3
Modelagem por Grafos para Leilões de Demanda Unitária

Para que possamos analisar os diversos tipos de mecanismos existentes para leilões de demanda unitária, precisamos escolher uma maneira de modelar um leilão deste tipo. Para tal, é necessário representarmos os consumidores, os bens e os lances, que podem ser vistos como uma tripla ordenada $(consumidor, bem, valor)$.

Neste capítulo, descrevemos como podemos usar a teoria de grafos para representar um leilão de demanda unitária. Grafos são muito utilizados em matemática e ciência da computação para modelar diversos tipos de problemas, desde a solução do caminho mais curto (ou mais barato) entre dois pontos de um mapa utilizando as estradas existentes até decidir se é possível obter lucro através da realização de uma seqüência de operações cambiais.

Um grafo não-direcionado, $G = (V, E)$, é representado por um conjunto V de vértices, ou nós, e um conjunto E de arestas. Cada aresta é um par *não-ordenado* de dois vértices pertencentes a V. Pode-se pensar na representação gráfica de um grafo não-direcionado como um conjunto de pontos (os vértices) interligados por linhas (as arestas). Por simplicidade, consideraremos que não podem haver laços, isto é, arestas ligando um nó a si próprio, nem arestas paralelas, isto é, duas ou mais arestas que conectam um mesmo par de vértices. Em grafos não-direcionados, podemos nos movimentar a partir de um vértice utilizando qualquer aresta incidente nele.

Em um grafo direcionado, as arestas, ao invés de pares não-ordenados, passam a ser pares *ordenados* de vértices. Dessa maneira, só é permitido se mover a partir de um vértice utilizando arestas que "saiam" deste vértice. A representação gráfica de um grafo direcionado é muito parecida com a de um grafo não-direcionado, porém, as linhas que representam as arestas devem ter uma seta indicando a direção da aresta. Mais uma vez, consideraremos que não existem laços nem arestas paralelas, cuja definição, no caso de grafos direcionados, passa a ser arestas com mesma origem e mesmo destino.

O grau de um vértice é igual ao número de arestas que incidem no vértice. O grau de entrada de um vértice indica o número de arestas que

"entram" no vértice e, semelhantemente, o grau de saída indica o número de arestas que "deixam" o vértice. No caso de grafos não-direcionados, o grau, o grau de saída e o grau de entrada têm o mesmo valor. No caso de grafos direcionados, o grau de um vértice é igual ao grau de entrada mais o grau de saída. É fácil perceber que a soma dos graus de todos os vértices de um grafo é exatamente igual ao dobro do número de arestas.

Cada vértice e cada aresta de um grafo pode ter um peso, ou custo, associado a eles. Este peso normalmente é utilizado para representar algum valor do problema sendo modelado, como, por exemplo, o peso de uma aresta pode ser equivalente ao tamanho de uma rua.

Um caminho em um grafo é uma seqüência de vértices, com a condição de que deve haver uma aresta que permita se mover de cada um dos vértices ao vértice seguinte da seqüência. Quando nenhum vértice do caminho se repete, dizemos que o caminho é simples. O custo de um caminho é, em geral, a soma dos custos das arestas utilizadas no caminho, sendo que arestas repetidas devem ser contadas múltiplas vezes. O comprimento de um caminho é igual ao número de arestas que pertencem ao caminho sendo que, novamente, arestas utilizadas mais de uma vez são contadas múltiplas vezes.

Quando um caminho começa e termina em um mesmo vértice, dizemos que o caminho é um ciclo, ou circuito. Se nenhum vértice se repete, a exceção do primeiro e do último, evidentemente, e o caminho tem comprimento maior ou igual a 3, o ciclo é dito simples. Um grafo é dito acíclico quando não possui nenhum ciclo.

Dizemos que um grafo é conexo quando existe um caminho entre quaisquer dois vértices distintos pertencentes ao grafo.

Um algoritmo de busca para um grafo, é um algoritmo que, começando em um vértice qualquer do grafo, passeia pelas arestas do grafo de forma a "visitar" todos os vértices que são alcançáveis a partir do nó inicial. Geralmente, algoritmos de busca são classificados de acordo com a ordem em que os vértices são visitados, sendo que os dois mais importantes são o de *busca primeiro em profundidade* e o de *busca primeiro em largura*.

Em termos simples, a busca primeiro em profundidade, ou, simplesmente, busca em profundidade, visita os "filhos" de um nó antes de visitar seus "irmãos". É o tipo de busca mais utilizada por ser de implementação muito mais simples que a busca em largura. A busca ocorre chamando o procedimento DFS a seguir passando um grafo G e um vértice $v \in V$:

DFS(G,v)
Para cada vértice $u \in V$
$visit[u] = 0$
$DFSAux(v)$

DFSAux(v)
Processe o vértice v
$visit[v] = 1$
Para cada vértice u tal que $e = (v, u) \in E$ e $visit[u] = 0$
$DFSAux[u]$

A busca primeiro em largura, ou busca em largura, se diferencia pelo fato que antes de visitar o "filho" de um nó, todos os seus "irmãos" são visitados. Isso pode ser obtido mantendo-se uma fila global indicando a ordem em que os nós devem ser visitados. Sempre se deve visitar o primeiro nó da fila e, quando um nó está sendo visitado, todos os "filhos" deste nó que ainda não foram visitados nem adicionados à fila, devem ser colocados ao final desta.

Um grafo $G' = (V', E')$ é dito um subgrafo de $G = (V, E)$ se:

1. $V' \subset V$;
2. $E' \subset E$; e
3. Se $(i, j) \in E'$, então $i \in V'$ e $j \in V'$.

Um grafo é dito completo quando existem arestas ligando todos os pares de vértices do grafo. Um grafo é bipartido quando é possível dividir o conjunto de vértices em dois, tal que não exista aresta conectando dois vértices pertencentes ao mesmo conjunto. Finalmente, um grafo é bipartido completo, se há arestas ligando todos os vértices de um grupo a todos os vértices do outro grupo.

Um conjunto de arestas M é um emparelhamento para o grafo $G = (V, E)$ quando $M \subset E$ e M não possui nenhum par de arestas adjacentes entre si. Duas arestas são adjacentes se ambas incidem em um mesmo vértice. O emparelhamento é dito perfeito quando todos os vértices de V são tocados por uma e somente uma aresta de M.

Mostraremos agora como podemos utilizar grafos para modelar um Leilão de Demanda Unitária. Veremos também como utilizarmos um subtópico da teoria de grafos, conhecido como Redes de Fluxo, para que possamos implementar nossos mecanismos de uma forma computacionalmente eficiente.

Utilizando grafos, podemos representar os consumidores e os bens utilizando vértices, enquanto os lances são representados por arestas. Como sugere a intuição, o lance do consumidor i pelo produto j deve ser representado por uma aresta entre o vértice que representa o consumidor i e o que representa o bem j. O peso da aresta pode variar, de acordo com o tipo de algoritmo que será utilizado mas deve ser alguma função do valor do lance. Para efeitos de modelagem, o peso da aresta é igual ao valor do lance. Para que não haja arestas paralelas, para cada par bem-consumidor consideramos apenas o lance de maior valor do consumidor pelo bem.

Como não haverá arestas ligando dois consumidores entre si e nem dois produtos entre si, o grafo é bipartido. Considerando que todo o consumidor oferece um lance de valor zero por todos os bens (esta presunção não prejudica os consumidores de nenhuma forma), passamos a ter um grafo bipartido completo.

Embora modelar o leilão como um grafo nos ajude a visualizar o problema, isto ainda não é suficiente para que possamos projetar algoritmos que sejam capazes de executar os mecanismos descritos anteriormente, tampouco os mecanismos que serão descritos no próximo capítulo. Para tal, como mencionado anteriormente, utilizaremos um sub-tópico da teoria de grafos conhecido como redes de fluxo, mais especificamente, nos deteremos em redes de fluxo unitárias.

3.1 Redes de Fluxo Unitárias

Uma rede de fluxo é um grafo direcionado onde as arestas possuem um peso e uma capacidade. O peso de uma aresta é equivalente ao custo de atravessar uma unidade de fluxo através dela. A capacidade de uma aresta indica a quantidade máxima de fluxo que pode atravessar uma aresta em qualquer instante de tempo, sendo que este fluxo sempre deve possuir a mesma orientação da aresta. Quando todas as arestas têm capacidade igual a um, a rede de fluxo é dita unitária.

Dado que os algoritmos para redes de fluxo sempre são válidos para o caso unitário e, de fato, muitos deles possuem uma melhor complexidade computacional nesse caso, e, como veremos, é bastante intuitiva a modelagem de um leilão de demanda unitária com a restrição de que todas as capacidades devem ser iguais a um, concentraremos nossos estudos apenas no caso específico das redes de fluxo unitárias.

Um fluxo em rede é uma atribuição de valores de fluxos para as arestas da rede. A soma dos fluxos das arestas que chegam em um vértice deve ser

igual à soma dos fluxos das arestas que deixam este mesmo vértice (restrição de conservação de fluxo), exceção feita aos nós *fonte* e *sumidouro*, que são capazes de "injetar" e de "remover" fluxo, respectivamente.

Mais formalmente, de acordo com (04, 01), uma rede de fluxo é um grafo orientado $G = (V, E)$, onde toda aresta $e = (u, v) \in E$ possui uma capacidade não-negativa $c(e) \geq 0$. Se (u, v) não pertence a E, então assumimos que $c(u, v) = 0$. Existem dois vértices especiais, o nó fonte s e o sumidouro t. O grau de entrada do nó fonte s é igual a zero e, de forma equivalente, o grau de saída do nó sumidouro t também é zero. Consideramos que para qualquer nó u do grafo, existe um caminho entre s e t que passe por u. Um fluxo em G é uma função $f : V \times V \rightarrow \mathcal{R}$ que satisfaz as seguintes restrições:

$$\sum_{v \in V} f(u, v) = 0, \text{ para todo } u \in V - \{s, t\} \text{ (Conservação de Fluxo)}$$

$$f(u, v) \leq c(u, v), \text{ para todo } e = (u, v) \in E \text{ (Restrição de Capacidade)}$$

$$f(u, v) = -f(v, u), \text{ para todo } e = (u, v) \in E \text{ (Simetria Oblíqua)}$$

O valor do fluxo f é definido como:

$$|f| = \sum_{v \in V} f(s, v),$$

isto é, o fluxo que flui do nó fonte s até o sumidouro t é igual ao fluxo que "deixa" o nó fonte. O fato que este fluxo "chega" ao sumidouro é garantido pelas três restrições acima.

Sendo $w_{u,v}$ o peso da aresta (u, v), o custo w de um fluxo em rede f é definido por:

$$w = \sum_{e=(u,v) \in E} f(u, v) * w_{u,v}.$$

Algumas modificações são necessárias no grafo da seção anterior que modelava um leilão de demanda unitária para que este seja uma rede de fluxo unitária. Para tal, precisamos decidir a direção de cada aresta, bem como determinar quais são os nós fonte e sumidouro. Uma maneira simples e bastante intuitiva de realizarmos isso é a seguinte:

– As arestas que representam os lances deixam o grupo de consumidores e chegam no grupo de bens, sendo que o peso dessas arestas deve ser uma função do valor de cada lance;

- Devemos criar um nó fonte s capaz de injetar até v unidades de fluxo na rede, onde v é a quantidade máxima de bens que se deseja vender;
- Para cada consumidor, conectamos uma aresta de peso zero que deixa s e chega ao consumidor;
- Criamos também um nó sumidouro t capaz de absorver v unidades de fluxo; e
- Para cada bem, ligamos uma aresta de peso zero que deixa o bem e chega em t.

Evidentemente, a capacidade de todas as arestas do grafo deve ser igual a um. Apenas a título de curiosidade, descrevemos a seguir o significado da capacidade de cada aresta, e como poderíamos utilizar a mesma rede com capacidades diferentes para modelar outros tipos de leilões:

- A capacidade das arestas que representam os lances pode modelar o número máximo de cópias do bem no qual o consumidor está interessado;
- A capacidade das arestas que ligam o nó s aos consumidores representa o número de bens que cada consumidor pode adquirir; e
- A capacidade das arestas que conectam cada bem ao vértice t indica o número de cópias existentes de cada bem.

Da maneira como projetamos a rede, enviar x unidades de fluxo do nó fonte para o sumidouro equivale a fazer x atribuições válidas bem-consumidor. Um conjunto de atribuições é válida se ela respeita as regras do leilão, que, neste caso, são de que nenhum produto pode ser vendido a dois consumidores diferentes e de que nenhum consumidor pode comprar dois produtos diferentes nem mais do que uma cópia de um mesmo produto.

A seguir descrevemos alguns conceitos e alguns problemas que podem ser abordados (e resolvidos) utilizando a teoria de de fluxo em redes.

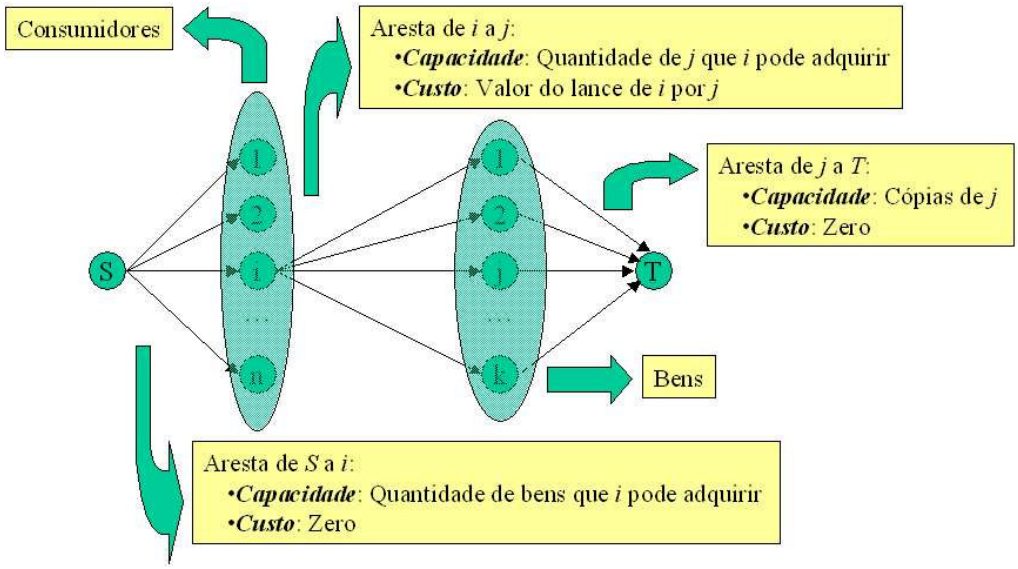

Figura 3.1: Utilizando grafos de fluxo para modelar um leilão

3.1.1 Redes Residuais

Uma rede residual para um fluxo em rede pode ser vista como uma rede de fluxo em que a aresta (u, v) só está presente caso seja possível, no fluxo em rede original, enviar fluxo de u para v, ou, alternativamente, deixar de enviar fluxo de v para u.

Dado um fluxo em rede, é possível derivar a rede residual correspondente com os seguintes passos:

- Os vértices das duas redes são os mesmos;
- Para cada aresta com capacidade c, fluxo corrente $f > 0$ e custo p, criar uma aresta de direção oposta a da rede original, com capacidade f e custo $-p$; e
- Para cada aresta com capacidade c, fluxo corrente $f < c$ e custo p, criar uma aresta de mesma direção que a da rede original, com capacidade $c - f$ e custo p.

Repare que é possível termos uma aresta na rede residual realizando uma ligação inexistente na rede original (uma aresta que deixa um bem j e chega em um consumidor i, por exemplo). Caso passemos fluxo por esta "nova" aresta, isto significa que estamos deixando de atribuir j para i.

Vale ressaltar que sempre que houver modificação no fluxo da rede original, devemos recalcular a rede residual. De fato, isto pode ser simplificado modificando apenas as arestas nas quais houve mudança de fluxo.

A grande vantagem da utilização de redes residuais é que este artifício torna muito mais simples a elaboração e a implementação de algoritmos que precisam operar sobre redes de fluxo. Em geral, os algoritmos precisam encontrar um caminho entre os nós s e t na rede residual, sem ter que se preocupar com o fluxo atual e a capacidade de cada aresta.

3.1.2 Caminhos Aumentantes

Um caminho aumentante de um fluxo em rede é qualquer caminho que exista entre s e t na rede residual correspondente. A capacidade de um caminho aumentante é igual a menor capacidade das arestas que pertencem ao caminho. No nosso caso, como todas as capacidades são unitárias, a capacidade de qualquer caminho aumentante sempre será igual a um. O custo de um caminho aumentante é igual à soma dos custos das arestas que fazem parte do caminho. Cada vez que enviamos uma unidade de fluxo por um caminho aumentante, elevamos o custo de nosso fluxo em uma quantidade igual ao custo do caminho aumentante e acrescemos uma unidade ao valor do fluxo.

3.1.3 Fluxo Máximo

Como o próprio nome sugere, o fluxo máximo numa rede de fluxo é a maior quantidade de unidades de fluxo que é possível enviar do nó fonte s para o nó sumidouro t. Ele é muito utilizado para calcular tamanhos de emparelhamentos máximos de grafos bipartidos.

Uma maneira simples de calcular o fluxo máximo é repetidamente encontrar um caminho aumentante e enviar fluxo através dele. Quando não existir mais nenhum caminho aumentante, significa que já encontramos o fluxo máximo que pode ser enviado de s a t. Quando utilizamos um mecanismo de busca qualquer para encontrar os caminhos aumentantes, temos o algoritmo de *Ford-Fulkerson* (04). Quando é utilizada busca em largura, temos o algoritmo de *Edmonds-Karp* (04), que apresenta uma melhor complexidade. Para o caso de redes com capacidades unitárias, no entanto, usar tanto busca em largura quanto busca em profundidade resulta na mesma complexidade computacional.

3.1.4 Fluxo de Custo Mínimo

Um fluxo de valor f é dito de custo mínimo quando não existe nenhum outro fluxo de mesmo valor e menor custo. Logo, para buscar fluxos de custo mínimo é necessário antes estabelecer o valor do fluxo desejado, sendo que este nunca deve exceder o tamanho do fluxo máximo.

Calcular um fluxo de custo mínimo é útil quando desejamos, por exemplo, encontrar o emparelhamento de menor custo de um grafo bipartido.

Existem diversos algoritmos para calcular o fluxo de custo mínimo. Aqui estudaremos dois desses algoritmos, o de caminhos mais curtos sucessivos, que começa com um fluxo de valor zero e o aumenta repetidamente até atingir o valor de fluxo desejado, e o de eliminação de ciclos negativos, que recebe um fluxo qualquer de valor igual ao desejado e busca diminuir o custo deste fluxo até encontrar o fluxo de custo mínimo. Para um aprofundamento no tema, recomenda-se a leitura de (01).

Caminhos Mais Curtos Sucessivos

O algoritmo de caminhos mais curtos sucessivos deve receber como parâmetros uma rede de fluxo e um valor de fluxo alvo f, menor ou igual ao fluxo máximo que pode ser transmitido pela rede em questão. Por simplicidade, como fizemos até aqui, assumiremos que existe apenas uma fonte e um sumidouro, já que redes com várias fontes e vários sumidouros podem, através de uma transformação simples, sempre ser representadas utilizando apenas uma fonte e um sumidouro (ver (04, 01)).

A idéia por trás do algoritmo é bastante simples. A cada passo é calculada a rede residual correspondente ao fluxo em rede corrente de valor f' e, em seguida, utilizando os pesos das arestas da rede residual, busca-se o caminho de menor peso entre s e t. Encontrado este caminho, calcula-se a menor capacidade c de uma aresta pertencente ao caminho e são enviadas $\min\{c, f - f'\}$ unidades de fluxo através do caminho encontrado. Repete-se o algoritmo até alcançarmos o valor de fluxo desejado.

Como estamos nos atendo a redes de fluxo unitárias, sabemos que todo caminho encontrado terá capacidade igual a um. Também sabemos, pelas características do nosso grafo, que o valor do fluxo máximo será o menor dentre o número de bens e de consumidores. Logo, é fácil perceber que este algoritmo de fato termina.

Para finalizarmos a análise do algoritmo, portanto, falta determinarmos a complexidade de encontrarmos o caminho de menor custo entre s e t. Devemos observar que, por construção, é possível que tenhamos arestas de

peso negativo na rede residual, logo, precisamos utilizar um algoritmo de caminho de menor custo que possa lidar com essa situação. Uma escolha natural é o algoritmo de *Bellman-Ford* (04), que encontra tal caminho em $O(|V||E|)$. Sendo n o número de consumidores e m o número de bens, significa que cada iteração do algoritmo tem complexidade $O(n^2.m + m^2.n)$. Como podem ser necessárias até $\min\{n, m\}$ iterações, a complexidade final será $O(m^2.n^2)$.

Sabemos, no entanto, que utilizando o algoritmo de *Dijkstra* (04), considerando que o nosso grafo é denso (número de arestas próximo ao quadrado do número de vértices) e dado que tivéssemos uma maneira de mudar os pesos das arestas de forma que fossem todos eles não-negativos, buscar o caminho de menor peso entre s e t na rede residual teria complexidade $O(n^2 + m^2)$. Sendo assim, a complexidade final seria $O(\min\{n^2.m, m^2.n\})$.

De fato, existe uma correção no peso das arestas que faz com que todas tenham pesos positivos ao mesmo tempo que garante que o caminho encontrado pelo algoritmo de Dijkstra é mínimo mesmo quando considerados os pesos originais. Esta correção se dá pela introdução do conceito de potenciais dos nós: sendo p_i o potencial do nó i, p_j o potencial do nó j e $w(i, j)$ o peso da aresta (i, j) na rede residual, o peso corrigido da aresta (i, j) é $w(i, j) - p_i + p_j$. Todos os nós começam com potencial zero e, quando é enviado fluxo de s a t pelo caminho de menor custo encontrado, usando os pesos corrigidos, deve-se atualizar os potenciais da seguinte maneira:

- Terminada a execução do algoritmo de Dijkstra, terá sido encontrado um caminho de peso mínimo de s a qualquer nó do grafo;
- Considere que w'_i é igual ao custo (considerando pesos corrigidos) do caminho de menor peso entre s e i encontrado pelo algoritmo de Dijkstra, sendo que, evidentemente, w'_s é zero; e
- O potencial de i deve ser atualizado subtraindo-se dele o valor w'_i, assim o novo potencial do nó i passa a ser igual a $p_i - w'_i$.

Repare que, para podermos atualizar os potenciais com sucesso, é necessário que continuemos o algoritmo de Dijkstra mesmo após o caminho de menor custo entre s e t ter sido encontrado. É possível, porém, fazer uma ligeira modificação que, embora não melhore a complexidade computacional do algoritmo, melhora seu tempo de execução na prática:

- Interrompa a execução do algoritmo de Dijkstra assim que tiver sido encontrado o caminho de menor custo entre s e t;

– Quando da interrupção, caminhos mais curtos de s a outros nós possivelmente terão sido encontrados e, para estes nós, deve ser aplicada a mesma regra anterior de atualização dos potenciais; e

– O potencial de qualquer nó i para o qual não foi encontrado um caminho de custo mínimo a partir de s deve ser atualizado subtraindo-se dele o valor w'_t, isto é, o custo do caminho entre s e t, ao invés de w'_i.

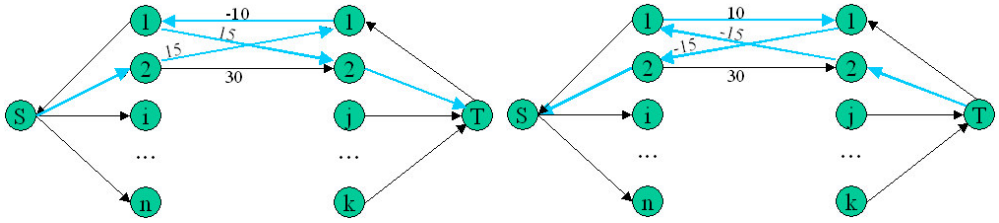

Figura 3.2: Caminhos mais curtos sucessivos

Para uma prova da corretude deste algoritmo ou para um aprofundamento no tema, recomenda-se a leitura de (01).

Eliminação de Ciclos Negativos

Este algoritmo baseia-se no teorema que afirma que um fluxo é de custo mínimo se e somente se sua rede residual não possui ciclos de custo negativo (01). A idéia deste algoritmo é, portanto, encontrar sucessivamente ciclos negativos na rede residual de um fluxo em rede que já possui o valor desejado, ou seja, este algoritmo não altera o valor do fluxo enviado entre s e t.

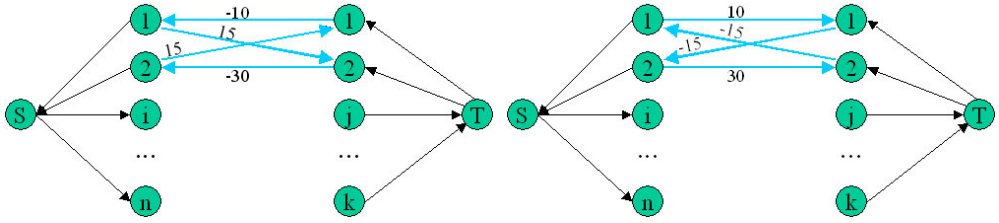

Figura 3.3: Eliminação de ciclos negativos

Como consideramos o caso em que só existe um sumidouro, sabemos que, caso exista um ciclo de custo negativo, este pode ser detectado executando o algoritmo de Bellman-Ford na rede residual para encontrar os caminhos de custo mínimo para todos os nós partindo do sumidouro. Encontrado um ciclo qualquer de custo $w < 0$ na rede residual, enviar fluxo por este ciclo significa reduzir o custo total do nosso fluxo em rede de exatamente

w. Após enviar fluxo pelo ciclo encontrado, deve-se repetir o algoritmo até que não seja mais possível encontrar um ciclo de custo negativo.

Apesar da complexidade computacional deste algoritmo ser muito pior que a do algoritmo de caminhos mais curtos sucessivos, em algumas situações ele pode apresentar um melhor desempenho. Resumidamente, deve-se considerar a possibilidade de utilização deste algoritmo quando temos um fluxo em rede de custo muito próximo ao mínimo como, por exemplo, quando removemos um nó de um fluxo de rede de custo mínimo.