O Problema do Fluxo Máximo de Custo Mínimo e suas Aplicações no Transporte de Recursos em Sistemas Distribuídos

Luís Fernando Schultz Xavier da Silveira
Filipe Hoss Lellis

27 de Novembro de 2008

Resumo

O problema do fluxo máximo de custo mínimo é bastante famoso em pesquisa operacional, logística e, principalmente, ciência da computação, pois é um problema que generaliza diversos outros ligados à teoria de grafos, inclusive o de obter um caminho de custo mínimo e o de encontrar um fluxo máximo. Além disso, existem algoritmos polinomiais assintoticamente eficientes para resolvê-lo, muito embora esses sejam uns dos mais complexos em teoria de grafos. Nosso trabalho será mostrar como esse problema pode ser ajustado para resolver um problema de otimização que costuma aparecer em sistemas distribuídos, o de balancear a carga nos nós do sistema.

1 O Problema do Fluxo Máximo

Seja $G = (V, E)$ um grafo direcionado, s (a fonte) e t (o sorvedouro) dois vértices distintos de G e $c : E \to \mathbb{R}^+$ uma função real não negativa representando as capacidades das arestas. Estenda c para $V \times V$ fazendo $c(u, v) := 0$ sempre que $(u, v) \in V \times V \setminus E$. Um fluxo f em G é uma função real $f : V \times V \to \mathbb{R}$ satisfazendo

- $f(u, v) \leq c(u, v)\ \forall (u, v) \in V \times V$ (respeito às capacidades)
- $f(u, v) = -f(v, u)\ \forall (u, v) \in V \times V$ (antissimetria)
- $\sum_{v \in V} f(u, v) = 0\ \forall u \in V \setminus \{s, t\}$ (conservação da "matéria")

Abusamos a notação e definimos $f(A, B) := \sum_{u \in A} \sum_{v \in B} f(u, v)$ se $A, B \subseteq V$ forem dois conjuntos de vértices. Ainda, definimos $f(u, B) := f(\{u\}, B)$ e $f(A, v) := f(A, \{v\})$ caso $u, v \in V$ sejam vértices quaisquer.

Dessa forma, definimos o valor do fluxo f como $|f| := f(s, V)$. Naturalmente, dizemos que um fluxo f^* é máximo quando quer que para todo fluxo f, $|f| \leq |f^*|$. O problema do fluxo máximo consiste em, dado (G, s, t, c), encontrar um fluxo máximo f^*.

Ao leitor que deseja uma visão mais abrangente das noções básicas de teoria de grafos, indicamos [0]. Nessa referência também serão encontradas propriedades de fluxos não citadas aqui com rigorosa demonstração.

Diversas são as razões que tornam esse problema interessante. Uma delas é que não só ele generaliza o problema do emparelhamento máximo em grafos bipartidos, mas também o algoritmo atualmente mais eficiente para esse problema, o de Hopcroft e Karp [1], foi desenvolvido com as idéias por trás do fluxo máximo em mente. [0] mostra a relação precisa entre esses dois problemas.

Para entender uma outra razão e já projetar nossa discussão adiante, devemos examinar os conceitos de cortes e de caminhos aumentantes. Um corte de um grafo G é uma partição $V = S \cup T$ dos vértices de forma que $S \cap T = \{\}$, $s \in S$ e $t \in T$. O valor de um corte $C = (S, T)$ é definido como $|C| := \sum_{u \in S} \sum_{v \in T} c(u, v)$. Uma propriedade interessante dos cortes é que para qualquer fluxo f, $|f| = f(S, T)$. Essa e outras propriedades de cortes encontram-se demonstradas em [0]. Com isso temos que

$$|f| = f(S, T) = \sum_{u \in S} \sum_{v \in T} f(u, v) \leq \sum_{u \in S} \sum_{v \in T} c(u, v) = |C|,$$

de forma que o valor de qualquer corte majora o valor de qualquer fluxo.

Um caminho aumentante em G com respeito a f é um caminho $P = [s = v_0, \cdots, v_k = t]$ da fonte ao sorvedouro respeitando $f(v_i, v_{i+1}) < c(v_i, v_{i+1})$ para todo i, $0 \leq i < k$. O nome "aumentante" se deve ao fato de que se definirmos $\delta := \min_{0 \leq i < k} \left\{ c(v_i, v_{i+1}) - f(v_i, v_{i+1}) \right\}$ e incrementarmos o fluxo f ao longo de P no valor de δ, estaremos obtendo um fluxo com valor estritamente maior, pois $\delta > 0$. Isso está formalmente verificado em [0].

Com esses dois conceitos claros em mente, podemos enunciar o principal Teorema concernente aos fluxos máximos.

Teorema 1. *(Max-Flow Min-Cut Theorem)*
As três seguintes afirmações são equivalentes.

(a) f é um fluxo máximo em G.

(b) Não existem caminhos aumentantes em G com relação a f.

(c) Existe um corte C de G com $|f| = |C|$.

Demonstração. $((a) \implies (b))$. Caso houvesse um caminho aumentante em G, o fluxo f poderia ser aumentado, contradizendo f ser máximo.

$((b) \implies (c))$. Defina S como o conjunto dos vértices atingíveis a partir do vértice s apenas por arestas (u, v) não saturadas, i.e., $f(u, v) < c(u, v)$, e

defina $T := V \setminus S$. Certamente $t \in T$, pois caso contrário haveria um caminho aumentante. Ainda, trivialmente se verifica que $s \in S$. Logo $C := (S, T)$ é um corte. Para cada $(u, v) \in S \times T$, temos que $c(u, v) = f(u, v)$, pois caso contrário v seria atingível. Então

$$|f| = \sum_{u \in S} \sum_{v \in T} f(u, v) = \sum_{u \in S} \sum_{v \in T} c(u, v) = |C|.$$

$((c) \implies (a))$. Segue do resultado estabelecido sobre valores de cortes majorarem valores de fluxos. □

Esse resultado coloca o problema de achar um corte mínimo como um problema paralelo ao de achar um fluxo máximo. Na verdade, eles são problemas duais no sentido de programação linear.

O método de Ford-Fulkerson nada mais é do que a aplicação desse teorema, pois ele encontra caminhos aumentantes e os aumenta até não poder mais. No entanto, escolhendo algumas capacidades como números irracionais de forma cuidadosa, podemos fazer esse algoritmo não parar. Já o algoritmo de Edmonds e Karp [0] aumenta caminhos aumentantes de comprimento (número de arestas) mínimo, e isso garante que ele leva no máximo $O(|V||E|)$ iterações, levando a uma complexidade $O(|V||E|^2)$.

Existem algoritmos que rodam sob $O(|V|^3)$, um dos quais está provado correto em [0]. Porém, esses são substancialmente complexos e fogem ao nosso escopo.

Porém, intuitivamente, nesse problema não há noção de custo nas arestas, pois não pagamos nada para transportar matéria por elas. Dessa forma, num certo sentido, nossos fluxos podem ficar "ineficientes", pois eles podem conter "ciclos", que nos custariam mais caro.

É essa observação que nos leva ao problema do fluxo máximo de custo mínimo.

2 Fluxos Máximos de Custo Mínimo

Dado um grafo direcionado $G = (V, E)$, os vértices distintos $s, t \in V$, uma função capacidade real não negativa $c : E \to \mathbb{R}^+$ e uma função custo real não negativa $\$: E \to \mathbb{R}^+$, estendemos c como da primeira vez para $V \times V$ e estendemos $\$$ de forma qualquer também para $V \times V$. Definimos então o custo de f como $\$(f) := \sum_{\substack{(u,v) \in V \times V \\ f(u,v) \geq 0}} f(u, v)\$(u, v)$.

Note que a arbitrariedade na extensão de $\$$ não prejudica a unicidade da definição de $\$(f)$ porque $f(u, v) \leq 0$ sempre que $(u, v) \notin E$.

Diremos que um fluxo f é de custo mínimo se para qualquer fluxo f' com $|f| \leq |f'|$, valer $\$(f) \leq \$(f')$. Assim, podemos enunciar o problema do fluxo máximo de custo mínimo como, dado $(G, s, t, c, \$)$, obter um fluxo f^* máximo e de custo mínimo.

Em primeiro lugar, correndo o risco de citar o óbvio, vale notar que esse problema generaliza o problema do fluxo máximo e que, portanto, devemos esperar um problema mais difícil e algoritmos menos eficientes. De fato, isso se verifica.

Algo interessante sobre o problema do fluxo máximo de custo mínimo é que podemos assumir, sem perda de generalidade, que não existem vértices $u, v \in V$ tais que ambas as arestas (u, v) e (v, u) estão em E. Isso ocorre porque fixado G, podemos montar o grafo $G' = (V', E')$ onde $V' = \bigcup_{v \in V} \{\overleftarrow{v}, \overrightarrow{v}\}$ é o conjunto de vértices obtidos trocando cada vértice de V por um vértice de entrada, $\overleftarrow{v}$, e um vértice de saída, $\overrightarrow{v}$. Além disso, $E' = \{(\overrightarrow{u}, \overleftarrow{v}) : (u, v) \in E\} \cup \{(\overleftarrow{v}, \overrightarrow{v}) : v \in V\}$ é o conjunto de arestas obtido ligando a saída de u na entrada de v para cada aresta $(u, v) \in E$ e adicionando ligações da entrada para a saída de cada vértice.

Para completar a construção, devemos definir uma função capacidade c' e uma função custo $\$'$. Se $(u, v) \in E$, definimos $c'(\overrightarrow{u}, \overleftarrow{v}) := c(u, v)$ e $\$(\overrightarrow{u}, \overleftarrow{v}) := \$(u, v)$. Ainda, para cada vértice $v \in V$, definimos $c'(\overleftarrow{v}, \overrightarrow{v}) := \sum_{(u,v) \in E} c(u, v)$ e $\$(\overleftarrow{v}, \overrightarrow{v}) := 0$.

É deixado como exercício a verificação de que é possível computar um fluxo máximo de custo mínimo de s para t em G se for computado um fluxo máximo de custo mínimo de $\overleftarrow{s}$ para $\overrightarrow{t}$.

Finalmente, observamos que $|V'| = 2|V|$ e que $|E'| = |E| + |V|$. Logo, se acharmos um algoritmo polinomial para resolver esse problema mais restrito, teremos achado um algoritmo assintoticamente tão eficiente quanto o primeiro para resolver o problema geral.

A próxima seção irá ilustrar uma aplicação desse problema no transporte de recursos em sistemas distribuídos. As seções subsequentes irão tratar de algoritmos para resolvê-lo. Esses algoritmos farão uso do resultado acima.

3 Aplicações

Nossa primeira aplicação é concernente a um servidor que funciona como repositório de códigos de software livre não orientado a objetos. Seus processos usam `void*` e mesmo assim seus administradores conhecem ciência da computação, ao contrário de certas pessoas que alegam o oposto e justificam sua tese dizendo que escreveram 200 papers e sendo autoritários e arrogantes. Todos os dias após o expediente os administradores se reúnem para contar piadas sobre os animais que ficam usando software de fundo de quintal e reclamam de serem invadidos, da máquina dar pau, de ela ficar lerda e do seu sistema ser um lixo.

Sendo um servidor muito sério e requisitado, é normal que ele possua muitos códigos armazenados. Dessa forma, foi necessário distribuir a informação em vários nós. O problema é que com o passar do tempo, os nós estão "desbalanceados", i.e., alguns nós estão muito cheios e outros muito vazios.

Mais formalmente, a cada nó v é associado o valor $r(v)$, onde $r : V \to \mathbb{R}$ é a função recurso. Caso $r(v) > 0$, v é um nó fornecedor e $r(v)$ unidades de

armazenamento devem ser retiradas desse nó. Caso $r(v) < 0$, v é um nó receptor, e $-r(v)$ unidades de armazenamento devem entrar em v. Caso $r(v) = 0$, dizemos que v é um nó intermediário ("transshipment") e o total armazenado em v deve permanecer constante.

Entre dois nós u e v, pode existir uma rede (aresta) (u, v) com velocidade $\sigma(u, v)$, onde "velocidade" é medido em unidades de armazenamento por unidade de tempo. O custo de manter essa rede ativa é $\$(u, v)$, onde esse valor é medido em dinheiro por unidade de armazenamento.

Foi destinado ao sistema um tempo τ para que a locomoção seja concluída. Gostaríamos de, nesse intervalo de tempo, satisfazer os requerimentos (dados por r) e, caso isso não seja possível, fazer com que o resultado seja o mais próximo do ideal, i.e., que o máximo de dados possível seja transferido de nós cheios para nós vazios. Iremos assumir, contudo, que não devemos transferir mais do que $r(v)$ a partir de um nó v caso $r(v) > 0$ e que não devemos fazer um nó v receber mais que $-r(v)$ caso $r(v) < 0$.

Para resolver esse problema, montaremos um grafo direcionado $G = (V, E)$. V será o conjunto dos nós mais dois vértices s e t. Para cada rede (u, v), iremos inserir uma aresta (u, v) em E com $c(u, v) = \tau\sigma(u, v)$. $\$$ já foi definido para a rede. Agora, para cada vértice $v \in V \setminus \{s, t\}$ com $r(v) > 0$, colocamos a aresta (s, v) com custo $\$(s, v) = 0$ e capacidade $c(s, v) = r(v)$. Ainda, para cada vértice $u \in V \setminus \{s, t\}$ com $r(u) < 0$, colocamos a aresta (u, t) com custo $\$(u, t) = 0$ e capacidade $c(u, t) = -r(u)$. Deixamos como exercício verificar que um fluxo de custo mínimo nesse grafo representa transações que levam a um transporte ideal com custo de utilização das redes mínimo.

Outra aplicação pode ser tentar minimizar o tempo levado para pelo servidor concluir a tarefa. Nesse caso a estrutura do nosso grafo seria a mesma. Os custos nas redes teriam de ser trocados para $\$(u, v) = \dfrac{1}{\sigma(u, v)}$, pois o tempo levado seria o inverso da velocidade. As capacidades seriam infinitas. Nas arestas (s, v), os custos seriam 0 e as capacidades seriam $r(v)$. Para as arestas (u, t) é análogo. Nesse problema, as capacidades não são limitadas fora das arestas em contato com a fonte e o sorvedouro, e existem algoritmos mais eficientes para isso que o fluxo máximo de custo mínimo. Trataremos dele adiante.

Temos que notar, no entanto, que devemos redefinir o custo de um fluxo para $\$(f) := \max\limits_{\substack{(u,v)\in E \\ f(u,v)\geq 0}} \left\{ f(u, v)\$(u, v) \right\}$ de forma que o problema faça sentido. Isso será um problema apenas para o primeiro algoritmo. No entanto, para os outros dois, que não usam propriedades sérias dos números reais, isso será facilmente contornado se a operação + for trocada por max.

De qualquer modo, ao longo desse trabalho, iremos nos concentrar na primeira situação, pois ela é mais comum em sistemas de tempo real.

Na última seção, discutiremos como executar as transferências uma vez dado o fluxo máximo de custo mínimo em questão.

4 Solução por Programação Linear

Um conhecimento básico de programação linear é necessário para acompanhar esta seção. Para os leitores que desejam adquiri-lo, recomendamos [2] para uma visão sob o ponto de vista de álgebra linear e [0] para uma visão algorítmica.

Assumindo que o grafo G não possua arestas em ambos os sentidos para qualquer par de vértices, para cada aresta $(u, v) \in E$, o fluxo $f(u, v)$ é não negativo, pois $f(u, v) = -f(v, u) \leq 0$, pois $c(v, u) = 0$. Dessa forma, podemos propor o seguinte programa linear para resolver o problema para esse tipo de grafos.

$$\min\left\{q^T x\right\} \text{ sujeito à}$$

$$\begin{cases} x_i \leq c_i, & 0 \leq i < |E|, \\ \sum_{e_i=(u,v)} x_i = 0, & u \in V \setminus \{s, t\}, \\ x_i \geq 0, & 0 \leq i < |E|, \end{cases}$$

onde $e_0, \cdots, e_{|E|-1}$ são as arestas, $q \in \mathbb{R}^{|E|}$ é o vetor de custos e $x \in \mathbb{R}^{|E|}$ é o vetor de fluxos.

Dada uma solução factível ótima para esse problema, podemos montar facilmente um fluxo de custo mínimo em G, pois o custo é minimizado por esse programa linear e uma solução factível para ele claramente corresponde a um fluxo.

Existe um outro programa linear, dessa vez com menos equações e variáveis, que corresponde a pensar num fluxo como definido nas arestas e reescrever os axiomas de fluxos, e ele irá valer porque $\$(u, v) \geq 0$ e um fluxo tanto num sentido como no outro seria um desperdício. Mas não iremos entrar nele aqui.

Infelizmente, os algoritmos de programação linear disponíveis não são bons o suficiente para nossas aplicações. O Simplex, o mais conhecido, é exponencial no pior caso e caso seja programado por algum programador OO pode não parar, pois como eles odeiam matemática, não vão notar uma sutileza na inspeção das variáveis que entram e saem da base. Já o método de Karmakar [2], mais recentemente desenvolvido, possui pior caso polinomial mas sofre de instabilidade numérica. Ele já é, no entanto, suficiente para mostrar que esse problema pode ser resolvido em tempo polinomial.

Ainda, existe um algoritmo chamado *Network Simplex* que é uma variação do Simplex comum e obtém tempo polinomial para esse programa linear específico.

5 Solução pelo Método de Ford-Fulkerson

O método de Ford-Fulkerson, como dito antes, encontra caminhos aumentantes e os aumenta até não poder mais para encontrar um fluxo máximo. Mas será que não poderíamos escolher caminhos aumentantes específicos para garantir que o fluxo máximo gerado é também de custo mínimo? A resposta, felizmente,

é sim. Se escolhermos caminhos de custo mínimo da fonte até o sorvedouro, o fluxo gerado pelo método de Ford-Fulkerson possui custo mínimo.

A verificação desse fato não é trivial. Não será possível deduzi-la aqui e não deu tempo para encontrar referências, sinto muito. No entanto, um verificador será apresentado adiante, i.e., um algoritmo que checa se um fluxo é de custo mínimo. Logo você pode colá-lo ao fim do seu algoritmo para garantir a corretude.

O problema é que não definimos direito que raios é um caminho de custo mínimo. Assuma que não existem arestas em ambas as direções. Pois bem, dado o grafo G e o resto, monte um grafo $G' = (V, E')$ onde $E' = X \cup Y$ onde $X = \{(u, v) \in E : f(u, v) < c(u, v)\}$ e $Y = \{(v, u) : (u, v) \in E \wedge f(u, v) > 0\}$ são conjuntos de arestas não saturadas. Defina uma função peso $w : E' \to \mathbb{R}$ fazendo $w(u, v) = \$(u, v)$ se $(u, v) \in X$ e $w(u, v) = -\$(v, u)$ caso $(u, v) \in Y$. w está bem definido porque $X \cap Y = \{\}$. Queremos um caminho mínimo em G em relação à função peso w.

Quando encontramos esse caminho, se ele existir, temos dois problemas. Primeiro, o peso das arestas é um número real, que pode ser negativo. Isso pode ser contornado usando o algoritmo Bellman-Ford [0], que ainda funciona com pesos negativos, mas é mais lerdo que o Dijkstra [0], que funciona com pesos não negativos apenas.

O segundo problema é que poderia haver um ciclo com soma dos pesos negativa, e se ele fosse atingível a partir da origem, não existiria um caminho mais curto. No entanto, como os fluxos gerados por esse Ford-Fulkerson são de custo mínimo. Um ciclo negativo poderia ser aumentando gerando um fluxo com mesmo valor e custo menor, pois os custos das arestas ($) são não negativos. Logo isso nunca ocorre.

Quando um desses caminhos for encontrado, devemos definir

$$\ell(u, v) := \begin{cases} c(u, v) - f(u, v), & (u, v) \in X \\ f(v, u), & (u, v) \in Y \end{cases}$$

como o máximo a ser aumentado numa aresta. Se fizermos $\ell := \min_{(u,v)\in E} \left\{ \ell(u, v) \right\}$, poderemos aumentar o fluxo ao longo desse caminho aumentante sem problemas. Conseguiremos então um fluxo maior.

A grande dificuldade no método de Ford-Fulkerson é, como visto antes, que ele pode não parar. Aqui nosso objetivo será impor condições no problema para que ele pare, e pare rápido.

Em primeiro lugar, em [0] está provado que se as arestas tiverem capacidades inteiras, existirá um fluxo máximo com valores inteiros nas arestas e valor inteiro, e esse será achado pelo algoritmo de Ford-Fulkerson. Com isso, temos que se um fluxo tem valor $|f|$, então no máximo $|f|$ iterações serão executadas. Se for usado o algoritmo de Bellman-Ford, temos que a complexidade nesse caso é $O(|f||V||E|)$.

Ainda, definindo $\alpha := \sum_{\substack{v\in V \\ r(v)\geq 0}} r(v)$ temos que $|f| \leq \alpha$, pois o corte $C = (S, T)$

onde $S = \{s\}$ e $T = V \setminus S$ tem valor α. Note que $\beta := -\sum_{\substack{v \in V \\ r(v)<0}} r(v)$ também poderia ser usado por argumento similar.

Dessa forma, podemos aproximar os valores da funçao recurso para inteiros suficientemente pequenos para que o algoritmo pare rapidamente.

Existem várias melhoras a esse algoritmo, mas nenhuma delas será citada aqui, sinto muito.

6 Soluções por Cancelamento de Ciclos

O último algoritmo que iremos apresentar é o de cancelamento de ciclos. Novamente, iremos assumir um grafo sem arestas "bidirecionais". Ele começa achando um fluxo máximo qualquer e procede encontrando ciclos (série fechada de arestas por onde pode-se aumentar o fluxo) com soma dos custos negativa (sobre o mesmo grafo da seção passada).

Claramente, dado um ciclo desses, pode-se obter um fluxo com o mesmo valor (logo ainda máximo) porém com um custo possivelmente menor, pois os custos são não-negativos. A pergunta dessa vez é se, ao fim desse procedimento, o fluxo obtido possuirá custo mínimo. Novamente a resposta é sim.

E novamente a demonstração disso é não trivial (principalmente para quem usa W*ndoze e programa em Java), então apenas indicaremos [3] como referência.

Bom, como se encontram ciclos e soma negativa ou se verifica que eles não existem? Pode-se digitar cinco linhas de código e implementar o algoritmo de Floyd-Warshall, pois ele detecta esses ciclos (lembrar de MTM discreta). Mas o algoritmo de Bellman-Ford também faz isso, e nesse caso ele é uma saída melhor, pois é mais barato para grafos esparsos e igualmente caro para grafos densos.

Outra coisa. Pode-se criar grafos com número de ciclos exponencial na sua representação em bits. Aumentando 1 de fluxo ao longo de cada um desses ciclos num fluxo máximo, nada nos garante que nosso algoritmo irá encontrar justamente esses ciclos em ordem, levando tempo exponencial.

No entanto, [3] mostra como escolher os ciclos a aumentar de modo que o algoritmo leve apenas $O(m(m + n \log n) \min\{\log(nU), m \log n\})$, onde U é qualquer limite superior para r e c simulteneamente. Isso é não só polinomial, mas também bastante rápido.

Lembra ainda do problema de minimizar o tempo total na operação? Pois é, [3] mostra um algoritmo para resolver o problema em apenas $O(n(m + n \log n) \log(nU))$. No entanto, realmente eu não sei se poderíamos trocar $+$ por max nesse algoritmo. Teríamos que ter acompanhado mais de perto o paper. Desculpem-nos.

Com isso foram apresentados algoritmos no estado da arte para resolver esse problema. Infelizmente, qualquer tentativa de ir além disso em 20 minutos será inútil.

7 Transferindo os Dados

Muito bem, até agora aprendemos a calcular fluxos máximos de custo mínimo, mas eles apenas representam quanto devemos transferir entre os nós, mas não como. Essa seção se propõe a, dado um fluxo máximo de custo mínimo, transferir os dados no sistema de forma ótima.

Antes precisamos esclarecer o porque essa preocupação é importante. Basicamente, se um vértice tem armazenamento inicial α_0 e armazenamento final α_τ, então, num dado instante de tempo $t \in [0; \tau]$, devemos ter $\alpha_0 \leq \alpha_t \leq \alpha_\tau$ ou $\alpha_\tau \leq \alpha_t \leq \alpha_0$, dependendo de se o armazenamento aumentou ou diminuiu. Caso isso não ocorresse, poderia existir um nó que fosse enchido além da sua capacidade, um problema. Similarmente, um nó poderia ficar com armazenamento negativo (wtf?).

Vamos nos valer da convexidade de blocos no $\mathbb{R}^{|V|}$. A estratégia é simples. Iremos subutilizar as redes pouco usadas e dispersar as requisições no tempo. Em outras palavras, se uma rede (u, v) estiver destinada a transportar $f(u, v)$ no intervalo τ, faremos com que ela tenha transportado $\frac{t}{\tau} f(u, v)$ no tempo t.

Com isso, fixado u, obtemos

$$\alpha_t = \alpha_0 + \frac{t}{\tau} \sum_{(u,v)\in E} f(u, v) = \alpha_0 + \frac{t}{\tau}(\alpha_\tau - \alpha_0) =$$

$$\left(1 - \frac{t}{\tau}\right) \alpha_0 + \frac{t}{\tau} \alpha_\tau$$

e nosso problema está resolvido.

8 Referências

[0] "Introduction to Algorithms", 2^{nd} edition. Thomas H. Cormen, Ronald Rivest (o "R" do RSA), Charles Leiserson, Stein.

[1] `http://en.wikipedia.org/wiki/Hopcroft-Karp_algorithm`

[2] "Linear Algebra", Gilbert Strang. MIT press.

[3] `dspace.mit.edu/bitstream/handle/1721.1/2630/SWP-3914-35650575.pdf?sequence=1`