2 = 0,\ r_3 = 7$
  - Colunas: $c_2 = 3,\ c_3 = 1$

- Maior penalidade: **linha O3** ($r_3 = 7$). Menor custo nessa linha: $c_{3,2} = 5$.
- Alocação: $x_{3,2} = \min(25, 10) = 10$.

Tabela 2.4: Vogel - Iteração 2

	D1	D2	D3	Oferta
O1	6	8	10	20
O2	7 (10)	11	11	5
O3	4	**5 (10)**	12	15
Demanda	0	0	40	

Vogel — Iterações 3, 4 e 5 (coluna D3)

- Agora resta apenas a coluna D3 (demanda 40).
- Passos realizados:
  1. Alocamos $x_{3,3} = 15$ (restante de O3).
  2. Alocamos $x_{2,3} = 5$ (restante de O2).
  3. Alocamos $x_{1,3} = 20$ (restante de O1).

Tabela 2.5: Vogel - Iterações 3, 4 e 5

	D1	D2	D3	Oferta
O1	6	8	**10 (20)**	0
O2	7 (10)	11	**11 (5)**	0
O3	4	5 (10)	**12 (15)**	0
Demanda	0	0	40	

Solução inicial (Vogel) e Custo Total

$$X = \begin{pmatrix} 0 & 0 & 20 \\ 10 & 0 & 5 \\ 0 & 10 & 15 \end{pmatrix}$$

- Cálculo do custo:

$$Z = 7 \cdot 10 + 10 \cdot 20 + 5 \cdot 10 + 11 \cdot 5 + 12 \cdot 15 = 555$$

- Solução encontrada pelo Método de Vogel (alocação inicial): $\boxed{Z = 555}$.

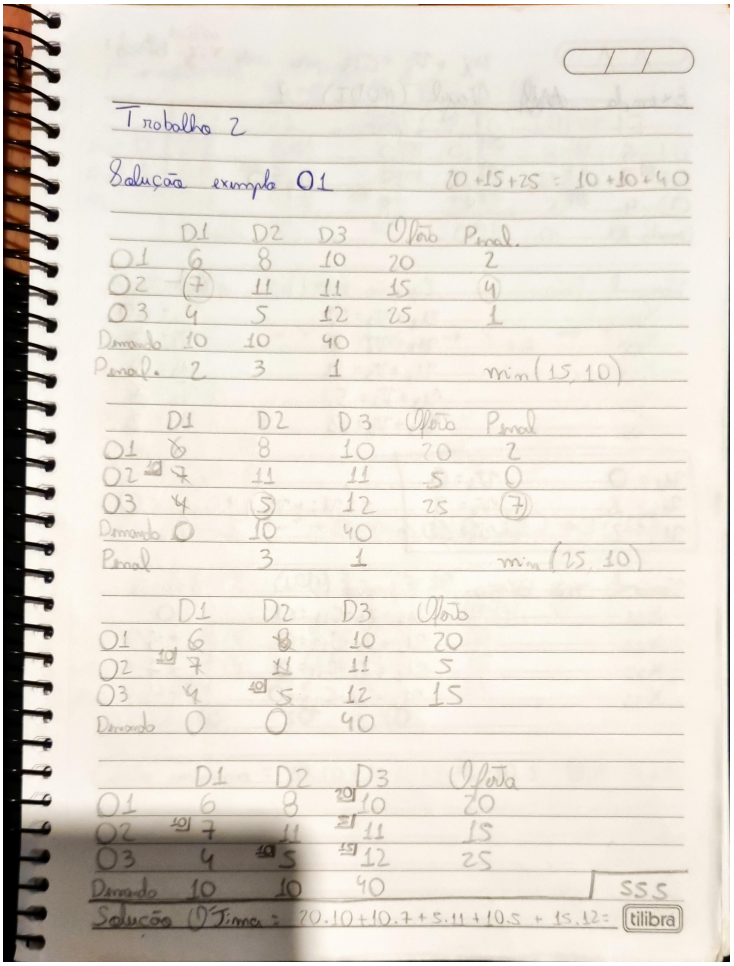

Figura 2.2: Resolução exemplo 1.

Exercício - Método de Vogel

Tabela 2.6: Tabela de custos e demandas (Exercício)

	D1	D2	D3	D4	Oferta
O1	5	8	6	10	30
O2	9	7	4	8	40
O3	6	5	8	9	50
Demanda	20	30	25	45	

Enunciado

Determine o plano de transporte ótimo utilizando o **Método de Aproximação de Vogel**, de modo a minimizar o custo total.

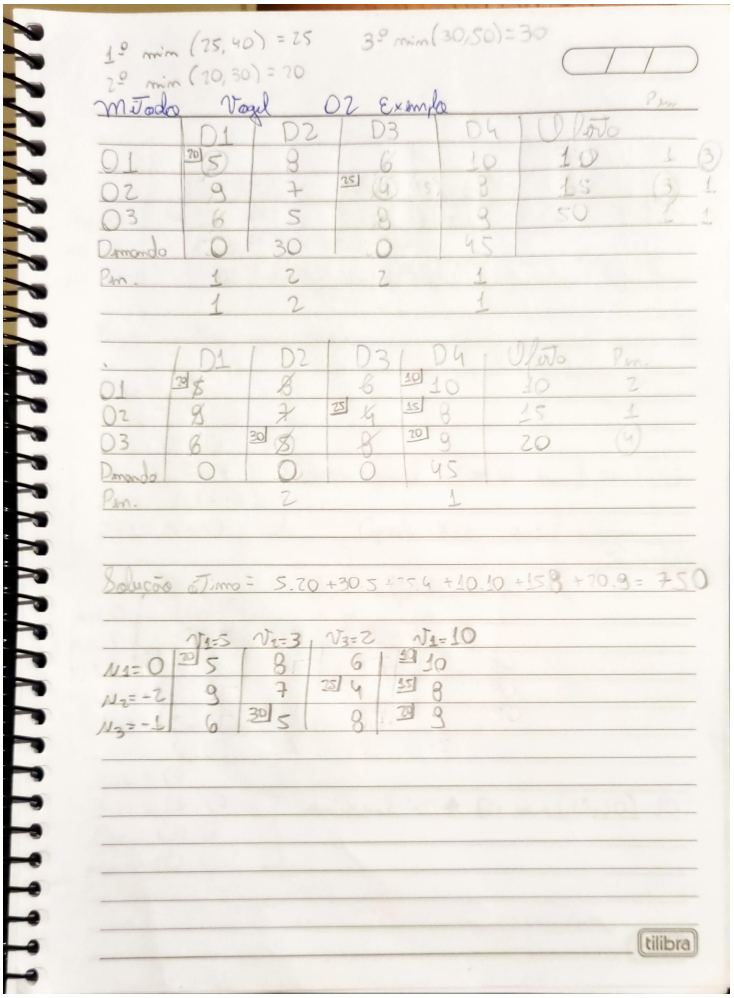

Figura 2.3: Resolução exercício 1.

Características do Método de Vogel

- Fornece soluções iniciais geralmente **mais próximas da ótima** do que o Método do Canto Noroeste ou o do Custo Mínimo;
- Requer cálculos adicionais, mas reduz o número de iterações posteriores no método de otimização (Simplex);
- É amplamente utilizado em problemas de transporte reais devido ao seu **bom equilíbrio entre precisão e esforço computacional**.

2.0.6 MÉTODO U-V (MODI)

- Também conhecido como **Método MODI** (*Modified Distribution Method*);
- É utilizado para **verificar a otimalidade** de uma solução viável inicial do problema do transporte;

- Permite identificar se é possível **reduzir o custo total** ajustando a alocação das variáveis não básicas;
- É uma forma simplificada de aplicar o **Simplex** ao problema do transporte.

Ideia do Método u-v

- O método se baseia na atribuição de dois conjuntos de variáveis:

$$u_i \text{ (para as origens) e } v_j \text{ (para os destinos)}$$

- Para cada célula básica (i, j), temos:

$$c_{ij} = u_i + v_j$$

- As células não básicas são então avaliadas pelo **custo reduzido**:

$$\Delta_{ij} = c_{ij} - (u_i + v_j)$$

Critério de Otimalidade

Condição de Ótimo: Uma solução é **ótima** se e somente se:

$$\Delta_{ij} \geqslant 0 \quad \forall (i, j) \text{ não básicos.}$$

- Se todos os Δ_{ij} forem positivos ou nulos, o custo total atual é mínimo;
- Caso exista algum $\Delta_{ij} < 0$, há oportunidade de melhoria no custo, a solução ainda não é ótima;
- Nesse caso, deve-se ajustar a alocação construindo um **ciclo fechado** para redistribuir as quantidades.

Etapas do Método MODI

1. **Obter uma solução inicial viável** (ex: Método de Vogel);
2. **Calcular os valores de u_i e v_j:**
   - Escolha arbitrariamente $u_1 = 0$;
   - Para cada célula básica, use $c_{ij} = u_i + v_j$ para determinar os demais valores;
3. **Calcular os custos reduzidos:**

$$\Delta_{ij} = c_{ij} - (u_i + v_j)$$

4. **Verificar a otimalidade:**

   - Se todos os $\Delta_{ij} \geqslant 0$, pare, a solução é ótima;
   - Caso contrário, vá para o próximo passo.

Etapas do Método MODI (continuação)

5. **Identificar a célula com o menor Δ_{ij}:**

   - Essa será a célula **candidata a entrar na base**;

6. **Construir o ciclo fechado:**

   - Alterne entre células básicas, formando um caminho fechado;
   - Sinalize os movimentos com + e −;

7. **Atualizar as alocações:**

$$x_{ij} = x_{ij} \pm \theta$$

   onde θ é o menor valor das células marcadas com −;

8. **Repetir o processo** até que todos os $\Delta_{ij} \geqslant 0$.

Exemplo - Método u-v (MODI)

Tabela 2.7: Tabela de custos e solução inicial (via Vogel)

	D1	D2	D3	Oferta
O1	6	8	10	20
O2	7	11	11	15
O3	4	5	12	25
Demanda	10	10	40	

Objetivo

A partir da solução inicial obtida pelo Método de Vogel, aplique o **Método MODI** para verificar se a solução é ótima e, se necessário, encontrar uma solução de custo menor.

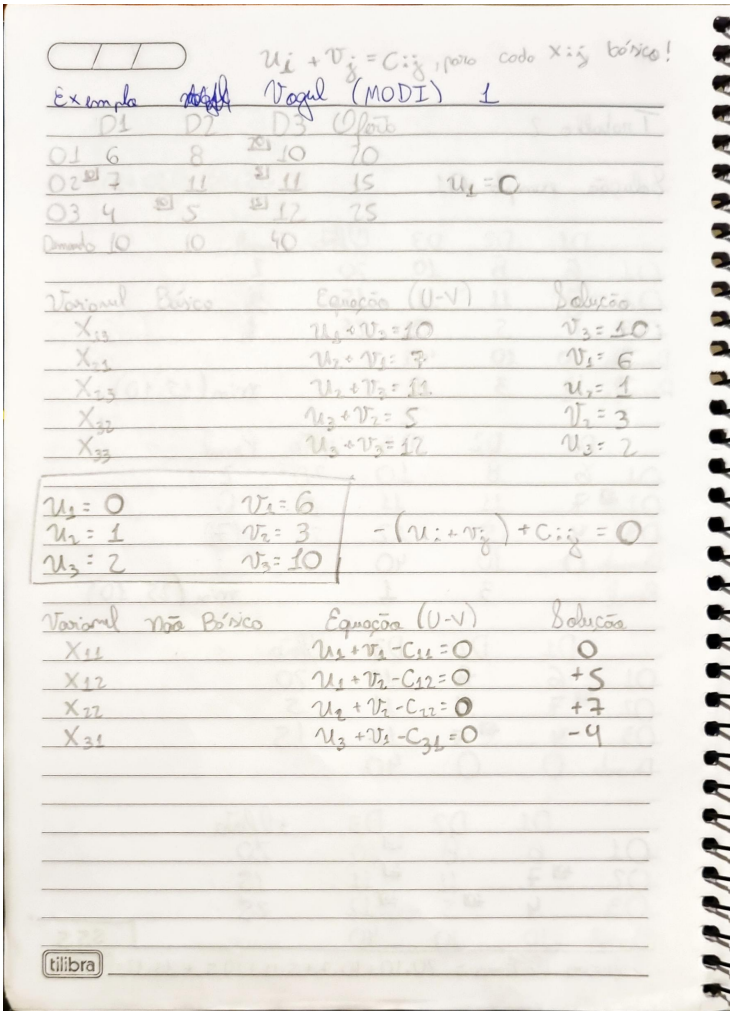

Figura 2.4: Resolução exemplo 1 (MODI).

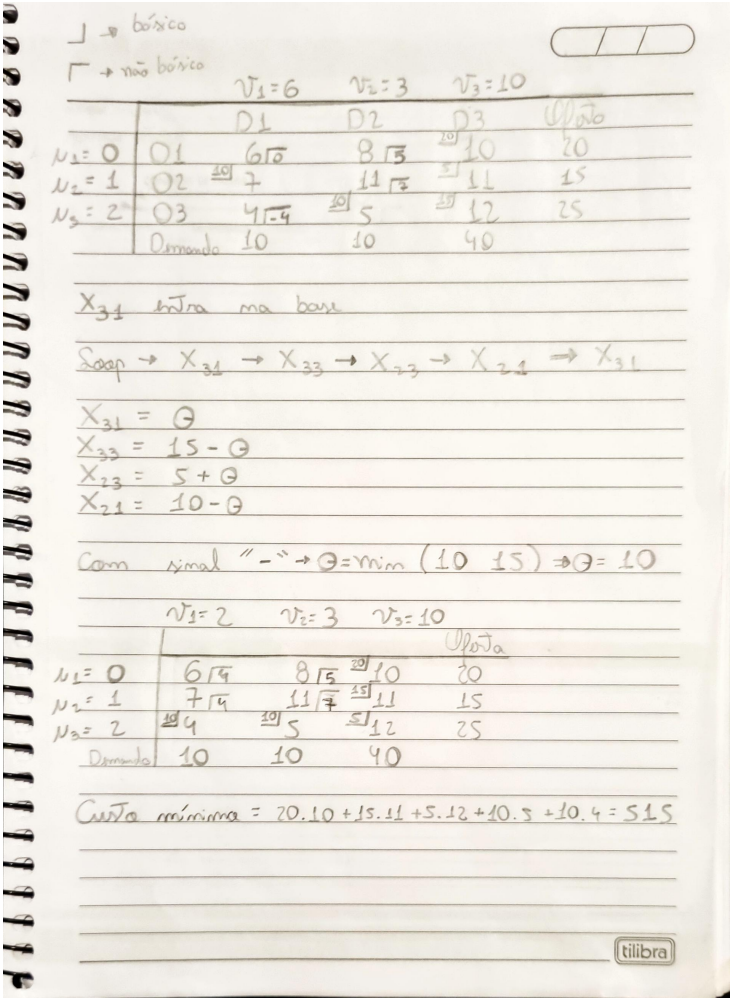

Figura 2.5: Continuação resolução exemplo 1 (MODI).

Exercício - Método u-v (MODI)

Tabela 2.8: Tabela de custos e demandas (Exercício MODI)

	D1	D2	D3	D4	Oferta
O1	10 (5)	2 (10)	20	11	15
O2	12	7 (5)	9 (15)	20 (5)	25
O3	4	14	16	19 (10)	10
Demanda	5	15	15	15	

Enunciado

Determine o plano de transporte ótimo utilizando o **Método de Aproximação de Vogel**, de modo a minimizar o custo total.

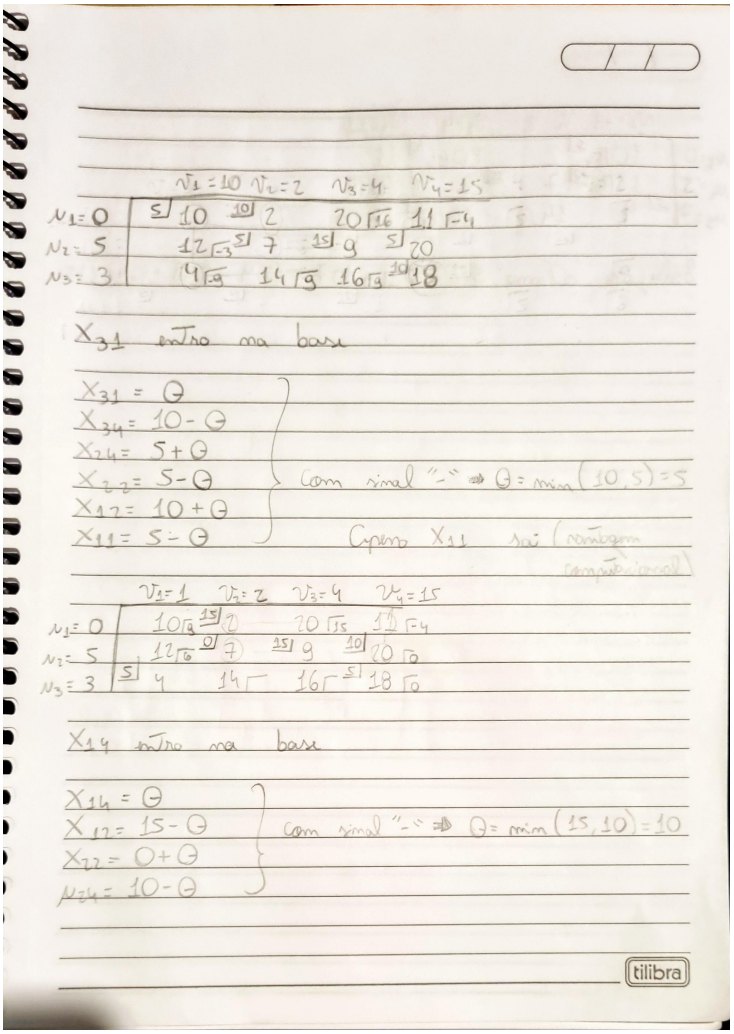

Figura 2.6: Resolução exercicio 1 (MODI).

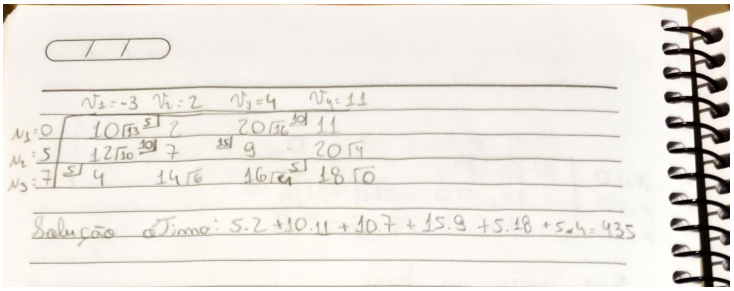

Figura 2.7: Continuação resolução exercicio 1 (MODI).

Vantagens do Método u-v (MODI)

- É o **método mais eficiente** para testar e melhorar soluções do problema do transporte;
- Evita o uso completo do **Simplex**, tornando o processo mais direto;
- Facilita a visualização da estrutura do problema;

Resumo do Método u-v (MODI)

Passos Principais

1. Obtenha uma solução inicial viável (Vogel, Canto Noroeste, etc.);
2. Calcule u_i e v_j com base nas células básicas;
3. Determine Δ_{ij} para as não básicas;
4. Verifique se há $\Delta_{ij} < 0$;
5. Ajuste as alocações por ciclos fechados até atingir a otimalidade.

2.0.7 CONCLUSÃO

- O problema do transporte é uma aplicação clássica da programação linear com grande relevância prática;
- Seu estudo auxilia na compreensão de modelos de otimização com restrições de capacidade e custo;
- Métodos como o de Vogel e o MODI permitem soluções eficientes e economicamente viáveis.

CAPÍTULO 3

O PROBLEMA DA ATRIBUIÇÃO

O Problema da Atribuição (PA) é um caso especial do Problema do Transporte, caracterizado pela necessidade de alocar um conjunto de n agentes (ou recursos) a um conjunto de n tarefas (ou atividades), de forma que cada agente execute exatamente uma tarefa e cada tarefa seja executada por exatamente um agente.

3.1 DEFINIÇÃO E APLICAÇÕES

A definição formal do Problema da Atribuição consiste em encontrar a alocação biunívoca entre agentes e tarefas que minimize (ou maximize) o custo total da operação. O custo c_{ij} representa o valor associado à designação do agente i para a tarefa j.

Este modelo é vastamente aplicado em cenários reais, tais como:

- Alocação de funcionários a máquinas ou postos de trabalho;
- Designação de motoristas a rotas de entrega (minimizando o tempo total);
- Atribuição de professores a disciplinas;
- Alocação de projetos a equipes de desenvolvimento.

3.2 MODELO MATEMÁTICO

Para formular o Problema da Atribuição como um Problema de Programação Linear, definem-se as variáveis de decisão, a função objetivo e as restrições.

Variáveis de Decisão: Seja x_{ij} uma variável binária:

$$x_{ij} = \begin{cases} 1 & \text{se o agente } i \text{ é atribuído à tarefa } j \\ 0 & \text{caso contrário} \end{cases}$$

Função Objetivo: O objetivo é minimizar o custo total, que é a soma dos custos de todas as atribuições realizadas:

$$\min Z = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{3.1}$$

Restrições: As restrições garantem que cada agente faça exatamente uma tarefa e que cada tarefa seja feita por exatamente um agente.

$$\sum_{j=1}^{n} x_{ij} = 1 \quad \forall i = 1, \ldots, n \quad \text{(Cada agente faz 1 tarefa)} \tag{3.2}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \quad \forall j = 1, \ldots, n \quad \text{(Cada tarefa é feita por 1 agente)} \tag{3.3}$$

$$x_{ij} \in \{0, 1\} \quad \forall i, j \tag{3.4}$$

3.3 MÉTODOS DE RESOLUÇÃO

Embora o PA possa ser resolvido por métodos genéricos de Programação Linear, existem algoritmos muito mais eficientes e especializados para sua estrutura particular.

3.3.1 RESOLUÇÃO POR PROGRAMAÇÃO LINEAR (SIMPLEX)

O modelo apresentado na Seção 3.2 pode ser resolvido diretamente pelo método Simplex. No entanto, esta abordagem apresenta limitações significativas. O número de variáveis (n^2) e de restrições ($2n$) cresce rapidamente com o tamanho do problema.

A construção e iteração do *tableau* Simplex para problemas de grande porte torna-se computacionalmente inviável ou, no mínimo, ineficiente. Embora o Simplex funcione (e, devido à propriedade de unimodularidade total da matriz, garanta solução inteira mesmo relaxando $x_{ij} \geq 0$), ele não é o método preferencial na prática.

3.3.2 O ALGORITMO HÚNGARO

O Algoritmo Húngaro é o método clássico e mais eficiente para resolver o Problema da Atribuição. Ele opera diretamente sobre a matriz de custos $C = [c_{ij}]$ e explora a propriedade de que a solução ótima não se altera se uma constante for somada ou subtraída de qualquer linha ou coluna da matriz de custos.

O algoritmo busca transformar a matriz de custos em uma matriz que contenha zeros, de forma que seja possível encontrar uma atribuição completa (uma para cada agente) usando apenas as posições de custo zero. As etapas principais são:

1. **Redução de Linhas:** Subtrair o menor valor de cada linha de todos os elementos daquela linha.

2. **Redução de Colunas:** Após a redução das linhas, subtrair o menor valor de cada coluna de todos os elementos daquela coluna.

3. **Verificação da Otimalidade (Cobertura por Linhas):** Tentar cobrir todos os zeros da matriz resultante com o menor número possível de linhas (horizontais ou verticais). Se o número mínimo de linhas for igual a n (ordem da matriz), uma solução ótima foi encontrada.

4. **Iteração (Se não for ótimo):** Se o número de linhas for menor que n, encontrar o menor elemento *não coberto* pelas linhas. Subtrair este valor de todos os elementos não cobertos e somá-lo a todos os elementos que estão na interseção de duas linhas de cobertura. Retornar ao Passo 3.

Este método garante a otimalidade e é executado em tempo polinomial (tipicamente $O(n^3)$), sendo vastamente superior ao Simplex para esta classe de problema.

3.4 VARIAÇÕES DO PROBLEMA DE ATRIBUIÇÃO

O modelo clássico assume n agentes para n tarefas (problema balanceado). Na prática, cenários desbalanceados são comuns:

- **Atribuição Desequilibrada (Agentes $\neq$ Tarefas):** Se o número de agentes for diferente do número de tarefas, o problema é "balanceado"pela adição de agentes ou tarefas fictícias (*dummy*). Essas posições fictícias recebem custo zero (se a alocação não for obrigatória) ou um custo proibitivo (M).

- **Problema de Atribuição Generalizada (GAP):** Esta é uma variante NP-difícil onde os agentes podem executar múltiplas tarefas e as tarefas podem ter demandas diferentes, respeitando-se restrições de capacidade (ex: tempo, orçamento) de cada agente. Este problema é mais complexo e exige métodos como *branch and bound* ou heurísticas.

3.5 ESTUDO DE CASO: RESOLUÇÃO MANUAL E ALGORÍTMICA

Considere a seguinte matriz de custos 3x3 (Tabela 3.1):

Tabela 3.1: Matriz de Custos do Exemplo de Atribuição

	Tarefa 1	Tarefa 2	Tarefa 3
Agente 1	15	20	17
Agente 2	13	18	21
Agente 3	19	14	16

Resolução por Enumeração (Força Bruta) Para um problema 3x3, existem $3! = 6$ atribuições possíveis:

- Atribuição 1 (A1-T1, A2-T2, A3-T3): $15 + 18 + 16 = 49$
- Atribuição 2 (A1-T1, A2-T3, A3-T2): $15 + 21 + 14 = 50$
- **Atribuição 3 (A1-T3, A2-T1, A3-T2):** $17 + 13 + 14 = 44$
- Atribuição 4 (A1-T3, A2-T2, A3-T1): $17 + 18 + 19 = 54$
- Atribuição 5 (A1-T2, A2-T1, A3-T3): $20 + 13 + 16 = 49$
- Atribuição 6 (A1-T2, A2-T3, A3-T1): $20 + 21 + 19 = 60$

A solução ótima é a Atribuição 3, com custo total mínimo de **44**.

Resolução pelo Algoritmo Húngaro

1. **Matriz Original:**
$$\begin{bmatrix} 15 & 20 & 17 \\ 13 & 18 & 21 \\ 19 & 14 & 16 \end{bmatrix}$$
2. **Passo 1: Redução de Linhas** (Subtrair 15, 13 e 14, respectivamente):
$$\begin{bmatrix} 0 & 5 & 2 \\ 0 & 5 & 8 \\ 5 & 0 & 2 \end{bmatrix}$$
3. **Passo 2: Redução de Colunas** (Subtrair 0, 0 e 2, respectivamente):
$$\begin{bmatrix} 0 & 5 & 0 \\ 0 & 5 & 6 \\ 5 & 0 & 0 \end{bmatrix}$$
4. **Passo 3: Alocação Ótima:** Nesta matriz, é possível encontrar uma alocação com 3 zeros em posições independentes (o número mínimo de linhas para cobrir os zeros é 3).

- A Linha 2 (Agente 2) só tem um zero na Coluna 1 → **Alocar A2-T1**.
- Eliminando L2 e C1, a Linha 3 (Agente 3) só tem um zero na Coluna 2 → **Alocar A3-T2**.
- Resta o Agente 1, que deve ser alocado à Tarefa 3 → **Alocar A1-T3**.

A solução (A1-T3, A2-T1, A3-T2) corresponde exatamente ao custo mínimo de 44 (17 + 13 + 14) encontrado na enumeração.

CAPÍTULO 4

O PROBLEMA DO MENOR CAMINHO

O Problema do Menor Caminho (PCM), ou *Shortest Path Problem* (SPP), é um dos problemas mais estudados e aplicados na teoria dos grafos e na Pesquisa Operacional. Ele é a base de muitas tecnologias modernas, desde sistemas de GPS até o roteamento de dados na internet.

4.1 DEFINIÇÃO E APLICAÇÕES

Formalmente, o problema é definido em um grafo ponderado $G = (V, E)$, onde V é o conjunto de vértices (nós) e E é o conjunto de arestas (ou arcos, em grafos direcionados). Cada aresta $(i, j) \in E$ possui um peso (ou custo) w_{ij} associado.

Dado um nó de origem s e um nó de destino t, o Problema do Menor Caminho consiste em encontrar a sequência de arestas (o caminho P) que conecta s a t de forma que a soma dos pesos das arestas nesse caminho seja minimizada.

$$P^* = \arg\min_{P:\, s \to t} \sum_{(i,j) \in P} w_{ij} \tag{4.1}$$

As aplicações são diversas e impactam diretamente o cotidiano:

- **Logística e Transportes:** Aplicações como Google Maps e Waze usam algoritmos de menor caminho para encontrar a rota mais rápida (custo = tempo) ou mais curta (custo = distância).
- **Redes de Computadores:** Protocolos de roteamento, como o OSPF (Open Shortest Path First), determinam como os pacotes de dados devem trafegar na internet, minimizando a latência.

- **Finanças:** Detecção de oportunidades de arbitragem em mercados (encontrar ciclos de custo negativo).
- **Análise de Redes Sociais:** Medir o "grau de separação"entre duas pessoas.

Frequentemente, os algoritmos de PCM resolvem o problema da **Origem Única** (*Single-Source Shortest Path* - SSSP), ou seja, encontram o caminho mínimo da origem s para *todos* os outros nós do grafo.

4.2 MODELO MATEMÁTICO POR PROGRAMAÇÃO LINEAR

O PCM pode ser formulado como um Problema de Programação Linear, especificamente como um Problema de Fluxo de Custo Mínimo. A ideia é enviar 1 unidade de "fluxo"da origem s para o destino t, minimizando o custo total desse envio.

Variáveis de Decisão: $x_{ij} = 1$ se o arco (i, j) for usado no caminho mínimo, e 0 caso contrário.

Função Objetivo: Minimizar o custo total do caminho percorrido pelo fluxo.

$$\min Z = \sum_{(i,j)\in E} w_{ij} \cdot x_{ij} \tag{4.2}$$

Restrições (Conservação de Fluxo): As restrições garantem que o fluxo de 1 unidade saia de s, chegue em t e seja conservado em todos os nós intermediários. Para um grafo com origem s e destino t:

$$\sum_j x_{sj} - \sum_i x_{is} = 1 \qquad \text{(Nó Origem s)} \tag{4.3}$$

$$\sum_j x_{tj} - \sum_i x_{it} = -1 \qquad \text{(Nó Destino t)} \tag{4.4}$$

$$\sum_j x_{kj} - \sum_i x_{ik} = 0 \qquad \forall k \neq s, t \text{ (Nós Intermediários)} \tag{4.5}$$

$$x_{ij} \geq 0 \tag{4.6}$$

Uma propriedade notável deste modelo, conhecida como **Total Unimodularidade** da matriz de restrições de fluxo em rede, garante que, se as ofertas e demandas (neste caso, 1 e -1) forem inteiras, a solução ótima do PPL (com $x_{ij} \geq 0$) será *automaticamente* inteira (0 ou 1). Portanto, não é necessário impor a restrição $x_{ij} \in \{0, 1\}$.

4.3 MÉTODOS DE RESOLUÇÃO ALGORÍTMICOS

Embora a formulação PPL seja academicamente correta, na prática, o PCM é resolvido por algoritmos de grafos muito mais eficientes. Os dois principais algoritmos SSSP são Dijkstra e Bellman-Ford.

4.3.1 O ALGORITMO DE DIJKSTRA

O Algoritmo de Dijkstra é uma abordagem "gulosa"(*greedy*) clássica para resolver o problema do caminho mais curto de origem única (*Single-Source Shortest Path*). Ele encontra o caminho de menor custo de um nó de origem s para todos os outros nós v em um grafo ponderado $G = (V, E)$.

Para sua execução, o algoritmo mantém duas estruturas de dados principais para cada nó $v \in V$:

- $d[v]$: A estimativa do custo (distância) do caminho mais curto de s até v.
- $\pi[v]$: O nó predecessor de v no caminho mais curto encontrado até o momento.

O algoritmo funciona mantendo um conjunto de nós "fechados"(cujo caminho mais curto já foi determinado) e um conjunto de nós "abertos"(ainda não visitados).

Passos Fundamentais do Algoritmo

1. **Inicialização:**
   - Para cada nó v no grafo, inicializa-se a distância como infinita e o predecessor como nulo:
   $$d[v] \leftarrow \infty, \quad \pi[v] \leftarrow \text{NULO}$$
   - Define a distância da origem s para ela mesma como zero:
   $$d[s] \leftarrow 0$$
2. **Fila de Prioridade (Q):** O conjunto de nós "abertos"Q é implementado como uma **fila de prioridade** (Min-Heap), que armazena todos os nós do grafo, priorizados pelo seu valor $d[v]$.
3. **Loop Principal (Processamento):** Enquanto a fila de prioridade Q não estiver vazia:
   (a) Extrai de Q o nó u com o **menor valor** $d[u]$. Este nó u é agora "fechado"(visitado), e sua distância $d[u]$ é considerada final.
   (b) Para cada vizinho v de u (ou seja, para cada aresta (u, v) com peso $w(u, v)$):

- Executa o passo de **Relaxação (Relaxation)** da aresta (u, v).

O Processo de Relaxação

O algoritmo realiza sucessivas "relaxações". A relaxação é o núcleo da operação e consiste em testar se o caminho até um nó v *passando por* u é melhor do que o caminho já conhecido até v.

Dado que o nó u acabou de ser fechado (com distância final $d[u]$), verificamos seus vizinhos v. Se o custo para chegar em u somado ao custo da aresta de u para v for menor que o custo atual para chegar em v, atualizamos a rota para v:

Condição de Relaxação:
Se $d[u] + w(u, v) < d[v]$, então:

$$d[v] \leftarrow d[u] + w(u, v) \quad \text{(Atualiza a distância)}$$
$$\pi[v] \leftarrow u \quad \text{(Define u como o novo predecessor de v)}$$

Ao atualizar $d[v]$, a posição de v na fila de prioridade Q é reajustada (operação `Decrease-Key`).

Restrição Crítica e Complexidade

Restrição Crítica: O Algoritmo de Dijkstra só funciona corretamente se o grafo **não possuir arestas com pesos negativos**.

A lógica "gulosa"do algoritmo baseia-se na premissa de que, uma vez que um nó u é extraído da fila (sendo o mais próximo de s entre os não visitados), seu caminho $d[u]$ é o mais curto definitivo. Se existisse uma aresta negativa em algum lugar do grafo, um caminho futuro (passando por essa aresta) poderia, eventualmente, encontrar um caminho total para u que fosse mais curto do que $d[u]$, invalidando a decisão "gulosa"já tomada. Para grafos com arestas de peso negativo (mas sem ciclos negativos), deve-se utilizar o Algoritmo de **Bellman-Ford**.

Complexidade: A eficiência do algoritmo depende da implementação da fila de prioridade Q.

- Usando um **array simples**, a complexidade é $O(V^2)$, pois encontrar o mínimo leva $O(V)$ e isso é feito V vezes.
- Usando uma **fila de prioridade (Heap Binário)**, a complexidade total é $O((V + E) \log V)$, ou $O(E \log V)$ para grafos conectados (onde $E \geq V - 1$).

4.3.2 O ALGORITMO DE BELLMAN-FORD

O Algoritmo de Bellman-Ford é uma abordagem robusta para o problema do caminho mais curto de origem única s, fundamentada em **Programação Dinâmica**. Em vez de ser "guloso"como Dijkstra, ele é mais "pessimista"e recalcula as distâncias iterativamente.

Assim como Dijkstra, ele mantém duas estruturas de dados para cada nó $v \in V$:

- $d[v]$: A estimativa do custo do caminho mais curto de s até v.
- $\pi[v]$: O nó predecessor de v no caminho mais curto.

Passos Fundamentais do Algoritmo

1. **Inicialização:**
   - A distância da origem s é zero. Todas as outras são infinitas.

$$d[s] \leftarrow 0$$
$$d[v] \leftarrow \infty, \quad \forall v \in V, v \neq s$$
$$\pi[v] \leftarrow \text{NULO}, \quad \forall v \in V$$

2. **Iterações e Relaxação (Núcleo da PD):**
   - O algoritmo "relaxa"**todas** as arestas do grafo por $V - 1$ iterações (onde V é o número de nós).
   - **Para i de 1 até $|V| - 1$:**
     - **Para cada aresta (u, v) com peso $w(u, v)$ em E:**
       - Executa a Relaxação:

Se $d[u] + w(u, v) < d[v]$, então:

$$d[v] \leftarrow d[u] + w(u, v)$$
$$\pi[v] \leftarrow u$$

A lógica da Programação Dinâmica aqui é que, após a k-ésima iteração do loop principal, o algoritmo garante ter encontrado o caminho mais curto para todos os nós v que utilizam, no máximo, k arestas. Como o maior caminho mínimo possível sem um ciclo tem, no máximo, $V - 1$ arestas, o loop de $V - 1$ iterações garante que o caminho mínimo final (se não houver ciclos negativos) seja encontrado.

Detecção de Ciclos Negativos

Sua principal vantagem é que ele **funciona com arestas de pesos negativos**. Além disso, ele é capaz de **detectar ciclos negativos** — um loop no grafo cujo custo total é

negativo.

A detecção ocorre em uma etapa final, após o término das $V - 1$ iterações:

3. **Verificação de Ciclos Negativos:**
    - O algoritmo realiza uma V-ésima passagem por todas as arestas.
    - **Para cada aresta** (u, v) **em** E**:**
        - Se $d[u] + w(u, v) < d[v]$:
        - **Retorne FALSO** (um ciclo negativo foi detectado).
    - Se o loop terminar sem encontrar relaxações, **Retorne VERDADEIRO** (não há ciclos negativos).

Se, após $V-1$ iterações (onde todos os caminhos simples já deveriam estar resolvidos), ainda for possível "relaxar"uma aresta na V-ésima passagem, isso prova que essa aresta faz parte ou é alcançável por um ciclo de custo negativo. Esse ciclo permitiria que o custo do caminho diminuísse indefinidamente (indo para $-\infty$), pois se pode percorrê-lo quantas vezes for necessário.

Complexidade

A complexidade do Algoritmo de Bellman-Ford é fácil de analisar:

- O loop principal executa $O(|V|)$ vezes (especificamente, $|V| - 1$ vezes).
- Dentro de cada iteração, ele verifica todas as arestas, o que leva $O(|E|)$ tempo.
- A verificação de ciclo negativo também leva $O(|E|)$ tempo.

A complexidade total do algoritmo é, portanto, $O(|V| \cdot |E|)$. Isso o torna mais lento que o Dijkstra (que é $O(E \log V)$), mas é o preço pago por sua capacidade de lidar com pesos negativos e detectar ciclos negativos.

4.3.3 COMPARATIVO DOS MÉTODOS

A escolha do algoritmo depende fundamentalmente da presença de custos negativos no grafo, conforme detalhado na Tabela 4.1.

Em resumo, Dijkstra é significativamente mais rápido e preferível para a maioria das aplicações (como GPS, onde o tempo/distância nunca são negativos). Bellman-Ford é mais robusto e "seguro"para problemas onde custos negativos são uma possibilidade.

Tabela 4.1: Comparativo: Dijkstra vs. Bellman-Ford

Característica	**Algoritmo de Dijkstra**	**Algoritmo de Bellman-Ford**
Abordagem	Gulosa (Greedy)	Programação Dinâmica
Pesos Negativos?	**Não** (falha)	**Sim**
Ciclos Negativos?	Não (não detecta)	**Sim** (detecta)
Complexidade	$O(E \log V)$ ou $O(V^2)$	$O(V \cdot E)$
Quando usar?	Grafos sem pesos negativos (Ex: GPS, redes OSPF)	Grafos com pesos negativos (Ex: Detecção de arbitragem)

CAPÍTULO 5

CONCLUSÃO

Este trabalho explorou três problemas centrais da Pesquisa Operacional: Transporte, Atribuição e Menor Caminho. Foi demonstrado que, embora todos possam ser modelados como Problemas de Programação Linear, suas estruturas únicas levaram ao desenvolvimento de algoritmos especializados que superam em muito a eficiência de métodos genéricos como o Simplex.

O Problema da Atribuição é eficientemente resolvido pelo Algoritmo Húngaro. O Problema do Menor Caminho, por sua vez, depende das características do grafo: o Algoritmo de Dijkstra é a escolha padrão pela sua rapidez, mas cede lugar ao Algoritmo de Bellman-Ford na presença de custos negativos.

A compreensão desses modelos e de seus respectivos métodos de solução é fundamental para qualquer profissional de engenharia, ciência da computação ou administração, permitindo a otimização eficiente de recursos em sistemas logísticos, redes e alocações de tarefas.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] ARENALES, M. N. et al. **Pesquisa operacional**. 2. ed. Rio de Janeiro: Elsevier, 2015.

[2] BAZARAA, M. S.; JARVIS, J. J.; SHERALI, H. D. **Linear Programming and Network Flows**. New York: John Wiley & Sons, 1990.

[3] BELFIORE, Patrícia; FÁVERO, Luiz Paulo. **Pesquisa operacional para cursos de engenharia**. Rio de Janeiro: Elsevier Brasil, 2013.

[4] BYJU'S. Vogel's Approximation Method. Disponível em: `https://byjus.com/maths/vogels-approximation-method/`.

[5] GOLDBARG, M. C.; LUNA, H. P. L. **Otimização Combinatória e Programação Linear: Modelos e Algoritmos**. Rio de Janeiro: Campus, 2000.

[6] MACAMBIRA, A. F. U. S.; MACULAN, N. et al. **Programação linear**. João Pessoa: Editora da UFPB, 2016.

[7] OLIVEIRA, Jéssica Moia de. O problema de transporte. 2016. Projeto Supervisionado II (Graduação em Matemática, Estatística e Computação Científica) – Universidade Estadual de Campinas, Instituto de Matemática, Estatística e Computação Científica, Campinas, 2016. Orientador: Aurelio Ribeiro Leite de Oliveira.

[8] TAHA, Hamdy A. **Operations research: an introduction**. 10. ed. Global Edition. Harlow: Pearson Education, 2017.