Algoritmos polinomiais para o problema de Fluxo Máximo

Notas de aula para disciplina Fluxo em Redes – Natã Goulart da Silva

Aulas dias 8, 10 e 15 e 22/04 prova.

Este documento são notas de aula tiradas do livro *"Network Flows: Theory, Algorithms, and Applications , Ahuja, Magnanti e Orlin"*, principalmente do capítulo 7, que pretende apresentar algoritmos que resolvem o problema de fluxo máximo em tempo polinomial. Um algoritmo é considerado polinomial se o consumo tempo na sua execução para resolver um problema, no pior caso, é limitado por uma função polinomial com variável igual ao tamanho da instância do problema. Um algoritmo que resolve um problema em no máximo 100 * N^4 + 30 * N^2 unidades de tempo com N sendo o tamanho da instância é polinomial. Algoritmos polinomiais são desejáveis, com exceção de algoritmos do tipo N^500.

Em um capítulo anterior foi apresentado o algoritmo "generic augmenting path" para resolver o Problema de Fluxo Máximo – PFM. O PFM consiste em uma rede com capacidade nos links, com um custo associado a cada arco c_{ij} e dados dois nós, uma fonte e um destino, enviar o máximo de fluxo possível entre este par de nós, respeitando os limites de capacidade da rede. Algoritmos da classe "augmenting path" fazem incrementos de fluxo nos caminhos que ligam os nos de origem s e o destino t, mantendo o equilíbrio de entrada e saída de fluxo nos nós intermediários. Todo fluxo que entra em um nó intermediário deve sair deste nó.

Rede Residual

Suponha que um arco (i,j) transmita inicialmente x^0_{ij} unidades de fluxo. Seja u_{ij} a capacidade do arco, podemos mandar adicionalmente u_{ij} - x^0_{ij} entre i e j neste arco. Podemos enviar até x^0_{ij} unidades de fluxo de j para i através do arco (i,j), cancelando o fluxo existente no arco. O envio de uma unidade de fluxo do nó i para o nó j aumenta o custo do fluxo em c_{ij} e o envio no sentido contrário do fluxo, decrementa este mesmo valor de custo.

Uma rede residual com relação a um fluxo x^0 inicial, possivelmente nulo, é obtida substituindo cada arco (i,j) na rede original por dois arcos (i,j) e (j,i) . O arco (i,j) tem custo c_{ij} e capacidade residual $r_{ij} = u_{ij}$ - x^0_{ij} e o arco (j,i) custo - c_{ij} e capacidade residual $r_{ij} = x^0_{ij}$. Uma rede residual $G(x^0)$ correspondente a um fluxo x^0, consiste apenas de arcos com **capacidade residual positiva**. Se em uma rede representada por um grafo direcionado existir os arcos (i,j) e (j,i), a rede residual poderá conter dois arcos paralelos do nó i ao nó j com diferentes custos e capacidades residuais e/ou

dois arcos com mesmas características no sentido oposto. Para problemas de fluxo máximo, podemos agrupar as características destes arcos em apenas um, facilitando a notação e representação. Dado um fluxo x, a **capacidade residual r_{ij}** de um arco (i,j) pertencente a **A** é a máxima quantidade de fluxo adicional que pode ser enviada do nó i para o nó j, usando arcos (i,j) ou (j,i). A capacidade residual r_{ij} tem um componente $u_{ij} - x_{ij}$ (arco (i,j)) e o fluxo corrente x_{ji} que pode ser enviado no sentido (j,i) cancelando o fluxo de (i,j). Então $r_{ij} = u_{ij} - x_{ij} + x_{ji}$. A Figura 1 apresenta o exemplo de uma rede residual.

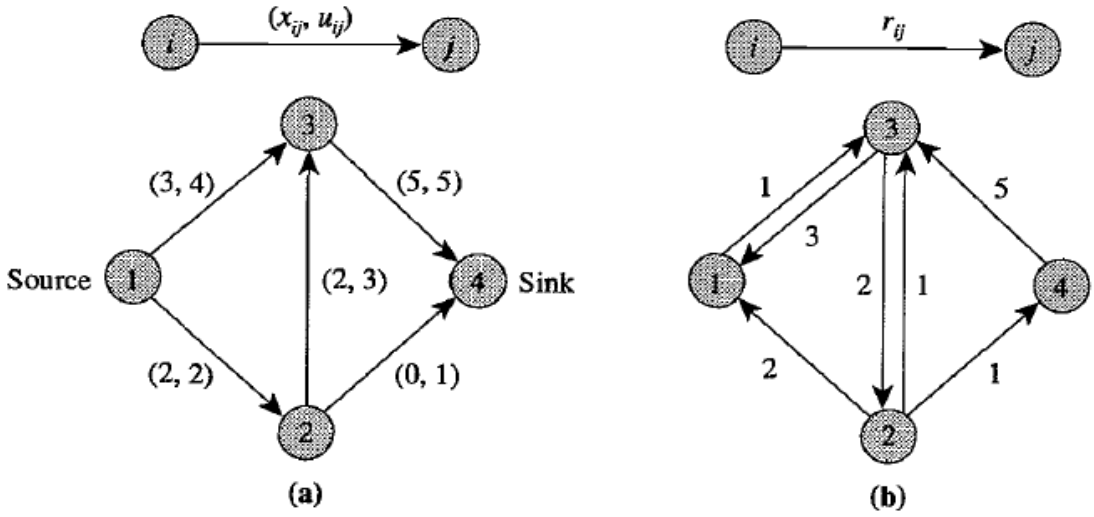


Figura1 – Exemplo rede residual

Generic Augmenting Path Algorithm

Define-se com caminho de aumento, *augmenting Path*, um caminho direto entre uma origem e um destino em uma rede residual. A capacidade residual de um caminho de aumento é igual a capacidade mínima considerando todos os arcos deste caminho. Ou seja, o máximo de fluxo que se pode enviar no caminho é limitado pelo arco com menor capacidade residual. A rede residual apresentada pela Figura 1 possui apenas um caminho de aumento, 1-3-2-4, e a capacidade residual deste caminho (sempre positiva), $\delta = \min (r_{13}, r_{32}, r_{24}) = \min \{1,2,1\} = 1$. O algoritmo Generic augmenting Path se baseia no fato de que enquanto houver capacidade residual em um caminho, é possível enviar fluxo entre a origem s e o destino t. A Figura 2 apresenta um pseudocódigo deste algoritmo.

Algoritmo *Augmenting Path*
Início
    x: = 0;
    Enquanto existir um caminho direto entre s e t **faça**
    Início
        Escolha um caminho de aumento P de s até t;

δ = min {r_{ij}:(i,j) in P};
Aumentar δ unidade de fluxo no caminho P e atualizar G(x);
Fim_enquanto
Fim_Algoritmo;

Figura 2 – Algoritmo Augmenting Path

A Figura 3 apresenta um exemplo do funcionamento do Algoritmo Generic *Augmenting Path*

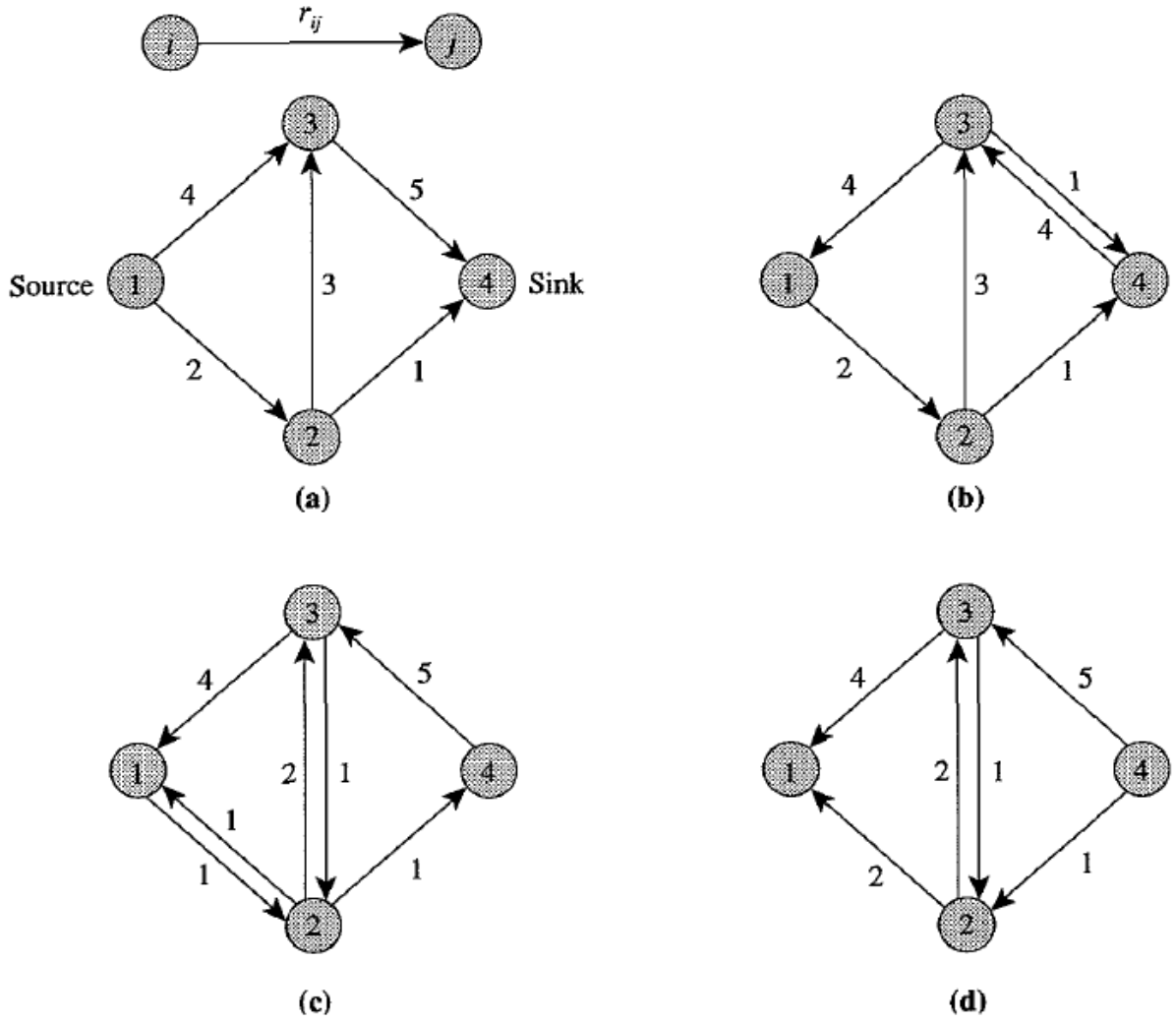


Figura 3 – Na parte b) foi selecionado o caminho 1-3-4 e foi possível enviar δ = 4 unidades de fluxo. Na parte c) foi escolhido o caminho 1-2-3-4 com δ = 1. Finalmente na parte d) após enviar 1 unidade de fluxo pelo caminho 1-2-4 a rede não contém mais caminhos de aumento e o algoritmo termina.

O algoritmo Generic Augmenting Path – GAP apresenta complexidade de pior caso de O(nmU), onde n é o número de nós, m é o número de arestas e o a máxima capacidade U. Para problemas com valor significativamente alto de capacidade, algo muito maior que n e m como U = 2^n, este algoritmo deixa de ser atrativo. Além disso, para problemas com valores de capacidade irracionais, o algoritmo pode convergir para uma solução não ótima. Então, tanto na teoria quanto na prática,

este algoritmo pode requerer muito tempo de execução em problemas com instâncias grandes, capacidades irracionais e capacidades com ordem de grandes muito superior aos parâmetros n e m.

Serão apresentados melhorias para o algoritmo e também apresentado outros algoritmos para a solução de problemas de fluxo máximo.

Teoria da decomposição de fluxo

A teoria de decomposição de fluxo mostra que, em princípio (se conhecêssemos o valor do fluxo máximo), seria possível desenvolver algoritmos de aumento de caminhos que encontram o valor do fluxo máximo em não mais que m iterações de aumento. Seja x um fluxo ótimo e x_0 o fluxo inicial. Pela propriedade 3.5 de decomposição de fluxo, podemos obter o fluxo x a partir de x_0 em no máximo m processos de aumento em caminhos de aumento que ligam os nós s e t mais fluxos passando por ciclos de aumento. Se definirmos x' como o vetor de fluxo obtido partindo do fluxo x_0 e utilizando apenas o envio de fluxo nos caminhos de aumento, x' é também um fluxo máximo. Fluxos em torno dos ciclos de aumento não alteram o valor de fluxo no nó de destino t. Por isso é possível obter o fluxo máximo usando no máximo m processo de aumento (augmentations). Para aplicar esta decomposição de fluxo seria necessário conhecer o valor do fluxo máximo que é o que procuramos. Por isso, nenhum algoritmo desenvolvido na literatura alcance o limite superior de m iterações de aumento.

Melhorando a performance do Generic Augmenting Path

Para criar algoritmos que necessitem de menos interações, menos augmentation, e assim ser mais eficiente, algumas estratégias foram utilizadas dentre as quais: Buscar aumentos de fluxo em grandes incrementos, relaxar a restrição de balanço de massa nos passos intermediários do algoritmo e relaxar que o processo de aumento de fluxo seja realizado entre a origem s e o destino t.

Para então buscar reduzir o número de iterações do Generic Augmenting Path foi desenvolvido o algoritmo Maximum Capacity Augmenting Path que busca sempre utilizar como caminho de aumento o caminho com a máxima capacidade residual. Outra opção é relaxar a exigência de máximo mas buscar um bom caminho de aumento, o que é realizado pelo algoritmo Capacity Augmenting Path. O algoritmo Preflow-Push relaxa o aumento do fluxo apenas em caminhos entre a origem e o destino e através de estratégias bem-sucedidas implementa o aumento de fluxo em caminhos intermediários. Veremos o conceito de rótulo de distâncias, Distance Labels, utilizado no desenvolvimento destes algoritmos.

Rótulos de Distância – Distance Labels

Uma função de distância d: N → Z^+ U {0} com relação a capacidade residual r_{ij} é uma função que relaciona os nós do conjunto de nós de uma rede residual com valores inteiros não negativos. Uma função de distância d é válida com relação a um fluxo x se ela satisfaz as seguintes condições:

d(t) = 0, onde t é o nó de destino;

d(i) <= d (j) + 1, para cada arco (i,j) na rede residual G(x) e d(i) é o rótulo de distância de um nó i.

Propriedade 7.1: Se os rótulos de distância são válidos, d(i) é um limite inferior do caminho mais curto direto do nó i até o no t na rede residual.

Seja $i = i_1 – i_2 – i_3 – . . . – i_k – i_{k+1} = t$ *um caminho qualquer de tamanho k do nó i até o nó t na rede residual. As condições de validade implicam que:*

$d(i_k) <= d (i_{k+1}) + 1 = d (t) + 1 = 1$

$d(i_{k-1}) <= d (i_k) + 1 <= 2$

$d(i_{k-2}) <= d (i_{k-1}) + 1 <= 3$

...

$d(i) = d(i_1) <= d (i_2) + 1 <= k$

Propriedade 7.2: Se d(s) >= n, a rede residual não contém nenhum caminho direcionado partindo do nó de origem s até o nó de destino t.

Se d(s) é um limite inferior do tamanho do caminho entre s e t na rede residual, então, nenhum caminho direto pode ter mais do que (n – 1) arcos. Além disso, se d (s) >= n, não existem caminhos diretos na rede residual que conectam s a t.

Os rótulos de distância são exatos se para cada nó i, d(i) é igual ao comprimento do caminho mais curto entre i e t. Na Figura 3, por exemplo, se o nó 1 é a origem e o nó 4 o destino, então d = (0, 0, 0, 0) é um vetor de distância de rótulos válido e d = (3, 1, 2, 0) é um vetor de rótulos das distâncias exato. Um vetor de rótulo de distâncias exato pode ser determinado pelo método de busca em largura aplicado de forma retroativa partindo no nó t, no exemplo, o nó 4.

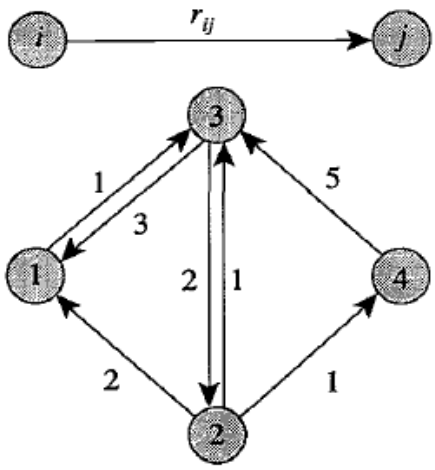


Figura 3 – Vetores de rótulos de distância

Um arco (i,j) em uma rede residual é dito admissível se ele satisfaz a condição d(i) = d(j) + 1 e inadmissível caso contrário. Um caminho admissível é aquele formado apenas por arcos admissíveis.

Propriedade 7.3.: Um caminho admissível é um caminho de aumento mais curto de uma origem s até um destino t.

Como cada arco (i,j) em um caminho admissível P é admissível, a capacidade residual do arco e *os rótulos de distância dos seus nós finais satisfazem as condições* $r_{ij} > 0$ *(caso contrário não existiria o arco) e d(i) = d(j) + 1. A primeira condição implica que P é um caminho de aumento e a segunda, que P tem k arcos, então d(s) = k. Como d(s) é um limite inferior para qualquer comprimento de qualquer caminho entre a origem s e o destino t em uma rede residual, P deve ser um caminho mais curto de aumento de fluxo.*

Algoritmo Capacity Scaling

Este algoritmo escolhe como caminho de aumento aquele(s) que possuí(em) um valor suficiente alto de capacidade residual ao invés do caminho da rede com máxima capacidade residual. Um caminho com uma alta capacidade residual pode ser obtido em O(m) execuções. Este algoritmo utiliza um parâmetro Δ com relação a um fluxo x. Uma rede Δ residual é uma rede que contém apenas arcos cuja capacidade é de pelo menos Δ e será definida por G(x,Δ). G(x, Δ) é um subgrafo de G(x) e se Δ = 1, G(x,1) = G(x). A Figura 4 apresenta uma rede residual G(x) e uma rede Δ residual para Δ = 8.

O algoritmo *Capacity Scaling* pode ser descrito pelos seguintes passos:

Algoritmo *Capacity Scaling*
1 - **Início**
2 - x: = 0;
3 - Δ:= $2^{\lfloor \log U \rfloor}$
4 - **Enquanto** $\Delta >= 1$ **faça**
5 - **Início**
6 - **Enquanto** Existir um caminho direto de s até t em G(x,Δ) **faça**
7 - **Início**
8 - Escolha um caminho P em G(x,Δ);
9 - δ:= min{r_{ij}: (i,j) in P}
10 - Aumente δ unidades de fluxo ao longo de P e atualize G(x,Δ);
11 - **Fim_enquanto**
12 - Δ:= Δ/2;
13 - Fim_enquanto
Fim_Algoritmo;

Figura 2 – Algoritmo Capacity Scaling

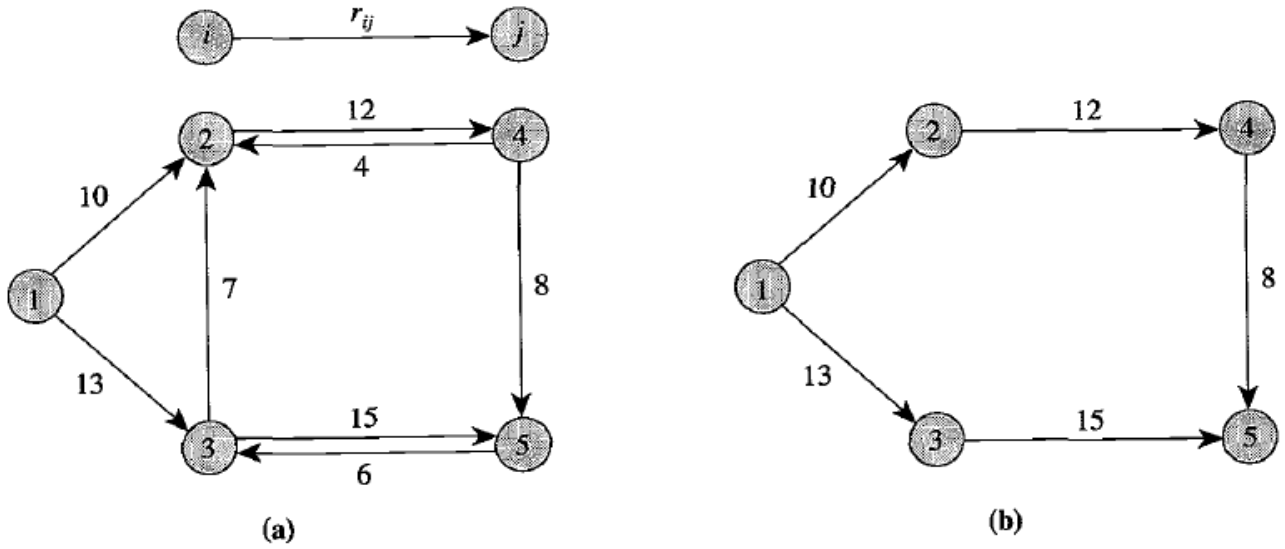


Figura 4 - Rede Δ residual

Definimos como fase de escalonamento, scaling phase, a etapa do algoritmo onde Δ permanece constante, linhas 6 a 11. Uma fase com um valor específico de Δ é chamada de Δ-scaling phase. Verifica-se que em uma Δ-scaling phase, cada aumento de fluxo ocorre com no mínimo Δ unidades de fluxo. O algoritmo se inicia com Δ:= $2^{\lfloor \log U \rfloor}$. Este valor é dividido pela metade após o término de cada Δ-scaling phase, linha 12, até que Δ = 1. Consequentemente, o algoritmo executa 1 + $\lfloor \log U \rfloor$= O(log U) scaling phases. Na última fase, Δ = 1 e G(x, Δ) = G(x) e o algoritmo termina encontrando o fluxo máximo.

A eficiência do algoritmo depende do fato que ele executa no máximo 2m aumentos em cada scaling phase. Para alcançar este resultado, consideremos x´ o fluxo no final de uma dada Δ-scaling phase e seja v' o valor deste fluxo. Seja S o conjunto de nós que podem ser atingidos a partir do no

s, origem, no grafo G(x', Δ). Como não existem caminhos de s até t em G(x', Δ), t não pertence a S. Assim, [S,$\overline{S}$] forma um corte s-t e a capacidade residual de cada arco em [S,$\overline{S}$] é estritamente menor que Δ (por definição). Então, a capacidade residual do corte [S,$\overline{S}$] é menor que mΔ. Assim, v* - v' < mΔ (propriedade 6.2 do livro). Na próxima scaling phase, cada caminho de aumento poderá transportar no máximo Δ/2 unidade de fluxo e esta fase poderá executar no máximo 2m ações de aumento, mΔ /(Δ/2).

O algoritmo Labeling, seção 6.5, necessita de O(m) para identificar uma caminho de aumento e atualizar a rede residual ao final de uma fase requer O(m). Então, o a execução do laço da linhas 6 a 11 consomem O(m) + O(m) = O(m). Podemos dizer que:

Teorema 7.4: O algoritmo Capacity Scaling resolve o problema de fluxo máximo em O(m logU) ações de aumento e é executado em O(m² logU).

Obs.: O(log U) scaling phases * O(m) para realizar o aumento * O(m) para identifica caminho de aumento e atualizar a rede.

Exercício: Resolva o problema da figura a seguir utilizando o método Capacity Scaling e utilize Δ = 8 como valor inicial. Resposta: fluxo máximo = 19

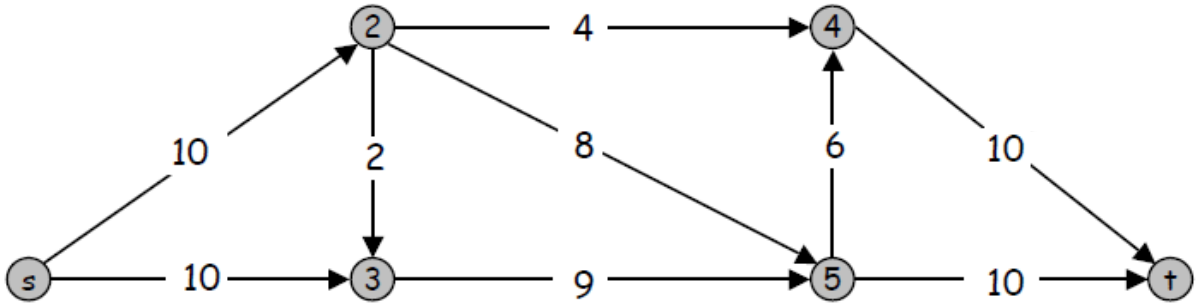


Figura 5 – Exercício Capacity Scaling

Algoritmo Shortest Augmenting Path – SAP

Este algoritmo envia o fluxo pelo menor caminho disponível entre a origem t e o destino s na rede. O algoritmo inicialmente calcula as **distance labels** através de, por exemplo, um algoritmo de busca em largura partindo no nó de destino t. Posteriormente, procura-se construir um caminho admissível, formado apenas por arcos admissíveis(d(i) = d(j) + 1), a partir da origem s. A cada iteração nesse processo de construção, é definido o **nó corrente i que inicialmente é o nó s,** verifica-se todos os arcos (i,j) na busca por um caminho admissível parcial. Caso em determinado momento um dado nó i não tenha caminhos admissíveis, é realizado um retrocesso no caminho admissível parcial até que outro caminho admissível seja encontrado ou verificando que não

existem mais caminhos admissíveis o algoritmo é encerrado.

Usaremos a Figura 6 que apresenta uma rede com os valores de **distance labels** já calculados, apresentados juntos aos nós, para descrever o funcionamento do método.

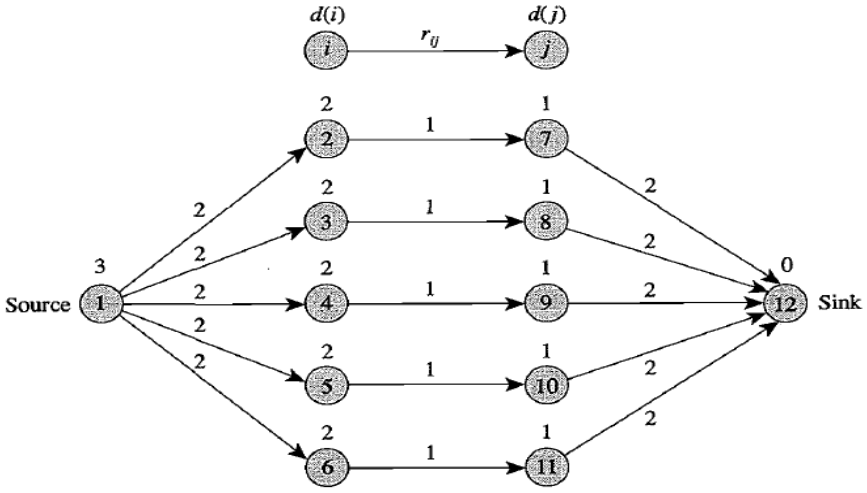


Figura 6 - Shortest Augmenting Path

Neste exemplo, a ordem de verificação dos arcos admissíveis é definida pelo menor valor de j. Começando com o nó corrente inicial i = s e caminho admissível parcial vazio, verifica-se se o arco (i,j) = (1,2) é admissível. A resposta é positiva e o arco (1,2) é adicionado ao caminho admissível parcial. Define-se como **predecessor do nó 2 o nó 1, pred(2) = 1 e o nó 2 passa a ser o nó corrente**. Verifica-se agora todos os arcos admissíveis do nó 2 e é identificado apenas o arco (2,7). O caminho admissível parcial é definido até o momento por 1 – 2 – 7 **e o nó corrente passa a ser o nó 7 e pred(7) = 2**. Na próxima iteração é adicionado o arco (7,12) e é atingido o destino. É então realizado o transporte de 1 unidade de fluxo que é a maior capacidade disponível considerando todos os arcos do caminho. A Figura 7 apresenta a rede residual após este primeiro aumento.

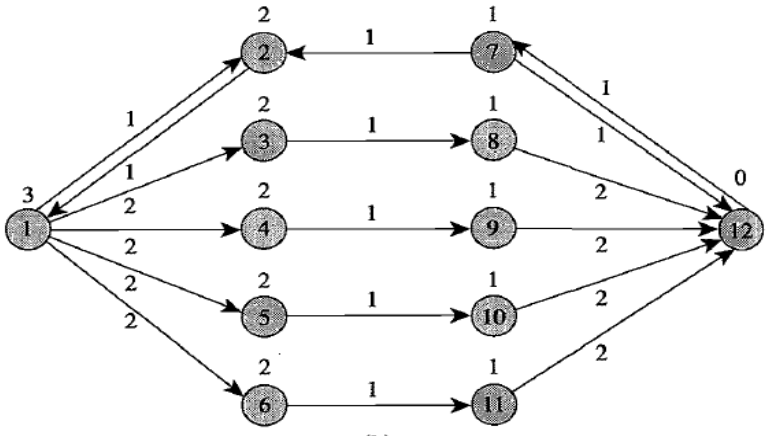


Figura 7 - Shortest Augmenting Path

O processo é iniciado novamente fazendo o nó corrente i = s. O algoritmo adiciona o arco (1,2) e o nó corrente passa a ser o nó 2. Neste caso, não existe arco admissível para i = 2 e o **distance**

label do nó 2 é incrementado. A operação realizada é: d(i) = min{d(j) + 1(i,j) in A(i) e $r_{ij} > 0$}, ou, d(2) = min{d(1) + 1} = 4. Então, nesta operação chamada **relabel, o distance label do nó 2 passa a ser 4**. Este aumento no **distance label** do nó 2 faz com que o arco (1,2) passe a ser inadmissível sendo retirado do caminho admissível parcial e não sendo mais avaliado. Os próximos caminhos admissíveis que serão gerados pelo algoritmo: **1-3-8-12, 1-4-9-12, 1-5-10-12 e 1-6-11-12.**

A seguir é apresentado o pseudocódigo do algoritmo.

Algoritmo *Shortest Augmennting Path*
1 - **Início**
2 - x: = 0;
3 - Calcular os **distance labels** para cada nó do grafo;
4 - i := s;
5 - **Enquanto** d(s) < n **faça**
6 - **Início**
7 - **Se** i tem um arco admissível **então**
8 - **Início**
9 - ***avançar(i)****;*
10 - **Se** i = t **então**
11 - ***aumentar***;
12- **i = s;**
13 - **Fim_Se**
14 - **Fim_Se**
15 - **Senão**
16 - r***etroceder(i);***
17 - **Fim_enquanto**
18- **Fim_Algoritmo**

Procedimento *avançar(i)*
1 - **Início**
2 - selecione um arco admissível (i,j) in A(i);
3 - pred(j) := i;
4 - i := j;
5 – **Fim_Procedimento**

Procedimento *aumentar*
1 - **Início**
2 - Usando os predecessores, pred, identificar uma caminho de s até t;
3 - δ:= min{r_{ij}: (i,j) in P};
4 - transportar δ unidade de fluxo de s até t;
5 - **Fim_Procedimento**

Procedimento *retroceder(i)*
1 - **Início**
2 - d(i) = min{d(j) + 1(i,j) in A(i) e $r_{ij} > 0$};
3 - Se **i <> s** então
4 - i := pred(i) ;
5 - **Fim_Procedimento**

Teorema 7.6: *O algoritmo Shortest Augmennting Path obtêm o valor de um fluxo máximo após*

um número finito de operações.

O algoritmo termina quando d(s) > = n o que implica que não existem mais caminhos de aumento da rede do nó de origem s até o nó de destino t. Então, o fluxo obtido é o fluxo máximo.

Complexidade do Algoritmos

O algoritmo tem complexidade de tempo de pior caso de $O(n^2m)$.

Propriedade 7.7: *Se são realizadas no máximo k alterações nos valores dos distance labels dos nós, o tempo total gasto para buscar arcos admissíveis na lista A(i) e alterar os rótulos dos nós é O(km)*.

Lema 7.8: *Se são realizadas no máximo k alterações nos valores dos distance labels dos nós, o valor da capacidade residual é reduzida a zero em no máximo km/2.* ***(prova página 218)***

Lema 7.9: No algoritmo SAP *os valores dos distances labels são aumentados em no máximo n vezes e o total de alterações é n * n.*

Teorema 7.10*: O algoritmo SAP tem complexidade de tempo de O (n² m).*

O número de operações de aumento para o algoritmo é de O (nm) e como cada aumento requer O (n), o custo em aumento é de O(n²m).

O algoritmo necessita de O(n²) operações para alterações de rótulos e O(n²m) aumentos. Então, O(n²) + O(n²m) = O(n²m)

Melhorando o algoritmo

O critério de parada do algoritmo, d(s) >=n, não é eficiente na prática. Verificou-se que o algoritmo passa muito tempo alterando os **distance labels** dos nós mesmo após o fluxo máximo ter sido alcançado. Será apresentada uma técnica através de um exemplo baseado na Figura 8, capaz de detectar a presença de um corte mínimo [S, $\overline{S}$] com s ∈ in S e t ∈ $\overline{S}$, consequentemente, de um fluxo máximo, encerrando o algoritmo de uma forma mais eficiente.

A Figura 8 apresenta uma rede residual do exemplo anterior, após terem sido realizados todos os aumentos de fluxo possíveis.

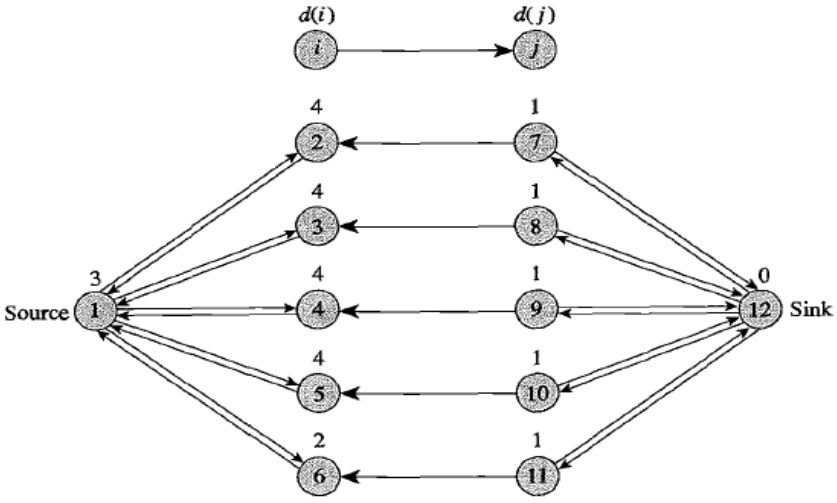


Figura 8 – Aprimorando o Shortest Augmenting Path

Na fase do algoritmo que se encontra apresentada na Figura 8, foi atingido o fluxo máximo pois não existe um caminho admissível entre s e t mas o algoritmo tradicionalmente seria encerrado apenas quando d(1) > = 12 e muitas alterações dos distance labels seriam necessárias.

A técnica utilizada para melhorar a performance do algoritmo faz uso de um vetor adicional o qual chamaremos de **labels** com n posições e índices variando de 0 a n-1. O valor presente na posição k do vetor, labels(k), representa o número de nós cujo distance label é igual a k. O algoritmo inicializa este vetor quando calcula os valores iniciais dos distance labels. Para a Figura 8, o vetor teria os seguintes valores: labels(0) =1, labels(1) =5, labels(2) =1, labels(3) =1, labels(4) =4 e o restante das entradas iguais a zero. Quando o algoritmo altera o valor de um distance label, por exemplo, passado de 2 para 3, adicionamos uma unidade na posição labels(3) e retiramos uma unidade de labels(3). Se alguma das posições do vetor que era não nula se tornar zero, o algoritmo se encerra.

O próximo passo do algoritmo a partir do estado apresentado na Figura 8 seria encontrar um arco admissível partido de i = s. Este arco seria (1,6). Fazendo i = 6, verifica-se que não existem arcos admissíveis para i. O valor do distance label de i = 6 seria alterado de 2 para 4. Isso fará com que a posição 2 do vetor labels se torne zero e o algoritmo seria encerrado.

Algoritmo Generic Preflow-Push – GPP

Seja a rede residual presente na Figura 9 especialmente definida para exemplificar um ponto fraco de algoritmos de caminhos de aumento. Neste exemplo, os algoritmos de caminhos de aumento identificariam dez caminhos de aumento, com comprimento de dez saltos e cada aumento seria de uma unidade

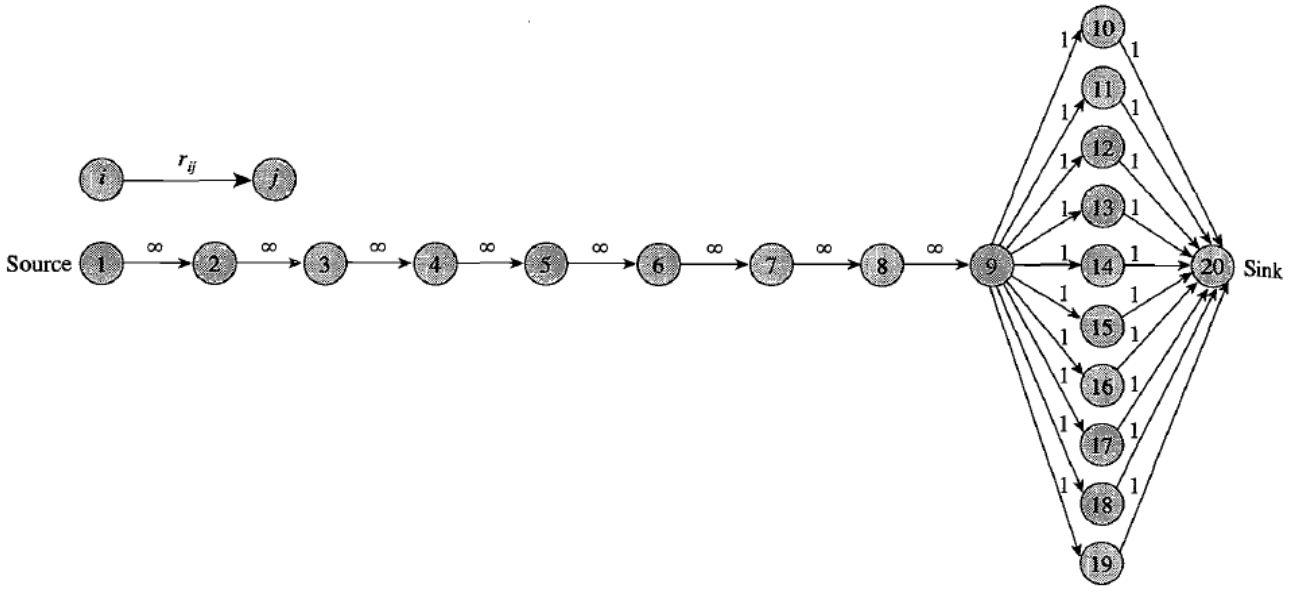


Figura 9 – Exemplos de falha de algoritmos de caminhos de aumento

Se fossem enviados dez unidades de fluxo de uma única vez entre os nós 1 e 9, haveria uma economia de tempo de execução do algoritmo. É baseado neste princípio que o algoritmo *Preflow-Push* é construído. Segundo o livro texto, esta classe de algoritmos, GPP, são mais eficientes e flexíveis do que os algoritmos de caminho de aumento. Nesta seção definimos como ação de ***push*** o processo de enviar δ unidades de fluxo ao longo de um arco. O algoritmo GPP promove operações de push nos arcos em vez de enviar fluxo por caminhos de aumento. O algoritmo GPP relaxa a restrição de balanço de fluxo nos nós intermediários da rede, devido as operações de push. Neste caso é permitido que o volume de fluxo entrando em um nó seja superior ao fluxo saindo e a este tipo de fluxo é dado o nome de **Preflow**. Podemos definir então um Preflow como uma função x: A → R que satisfaz as seguintes restrições:

$0 <= x_{ij} <= U_{ij}$ para cada arco $(i,j) \in A$.

$\sum_{\{j:(j,i) \in A\}} x_{ji} - \sum_{\{j:(i,j) \in A\}} x_{ij} >= 0$, para todo $i \in N - \{ s , t\}$

No algoritmo GPP define-se como excesso de fluxo **e(i),** a diferença entre o fluxo que entra e o que sai de um nó. Com exceção do nó s, todos os outros nós apresentam valor de **e(i) >= 0**. O **nó s tem um valor de excesso negativo**.

Definimos como nós ativos aqueles com valor de excesso positivo, ou seja, **e(i) >= 0**. Convencionamos que os nós de origem s e de destino t nunca serão ativos.

O objetivo do algoritmo GPP é retirar o excesso de fluxo em um nó ativo transportando este excesso para seus nós adjacentes que façam parte de um arco admissível. Caso não existam arcos adjacentes admissíveis, o valor do distance label do nó ativo é incrementado com o objetivo de criar pelo menos um arco admissível. O algoritmo GPP termina quando não existir mais nenhum nó ativo. Então, o algoritmo GPP parte geralmente de uma solução inviável e busca viabilizar uma solução gerando o balanço de fluxo nos nós intermediários.

O funcionamento do algoritmo GPP pode ser descrito pelas seguintes três funções:

Algoritmo *Preflow-Push*
1 - **Início**
2 - *preprocess*;//procedimento
3 - **Enquanto** *a rede contenha um nó ativo* **faça**
4 - **Início**
5 - *Selecione um nó ativo i;*
6 - *Push_relabel(i);//procedimento*
7 - **Fim_enquanto**
8- **Fim_Algoritmo**

Procedimento *preprocess*
1 - **Início**
2 - x :=0;
3 - Calcular os distances labels d(i);
4 - x_{ij} := U_{ij} para cada arco (s, j) $\in$ A(s);
5 - d(s) := n;
Fim_Procedimento

Procedimento *Push_relabel(i)*
1 - **Início**
2 - **Se** *a rede possui um arco admissível (i,j)* **então**
3 - *push* δ*:= min{e(i),* r_{ij}*} unidade de fluxo de i para j;*
4 - **Senão**
5 - *substitua d(i) por min{d(j) + 1: (i,j)* $\in$ *A(i) e* $r_{ij} > 0$*};*
Fim_Procedimento

Uma ação de push δ unidades de um nó i para um nó j diminui o valor de e(i) e r_{ij} em δ unidades e aumenta a mesma quantidade em e(j) e r_{ji}. Define-se como um push saturado se $\delta = r_{ij}$ e não saturado caso contrário. Um push não saturado em um nó i reduz e(i) a zero. Chamamos o processo de aumentar o valor do distance label de um nó de relabel que tem como objetivo criar arcos admissíveis quando estes não existirem para um determinado nó i. **No algoritmo, os excessos de fluxo no nós intermediários serão encaminhados quando possível para o nó de destino t ou serão levados de volta a origem s com as operações de relabel**.

O procedimento *preprocess* realiza as seguintes operações: i) Dá excesso de fluxo a todos os nós adjacentes ao nó s e satura todos os arcos adjacentes a s tornando-os não admissíveis. ii) Define d(s) = n e como os valores dos distance labels nunca são reduzidos, não existe uma caminho de aumento entre s e t e não será necessário/possível enviar mais unidades de fluxo a partir de s.

A rede apresentada na Figura 10 é utilizada para descrever o funcionamento do algoritmo GPP. A Figura 10(a) apresenta a rede residual inicial com os valores dos distance labels e os e(i). A Figura 10(b) apresenta o estado da rede após a execução do procedimento *preprocess*.

A Figura 10(c) apresenta o estado da rede após o algoritmo selecionar de maneira aleatória o nó ativo 2 que contém apenas um arco admissível, (2, 4). Então verifica-se que pode ser enviado δ = min{e(2), r_{24}} = min{2, 1} = 1. Esta ação de push satura o arco (2,4). Suponha que o nó 2 que continua ativo é selecionado novamente na próxima iteração. Como não existem arcos saindo deste nó, é realizada a ação de relabel com o objetivo de alterar o distance label. Então, d(2) = min {d(3) + 1, d(1) + 1} = min { 2, 5} = 2. Neste momento então, o valor do distance label do nó 2 é alterado.

Seguindo o exemplo, suponha que agora seja escolhido o nó ativo 3 e o arco (3,4) é o único admissível saindo de 3. O algoritmo consegue efetuar uma operação de push com δ = 4 que não satura o arco. Esta situação é apresentada na Figura 10(d).

Se agora for selecionado o nó 2, é possível realizar uma operação de push no arco (2, 3) com δ = min{1,3}= 1 e a situação da rede residual é a apresentada na Figura 10(e). O nó 2 deixa de ser ativo.

O último nó ativo da rede é o nó 3. É possível realizar uma operação de push com δ = min{1,1}= 1, saturando o arco (3, 4) e como não existe nenhum nó ativo, o algoritmo é encerrado. Foram enviados seis unidades de fluxo, e(4) = 4, e a situação final da rede é apresentada na Figura 10(f).

Quando o algoritmo termina, apenas há excessos nos nós de origem ou destino. Como d(s) = n, não há um caminho de aumento entre a origem e o destino e o excesso apresentado no nó de destino apresenta o fluxo máximo.

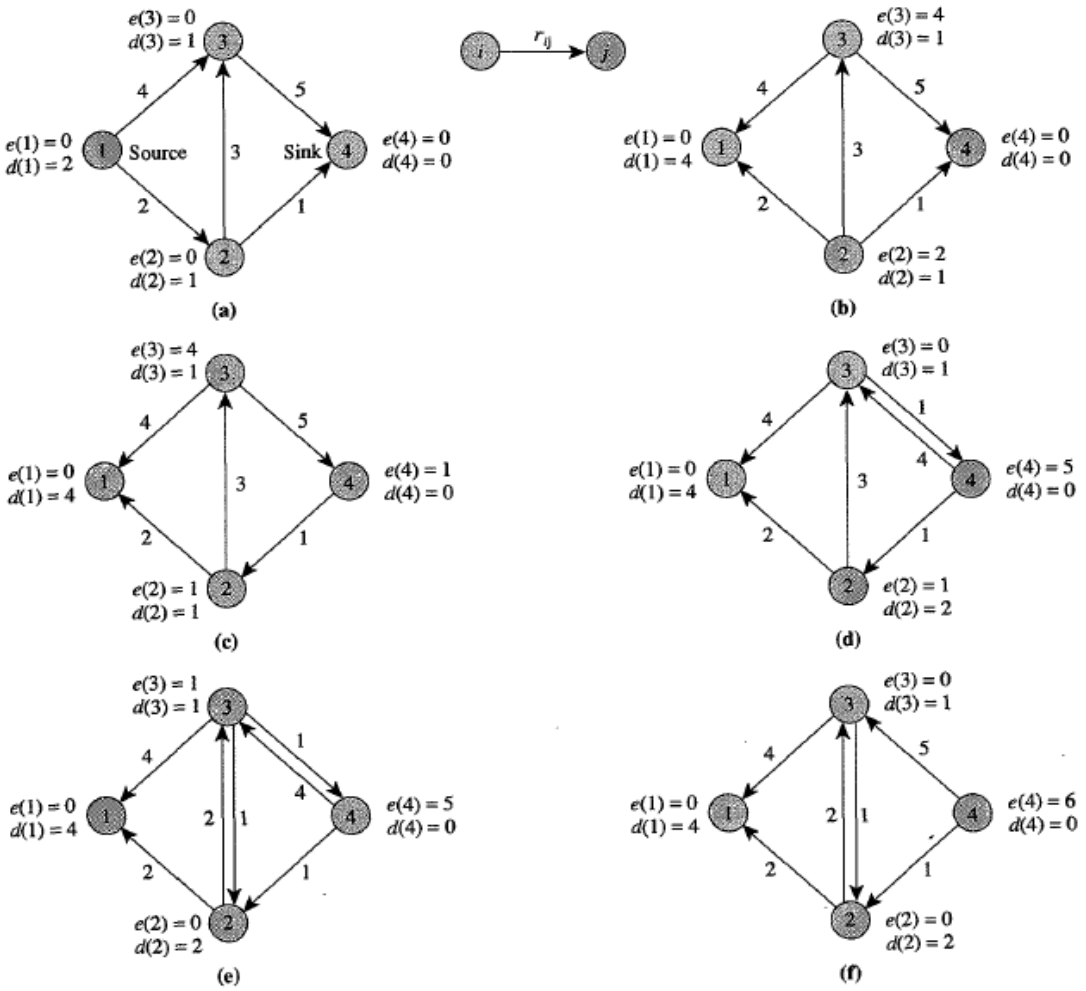


Figura 10 – Exemplos de falha de algoritmos de caminhos de aumento

Complexidade do Algoritmo

Lema 7.11: *Em qualquer fase do algoritmo GPP, cada nó com excesso positivo está conectado ao nó s por um caminho direto entre o nó i e o nó s.*

Lema 7.12: *Para cada nó* $i \in N$, $d(i) < 2n$.

Lema 7.13: *Cada distance label é aumentado em no máximo 2n vezes.Então, o número total de operações de relabel é de* $2n^2$.

Lema 7.14: *O algoritmo GPP executa no máximo nm operações de push que geram saturação.*

Lema 7.15: *O algoritmo GPP executa no máximo* $O(n^2m)$ *operações de push que* ***não*** *geram saturação.*

Teorema 7.16: *O algoritmo GPP tem complexidade de tempo de* $O(n^2m)$.

Implementações específicas para o algoritmo GPP

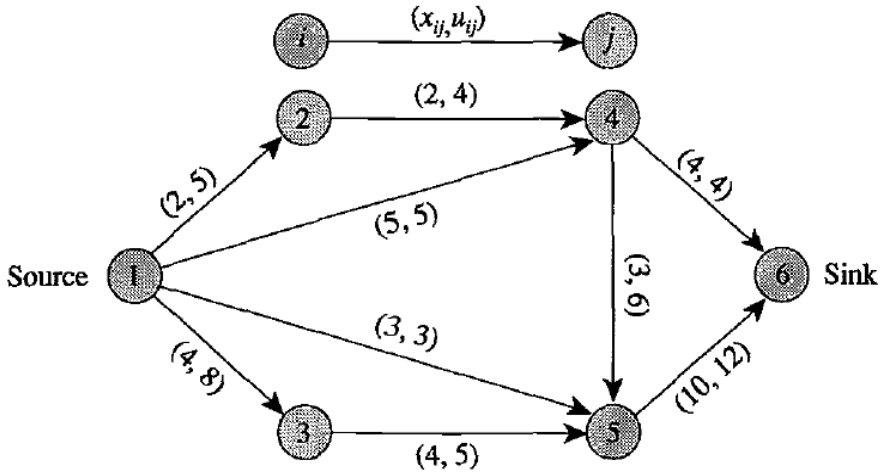
A implementação do GPP tem complexidade semelhante à do algoritmo Shortest Augmenting Path. Porém, é possível utilizar diferentes técnicas para a escolha de um entre todos os nós ativos disponíveis na rede. Três tipos diferentes de escolha levam a três complexidade de pior caso diferentes. Um ponto fraco do algoritmo GPP é quando são escolhidas operações de push que não saturam o arco envolvido. É altamente desejável reduzir o número de operações de push que não saturam um arco. A seguir são descritos de maneira breve, duas especializações do algoritmo GPP.

FIFO Preflow-Push examina os nós ativos através de uma fila onde o primeiro a entrar é examinado primeiro. A complexidade deste algoritmo é $O(n^3)$.

Highest label Preflow-Push executa as operações de push em nós com os maiores valores de distance label. Este algoritmo tem complexidade $O(n^2 m^{1/2})$ e tem melhor desempenho do que o FIFO.

Exercícios:

1) Apresente o grafo da rede residual corresponde a figura e coloque para cada nó do grafo o "distance label". Além disso, altere o fluxo nos arcos de forma que o "distance label" do nó 1 seja incrementada em 1 unidade e depois decrementada em 1 unidade sem alterar o fluxo saindo da origem.



Resposta:

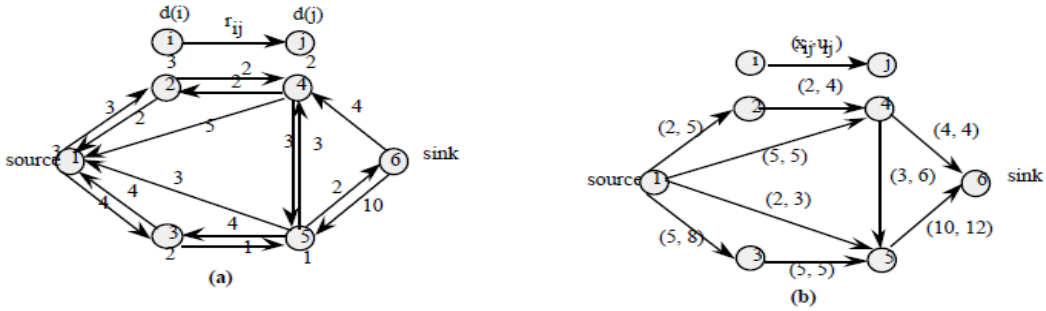
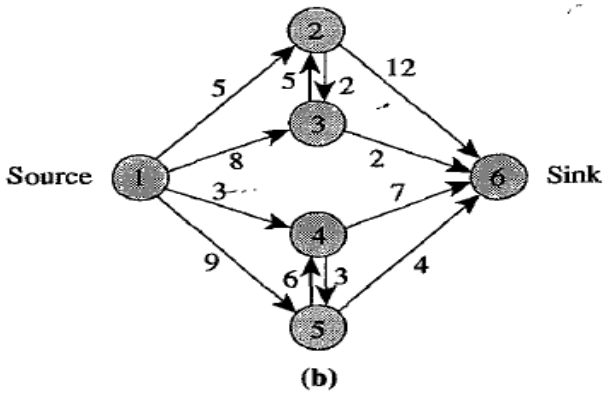


Figure S7.1

2) Determine o fluxo máximo utilizando todos os algoritmos apresentados neste material para o grafo residual da figura.



Resposta: 23 unidades de fluxo.