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

10 de Dezembro de 2024

1 Introdução

O problema da *r-arborescência de custo mínimo* consiste em, dado um grafo dirigido com custos nas arestas, encontrar um subgrafo orientado enraizado em r que conecte r a todos os demais vértices por caminhos direcionados, minimizando a soma dos custos das arestas selecionadas.

Ao longo das últimas décadas, consolidaram-se duas famílias clássicas de métodos: (i) o algoritmo de Chu–Liu/Edmonds, que opera por normalização dos custos das arestas de entrada, seleção sistemática de arcos de custo zero e contração de ciclos até obter um grafo reduzido, seguido de 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 0-arestas 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 as relações estruturais em grafos dirigidos e a noção de r-arborescência.

Neste trabalho, realiza-se a análise e implementação desses algoritmos, que envolvem a busca de *r-arborescência inversa de custo mínimo*, A dissertação explora tanto os aspectos teóricos quanto os computacionais da estrutura e dos seus respectivos algoritmos de busca. Adicionalmente, desenvolveu-se uma aplicação web didática com interface interativa que possibilita a visualização passo a passo da execução dos algoritmos em instâncias de diferentes grafos dirigidos, tornando-os mais acessíveis a estudantes e educadores.

A proposta alia, portanto, três dimensões complementares:

1. **Análise teórica**: revisão de fundamentos de grafos, arborescências e formulações primal e dual;
2. **Implementação computacional**: tradução das etapas dos algoritmos em código Python e validação experimental em instâncias geradas aleatoriamente;

3. **Aplicação pedagógica**: desenvolvimento de uma ferramenta interativa para experimentação e aprendizado visual.

Dessa forma, este trabalho contribui para o aprofundamento conceitual sobre arborescências de custo mínimo, e para a produção de recursos didáticos que favorecem a compreensão e a difusão de algoritmos fundamentais da teoria dos grafos.

2 Definições Preliminares

Neste capítulo, apresentamos os conceitos essenciais para compreender o problema de obter uma r-arborescência de custo mínimo em grafos dirigidos, bem como os algoritmos empregados em sua resolução. Abordaremos, em ordem, (i) a definição de grafo dirigido e noções correlatas; (ii) caminhos, ciclos e subdigrafos; (iii) o conceito de r-arborescência; (iv) a função de custo e a formulação do problema de r-arborescência de custo mínimo; e (v) princípios de algoritmos gulosos e sua aplicação a esse problema. Na sequência, enunciamos lemas e teoremas fundamentais que embasam as ideias algorítmicas desenvolvidas nas seções posteriores.

2.1 Definições

Formalmente, um **grafo dirigido** D ou abreviadamente **dígrafo** [1] é um triplo ordenado $D = (V(D), A(D), \varphi)$, onde:

- $V(D)$ é um **conjunto não vazio de vértices**, aqui nos referiremos apenas como V quando estivermos trabalhando apenas com um objeto;
- $A(D)$ é um **conjunto de arcos (ou arestas direcionadas)**, disjunto de $V(D)$, chamaremos apenas de A, e nesse texto sempre que estivermos falando de uma estrutura em dígrafo, utilizaremos apenas o termo **aresta**, para nos referir à arcos ou arestas direcionadas;
- $\varphi : A(D) \to V(D) \times V(D)$ é **uma função de incidência** que associa a cada arco $a \in A(D)$ um par ordenado de vértices de $V(D)$, possivelmente repetidos (ou seja, os dois vértices não precisam ser distintos).

Um **digrafo** pode ser representado por um diagrama de pontos como vértices e setas como suas arestas. Cada seta indica a orientação da aresta, logo, conceituamos essa direção através da determinação do que vem a ser a cauda e a cabeça de uma aresta direcionada.

Seja $a \in A$ uma aresta tal que $\varphi(a) = (u, v)$, onde $u, v \in V$. Nesse caso:

- a é dito conectar u a v;
- u é chamado de **cauda/tail** de a se for a origem da aresta. Formalmente, $\text{tail} : A \to V$, onde $\text{tail}(a) = u$.
- v é chamado de **cabeça/head** de a, ou o destino da aresta (ponta da seta em uma representação pictográfica). Formalmente, $\text{head} : A \to V$, onde $\text{head}(a) = v$.

[1] O termo é abreviado do inglês: *directed graph*.

Antes de definirmos o conceito de uma r-arborescência, precisamos estabelecer o conceito de **subgrafo dirigido** ou **subdigrafo**, uma vez que uma arborescência corresponde a um subdígrafo particular.

Desse modo, um **subdigrafo** D' de um dígrafo $D = (V(D), A(D), \varphi)$ é um dígrafo $D' = (V(D'), A(D'), \varphi')$ tal que:

- $V(D') \subseteq V(D)$;
- $A(D') \subseteq A(D)$;
- φ' é a restrição de φ ao conjunto $A(D')$, ou seja, $\varphi'(a) = \varphi(a)$ para todo $a \in A(D')$.

Logo, D' herda as relações de incidência do dígrafo D, apresentando um subconjunto de vértices e arcos do dígrafo original D, e podemos defini-lo apenas como um **subgrafo** em um dígrafo.

A definição formal de uma r-arborescência depende do conceito de ciclos; esse, por sua vez, depende da noção de caminho. Apresentamos ambas a seguir.

Um **caminho** em um dígrafo $D = (V, A)$ é uma sequência ordenada de vértices $v_1, v_2, \ldots, v_k$, onde:

- Cada par consecutivo $(v_i, v_{i+1}) \in A$ para $1 \leq i < k$.
- $a \ \in A$ é dito conectar v_i a v_{i+1};
- $\text{tail}(a) = v_i$;
- $\text{head}(a) = v_{i+1}$;

Seja $C = (v_0, v_1, \ldots, v_n)$ **um caminho em um dígrafo** de comprimento n, no qual $v_0 = v_n = u$. Caracterizamos um **ciclo** a seguir.

Se existir um vértice v tal que $v \notin V(C)$, mas $v \in V(D)$, ou seja, v pertence a D, mas não a C, e simultaneamente:

- v recebe um arco de um vértice de C (existe um arco $(v_i, v) \in A(D)$ para algum $v_i \in C$);
- v emite um arco para um vértice de C (existe um arco $(v, v_{i+1}) \in A(D)$ para algum $v_{i+1} \in C$);

então v está inserido em um **ciclo direcionado** ou **circuito** de comprimento $n + 1$, dado por:

$$(v_a, v_1, \ldots, v_i, v, v_{i+1}, \ldots, v_n).$$

Agora, já podemos caracterizar formalmente uma r-arborescência. Para isso, temos o seguinte resultado:

Lema 2.1.1: Um subgrafo $T = (V, F)$ de G é uma arborescência enraizada em r se e somente se T não contém ciclos e, para cada nó $v \neq r$, há exatamente uma aresta em F que entra em v.

Prova: Provaremos primeiro a ida e depois a volta.

Se T é uma arborescência enraizada em r, então, por definição, há um caminho único de r para qualquer nó v. A última aresta desse caminho é a única que entra em v, o que satisfaz a condição enunciada.

Por outro lado, supondo que T não contém ciclos e que cada nó $v \neq r$ possua exatamente uma aresta entrando. Para provar que T é uma arborescência, basta demonstrar que há um caminho direcionado de r para cada nó v.

A prova se dá por construção, começamos em v e seguimos as arestas no sentido contrário. Como T não contém ciclos, o processo não pode continuar indefinidamente e deve terminar em um nó sem arestas de entrada. Sendo r o único nó sem arestas de entrada, isso implica que existe um caminho de r até v, garantindo que T é de fato uma arborescência.

Agora que temos a definição formal de uma r-arborescência, podemos falar sobre o problema de busca de uma r-arborescência de custo mínimo, antes precisamos da definição de **função de custo total** $c(F)$.

Dada uma função de custo $c : A \to \mathbb{R}^+$ associada às arestas de um grafo $D = (V, A)$, o **custo total** de um subgrafo $F \subseteq D$ é definido como:

$$c(F) = \sum_{a \in F} c(a),$$

onde F é o conjunto de arestas do subgrafo.

Assim sendo, o **problema da R-arborescência de custo mínimo** consiste em determinar a r-arborescência $F \subseteq D$ que minimiza o custo total $c(F)$, sujeito à condição de que F seja uma r-arborescência válida.

Nesse contexto, **algoritmos gulosos** são uma heurística [2] comumente utilizada na resolução desse tipo de problema.

De acordo com os autores do livro Algorithm Design, um algoritmo é considerado guloso quando constrói uma solução de maneira incremental, realizando pequenas escolhas sucessivas. Em cada etapa, a decisão é tomada de forma *míope*, visando otimizar algum critério subjacente de maneira local.

[2] método aproximado utilizado para resolver problemas de otimização ou decisão de forma eficiente, quando uma solução exata é computacionalmente inviável

Dito isso, dado um digrafo D e um conjunto de custos não negativos associados às suas arestas, os autores sugeriram um algoritmo guloso de complexidade $O(n^3)$ responsável por modificar esses custos para o mínimo possível visando garantir que T, uma r-arborescência específica de G, seja a arborescência mais curta, a construção dessa solução de tempo polinomial depende de algumas propriedades fundamentais sobre r-arborescências que apresentaremos na próxima seção.

2.2 Propriedades Fundamentais

Agora, vamos apresentar alguns lemas que estabelecem as bases para o desenvolvimento do algoritmo guloso de múltiplas fases para encontrar uma r-arborescência específica em D de custo mínimo.

Assim, antes de buscarmos uma arborescência de custo mínimo, é necessário garantir a existência de uma r-arborescência. Para isso, tem-se o seguinte resultado:

Lema 2.2.1: Um digrafo D contém uma arborescência enraizada em r se e somente se há um caminho direcionado de r para cada outro nó em D.

Prova: Suponha que $D = (V, A)$ seja um grafo dirigido e que possua uma arborescência enraizada em um nó $r \in V$. Por definição, uma arborescência é uma árvore geradora enraizada em r.

Provando a ida, supomos para fins de contradição, que $\exists v \in V$ tal que não há um caminho direcionado de r até v. Isso implica que v não pode ser alcançado a partir de r. No entanto, como $T = (V, F)$ é uma arborescência, ele deve conter **exatamente um caminho direcionado** de r até cada nó v. A inexistência de tal caminho contradiz a própria definição de arborescência, levando a uma contradição. Assim, concluímos que deve existir um caminho direcionado de r para cada $v \in V$.

Agora provamos a recíproca, supomos que D é um digrafo em que existe um caminho direcionado de r para cada nó $v \in V$. Queremos provar que D contém uma arborescência $T = (V, F)$ enraizada em r.

Definimos F da seguinte forma: para cada nó $v \neq r$, escolhemos uma única aresta (u, v) tal que u seja um nó no caminho direcionado mínimo de r para v. Isso é possível porque, por hipótese, existe um caminho direcionado de r até cada v, garantindo que v tem pelo menos um nó predecessor. Isso garante que cada nó $v \neq r$ tem exatamente uma aresta de entrada em T. Além disso, como T contém apenas arestas que fazem parte dos caminhos direcionados partindo de r, ele não pode conter ciclos. Portanto, T é uma arborescência em D

Logo, garantimos que de fato possuímos a estrutura pretendida: r-arborescência, uma vez que acessamos todos os vértices de um dígrafo a partir de um vértice raiz r.

Esse resultado é necessário mas, não suficiente, pois é preciso encontrar uma **arborescência de custo mínimo**. Entretanto, ao selecionar um conjunto de arestas candidatas, pode ocorrer que esse conjunto não seja uma arborescência com tal característica. Então, nosso algoritmo de busca precisa garantir o seguinte:

Conjectura 2.2.1: Se (V, F^*) é uma arborescência, então ela é uma arborescência de custo mínimo.

Essa conjectura parte da seguinte observação: toda arborescência contém exatamente uma aresta entrando em cada nó $v \neq r$. Caso contrário, não seria uma arborescência e portanto ou conteria ciclos ou não atingiria todos os vértices do dígrafo a partir de um nó raiz.

Portanto uma vez que exploremos um dígrafo D a partir de um nó raiz e alcancemos todos os vértices, pelo lema 2.2.1, sabemos que temos uma r-arborescência, queremos encontrar uma que seja de custo mínimo. Nos restando dois casos:

- O digrafo $(D = V, A)$ não possui ciclos, e portanto possui apenas uma possibilidade de construção de uma r-arborescência digamos $T = V, A$;
- O digrafo $(D = V, A)$ possui ciclos, e portanto possui mais uma possibilidade de construção de uma r-arborescência digamos $T' = V, A'$ e $T^* = V, A^*$ com $A' \neq A^*$.

Logo, uma operação de contração de ciclos pode estar envolvida na busca por r-arborescência de custo-mínimo. Desse modo, vamos primeiro definir uma operação de contração de vértices e depois caracterizar a operação de contração de ciclos.

A **contração de vértices** é uma operação que reduz um conjunto de vértices conectados C em um digrafo $D = (V, A)$ a um único vértice v_c, preservando a estrutura das arestas adjacentes. Formalmente, para cada conjunto $C \subseteq V$:

- Todas as arestas que entram ou saem de C são reconfiguradas para conectar-se diretamente ao novo vértice v_c;
- Arestas internas em C são removidas;
- O vértice v_c substitui os vértices de C no grafo resultante.

Desse modo, se a estrutura inicial contém ciclos, precisamos removê-los através de uma operação de contração de vértices sem perder a conectividade da arborescência, essa operação envolve uma modificação de custos nas arestas com a contração do ciclo C em um único **super nó**, obtendo um grafo reduzido $D' = (V', A')$. Aqui, V' contém os nós de $V - C$, além de um único nó c^* que representa C.

E, cada aresta $e \in E$ é transformada em uma aresta correspondente $e' \in E'$, substituindo cada extremidade de e que pertence a C pelo novo nó c^*. Esse processo pode gerar **arestas paralelas** (ou seja, arestas com as mesmas extremidades), o que não representa um problema, uma vez que pode-se remover os **auto-laços** de E', ou seja, arestas cujas duas extremidades são c^*.

Contudo, ainda precisamos estabelecer que certas transformações nos custos das arestas, visando a contração de ciclos de forma eficiente não alteram a solução ótima.

Portanto, se escolhermos um nó v e subtrairmos uma mesma quantidade do custo de todas as arestas que entram em v, observamos que o custo total de qualquer arborescência deve mudar exatamente pelo mesmo valor.

Isso significa que o custo real da aresta mais barata que entra em v não é relevante por si só; o que importa é o custo relativo das outras arestas que entram em v. Definimos, então, y_v como o menor custo de qualquer aresta que entra em v:

$$y_v = \min\{c_a \mid a = (u, v) \in A, u \in V\}$$

Para cada aresta $a = (u, v)$, cujo custo original é $c_a \geq 0$, definimos o custo modificado c'_a como:

$$c'_a = c_a - y_v$$

Observamos que, como $c_a \geq y_v$, todos os custos modificados continuam sendo não negativos:

$$c'_a \geq 0, \quad \forall e \in A.$$

Tendo em vista esses conceitos, é possível estabelecer o lema abaixo.

Lema 2.2.2: T é uma arborescência ótima em D com relação aos custos $\{c_a\}$ se e somente se também é uma arborescência ótima com relação aos custos modificados $\{c'_a\}$.

Prova: Seja T uma arborescência arbitrária em D. Definimos os custos originais das arestas como $\{c_a\}$ e os custos modificados como $\{c'_a\}$, onde cada custo modificado é dado por:

$$c'_a = c_a - y_v, \quad \text{para toda aresta } a = (u, v) \in A.$$

Queremos mostrar que o custo total de qualquer arborescência T sob $\{c_a\}$ difere do custo sob $\{c'_a\}$ por uma constante fixa, independentemente da escolha de T.

O custo total de T sob os pesos originais é:

$$C(T) = \sum_{a \in T} c_a.$$

O custo total de T sob os pesos modificados é:

$$C'(T) = \sum_{a \in T} c'_a = \sum_{a \in T} (c_a - y_v).$$

Expandindo a soma:

$$C(T) - C'(T) = \sum_{a \in T} c_a - \sum_{a \in T} (c_a - y_v).$$

Distribuindo os termos:

$$C(T) - C'(T) = \sum_{a \in T} c_a - \sum_{a \in T} c_a + \sum_{a \in T} y_v.$$

Cancelando os termos c_a:

$$C(T) - C'(T) = \sum_{e \in T} y_v.$$

Ou seja, a diferença entre seu custo com os pesos $\{c_a\}$ e os pesos modificados $\{c'_a\}$ é dada por:

$$\sum_{e \in T} c_a - \sum_{e \in T} c'_a = \sum_{v \neq r} y_v.$$

Isso ocorre porque toda arborescência contém exatamente uma aresta entrando em cada nó v. Como essa diferença é independente da escolha de T, segue-se que T tem custo mínimo sob $\{c_a\}$ se e somente se tem custo mínimo sob $\{c'_a\}$.

Assim sendo, basta-se aplicar recursivamente a operação de contração de ciclos para encontrar uma arborescência ótima no grafo reduzido G', considerando os custos modificados $\{c'_e\}$. A arborescência retornada por essa chamada recursiva pode ser convertida em uma arborescência de D incluindo todas as arestas do ciclo C, exceto uma.

Baseado nessas propriedades fundamentais, apresentamos na seção seguinte uma contextualização e descrição detalhada do algoritmo.

2.3 Descrição do Algoritmo

Em otimização combinatória, é comum o uso de uma aplicação do método primal-dual [3], que permite resolver o problema por meio de ajustes iterativos nos custos.

Nesse contexto, o algoritmo de busca da arborescência de custo mínimo aqui abordado é uma aplicação da técnica primal-dual, o mesmo trabalha com a modificação dos custos das arestas, seleciona um conjunto candidato de arborescência e, caso necessário, contrai ciclos e repete o procedimento recursivamente, até que uma r-arborescência seja identificada. O procedimento é descrito a seguir:

1. **Inicialização e alteração dos custos das arestas:** Para cada nó $v \neq r$:
   (a) Define-se y_v como o custo mínimo de qualquer aresta que entra em v.
   (b) Modificam-se os custos das arestas e que entram em v para $c'_e = c_e - y_v$.

2. **Construção da arborescência candidata:** Escolhe-se uma aresta de custo zero entrando em cada nó $v \neq r$, formando um conjunto F^*.

3. **Verificação da arborescência:**
   (a) Se F^* forma uma arborescência, retorna-se F^* como a solução ótima.
   (b) Caso contrário, existe um ciclo dirigido $C \subseteq F^*$.

[3] A dualidade lida com pares de programas lineares e as relações entre suas soluções. Um desses problemas é chamado de **primal** e o outro, de **dual**.

(a) Contrair C em um único **supernó** c^*.

(b) Criar um novo grafo reduzido $G' = (V', E')$, onde:
   - V' contém os nós de $V - C$ mais o supernó c^*.
   - Cada aresta $e \in E$ é transformada em uma aresta correspondente $e' \in E'$, substituindo as extremidades que pertencem a C pelo nó c^*.
   - Auto-laços são removidos de E'.

4. **Recursão no grafo reduzido:** Encontrar recursivamente uma arborescência ótima (V', F') em G', considerando os custos modificados $\{c'_e\}$.

5. **Reconstrução da solução original:** Expandir (V', F') para obter uma arborescência (V, F) em G, adicionando todas as arestas de C exceto uma.

Esse algoritmo permite uma implementação de complexidade polinomial e se encaixa na classe de algoritmos de otimização combinatória.

2.4 Correção do Algoritmo

WIP

extbfReferência: [4]. *(Definição localizada em Algorithm Design, pág. 177).*

2.5 Implementação do Algoritmo

Link do colab notebook atualizado - wip

3 Algoritmo de Fulkerson - 1973

Nesse capítulo, trataremos do problema de busca da r-arborescência de custo mínimo, através de uma abordagem dual proposta por Fulkerson em 1973, fortemente baseada em um teorema min-max. Antes de apresentarmos tal teorema, precisamos da definição de uma função de custo c, uma função de conjunto [4] z atrelada a um conjunto X chamada, portanto, de $z(X)$ e do que significa essa função de $z(X)$ ser c-viável.

Comecemos pela definição da função de custo c. Seja, $c : A \to \mathbb{R}^+$ uma função que associa um custo positivo a cada aresta do conjunto A e z uma função de conjunto, dada por:

$$z : 2^{(V-r_0)} \to \mathbb{R}^+$$

[4] Uma *função de conjunto* é um tipo de função matemática que recebe um conjunto como entrada e retorna um valor associado a esse conjunto. Em termos formais, uma função de conjunto é uma função cuja entrada pertence ao conjunto das partes ($\mathcal{P}(X)$) de um conjunto X e cujo valor de saída pertence a um conjunto Y:

$$f : \mathcal{P}(X) \to Y$$

Em outras palavras, z é uma função que associa um valor real não negativo ao subconjunto das partes X de $V - r_0$, sendo definida portanto como $z(X)$. Essa função $z(X)$ é considerada c-viável se satisfizer a seguinte condição para todas as arestas $a \in A$:

$$c(a) \geq \sum_{\substack{X \subseteq V - r_0 \\ a \text{ entra em } X}} z(X)$$

O que significa que o custo $c(a)$ de cada aresta a que entra em um subconjunto X deve ser pelo menos igual à soma dos valores $z(X)$ para todos os subconjuntos X nos quais essa aresta é uma entrada.

A partir dessas definições, Fulkerson estabelece que o custo mínimo necessário para construir uma arborescência que conecta todos os vértices ao nó raiz r_0 pode ser determinado pela maximização do somatório da função de conjunto $z(X)$, desde que essa função satisfaça a condição de c-viabilidade, de acordo com o teorema abaixo:

Teorema 3.1: O custo mínimo de uma arborescência abrangente com raiz em r_0 é dado por:

$$\max \left\{ \sum_{X \subseteq V - r_0} z(X) \;\middle|\; z \text{ é } c\text{-viável} \right\}$$

Essa equação expressa um teorema min-max, onde a minimização do custo total da arborescência é alcançada por meio da maximização da soma dos valores $z(X)$. Isso significa que o custo ótimo da arborescência pode ser obtido maximizando uma função dual z que respeita a condição de c-viabilidade. Além disso, de acordo com o autor, se os custos $c(a)$ forem inteiros, existe uma função z ótima que também assume apenas valores inteiros.

Outra forma de entender essa abordagem é pensar que o problema da determinação do custo mínimo de uma r-arborescência abrangente envolve encontrar um empacotamento máximo de cortes r-direcionados.

Pois, um corte r-direcionado é um subconjunto de arestas que, quando removido do grafo, separa o nó raiz r_0 de algum subconjunto de vértices. Um **empacotamento máximo de cortes r-direcionados** é aquele em que a soma total dos pesos atribuídos a esses cortes é a maior possível, respeitando as restrições impostas pela estrutura do grafo.

Em resumo, o principal problema tratado no artigo do Fulkerson consiste em construir um empacotamento máximo de cortes r - direcionados [5] que permita determinar o custo mínimo de uma arborescência abrangente.

Na sessão seguinte, apresentaremos a descrição do algoritmo de tempo polinomial proposto pelo autor que leva em consideração o argumento do teorema 3.1.

[5] direcionados da raíz para fora, ou seja, nenhuma aresta entra em r ou não é possível chegar em r a partir de nenhum outro vértice.

3.1 Descrição do Algoritmo

O algoritmo é composto por duas fases principais. A primeira fase tem como objetivo construir a função $z(X)$ de maneira gulosa, garantindo sua c-viabilidade, enquanto a segunda fase utiliza os valores obtidos para construir a arborescência F exclusivamente a partir das arestas de custo reduzido a zero.

3.1.1 Fase 1: Construção da Função $z(X)$

A fase 1, proposta por Fulkerson, consiste na construção da função $z(X)$ de maneira gulosa. Durante esse processo, a função de custo é ajustada repetidamente até que cada subconjunto de vértices tenha ao menos uma aresta de custo zero entrando nele. A função de custo ajustada é denotada por c', sendo que uma aresta é chamada de **0-aresta** se seu custo atualizado for:

$$c'(a) = 0$$

O procedimento consiste nos seguintes passos:

1. Escolher um subconjunto mínimo $X \subseteq V - r_0$ que ainda não tenha uma aresta de custo zero entrando nele.
2. Definir $z(X)$ como o menor custo das arestas que entram em X:

$$z(X) := \min\{c'(a) \mid a \text{ entra em } X\}$$

3. Atualizar a função de custo para todas as arestas a que entram em X:

$$c'(a) := c(a) - z(X)$$

4. Repetir os passos acima até que cada subconjunto $X \subseteq V - r_0$ tenha pelo menos uma aresta de custo zero entrando nele.

Ao final da Fase 1, garantimos que a função $z(X)$ construída seja c-viável e que todas as arestas necessárias tenham sido transformadas em 0-arestas.

3.1.2 Fase 2: Construção da Arborescência F

A fase 2, proposta por Frank, consiste na construção da arborescência F utilizando apenas as 0-arestas identificadas na fase 1. Essa fase busca garantir que todos os nós sejam conectados à raiz r_0 por meio de arestas cujo custo foi reduzido a zero na fase anterior.

O procedimento de construção de F segue os seguintes passos:

1. Iniciar a arborescência em r_0.
2. Adicionar 0-arestas ao conjunto F, garantindo que cada nova aresta conecte um novo nó à arborescência.

3. Se houver múltiplas opções de 0-arestas, escolher aquela que se tornou uma 0-aresta mais cedo na fase 1.
4. Repetir o processo até que todos os nós estejam conectados à raiz r_0.

A seguir, mostraremos um exemplo de como essa abordagem funciona.

3.2 Exemplo: algoritmo de Fulkerson

Consideremos o grafo direcionado $D = (V, A)$ representado abaixo:

- **Conjunto de vértices:**
$$V = \{r_0, v_1, v_2, v_3, v_4\}$$
- **Conjunto de arestas e seus custos:**
$$A = \{(r_0, v_1, 3), (r_0, v_2, 2), (v_1, v_3, 4), (v_2, v_3, 1), (v_2, v_4, 5), (v_3, v_4, 2)\}$$

Cada tupla (u, v, c) representa uma aresta do vértice u para v com custo c. Nosso objetivo é encontrar a **arborescência de custo mínimo enraizada em** r_0, garantindo que cada vértice tenha um caminho único partindo de r_0.

Passo 1: Definição da Função $z(X)$ e c-Viabilidade

A função de conjunto $z(X)$ deve ser construída de maneira a respeitar a c-viabilidade. Nesse exemplo, vamos construir $z(X)$ iterativamente, sempre escolhendo o menor custo entre as arestas que entram em X.

Escolha do primeiro conjunto X: Consideramos $X = \{v_1\}$ e $X = \{v_2\}$, pois são os primeiros vértices a serem conectados a r_0. Definimos:

$$z(\{v_1\}) = \min\{c(r_0, v_1)\} = 3, \quad z(\{v_2\}) = \min\{c(r_0, v_2)\} = 2.$$

Escolha do próximo conjunto X: Agora consideramos $X = \{v_3\}$, que tem duas possíveis conexões:

- De v_1 com custo 4
- De v_2 com custo 1

Escolhemos o menor custo:

$$z(\{v_3\}) = \min\{c(v_1, v_3), c(v_2, v_3)\} = 1.$$

Escolha do último conjunto X: Para $X = \{v_4\}$, temos duas opções:

- De v_2 com custo 5
- De v_3 com custo 2

Escolhemos:

$$z(\{v_4\}) = \min\{c(v_2, v_4), c(v_3, v_4)\} = 2.$$

Agora temos a função $z(X)$ construída:

$$z(\{v_1\}) = 3, \quad z(\{v_2\}) = 2, \quad z(\{v_3\}) = 1, \quad z(\{v_4\}) = 2.$$

A soma total de $z(X)$ é:

$$\sum_{X \subseteq V - r_0} z(X) = 3 + 2 + 1 + 2 = 8.$$

Pelo **teorema 3.1**, essa soma corresponde ao custo mínimo necessário para formar uma arborescência abrangente.

Passo 2: Construção da Arborescência T com Custo Mínimo

Primeiro, realizamos o processo de construção das 0-arestas da seguinte forma:

- $X = \{v_1\} \Rightarrow z(X) = 3 \Rightarrow c'(r_0, v_1) = 3 - 3 = 0.$
- $X = \{v_2\} \Rightarrow z(X) = 2 \Rightarrow c'(r_0, v_2) = 2 - 2 = 0.$
- $X = \{v_3\} \Rightarrow z(X) = 1 \Rightarrow c'(v_2, v_3) = 1 - 1 = 0.$
- $X = \{v_3\} \Rightarrow z(X) = 1 \Rightarrow c'(v_4, v_3) = 4 - 1 = 3.$
- $X = \{v_4\} \Rightarrow z(X) = 2 \Rightarrow c'(v_2, v_4) = 5 - 2 = 3.$
- $X = \{v_4\} \Rightarrow z(X) = 2 \Rightarrow c'(v_3, v_4) = 2 - 2 = 0.$

Então, construímos a arborescência T, escolhendo apenas as 0-arestas que foram minimizadas primeiro para os valores $z(X)$:

$$T = \{(r_0, v_2), (v_2, v_3), (v_3, v_4), (r_0, v_1)\}$$

Essa estrutura forma uma **arborescência abrangente enraizada em** r_0 com custo total 8, que corresponde exatamente à soma da função $z(X)$.

3.3 Implementação do Algoritmo

WIP

3.4 Correção do Algoritmo

WIP

4 Algoritmo Dual Guloso de Frank - 1979

O algoritmo de András Frank, apresentado em sua obra sobre conexões em otimização combinatória, constitui uma abordagem alternativa ao problema da busca de uma r-arborescência de custo mínimo em um dígrafo. Diferente do algoritmo de Chu e Liu (1965), que se apoia em operações de contração de ciclos, a técnica de Frank é fundamentada em um procedimento iterativo de redução de custos que combina conceitos de cortes mínimos e propriedades de grafos dirigidos.

De modo geral, o algoritmo estrutura-se em duas fases. A primeira fase é responsável por identificar, em um grafo direcionado $D = (V, A)$ com custos não negativos, um subconjunto de arestas de custo zero, construído por meio de sucessivas reduções nos pesos das arestas incidentes a determinados subconjuntos de vértices. A segunda fase, por sua vez, utiliza esse conjunto de arestas de custo zero para reconstruir uma r-arborescência de custo mínimo, garantindo conectividade a partir do nó raiz r.

4.1 Descrição do Algoritmo

Podemos descrever o algoritmo de Frank em duas fases principais:

1. **Fase 1 – Identificação de arcos de custo zero:**
   - Inicialmente, define-se o grafo auxiliar D_0, composto pelos mesmos vértices de D, mas contendo apenas arestas de custo zero.
   - Para cada vértice $v \neq r$, considera-se o conjunto X formado por v e seus ancestrais em D_0.
   - Identifica-se o corte $\delta^-(X)$, determina-se o menor custo y_v dentre essas arestas e subtrai-se y_v de todas as arestas que entram em X.
   - As arestas cujo custo atinge zero são adicionadas a A_0 e incluídas em D_0.
2. **Fase 2 – Construção da arborescência:**
   - Utilizando o conjunto A_0, inicia-se a construção da arborescência mínima T.
   - Partindo do nó raiz r, adicionam-se sucessivamente arestas de A_0 que conectam novos vértices ainda não incluídos em T.
   - Esse processo se repete até que todos os vértices sejam alcançados a partir de r, formando uma arborescência de custo mínimo.

Dessa forma, o algoritmo evita explicitamente a contração de ciclos, substituindo-a por operações de redução de custos baseadas em cortes, o que o diferencia da abordagem de Chu e Liu.

4.2 Complexidade e Implementação - 1° versão

A implementação do algoritmo de Frank pode ser realizada em tempo polinomial, com complexidade $O(n \cdot m)$, onde $n = |V|$ é o número de vértices e $m = |A|$ o número de arestas. Essa eficiência decorre da aplicação sucessiva da operação de redução de custos e da verificação de ancestrais, realizada de forma incremental.

Na prática, assim como na implementação do Chu Liu, a implementação do Frank será em Python e será feita utilizando a biblioteca `NetworkX`, que oferece suporte para operações em dígrafos, detecção de ancestrais e construção de árvores. Abaixo, apresentamos um trecho da implementação desenvolvida neste trabalho:

```
def phase1_find_minimum_arborescence(D_original, r0):
    D = D_original.copy()
    A_zero = []
    D_zero, A_zero = build_D_zero(D)
    continue_execution = True

    while continue_execution:
        continue_execution = False
        for v in D.nodes():
            if v == r0:
                continue
            X = nx.ancestors(D_zero, v)
            X.add(v)
            arcs = get_arcs_entering_X(D, X)
            min_weight = get_minimum_weight_cut(arcs)
            if min_weight:
                continue_execution = True
            update_weights_in_X(D, X, min_weight, A_zero, D_zero)
    return A_zero
```

A segunda fase é responsável pela reconstrução da arborescência a partir das arestas de custo zero:

```
def phase2_find_minimum_arborescence(D_original, r0, A_zero):
    Arb = nx.DiGraph()
    Arb.add_node(r0)
    n = len(D_original.nodes())
    for _ in range(n):
        for u, v in A_zero:
            if u in Arb.nodes() and v not in Arb.nodes():
                edge_data = D_original.get_edge_data(u, v)
                Arb.add_edge(u, v, **edge_data)
                break
    return Arb
```

Assim, o algoritmo de Frank constitui uma abordagem alternativa elegante e eficiente para a resolução do problema da r-arborescência de custo mínimo, fornecendo resultados equivalentes à técnica de Chu e Liu, mas com uma estrutura conceitual baseada em cortes mínimos.

5 O Problema da Arborescência Inversa - Frank, Hadju, 2021

WIP

5.1 Descrição do Algoritmo

WIP

5.2 Correção do Algoritmo

WIP

5.3 Implementação do Algoritmo - Versão 1

WIP

6 Conclusão

WIP

7 Referências

1. **Frank and Hajdu's Inverse Arborescence Algorithm:**
   - Frank, A., Hajdu, G. (2014). *A Simple Algorithm and Min–Max Formula for the Inverse Arborescence Problem*. *Algorithms*, 7(4), 637-647. DOI: 10.3390/a7040637.
2. **Livro de Algoritmos - Teoria de Grafos:**
   - Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C. (2009). *Introduction to Algorithms* (3rd edition). MIT Press.
   - Kleinberg, J., Tardos, É. (2006). *Algorithm Design*. Addison-Wesley.

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`.

[4] J. Kleinberg e É. Tardos. *Algorithm Design*. Addison-Wesley, 2006.