Minimum cost flow problem

Abstract

We describe several interesting observation and connections related to minimum cost flow problem.

1 Minimum Cost Flow Problem and methods to solve them

There are several ways to define minimum cost flow and with different types of flows(non-negative and skew-symmetric), capacity or upper bound, edge demand or lower bound, edge cost, and with flow balance. The standard way to define:

$$\begin{array}{l}\textbf{min} < \textbf{cent}, \textbf{flow} > \\ \text{s.t} \sum_{u \to v} \text{flow}(u \to v) - \sum_{v \to w} \text{flow}(v \to w) = 0 \\ \quad \text{flow}(u \to v) = b(u \to v) \\ \quad flow \geq 0\end{array}$$

The easiest solution we can find is using augmenting cycle:
We can run any maximum flow algorithm to find feasible solution to the problem(neglecting the cost). Let the augmenting cycle be a cycle with negative cost. Then sending a flow through this cycle would reduce the total cost, while maintaining the feasible property.
We can also define reduced cost as follows:
Let $\phi(v)$ be a any potential function on a vertex. Then $\bar{\text{cost}}(u \to v) = \phi(u) - \phi(v) + \bar{\text{cost}}(u \to v)$ satisfies the condition that cost(C) = $\bar{\text{cost}}(C)$ for any cycle C.

Lemma. *A feasible flow f is optimal ↔ there is no augmenting cycle in the residual graph.*

Proof: → Suppose the flow is optimal. If there is an augmenting cycle C, then we can send flow through C and reduce the cost of current flow, contradicting the f is optimal.
← Suppose there is no augmenting cycle. Let $\phi(v)$ = [Shortest path from s to v with respect to the cost function]. Then $\bar{\text{cost}}(u \to v) = \phi(u) - \phi(v) + \text{cost}(u \to v) \geq 0$. For the new cost function, sending more flow in a new residual graph would increase the cost of any flow, therefore f is optimal. □

We can find the augmenting cycle in a graph systematically as follows:
Let T be any fixed spanning tree of G. Define new potential function for each vertex

$$\text{slack}_T(u \to v) = \begin{cases} 0 & \text{, if } u \to v \in T \\ \sum_{e \in \text{cycle in } T \cup \{u \to v\}} \text{cost}(u \to v) & \text{, Otherwise} \end{cases}$$

Above definition preserves the property that $\text{slack}_T(C) = \text{cycle}(C)$ for any cycle C in G. Therefore, we can essentially find negative reduced cost edge in residual graph and push flow through it and do pivoting. Spanning tree T will be updated as follows:

Update T:
- Find bottleneck capacity in a cycle
- Push the flow amount equal to bottleneck capacity through the cycle
- Pivot out the bottleneck capacity edge, and pivot in the dart with negative reduced cost
- Recompute vertex potentials

There are two ways to represent the flow in the graph:

- f(u → v) = - f(v → u), for all edges

- f(u $\to$ v) ≥ 0 and f(u $\to$ v) $= 0$ if f(u $\to v) > 0$

Transhipment problem

$$\boxed{\begin{array}{l}\textbf{min} < f, \$ > \\ \text{s.t } \partial f = b \\ \quad f \geq 0\end{array}}$$

Under generic assumption on $f, \$$:

- Basis spanning tree T, there is a unique flow$_T$ that satisfies the condition $\partial \text{flow}_T = b, flow_T(e)$ is nonzero only for edges in T.
- There exist a unique spanning tree T_{OPT} with $flow_{T_{\text{OPT}}}$ is optimal.

Subsequently, we can define slack as follows:
For fixed spanning tree T, there is unique cycle $C = T \cup \{e\}$ for edge e not in T. $slack(e) = \sum_{l \in C} \$(l)$, and 0 otherwise. **Observe that slack is not negative, since otherwise there is no optimal solution to our LP**
The main LP is:

$$\boxed{\begin{array}{l}\textbf{min} < f, \textbf{slack}_T > \\ \text{s.t } \partial f = \partial \text{ flow}_T \\ \quad f \geq 0\end{array}} \qquad \boxed{\begin{array}{l}\textbf{min} < s, \textbf{flow}_T > \\ \text{s.t } \partial s = \partial \text{ slack}_T \\ \quad s \geq 0\end{array}}$$

And in the case of non-planar embedded graph with genus g:

$$\boxed{\begin{array}{l}\textbf{min} < f, \textbf{slack}_T > \\ \text{s.t } \partial f = \partial \text{ flow}_T \\ \quad f \geq 0\end{array}} \qquad \boxed{\begin{array}{l}\textbf{min} < s, \textbf{flow}_T > \\ \text{s.t } \partial s = \partial \text{ slack}_T \\ \quad [s] = [\text{slack}_T] \\ \quad s \geq 0\end{array}}$$

Consider the primal LP, We can rewrite the constraints as follows:

$$\partial f = \partial \text{flow}_T \iff \sum_u f(u \to v) - f(v \to u) \iff$$

$$\gamma(u) \sum_u (f(u \to v) - f(v \to u)) = \sum_{u \to v} f(u \to v)(\gamma(u) - \gamma(v)) = \sum_{u \to v} f(u \to v)\gamma(u \to v)$$

Observe here that $\gamma(u \to v) = -\gamma(v \to u)$ We can define s in terms of γ by setting the negative values to be 0, and we will get the exact dual program defined above.
Couple questions regarding the slack and flow:

- If the fixed tree T is arbitrary tree (not necessarily the Holiest Tree, then the flow in the answer does not have to be optimal) but solution to the linear program $min < f, \text{slack}_T >$ is equal to the answer from fixed tree T, not necessarily the optimal solution. That is because for each vertex, the demand satisfies the constraint and if we consider the tree T, then the value of $min < f, \text{slack}_T >$ would be 0, implying it is the optimal solution. (Sum cannot be negative since otherwise there is no optimal solution)
- The reason we picked the slack as the way we defined is due to the fact that slack is not negative, ensuring that the nothing bad happens.
- What makes the non-planar case special with 2g extra constraints?
- How does the slack in dual representation help us to find the pivots quickly?

2 References:

- Cabello, Sergio, Erin W. Chambers, and Jeff Erickson. "Multiple-source shortest paths in embedded graphs." SIAM Journal on Computing 42.4 (2013): 1542-1571.
- Eisenstat, David, and Philip N. Klein. "Linear-time algorithms for max flow and multiple-source shortest paths in unit-weight planar graphs." Proceedings of the forty-fifth annual ACM symposium on Theory of computing. ACM, 2013.
- Erickson, Jeff. Maximum flows and parametric shortest paths in planar graphs. Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms. Society for Industrial and Applied Mathematics, 2010.