Trabalho 2 - Problema dos Caminhos Mínimos

Igor J. Rodrigues; Leonardo S. Coradeli; Lucas V. C. Ikeda; Marco V. M. Faria

Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciência e Tecnologia

12 de novembro de 2025

unesp

Sumário

1. Introdução e Definição do Problema
2. Modelo Matemático (PL)
3. Métodos de Resolução
4. Referências Bibliográficas

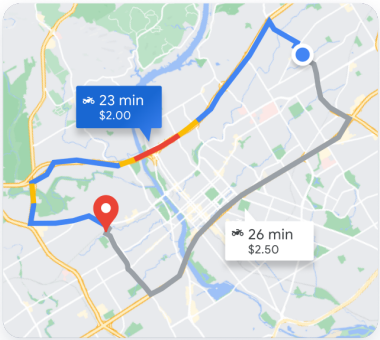
unesp

Qual é o melhor caminho?

A Pergunta Intuitiva

Como o Google Maps sabe a rota mais rápida para chegar na FCT?

- É o caminho mais rápido (menor tempo)?
- É o caminho mais curto (menor distância)?
- É o caminho mais barato (sem pedágios)?



A ideia central é modelar o mapa como um **grafo** e atribuir **custos** às ruas.

unesp

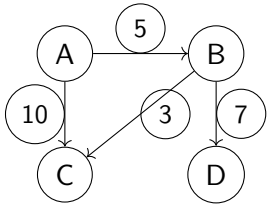
Onde encontrar o PCM

- **Redes de Computadores:** Roteamento de pacotes na internet (Protocolo OSPF) — o custo é a latência.
- **Logística e Transporte:** Otimização de rotas de entrega (Correios, Mercado Livre) — o custo é a distância ou tempo.
- **Finanças (Avançado):** Detecção de oportunidades de arbitragem em mercados (encontrar ciclos de custo negativo).
- **Análise de Redes Sociais:** Medir o "grau de separação"entre duas pessoas.

unesp

Traduzindo o Mapa para a Matemática

- **Grafo (ou Dígrafo):** $G = (V, E)$
- V **(Vértices):** O conjunto de "nós" (ex: cidades, roteadores).
- E **(Arestas):** O conjunto de "conexões" (ex: ruas, cabos).
- **Grafo Ponderado:** Cada aresta $(i, j) \in E$ possui um **peso** (ou custo) w_{ij} associado.



unesp

O Problema de Caminhos Mínimos

- **Caminho:** Uma sequência de arestas que conecta um nó origem s a um nó destino t.
  - Ex: $P = (A \rightarrow B \rightarrow D)$
- **Custo do Caminho:** A soma dos pesos w_{ij} das arestas que formam o caminho.
  - Ex: $Custo(P) = w_{AB} + w_{BD}$

Intuição

Queremos encontrar o caminho mais "barato" ou "curto" entre dois pontos em um grafo.

unesp

Formulação Matemática do Problema

O Grande Objetivo

Dado um grafo ponderado G, um nó origem s e um nó destino t, encontrar o caminho P de s para t que **minimiza o custo total**.

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

Observação Importante

Os algoritmos que vamos estudar (Dijkstra, Bellman-Ford) resolvem o problema de **Origem Única (SSSP)**: encontram o caminho mínimo de s para **TODOS** os outros nós.

unesp

Modelo PL: A Abordagem Genérica

- **Ideia:** Modelar como um **Problema de Fluxo** para enviar 1 unidade da origem s ao destino t.

unesp

Modelo PL: A Abordagem Genérica

- **Ideia:** Modelar como um **Problema de Fluxo** para enviar 1 unidade da origem s ao destino t.

Variáveis de Decisão

$$x_{ij} = \begin{cases} 1, & \text{se o arco } (i,j) \text{ for usado no caminho;} \\ 0, & \text{caso contrário.} \end{cases}$$

Modelo PL: A Abordagem Genérica

- **Ideia:** Modelar como um **Problema de Fluxo** para enviar 1 unidade da origem s ao destino t.

Variáveis de Decisão

$$x_{ij} = \begin{cases} 1, & \text{se o arco } (i,j) \text{ for usado no caminho;} \\ 0, & \text{caso contrário.} \end{cases}$$

Função Objetivo (Minimizar o Custo Total)

Minimizar a soma dos custos de todas as arestas usadas:

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

unesp

Modelo PL: Restrições de Conservação de Fluxo

Restrições

- **Origem (s):** Fluxo que **sai** $= 1$

$$\sum_{j:(s,j)\in E} x_{sj} = 1$$

Modelo PL: Restrições de Conservação de Fluxo

Restrições

- **Origem (s):** Fluxo que **sai** $= 1$

$$\sum_{j:(s,j)\in E} x_{sj} = 1$$

- **Nós intermediários (k):** Fluxo que **entra** $=$ Fluxo que **sai**

$$\sum_{i:(i,k)\in E} x_{ik} - \sum_{j:(k,j)\in E} x_{kj} = 0$$

Modelo PL: Restrições de Conservação de Fluxo

Restrições

- **Origem (s):** Fluxo que **sai** $= 1$

$$\sum_{j:(s,j)\in E} x_{sj} = 1$$

- **Nós intermediários (k):** Fluxo que **entra** $=$ Fluxo que **sai**

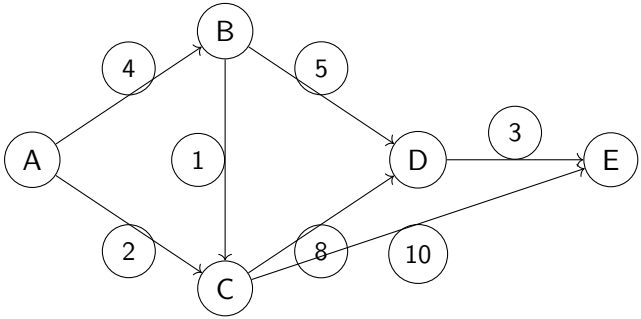
$$\sum_{i:(i,k)\in E} x_{ik} - \sum_{j:(k,j)\in E} x_{kj} = 0$$

- **Destino (t):** Fluxo que **entra** $= 1$

$$\sum_{i:(i,t)\in E} x_{it} = 1$$

Exemplo de Modelagem: Grafo Logístico

- Vamos modelar o problema de achar a rota de custo mínimo da Origem **A** para o Destino **E**.



Custos (w_{ij}):

- $w_{AB} = 4$
- $w_{AC} = 2$
- $w_{BC} = 1$
- $w_{BD} = 5$
- $w_{CD} = 8$
- $w_{CE} = 10$
- $w_{DE} = 3$

unesp

Exemplo de Modelagem: O PPL Completo

Função Objetivo (Minimizar Z):

$$\min Z = 4x_{AB} + 2x_{AC} + 1x_{BC} + 5x_{BD} + 8x_{CD} + 10x_{CE} + 3x_{DE}$$

Sujeito a (Conservação de Fluxo):

$$\begin{aligned}
x_{AB} + x_{AC} &= 1 && \text{(Nó A: Origem)} \\
x_{BC} + x_{BD} - x_{AB} &= 0 && \text{(Nó B: Intermediário)} \\
x_{CD} + x_{CE} - x_{AC} - x_{BC} &= 0 && \text{(Nó C: Intermediário)} \\
x_{DE} - x_{BD} - x_{CD} &= 0 && \text{(Nó D: Intermediário)} \\
x_{CE} + x_{DE} &= 1 && \text{(Nó E: Destino)} \\
x_{ij} &\in \{0,1\}, \quad \forall (i,j)
\end{aligned}$$

Interpretação: Cada variável x_{ij} indica se o arco (i,j) faz parte do caminho mínimo (1) ou não (0). O modelo garante que o fluxo parte de A, passa por nós intermediários e chega em E com o menor custo total.

unesp

Modelo PL: A Mágica da Integridade

- **Questão:** É necessário impor explicitamente que as variáveis x_{ij} assumam apenas valores inteiros (0 ou 1)?

$$x_{ij} \in \{0, 1\} \quad \text{(Programação Inteira)}$$

- **Observação:** Para este tipo de problema de rede, a imposição de integralidade não é necessária.
- Modelos de **Fluxo de Custo Mínimo** — como o problema de caminho mínimo — possuem uma estrutura matricial denominada **Totalmente Unimodular**.
- **Consequência:** Quando a matriz de coeficientes do sistema de restrições é totalmente unimodular e os termos do lado direito (ofertas e demandas) são inteiros, o **problema de Programação Linear relaxado** (isto é, considerando apenas $x_{ij} \geq 0$) apresenta uma **solução ótima inteira**.
- **Em síntese:** A propriedade de total unimodularidade garante a integridade natural da solução, tornando desnecessária a formulação explícita como um problema de programação inteira.

unesp

- Embora o modelo de PL seja academicamente correto, ele não é a forma mais *eficiente* de resolver o Problema de Caminhos Mínimos na prática.
- Na prática, usamos algoritmos especializados em grafos que são muito mais rápidos.
- Vamos focar em dois algoritmos fundamentais:
  1. **Algoritmo de Dijkstra**
  2. **Algoritmo de Bellman-Ford**

unesp

Algoritmo de Dijkstra

O Conceito

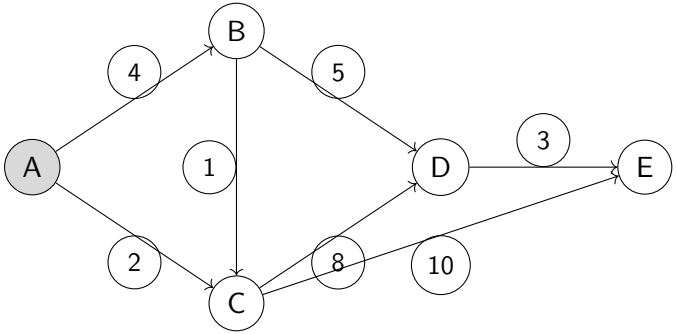
Uma abordagem "Gulosa"(Greedy). A cada passo, ele escolhe o caminho que *parece* ser o melhor (o mais curto) e expande a partir dele.

Restrição Crítica

O Algoritmo de Dijkstra **NÃO FUNCIONA** se o grafo tiver arestas com **pesos negativos**.

unesp

Exercício: Algoritmo de Dijkstra (Inicialização)



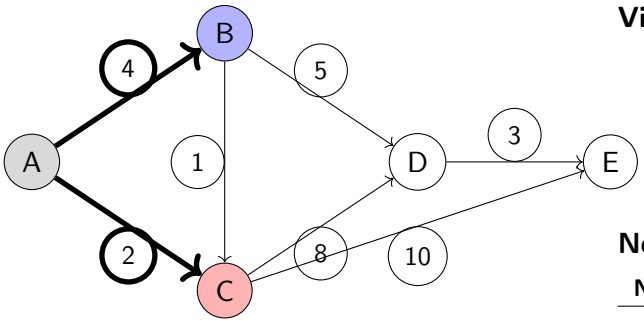
Nós Visitados: {}

Nó	dist[i]	prev[i]
A	**0**	-
B	∞	-
C	∞	-
D	∞	-
E	∞	-

Próximo nó: A (custo 0)

unesp

Exercício: Dijkstra - Iteração 1 (Nó A)



Visitando Nó: A (dist = 0)

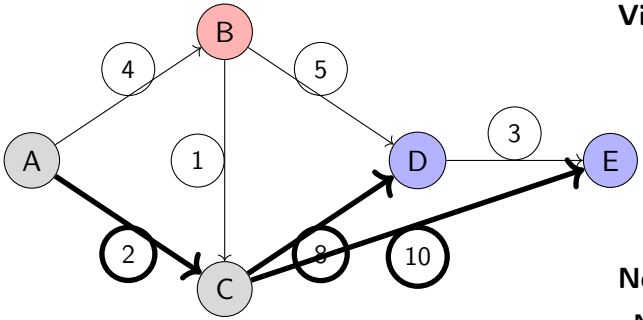
- Vizinho B: $0 + 4 = 4 \rightarrow$ Atualiza dist[B]=4, prev[B]=A
- Vizinho C: $0 + 2 = 2 \rightarrow$ Atualiza dist[C]=2, prev[C]=A

Nós Visitados: {**A**}

Nó	dist[i]	prev[i]
A	0	-
B	**4**	A
C	**2**	A
D	∞	-
E	∞	-

Próximo nó: C (menor dist = 2)

Exercício: Dijkstra - Iteração 2 (Nó C)



Visitando Nó: C (dist = 2)

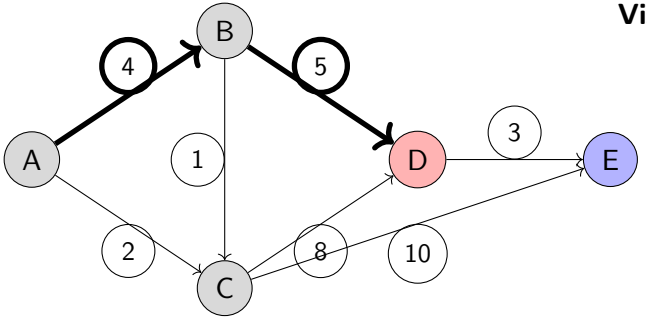
- Vizinho D: $2 + 8 = 10 \rightarrow$ Atualiza dist[D]=10, prev[D]=C
- Vizinho E: $2 + 10 = 12 \rightarrow$ Atualiza dist[E]=12, prev[E]=C

Nós Visitados: {**A, C**}

Nó	dist[i]	prev[i]
A	0	-
B	4	A
C	2	A
D	**10**	C
E	**12**	C

Próximo nó: B (menor dist = 4)

Exercício: Dijkstra - Iteração 3 (Nó B) - A "Relaxação"



Visitando Nó: B (dist = 4)

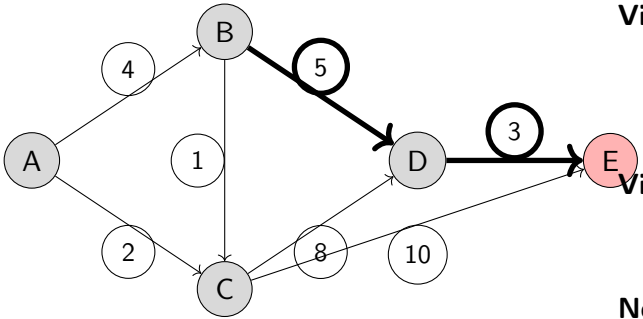
- Vizinho C: já visitado.
- Vizinho D: $4 + 5 = 9$.
- O custo atual de D é 10.
  Como 9 ¡ 10, atualizamos!
- Atualiza dist[D] = 9, prev[D] = B

Nós Visitados: {**A, C, B**}

Nó	dist[i]	prev[i]
A	0	-
B	4	A
C	2	A
D	**9**	**B**
E	12	C

unesp

Exercício: Dijkstra - Iterações 4 e 5 (Nós D, E)



Visitando Nó: D (dist=9)

- Vizinho E: $9 + 3 = 12$.
- O custo atual de E é 12. Como $12 \not< 12$, **não há atualização**.

Visitando Nó: E (dist=12)

- Fim do algoritmo (destino alcançado / fila vazia).

Nós Visitados: {**A, C, B, D, E**}

Nó	dist[i]	prev[i]
A	0	-
B	4	A
C	2	A
D	9	B
E	12	C

unesp

Exercício: Dijkstra — Conclusão e Caminho

Custos Mínimos a partir de A

Nó	Custo (dist)
B	4
C	2
D	9
E	12

Caminho Mínimo (Backtracking)

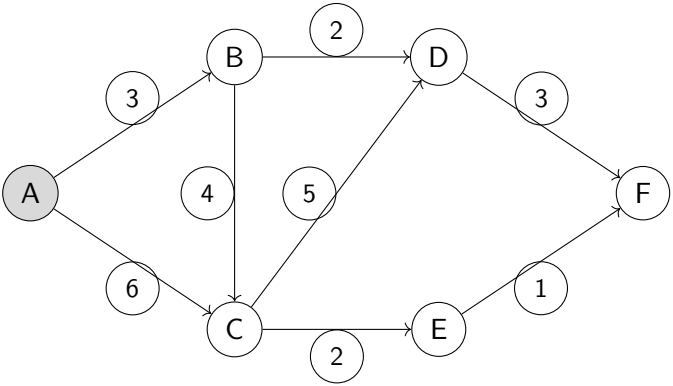
- **E** $\leftarrow$ C
- **C** $\leftarrow$ A
- Chegamos à origem: **A**

Caminho: $A \rightarrow C \rightarrow E$

Custo Total: $2 + 10 = 12$

unesp

Exercício: Algoritmo de Dijkstra (Inicialização)



Nós Visitados: {}

Nó	dist[i]	prev[i]
A	**0**	-
B	∞	-
C	∞	-
D	∞	-
E	∞	-
F	∞	-

Próximo nó: A (custo 0)

unesp

Algoritmo de Bellman-Ford

Ideia Geral

O algoritmo de **Bellman-Ford** segue uma abordagem de **Programação Dinâmica**. Diferente do Dijkstra, que é **guloso**, o Bellman-Ford é **cauteloso**: ele reavalia continuamente todas as arestas, garantindo o custo mínimo mesmo com pesos negativos.

Principais Vantagens

- **Funciona com pesos negativos.**
- **Detecta ciclos negativos**, isto é, caminhos que reduzem o custo infinitamente.

Ideia Principal

Repete o processo de **relaxar todas as arestas** do grafo exatamente $|V| - 1$ vezes, onde $|V|$ é o número de vértices.

Algoritmo de Bellman-Ford — Funcionamento

Por que repetir $|V| - 1$ vezes?

O maior caminho possível sem formar um ciclo contém, no máximo, $|V| - 1$ arestas. Após essas iterações, todos os menores caminhos já terão sido encontrados.

Detecção de Ciclos Negativos

Após as $|V| - 1$ passagens, o algoritmo faz uma **última verificação**:

- Se ainda for possível "relaxar" alguma aresta, significa que existe um **ciclo negativo**.
- Nesse caso, o algoritmo emite um alerta: **"Ciclo Negativo Detectado!"**

Aplicação Prática

Em finanças, é usado para detectar **arbitragem**: *converter moedas em sequência (A $\rightarrow$ B $\rightarrow$ C $\rightarrow$ A) e terminar com lucro.*

Comparativo: Dijkstra vs. Bellman-Ford

Característica	**Dijkstra**	**Bellman-Ford**
Abordagem	Gulosa (Greedy)	Programação Dinâmica
Pesos Negativos	x Não funciona	V Suportado
Detecta Ciclos Negativos	x Não	V Sim
Complexidade	$O(E \log V)$ ou $O(V^2)$	$O(V \cdot E)$
Uso Típico	Grafos sem pesos negativos	Grafos com pesos negativos
Exemplo	GPS, OSPF (redes)	Arbitragem financeira

Conclusão

Dijkstra é mais rápido e eficiente para grafos positivos. **Bellman-Ford**, embora mais lento, é mais **robusto e confiável** em cenários complexos.

Referências Bibliográficas

ARENALES, M. N. *et al.* **Pesquisa operacional**. 2. ed. Rio de Janeiro: Elsevier, c2015.

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

BELFIORE, Patrícia; FÁVERO, Luiz Paulo. **Pesquisa operacional para cursos de engenharia**. Rio de Janeiro: Elsevier Brasil, 2013. ISBN 978-85-352-6335-0.

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

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

unesp