

Algoritmos para r -Arborescências Geradoras Mínimas em Digrafos: Uma Aplicação Web Interativa

Lorena Sampaio, Samira Haddad
Orientador: Prof. Dr. Mário Leston Rey

Universidade Federal do ABC
Centro de Matemática, Computação e Cognição

26 de novembro de 2025

Sumário

- 1 Introdução
- 2 Algoritmo de Chu-Liu-Edmonds
- 3 Algoritmo de András Frank
- 4 Resultados Experimentais
- 5 Aplicação Web
- 6 Conclusões

O Problema

Encontrar uma r -Arborescência Geradora de Custo Mínimo

Dado um r -digrafo ponderado (D, w, r) :

- Encontrar uma r -arborescência geradora de custo mínimo de D

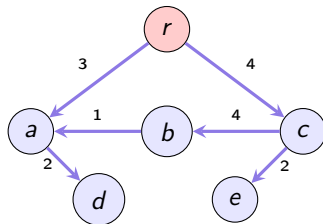
Algoritmos estudados:

- 1 Chu-Liu-Edmonds (1965-67)
- 2 András Frank (1981-2014)

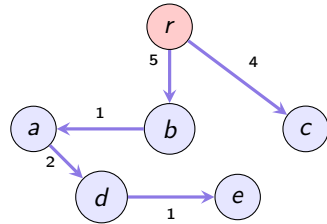
Exemplo: r -Arborescência Geradora Mínima



Digrafo Original

 r -Arborescência Geradora

Custo: 16



Geradora Mínima

Custo: 13

Chu-Liu-Edmonds

Algoritmo Recursivo: dado um r -digrafo ponderado (D, w, r)

$\text{chu-liu-edmonds}((D, w, r))$:

- 1 **Reduzir custos**: para cada vértice $v \neq r$, subtrair $\lambda(v) = \min\{w(a) : a \in \delta^-(v)\}$
- 2 **Construir** D_0 : escolhendo um arco a_v de custo reduzido zero para cada $v \neq r$
- 3 **Verificar**: se D_0 é uma r -arborescência \Rightarrow **devolver** D_0
Caso contrário:
- 4 **Contração**: encontrar ciclo C em D_0 e contrair
- 5 **Chamada recursiva**: Seja $D' = D/C$ e $w' = w_\lambda/C$. Calcular $T' = \text{chu-liu-edmonds}(D', w', r)$
- 6 **Devolver**: expandir(T')

Escolha Gulosa

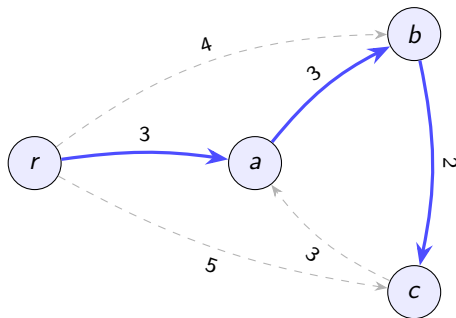
Definição:

Para cada $v \neq r$, escolher um arco a_v de custo mínimo que entra em v :

$$T := \{a_v : v \in V \setminus \{r\}\}$$

Propriedade:

Se T é uma r -arborescência, então T tem custo mínimo.



Resultado

$T = \{(r, a), (a, b), (b, c)\}$ é uma r -arborescência de custo mínimo!

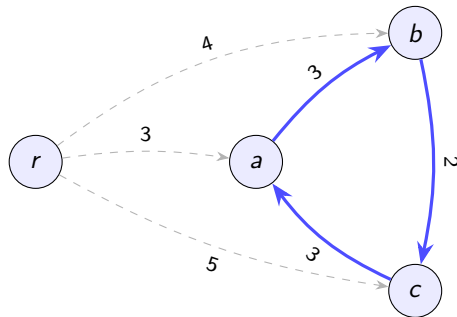
E quando a escolha gulosa falha?

Problema:

A escolha gulosa pode produzir um conjunto T que *não* é uma r -arborescência.

Exemplo:

Os arcos de custo mínimo formam um ciclo (a, b, c, a) sem alcançar r .



Arcos azuis formam um **ciclo**!

Passo 1: Redução de Custos

Definição:

Para cada $v \in V \setminus \{r\}$:

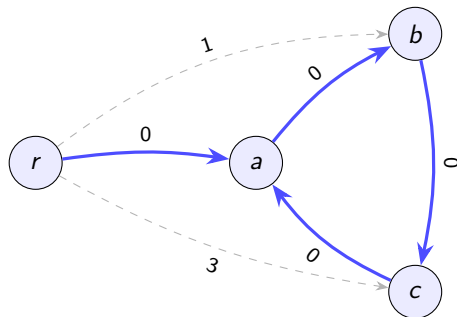
$$\lambda(v) := \min\{w(a) : a \in \delta^-(v)\}$$

Custo λ -reduzido:

$$w_\lambda(uv) := w(uv) - \lambda(v)$$

Valores de λ :

- $\lambda(a) = 3, \lambda(b) = 3, \lambda(c) = 2$



Arcos do ciclo têm custo zero!

Arcos com custo zero são candidatos para D_0

Passo 2: Construção de D_0

Formação de D_0 :

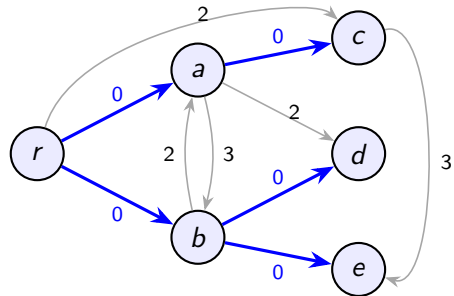
Para cada $v \neq r$, escolher um arco $a_v \in \delta^-(v)$
com $w_\lambda(a_v) = 0$

Formar:

$$D_0 := (V, \{a_v : v \in V \setminus \{r\}\})$$

Arcos escolhidos:

- $(r, a), (r, b)$
- $(a, c), (b, d), (b, e)$



D_0 é uma r -arborescência!

Neste exemplo, $D_0 = \{(r, a), (r, b), (a, c), (b, d), (b, e)\}$ forma uma r -arborescência.

Contração de Ciclos

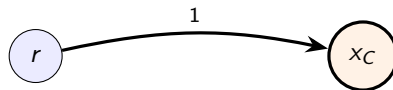
Operação:

Contrair ciclo C em supervértice x_C .

Novo problema:

(D', w', r) onde:

- $D' := D/C \mapsto x_C$
- $w' := w_\lambda/C \mapsto x_C$



Digrafo contraído D'

Recursão:

Resolver problema no digrafo menor recursivamente.

Propriedade

Uma solução ótima em D' pode ser expandida para uma solução ótima em D .

UFABC

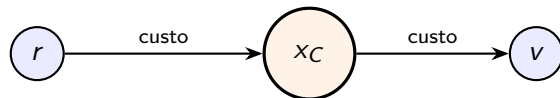
Passo 3: Contração de Ciclos

Quando D_0 tem ciclo:

Se após redução de custos, D_0 contém um ciclo C , contraímos C em supervértice x_C

Em outras palavras: todos os vértices de C viram um único vértice

Custos são ajustados usando os custos λ -reduzidos



Chamada Recursiva

Resolvemos $\text{chu-liu-edmonds}(D/C \mapsto x_C, w_\lambda/C \mapsto x_C, r)$

Em outras palavras: aplicamos o mesmo algoritmo no digrafo menor

UFABC

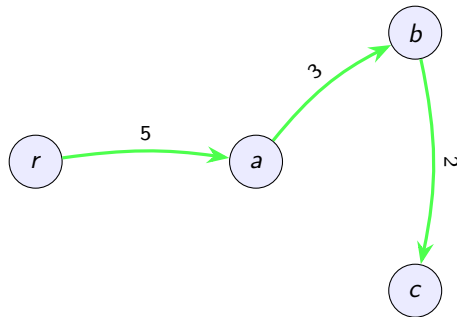
Reexpansão da Solução

Dado: T' ótima em (D', w')

Construir: T ótima em (D, w)

Procedimento:

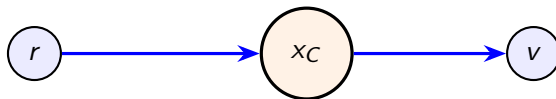
- 1 Seja uv o arco de D correspondente ao arco ux_C de T'
- 2 Incluir uv em T
- 3 Incluir todos os arcos de C exceto aquele que entra em v



r -arborescência final no digrafo original

Resultado: T é uma r -arborescência de custo mínimo

Passo 4: Chamada Recursiva



O algoritmo devolve T' (arborescência no digrafo contraído)

Em outras palavras: encontramos a r -arborescência ótima no grafo contraído

Próximo passo: expandir T' de volta para obter T no digrafo original

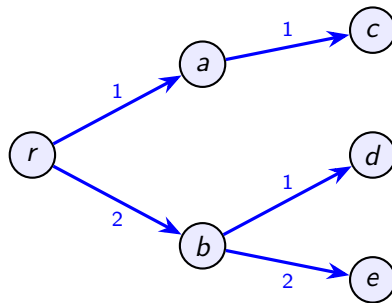
Solução Final – Exemplo Original

Para nosso exemplo inicial:

A escolha gulosa produziu uma r -arborescência diretamente!

Resultado:

- Não há ciclo
- Todos vértices alcançáveis de r
- Cada vértice tem grau de entrada 1
- Solução ótima encontrada!



Custo total: $1 + 2 + 1 + 1 + 2 = 7$

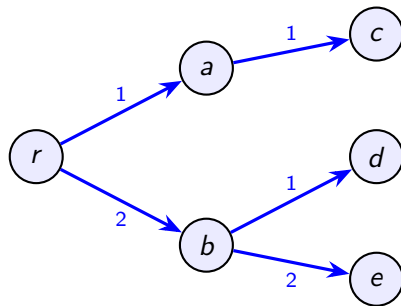
Solução Final – Exemplo Original

Para nosso exemplo:

Como D_0 já é uma r -arborescência, o algoritmo termina diretamente!

Em outras palavras:

- Não há ciclo
- Todos vértices alcançáveis de r
- Cada vértice tem grau de entrada 1
- Solução ótima encontrada!



Solução Ótima

Custo total: $1 + 2 + 1 + 1 + 2 = 7$

András Frank: Visão Geral

Abordagem em Duas Fases

Fase I: Construir cobertura de subconjuntos minimais via redução de custos

Fase II: Extrair arborescência da cobertura

Diferencial:

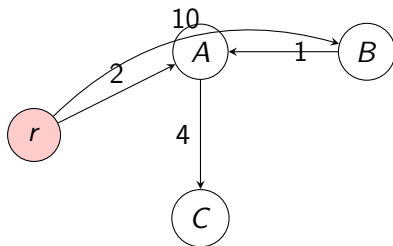
- Trabalha com múltiplos vértices simultaneamente
- Usa componentes fortemente conexas
- Redução sistemática de custos

Complexidade:

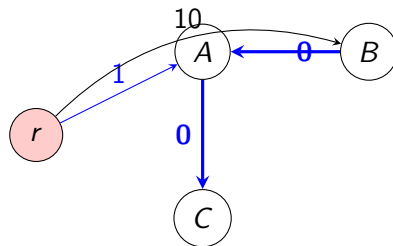
- Fase I: $O(nm)$
- Fase II v1 (lista): $O(n^2)$
- Fase II v2 (heap): $O(n \log n)$

Fase I: Redução de Custos

Para cada vértice $v \neq r$: subtrair o mínimo de entrada



Original



Após Redução

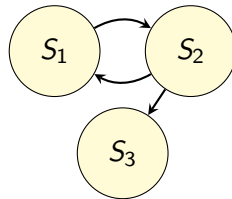
Arcos com custo **zero** formam o digrafo D_0

Fase I: Componentes Fortemente Conexas

Identificar componentes fortemente conexas (CFCs) em D_0

Cada CFC forma um **subconjunto minimal**

Construir sequência laminar de subconjuntos



Condição de Otimalidade

Sequência λ satisfaz: $|\delta^-(X)| = 1$ para cada X em λ

Fase II: Construção da Arborescência

Objetivo: Extrair arborescência de D_0 respeitando λ

- 1 Iniciar com conjunto $R = \{r\}$
- 2 Para cada v fora de R :
 - Selecionar arco (u, v) com $u \in R$ e custo reduzido zero
 - Adicionar v a R
- 3 Repetir até incluir todos os vértices

Resultado

Arborescência ótima com mesma solução: custo 14

Comparação de Desempenho

Experimentos: 2000 digrafos aleatórios, $|V| \in [101, 4996]$

Algoritmo	Tempo Mediano	Tempo Médio
Chu-Liu-Edmonds	0,25 s	0,58 s
Frank Fase I	8,93 s	12,40 s
Frank Fase II (lista)	0,98 s	1,34 s
Frank Fase II (heap)	0,016 s	0,020 s

Speedup Fase II

Heap vs Lista: aceleração de **58,12 vezes** (mediana)

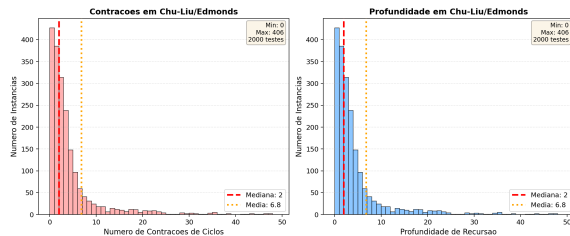
Características Estruturais

Contrações (Chu-Liu):

- Mediana: 2 contrações
- Média: 6,82
- Máximo: 406
- 93,8% com < 20

Muito abaixo do limite teórico $O(n)$

Consumo de memória: mediana 11,5 MB (Fase I)



Motivação Didática

Desafio

Algoritmos de grafos são **abstratos** e **difíceis de visualizar**

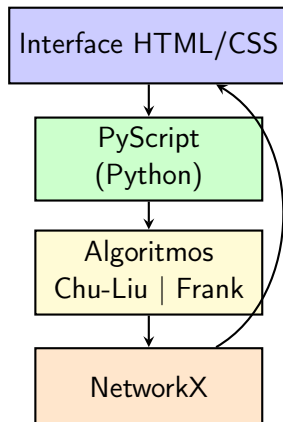
Solução Proposta:

- Visualização interativa
- Execução passo a passo
- Feedback imediato
- Acessível via navegador

Tecnologias:

- PyScript (Python no browser)
- JavaScript
- HTML5/CSS3
- NetworkX

Arquitetura da Aplicação



Interface: Página Principal

**ArboGraph**

 Home

 Chu-Liu-Edmonds

 András Frank (V1)

 András Frank (V2)

 Desenhe um digrafo

 Nossa dissertação

**Dúvidas ?**

Quer aprender mais sobre esses algoritmos, leia nossa tese :)

[Link](#)

Algoritmos para o problema da arborescência geradora mínima: uma aplicação didática interativa

Resumo

Este trabalho investiga e implementa algoritmos de busca de uma r -arborescência geradora mínima em digrafos. A partir da formulação clássica e da literatura de Chu-Liu-Edmonds e também da formulação de András Frank, desenvolvemos uma aplicação web que permite: (i) desenhar ou importar um digrafo ponderado, (ii) escolher o nó raiz r , (iii) executar o algoritmo passo a passo com visualização das contrações, seleção de arcos de custo mínimo e reconstrução da arborescência, e (iv) exportar resultados e logs. A solução combina PyScript e NetworkX para a lógica algorítmica, Cytoscape para edição e visualização interativa, e Tailwind/Flowbite na interface. Como contribuição, o sistema oferece um ambiente didático que torna transparentes as decisões do algoritmo e facilita a análise e comparação de soluções em diferentes instâncias, apoiando ensino, experimentação e validação.


Integrantes do Projeto







Interface: Desenho de Grafos


ArboGraph

Home

Chu-Liu/Edmonds

Andras Frank (V1)

Andras Frank (V2)

Desenhe um grafo

Nossa tese

Dúvidas ?

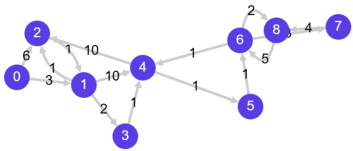
Quer aprender mais sobre esses algoritmos, leia nossa tese :)

Link

Desenhe seu grafo

1. Desenhe um grafo, [carregue](#) um exemplo ou [importe](#) um grafo já existente.


Grafo Original



Funcionalidades:

- Adicionar vértices e arestas
- Definir pesos

Interface: Chu-Liu-Edmonds


ArboGraph

Home

Chu-Liu/Edmonds

Andras Frank (V1)

Andras Frank (V2)

Desenhe um grafo

Nossa tese

Chu-Liu / Edmonds

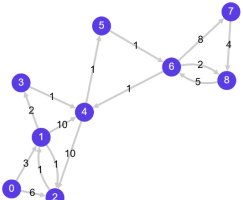
1 Crie um grafo
Desenhe um grafo, [carregue um exemplo](#) ou [importe um grafo](#) já existente.

2 Escolha o nó raiz

3 Execute o algoritmo

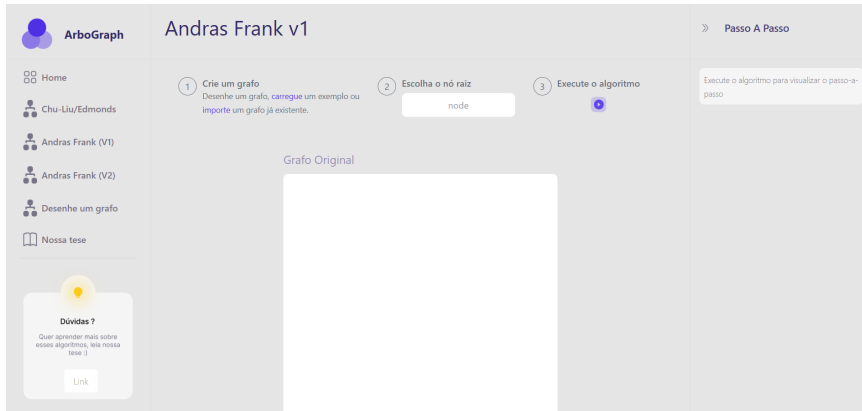
Execute o algoritmo para visualizar o passo-a-passo

Grafo Original



- Visualização passo a passo
- Destacamento de ciclos detectados
- Log detalhado das operações

Interface: András Frank



- Exibição das duas fases
- Visualização de CFCs
- Comparação entre versões (lista vs heap)

Princípios de Design

Teoria dos Registros de Representação (Duval)

Transitar entre diferentes representações:

- **Visual:** diagramas do grafo
- **Simbólico:** código Python
- **Textual:** log das operações

Feedback Imediato

Validação em tempo real das operações do usuário

Contribuições do Trabalho

1 Implementação completa de dois algoritmos clássicos

- Chu-Liu-Edmonds: recursivo com contração
- András Frank: duas fases com otimização heap

2 Análise experimental detalhada

- 2000 instâncias aleatórias
- Comparação de desempenho e características estruturais

3 Aplicação web interativa

- Ferramenta didática para visualização
- Execução passo a passo dos algoritmos
- Design centrado no usuário

Principais Resultados

- **Corretude validada:** custos idênticos em todas as instâncias
- **Chu-Liu-Edmonds** mais rápido para construção direta
 - Mediana: 0,25 s vs 8,93 s (Fase I Frank)
- **Otimização heap** fundamental na Fase II
 - Speedup: 58× (mediana), 61× (média)
- **Comportamento prático** muito melhor que limites teóricos
 - Contrações: mediana 2 (limite $O(n)$)
 - Memória modesta: 11,5 MB

Trabalhos Futuros

Extensões Possíveis

- Implementar outras variantes (Tarjan, Gabow)
- Análise em grafos com estruturas especiais
- Paralelização dos algoritmos
- Extensão para grafos dinâmicos

Melhorias na Aplicação

- Modo de edição visual de grafos
- Geração automática de casos de teste
- Exercícios interativos com correção automática
- Integração com plataformas de ensino (Moodle, Jupyter)

Obrigado!

Perguntas?

<https://github.com/lorenypsum/graph-visualizer>