Análise e Implementação de Algoritmos de Busca de uma r-Arborescência Inversa de Custo Mínimo em Grafos Dirigidos com Aplicação Didática Interativa

Orientador: Mário Leston
Discentes: Lorena Silva Sampaio, Samira Haddad

21 de setembro de 2025

1 Introdução

Encontrar uma *r-arborescência inversa de custo mínimo* em grafos dirigidos é um problema estudado em ciência da computação desde os anos 1960, com formulações fundamentais apresentadas por Jack Edmonds em 1967 [2].

Essa busca dialoga com um princípio formulado na Idade Média por Guilherme de Ockham: a navalha de Occam (princípio da parcimônia), uma heurística filosófica segundo a qual, entre explicações concorrentes para um fenômeno, devemos preferir a mais simples ou a que faz menos suposições.

Podemos pensar na navalha de Occam como critério de escolha entre explicações por meio de uma *teia explicativa mínima*: uma estrutura que conecta fatos ou observações com o mínimo de relações explicativas necessárias. Quando tais relações envolvem dependência ou causalidade, podemos representá-las pictograficamente como setas direcionadas entre os fatos (as hipóteses aparecem como rótulos dessas setas). Para refinar o modelo, associamos um custo a cada relação (por exemplo, o esforço para validar a relação ou a complexidade da explicação).

Encontrar a *teia explicativa mínima* equivale, nessa metáfora, a encontrar uma *r-arborescência de custo mínimo*: fixamos um vértice raiz r (a explicação inicial) e escolhemos um conjunto mínimo de relações explicativas de modo que todos os fatos tenham um caminho dirigido que leve a r, minimizando o custo total das arestas.

A *r-arborescência* (também chamada *out-arborescência*) orienta as arestas para fora de r: cada vértice $v \neq r$ tem exatamente uma aresta de entrada, e há um caminho dirigido único de r até v. Já a *r-arborescência inversa* (*in-arborescência*) orienta as arestas em direção a r: cada $v \neq r$ tem exatamente uma aresta de saída, e de cada vértice parte um caminho dirigido único até r [2, 3].

Com essa distinção em mente, este trabalho concentra-se na variante inversa. Formalmente, dado um grafo dirigido $G = (V, E)$ com custos $c : E \to \mathbb{R}^+$ nas arestas e um vértice raiz $r \in V$, procura-se uma *r-arborescência inversa* — isto é, uma árvore direcionada que atinja todos os vértices por caminhos dirigidos até r — que minimize o custo total das arestas selecionadas (cf. [2, 3]).

Nosso interesse, porém, não é apenas encontrar a arborescência mínima: o percurso até ela também importa, pois revela propriedades estruturais dos dígrafos e ilumina técnicas distintas de otimização. Por isso, investigamos duas rotas clássicas e complementares: (i) o algoritmo de Chu–Liu/Edmonds, que opera por normalização dos custos das arestas de entrada, seleção sistemática de arestas de custo zero e contração de ciclos até obter um grafo reduzido, seguida pela reexpansão para reconstrução da solução [1, 2]; e (ii) a abordagem dual, em duas fases, de András Frank, fundamentada em cortes dirigidos, na qual se maximiza uma função de cortes c-viável para induzir arestas de custo zero e, em seguida, extrai-se a arborescência apenas a partir dessas arestas [3]. Embora assentados em princípios distintos — contração de ciclos no plano primal versus empacotamento/dualidade por cortes —, ambos os paradigmas produzem soluções ótimas e tornam explícitas a variedade de abordagens matemáticas que podem ser empregadas para resolver o mesmo problema.

Analisamos e implementamos, em Python, essas duas abordagens, e apresentaremos detalhes das implementações, desafios enfrentados e soluções adotadas. Realizamos testes de volume com milhares de instâncias geradas aleatoriamente, registrando resultados em arquivos CSV e de log; os custos obtidos pelo Chu–Liu/Edmonds e pelas duas variantes de András Frank coincidem, corroborando a correção das implementações.

Adicionalmente, desenvolvemos uma aplicação web com fins didáticos, utilizando o framework PyScript e as bibliotecas NetworkX e Matplotlib, que permitem construir grafos dirigidos interativamente, escolher o vértice-raiz, executar o algoritmo de Chu–Liu/Edmonds e acompanhar, passo a passo, a evolução do grafo e o registro detalhado da execução (log). A interface inclui operações de adicionar arestas com pesos, carregar um grafo de teste e exportar a instância em formato JSON, facilitando a experimentação por estudantes e educadores.

1.1 Justificativa

A busca por uma *r-arborescência inversa de custo mínimo* em grafos dirigidos é um problema clássico com aplicações em diversas áreas, como redes de comunicação, planejamento de rotas, análise de dependências e modelagem de processos. Mas, não precisamos dessa justificação prática para nos interessarmos pelo problema: a riqueza estrutural dos dígrafos e a variedade de técnicas algorítmicas disponíveis o tornam um excelente caso de estudo em otimização combinatória.

Do ponto de vista didático, a metáfora da "teia explicativa mínima" torna concreto o porquê de estudarmos arborescências enraizadas: ela mapeia perguntas sobre explicação, alcance e economia de recursos para estruturas dirigidas, servindo de fio condutor nas implementações e nos experimentos que apresentamos.

1.2 Objetivos

O objetivo principal deste trabalho é analisar, implementar e comparar duas abordagens clássicas para o problema de *r-arborescência de custo mínimo* em grafos dirigidos oferecendo uma aplicação web interativa que facilite o entendimento e a experimentação com o algoritmo de Chu–Liu/Edmonds e o método de András Frank, tornando-o acessível para estudantes e educadores.

1.3 Estrutura do Trabalho

Resumidamente, o trabalho abrange as seguintes frentes:

1. **Fundamentação teórica**: revisão da literatura sobre arborescências em grafos dirigidos, incluindo definições, propriedades e resultados relevantes.

2. **Análise teórica**: consolidação dos conceitos de dígrafos e arborescências, compondo as formulações primal (normalização de custos e contração/reexpansão de ciclos no algoritmo de Chu–Liu/Edmonds) e dual (cortes dirigidos e função c-viável no método de András Frank), destacando resultados e intuições estruturais.

3. **Implementação computacional**: implementação em Python das rotinas de normalização dos custos de entrada, construção de F^*, detecção e contração de ciclos e reconstrução da solução (Chu–Liu/Edmonds), bem como das duas fases do método de András Frank; além de uma suíte de testes automatizados em larga escala sobre instâncias aleatórias com até centenas de vértices, verificando a coincidência dos custos entre os métodos e registrando resultados em CSV e log.

4. **Aplicação pedagógica**: desenvolvimento de uma aplicação web interativa (PyScript + NetworkX + Matplotlib) que permite montar instâncias, escolher o vértice-raiz e acompanhar, passo a passo, a execução do algoritmo com visualização do grafo e dos pesos das arestas, log textual e importação/exportação em JSON para facilitar a reprodução de experimentos.

Como validação empírica, realizamos testes de volume com milhares de instâncias geradas aleatoriamente, registrando resultados em arquivos CSV e de log; os custos obtidos pelo Chu–Liu/Edmonds e pelas duas variantes de András Frank coincidem, corroborando a correção das implementações. Deste modo, o trabalho entrega implementações verificadas de Chu–Liu/Edmonds e András Frank, um visualizador web interativo e testes de volume que confirmam a equivalência de custos, úteis ao estudo e ao ensino de arborescências.

2 Definições Preliminares

Neste capítulo, reunimos as noções básicas de teoria dos grafos dirigidos utilizadas ao longo do texto. O objetivo é fixar notações e conceitos (conjuntos, relações, funções, dígrafos, propriedade em dígrafos, dígrafos ponderados, ramificações geradoras, arborescências, funções de custo, dualidade, problemas duais e algoritmos), até chegar à formulação do problema da r-arborescência inversa de custo mínimo e adiamos descrições algorítmicas para capítulos posteriores. Mantendo a intuição, pense o dígrafo como essa teia orientada: fixamos um vértice-raiz r e custos nas arestas; a terminologia será introduzida gradualmente.

2.1 Conjuntos

Este trabalho depende profundamente da teoria dos conjuntos. Basicamente podemos dizer que todos os objetos matemáticos que iremos utilizar nessa dissertação se reduzem a conjuntos e operações entre eles.

Um **conjunto** é uma agregação de objetos distintos com características bem definidas, chamados elementos ou membros do conjunto. Os conjuntos são geralmente representados por letras maiúsculas (por exemplo, A, B, C) e seus elementos são listados entre chaves (por exemplo, $A = \{1, 2, 3\}$). Dois conjuntos são iguais se contêm exatamente os mesmos elementos.

Podemos ter conjuntos de qualquer tipo de objeto, incluindo números, letras, elementos da natureza, etc. Para motivar as definições ao longo do texto, adotaremos um exemplo-mestre com organismos: árvores, plantas e fungos, usando pertinência e inclusão para guiar a intuição.

2.1.1 Subconjuntos

Dizemos que A é um **subconjunto** de B, denotado $A \subseteq B$, quando todo elemento de A também pertence a B. Se, além disso, $A \neq B$, escrevemos $A \subset B$ e chamamos A de **subconjunto próprio** de B. Ex.: $\{1, 2\} \subseteq \{1, 2, 3\}$ e $\{1, 2\} \subset \{1, 2, 3\}$.Por convenção, o conjunto vazio $\varnothing$ é subconjunto de qualquer conjunto X (isto é, $\varnothing \subseteq X$), e todo conjunto é subconjunto de si mesmo ($X \subseteq X$).

No nosso exemplo-mestre, com $P = \{\text{todas as plantas}\}$, $T = \{\text{todas as árvores}\}$ e $F = \{\text{todos os fungos}\}$, temos $T \subseteq P$ (todas as árvores são plantas), enquanto $F \not\subseteq P$.

2.1.2 Pertinência e inclusão

Pertinência e inclusão são os conceitos mais fundamentais da teoria dos conjuntos.

Começando pela **noção de pertinência** denotado por $\in$: dizemos que um elemento x pertence a um conjunto X quando $x \in X$ e não pertence quando $x \notin X$.

Para ancorar ideias, fixemos um universo U de organismos e definamos três conjuntos: $P = \{\text{todas as plantas}\}$, $T = \{\text{todas as árvores}\}$ e $F = \{\text{todos os fungos}\}$. Se x é um carvalho, então $x \in T$ e, como toda árvore é uma planta, $x \in P$. Já se y é um cogumelo, então $y \in F$ e, na taxonomia moderna, $y \notin T$ e $y \notin P$. Agora, considere $A = \{\text{árvores com folhas verdes}\}$; a pertinência fica clara: $x \in A$ se, e somente se, x é árvore e tem folhas verdes.

Continuando, vem a **relação de inclusão** entre conjuntos denotada por $\subseteq$: escrevemos $X \subseteq Y$ quando todo elemento de X também pertence a Y (e $X \subset Y$ quando, além disso, $X \neq Y$). No nosso exemplo, $A \subseteq T \subset P$ e $T \cap F = \varnothing$ (árvores e fungos não se sobrepõem).

2.1.3 Operações entre conjuntos

Com essas definições de pertinência, inclusão e subconjuntos, apresentamos as operações básicas entre conjuntos, que usaremos ao longo do texto (mantendo o exemplo com P, T, F, A).

Outras operações comuns entre conjuntos incluem:

- **União** ($A \cup B$): o conjunto de todos os elementos que pertencem a A, a B, ou ambos.

  Exemplo: $T \cup F$ é o conjunto de todos os organismos que são árvores ou fungos (ou ambos, se existissem tais organismos).

- **União disjunta** ($A \uplus B$): o conjunto de todos os elementos que pertencem a A ou a B, mas não a ambos; é igual a $A \cup B$ quando A e B são disjuntos.

  Exemplo: $T \uplus F$ é o conjunto de todos os organismos que são árvores ou fungos, mas não ambos (o que é trivialmente igual a $T \cup F$ pois T e F são disjuntos).

- **Interseção** ($A \cap B$): o conjunto de todos os elementos que pertencem tanto a A quanto a B.

  Exemplo: $T \cap P = T$, pois todas as árvores são plantas.

- **Diferença** ($A \setminus B$): o conjunto de todos os elementos que pertencem a A mas não a B.

  Exemplo: $T \setminus A$ é o conjunto de todas as árvores que não têm folhas verdes.

- **Complemento** de X em um universo fixo U: $X^c := U \setminus X$ (também chamado de *complemento absoluto*); o *complemento relativo* de X em Y é $Y \setminus X$.

  Exemplo: $T^c = U \setminus T$ é o conjunto de todos os organismos que não são árvores. Ou seja, T^c inclui plantas que não são árvores, fungos e quaisquer outros organismos no universo U.

- **Diferença simétrica** ($A \Delta B$): $(A \setminus B) \cup (B \setminus A)$; é igual a $A \cup B$ quando A e B são disjuntos.

  Exemplo: $P \Delta F$ é o conjunto de todos os organismos que são plantas ou fungos, mas não ambos (o que é trivialmente igual a $P \cup F$ pois P e F são disjuntos).

- **Produto cartesiano** ($A \times B$): o conjunto de pares ordenados (a, b) com $a \in A$ e $b \in B$.

  Exemplo: suponha $T = \{t_1, t_2\}$ e $F = \{f_1, f_2\}$. Então $T \times F = \{(t_1, f_1), (t_1, f_2), (t_2, f_1), (t_2, f_2)\}$.

- **Conjunto das partes** (2^U): a família de todos os subconjuntos de U (inclui $\varnothing$ e o próprio U).

  Exemplo: se $U = \{x, y\}$, então $2^U = \{\varnothing, \{x\}, \{y\}, \{x, y\}\}$. Logo, $|2^U| = 4 = 2^{|U|}$.

Identidades úteis. Usaremos livremente as propriedades clássicas de conjuntos — comutatividade e associatividade de $\cup$ e $\cap$, distributividade e as **leis de De Morgan** — sem prova. Quando for relevante, explicitaremos a identidade no ponto de uso. Por exemplo, no nosso universo U, $(P \cup F)^c = P^c \cap F^c$.

2.1.4 Coleção

Entre os objetos que podem pertencer a um conjunto, estão também eles mesmos, outros conjuntos. Chamaremos tais conjuntos de **coleções** (ou **famílias**) de conjuntos. Por exemplo, $\mathcal{C} = \{P, T, F\}$ é uma coleção formada pelos conjuntos de organismos já definidos: plantas P, árvores T e fungos F. Note que $\mathcal{C}$ é um conjunto como outro qualquer; seus elementos são, cada um, um conjunto.

Coleções são úteis para agrupar subconjuntos relacionados de um mesmo universo. Por exemplo, considere $\mathcal{D} = \{A, B\}$, onde $A = \{\text{árvores com folhas verdes}\}$ e $B = \{\text{árvores com folhas vermelhas}\}$. Assim, $\mathcal{D} \subseteq 2^T$ é uma coleção de subconjuntos de T.

Uma coleção $\mathcal{F}$ é dita **laminar** quando, para quaisquer $X, Y \in \mathcal{F}$, vale que $X \subseteq Y$, $Y \subseteq X$ ou $X \cap Y = \varnothing$; isto é, quaisquer dois conjuntos são aninhados (um está contido no outro) ou são disjuntos.

Por exemplo, na coleção $\mathcal{C} = \{P, T, F\}$: P é o conjunto de todas as plantas, T o de todas as árvores (portanto $T \subseteq P$) e F o de todos os fungos (disjunto de plantas e, logo, de árvores). Assim, quaisquer dois conjuntos em $\mathcal{C}$ são aninhados ou disjuntos, e $\mathcal{C}$ é laminar. Na coleção $\mathcal{D} = \{A, T\}$: A é o conjunto de árvores com folhas verdes e T o de todas as árvores; como toda árvore de A é árvore de T, temos $A \subseteq T$ e a coleção é laminar. Já em $\mathcal{E} = \{A, R\}$: R é o conjunto de árvores frutíferas; há árvores que são ao mesmo tempo frutíferas e de folhas verdes (a interseção é não vazia), mas nenhuma das classes contém a outra, então $\mathcal{E}$ não é laminar.

Este é um importante conceito que aparecerá no restante do trabalho. A ideia de laminaridade retornará quando tratarmos de cortes dirigidos.

2.1.5 Comparando conjuntos: cardinalidade e maximalidade

Podemos comparar conjuntos através de relações de tamanho (cardinalidade) ou por relações de inclusão. Essas duas formas de comparação são distintas e importantes, especialmente quando lidamos com coleções de conjuntos.

A **cardinalidade** de um conjunto A, denotada por $|A|$, é o número de elementos de A. Para conjuntos finitos, é simplesmente a contagem dos elementos (por exemplo, se $A = \{1, 2, 3\}$, então $|A| = 3$). Para conjuntos infinitos, a cardinalidade pode ser mais complexa, envolvendo conceitos como infinito enumerável e não enumerável. Por exemplo, o conjunto dos números naturais $\mathbb{N}$ é infinito enumerável, enquanto o conjunto dos números reais $\mathbb{R}$ é infinito não enumerável.

Dizemos que $A \in \mathcal{C}$ tem **maior cardinalidade** se $|A| \geq |B|$ para todo $B \in \mathcal{C}$ (podendo haver empates). Esse critério não coincide, em geral, com a comparação por relação de inclusão. Em grafos, por exemplo, distinguem-se conjuntos independentes *maximais* (não ampliáveis) de conjuntos independentes *máximos* (de cardinalidade máxima).

Ao compararmos uma coleção $\mathcal{C}$ de conjuntos utilizando sua relações de inclusão $(\mathcal{C}, \subseteq)$, é imprescindível distinguir **maximal** de **máximo**.

Um conjunto $A \in \mathcal{C}$ é **maximal** se não existe $B \in \mathcal{C}$ tal que $A \subset B$. Em palavras: não dá para ampliar A estritamente dentro da coleção. Podem haver vários elementos maximais, e eles podem ser incomparáveis entre si. Ex.: em $\mathcal{C} = \big\{\{1\}, \{2\}\big\}$, ambos $\{1\}$ e $\{2\}$ são maximais, mas não existe máximo.

Um conjunto $A \in \mathcal{C}$ é **máximo** se $B \subseteq A$ para todo $B \in \mathcal{C}$. Se existe, é único. Ex.: em $\mathcal{C} = \big\{\{1\}, \{2\}, \{1, 2\}\big\}$, o conjunto $\{1, 2\}$ é o máximo.

Um bom exemplo para ilustrar a distinção entre conjuntos maximais e máximos é a coleção $\mathcal{C} = \big\{\{1\}, \{2\}, \{1, 2\}, \{3\}\big\}$. Aqui, $\{1, 2\}$ é o único conjunto máximo (contém todos os outros), enquanto $\{1\}$, $\{2\}$ e $\{3\}$ são todos maximais (não podem ser ampliados dentro da coleção).

Esses conceitos serão úteis mais adiante, quando lidarmos com coleções de cortes dirigidos e conjuntos de arestas em grafos.

2.2 Relações e Funções

Quando lidamos com conjuntos, frequentemente precisamos estabelecer conexões ou associações entre seus elementos. Essas conexões são formalizadas através de **relações** e **funções**, que são conceitos fundamentais em matemática e ciência da computação.

Na matemática, uma **relação** entre dois conjuntos A e B é uma maneira de associar elementos de A com elementos de B. Uma **função** é um tipo especial de relação que associa cada elemento de A a exatamente um elemento de B.

Na ciëncia da computação, relações e funções são usadas para modelar conexões entre dados, estruturas de dados e operações.

Uma **relação** R entre dois conjuntos A e B é um subconjunto do produto cartesiano $A \times B$. Ou seja, $R \subseteq A \times B$. Se $(a, b) \in R$, dizemos que a está relacionado a b pela relação R, denotado aRb.

No nosso exemplo-mestre, considere $P = \{\text{todas as plantas}\}$ e $F = \{\text{todos os fungos}\}$. Definimos a relação R como "é um organismo que compete com". Assim, se uma planta $p \in P$ compete com um fungo $f \in F$, então $(p, f) \in R$.

Em teoria dos grafos, uma relação pode representar conexões entre vértices. Por exemplo, em um grafo dirigido, a relação "existe uma aresta de u para v"pode ser representada como um conjunto de pares ordenados (u, v).

Uma **função** f de um conjunto A em um conjunto B é uma relação especial que associa cada elemento de A a exatamente um elemento de B. Denotamos isso como $f : A \to B$. Se $f(a) = b$, dizemos que b é a imagem de a sob f.

No nosso exemplo-mestre, considere a função $f : P \to \mathbb{N}$ que associa cada planta ao seu número de folhas. Se $p \in P$ é uma árvore com 100 folhas, então $f(p) = 100$.

Em teoria dos grafos, funções podem ser usadas para atribuir pesos ou capacidades às arestas. Por exemplo, se temos um grafo G com arestas $e_1, e_2, \ldots, e_n$, podemos definir uma função $c : E \to \mathbb{R}^+$ que atribui um peso $c(e_i)$ a cada aresta e_i.

Computacionalmente, relações são frequentemente implementadas como tabelas ou listas de pares, enquanto funções são frequentemente implementadas como métodos ou procedimentos que recebem entradas e retornam saídas. Elas são fundamentais para a modularidade e reutilização de código.

3 Considerações Finais

Referências

[1] Y. J. Chu e T. H. Liu. "On the Shortest Arborescence of a Directed Graph". Em: *Scientia Sinica* 14 (1965), pp. 1396–1400.

[2] J. Edmonds. "Optimum Branchings". Em: *Journal of Research of the National Bureau of Standards* 71B (1967), pp. 233–240.

[3] A. Frank e G. Hajdu. "A Simple Algorithm and Min–Max Formula for the Inverse Arborescence Problem". Em: *Algorithms* 7.4 (2014), pp. 637–647. DOI: 10.3390/a7040637.