Abastecimento de lojas à custo mínimo

Max William S. Filgueira e João Pedro de A. Paula

Junho 2019

Conteúdo

1 Introdução do Problema

A demanda incessante por produtos requer que fábricas por todo o mundo trabalhem à todo vapor para supri-la e isto chama atenção ao problema associado a este o qual não pode ser evitado: Como transportar adequadamente todos estes produtos? Uma má estratégia de entregas resultaria em prejuízos para todos os envolvidos, talvez seja mais caro enviar o mesmo caminhão para duas cidades em direções opostas do que enviar um caminhão para cada cidade.

No contexto de grafos, as fábricas e as lojas são tratadas como vértices e as arestas são as "estradas" entre as localizações de cada dois vértices.

2 Modelagem do problema

Como mencionado na seção 1, cada vértice representa ou uma fábrica ou uma loja, e as arestas entre cada dois vértices representariam um trajeto possível entre as duas localizações (uma "estrada"). Neste trabalho trataremos este como um problema de fluxo à custo mínimo. Assim, serão necessárias algumas modificações no grafo. A princípio, adicionaremos informações nas arestas, as quais conterão os valores da capacidade mínimo e máximo que pode percorrer esta aresta e também um valor inteiro não-negativo associado ao custo de transporte de uma unidade de um produto por esta estrada. Outro detalhe a ser corrigido é que, para que os vértices sejam conservativos, precisamos garantir de alguma maneira que o fluxo dos produtos ao chegarem em suas lojas "escorram" para algum sumidouro, apesar de efetivamente os produtos permanecerem nas lojas. Assim, cria-se uma aresta com custo 0 entre o vértice da loja e um novo sumidouro com fluxo máximo igual à capacidade de estoque da loja, analogamente cria-se uma fonte com arestas de peso 0 para os vértices de fábrica com fluxo mínimo e máximo igual à produção da fábrica em questão. Todas as arestas entre vértices que não são fonte nem sumidouro tem capacidade mínima igual a 0 e capacidade máxima infinita.

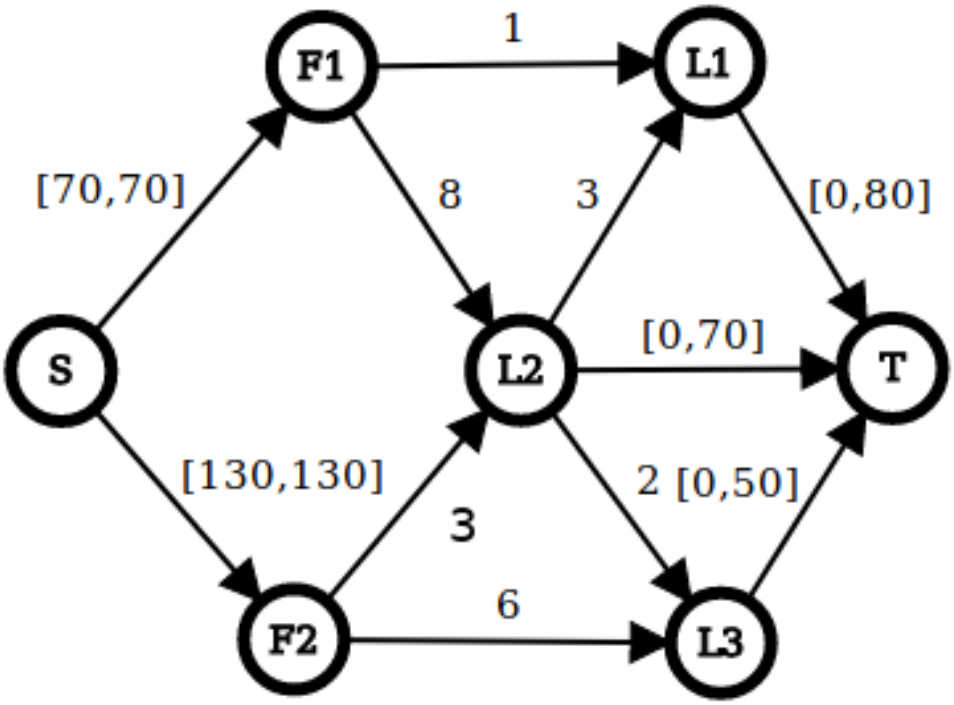

Figura 1: Um grafo modelado com o problema de fluxo a custo mínimo.

2.1 Estrutura do vértice

Dada a modelagem proposta, a estrutura dos vértices não difere da estrutura de um vértice em um problema qualquer de grafos, visto que a informação vital necessária para a solução do problema reside nas arestas, no mínimo seria necessário que cada vértice armazenasse um identificador e os vértices adjacentes, dependendo da forma de implementação.

2.2 Arcos

Os arcos, sendo os portadores das informações essenciais para a solução deste problema, possuem os valores das capacidades mínimas e máximas das arestas (em relação ao fluxo) e o valor do custo para transportar uma unidade do produto por meio deles. Os arcos entre a fonte e as fábricas possuem capacidade mínima igual à máxima, representativa da produção real da fábrica. Os arcos entre as lojas e o sumidouro tem capacidade mínima 0 e a capacidade máxima representativa da demanda real da loja. Todos os outros arcos tem capacidade mínima 0 e capacidade máxima infinita.

3 Estado da arte

3.1 Minimum Mean Cycle-Cancelling

A princípio, para cada aresta entre dois vértices (u, v) cria-se uma aresta (v, u) com custo $c(v, u) = -c(u, v)$. Definimos o custo de um ciclo como a soma dos pesos das arestas que o compõem. A idéia por trás deste algoritmo de Goldberg e Tarjan é: se não houver nenhum ciclo na rede residual cujo custo for negativo, então este fluxo é de custo mínimo.

Em teoria o algoritmo se resume em: encontre um ciclo de custo negativo, então cancele ele aumentando o fluxo neste ciclo (saturando assim, ao menos uma aresta), repita o processo até que não existam mais ciclos de custo negativo. De forma semelhante ao algoritmo original de Ford e Fulkerson, dependendo da magnitude do valor do fluxo, é possível que o algoritmo leve tempo exponencial (e para valores irracionais ele pode nunca terminar). Devido a isto, definimos a média de um ciclo como o custo do ciclo divido pela quantidade de arestas que o compõem.

Podemos então repetir o procedimento já definido porém com base numa regra: Buscando sempre o ciclo com menor custo, temos não só a garantia de termino do algoritmo, como para grafos com arestas de peso inteiro o algoritmo tem complexidade $O(mn \log(nC))$

3.2 Network Simplex

O algoritmo simplex mantém uma rede de fluxo viável a cada iteração. Uma solução básica para o problema de fluxo à custo mínimo é denotado por uma tripla $\langle T, L, U \rangle$, onde T, L e U são partições do conjunto de arcos. T é um

conjunto de arcos básicos, isto é, arcos que fazem parte da árvore geradora do grafo. L e U são respectivamente conjuntos com os arcos não básicos nos seus limites inferiores e superiores. Nos referimos à tripla $\langle T, L, U\rangle$ como uma *estrutura básica* o fluxo x associado à estrutura básica da seguinte forma:

- Para cada arco $(u, v) \in U$: $x_{ij} = u_{ij}$.
- Para cada arco $(u, v) \in L$: $x_{ij} = l_{ij}$.
- Para os outros arcos, obtém-se valores quaisquer com a restrição apenas de que os vértices se mantenham conservativos. Dizemos que esta estrutura tem circulação viável se para todo arco de T temos que $l_{ij} \leq x_{ij} \leq u_{ij}$.

Denotemos como $G^*(x)$ o subgrafo da rede residual $G(x)$ em relação ao fluxo x na qual todos os arcos da arvore geradora T e seus arcos reversos foram excluídos. Isto é: $(u, v) \in G^*(x)$ se $(u, v) \in G(x)$ mas $(v, u) \notin G(x)$.

Denotemos também uma arvore $T(v)$ como um subgrafo de $G(x)$ no qual todos os arcos estão direcionados ao nó raiz v, tendo os custos e capacidades dos arcos idênticos aos arcos definidos em $G(x)$.

Suponha que x seja uma circulação básica viável, e seja T a arvore geradora associada. Se $(k, l) \in G^*(x)$, então o ciclo básico W criado com a inserção do arco (k, l) em T é a união de (k, l) com o caminho pré-existente de l até k em $T(k)$. Um *pivoteamento simplex* consiste em adicionar o arco (k, l) à arvore, enviar δ unidades de fluxo em W, e retira-se o arco cuja capacidade residual foi reduzida para 0 no processo (existe ao menos um).

Por último, definimos uma **condição ótima**. Uma estrutura básica $\langle T, L, U\rangle$ é ótima se o custo de cada ciclo básico for não negativo.

Assim, podemos definir o algoritmo em sua totalidade como abaixo:

```
algorithm network-simplex;
begin
  find a feasible basis structure (T,L,U);
  let x be the basic feasible flow;
  while x is not optimum do
  begin
    find an arc (k,l) ∈ G*(x) for which the corresponding basic cycle W has a
    negative cost;
    perform a simplex pivot by sending δ = min(r_ij: (i,j) ∈ W) units of flow around
    W;
    update x and (T,L,U);
  end
end;
```

Figura 2: Algoritmo network simplex.

4 Menções importantes

Soluções para este problema cuja menção é indispensável encontram-se abaixo:

O algoritmo de Morton Klein [1] utiliza-se de propriedades de programação linear, mais especificamente caso a função de custo das arestas seja linearmente convexa.

Algoritmo de Goldberg & Tarjan [2] cuja solução proposta para o problema é acabar com todos os ciclos negativos no grafo.

Artigo de Edmonds & Karp [3] com solução para o problema de fluxo à custo mínimo, mais especificamente voltado para um subset deste, chamado assignment problem.

Algoritmo de Goldberg & Tarjan [4] que utiliza-se de programação linear e o conceito de dimensionamento (*scaling*) para encontrar um algoritmo de complexidade $O(n^2(\log n)\log(nC))$.

Algoritmo network simplex, de James B. Orlin [5], subconjunto do algoritmo simplex clássico da programação linear.

5 Solução escolhida

5.1 Dijkstra

Como a busca em largura necessita que os pesos das arestas sejam 1, precisamos utilizar um algoritmo de busca diferente para encontrar o menor ca-

minho, visto que dada modelagem do problema feita na seção 2 cada aresta tem um peso associado, que se refere ao custo de transporte de um vértice ao outro. O algoritmo escolhido para encontrar o menor caminho é o algoritmo de Dijkstra.

Chamemos o vértice de onde começamos de *vértice inicial* e *distância do nó Y* a distância entre o vértice inicial até o vértice Y. O Dijkstra pode ser implementado utilizando uma fila de prioridade que contenha os métodos `adicionarComPrioridade`, `extrairMenor` e `diminuirPrioridade` da seguinte forma:

Algorithm 1: Djikstra com fila de prioridade

```
Result: dist[],prev[]
Crie um conjunto de vertices nao visitados Q.
Para cada vertice v no Grafo faça:
dist[v] = INFINITO
prev[v] = indefinido
dist[fonte] = 0
Q.adicionarComPrioridade(dist[v],v)
while Q não está vazio do
    u = Q.extrairMenor()
    para cada vertice adjacente v de u:
        alt = dist[u] + peso(u,v)
        se alt < dist[v]:
        dist[v] = alt
        prev[v] = u
        Q.diminuirPrioridade(alt,v)
end
return dist[],prev[]
```

Em nossa implementação de fila de prioridade utilizamos uma heap binária com uma HashTable anexada para garantir acesso em O(1) aos indices dos elementos dentro da heap, de forma à permitir o ajuste de prioridade em $O(logn)$. A complexidade do algoritmo de Dijkstra é $O(m \log n + n)$, visto que visitamos cada nó uma vez ($O(n)$) e acessamos cada vértice adjacente exatamente duas vezes ($O(m)$), cada vez acessando a lista de prioridade no máximo duas vezes ($O(\log n)$).

5.2 Edmonds e Karp

A solução escolhida para o atual projeto é uma variante do algoritmo de Ford-Fulkerson, o algoritmo de Edmonds e Karp, onde invés de utilizar um grafo com arestas todas de peso 1, utilizamos arestas com o peso associado ao custo do problema em questão. Durante a execução deste algoritmo, é criado um novo grafo baseado no original, denominado de rede residual, onde os vértices são os mesmos que os do grafo anterior, porém é composto apenas por arestas nas quais haviam folga anteriormente (o fluxo que passava pela aresta era menor do que a capacidade máxima da aresta). Uma das características principais do algoritmo de Edmonds e Karp é que à cada iteração deste, é sempre encontrado o menor caminho possível entre a fonte e o sumidouro da rede residual, por meio de uma busca em largura. Entretanto, a busca em largura garante o menor caminho unicamente quando o peso das arestas é igual a um. Assim, para encontra-lo é necessário o uso de um algoritmo de menor caminho, como o Dijkstra por exemplo. Porém, como o peso das arestas em questão era o custo associado de transporte do produto, conseguimos a partir deste algoritmo o caminho de menor custo. Ao fim do algoritmo teremos, por corretude do algoritmo de Edmonds e Karp, o fluxo desejado na rede, e por meio da nossa escolha de caminhos de aumento de fluxo, teremos que este fluxo será o de menor custo associado.

Tendo que m é a quantidade de arestas e n é a quantidade de vértices, a complexidade do Edmonds e Karp é $O(mn) \times B(m, n)$, onde $B(m, n)$ é a complexidade da busca em largura. Como cada arco é visitado no máximo duas vezes ($O(m)$); existem, no máximo, m caminhos de aumento de fluxo em redes com $1 \leq k \leq m - 1$ arcos entre a fonte e o sumidouro, cada aumento de fluxo requer no máximo ($O(m)$) passos; temos, no máximo, $n-1$ iterações ($O(n)$) e a complexidade da busca em largura é $O(m)$, temos que a complexidade final do algoritmo é $O(m^2n)$.

Analogamente ao algoritmo de Edmonds e Karp original, a complexidade deste algoritmo é $O(mn) \times T(m, n)$ onde $T(m, n)$ é a complexidade do algoritmo de menor caminho utilizado. Visto que a complexidade do algoritmo de Dijkstra, dado no final da seção 5.1, é $O(m \log n + n)$, a complexidade final do nosso algoritmo será $O(m^2n \log n + mn^2)$.

6 Como usar o algoritmo

A implementação pode ser encontrada no GitHub. Após clonar o repositório ou baixar os arquivos, basta executar o comando

```
make
```

na pasta raiz do projeto (provavelmente com o nome `min-cost-flow`). Isto irá gerar um arquivo chamado `min-cost-flow` nesta mesma pasta, este é o arquivo usado para executar o projeto.

O programa recebe como argumento um arquivo de texto contendo a descrição das arestas de um grafo. Cada arquivo de texto é tratado como um grafo diferente e a pasta baixada no repositório já tem dois exemplos de grafos, `g.txt` e `g2.txt` para serem executados. A primeira linha do arquivo deve ser um número que representa a ordem desse grafo, ou seja, o número de arestas que ele tem. As linhas seguintes são descrições de arestas compostas por quatro números; o primeiro número é o rótulo do vértice de origem daquela aresta; o segundo número é o rótulo do vértice de destino; o terceiro número é o peso associado aquela aresta; e o último número é a capacidade máxima da aresta. Por exemplo, um arquivo contendo as seguintes descrições

```
4
0 1 8 8
0 2 6 6
1 2 1 1
1 3 1 1
2 3 6 6
```

irá gerar o seguinte grafo

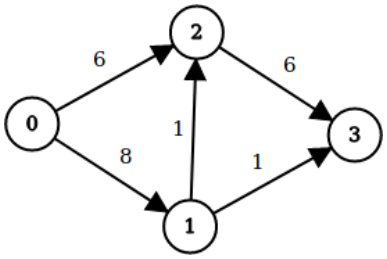

Figura 3: Grafo gerado a partir de um arquivo

onde cada aresta tem capacidade máxima igual ao peso.

Já o seguinte arquivo

```
7
0 1 0 70
0 2 0 130
1 3 8 500
1 4 1 500
2 3 3 500
3 4 3 500
3 5 2 500
3 6 0 70
4 6 0 80
5 6 0 50
```

gera o seguinte grafo

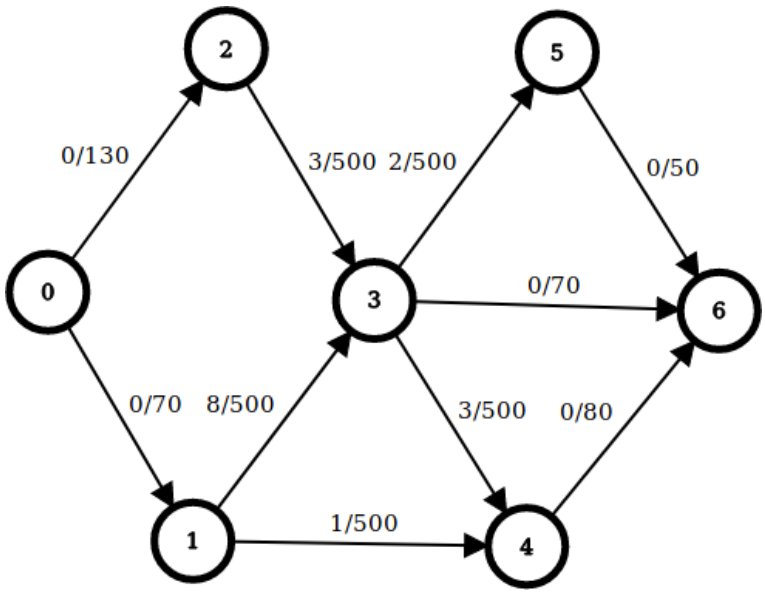

Figura 4: Grafo gerado a partir de um arquivo

onde o primeiro número em cada aresta é o peso e o segundo é capacidade máxima dela.

Note que as arestas de rótulo 0 e $n - 1$, onde n é o número de arestas especificado na primeira linha do arquivo, representam, respectivamente, os vértices fonte e sumidouro.

Para executar o programa basta colocar o caminho completo até o arquivo executável gerado após a execução do comando `make`, ou, caso esteja na pasta principal do projeto, basta executar

```
./min-cost-flow [arquivo de texto com a descrição do grafo]
```

Por exemplo

```
./min-cost-flow sample/g.txt
```

executa o programa no grafo especificado no arquivo `g.txt` na pasta `sample`.

É necessário ter uma versão recente do `gcc` instalada, ou uma versão recente do `clang`. Caso vá usar o `clang`, ao o `make` se faz necessário especificar que queremos usar o `c++` como compilador da seguinte forma

```
make CXX=c++
```

7 Conclusão

Além de categorizar o problema de abastecimento entre fábricas e lojas como um problema de fluxo à custo mínimo, apresentamos sua modelagem e algumas soluções que podem resolvê-lo em tempo polinomial. Mostramos que podem ser diversas as abordagens para o mesmo problema, enquanto uma variante de Edmonds e Karp juntamente com o algoritmo de Dijkstra pode ser o suficiente, este problema também pode ser visto como um de programação linear, como no algoritmo simplex.

Referências

[1] Morton Klein. A primal method for minimal cost flows, with applications to the assignment and transportation problems. 1967.

[2] Andrew V. Goldberg and Robert E. Tarjan. Finding minimum-cost circulations by canceling negative cycles. *J. ACM*, 36(4):873–886, October 1989.

[3] Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM*, 19(2):248–264, April 1972.

[4] Andrew V. Goldberg and Robert E. Tarjan. Finding minimum-cost circulations by successive approximation. *Math. Oper. Res.*, 15(3):430–466, August 1990.

[5] James B. Orlin. A polynomial time primal network simplex algorithm for minimum cost flows. In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '96, pages 474–481, Philadelphia, PA, USA, 1996. Society for Industrial and Applied Mathematics.