7825 Final Project - Cycle cancelling algorithm

Paul Guidas

May 3, 2024

Introduction

To understand the cycle-cancelling algorithm we need to understand what a minimum cost flow problem is as this is the type of problem this algorithm is intended for. A min-cost flow problem finds the minimum cost necessary to send the maximum flow through a network.

Notation and Assumptions (heavily adapted from (Ahuja et al., 1993))

Let:
$\mathbf{G} = (\mathbf{N}, \mathbf{A})$ be a directed network with N nodes and A arcs.
$\mathbf{c_{ij}} \geq \mathbf{0}$ represent the cost of arc $(i, j) \in A$
$\mathbf{u_{ij}}$ represent the capacity of arc $(i, j) \in A$
$\mathbf{s(i)}$ represent the supply or demand of node i depending on whether $s(i) > 0$ or $s(i) < 0$, $\forall i \in N$
$\mathbf{C}$ denote the largest magnitude of any arc cost.
$\mathbf{U}$ denote the largest magnitude of any supply/demand or finite arc capacity.

So, the minimum cost problem is:

$$\text{Minimize } z(x) = \sum_{(i,j)\in A} c_{ij}x_{ij}$$

subject to:

$$\sum_{\{f:(i,j)\in A\}} x_{ij} - \sum_{\{f:(j,i)\in A\}} x_{ji} = s(i) \quad \forall i \in N,$$

$$0 \leq x_{ij} \leq u_{ij} \quad \forall (i, j) \in A.$$

We will also assume:
The network is directed. (This is necessary for us to have feasible flow.)
The data is all integral. (Data being defined as arc capacities/costs and node supplies/demands)

Note: If we have integral values for all our data, our solution will also be integral which is important to note as many of these problems will not allow for a non-integral solution. The validity of this claim is discussed in Theorem 9.10 (Integrality Property) **(Ahuja et al., 1993)**

Feasibility and Algorithm Setup

First we need to verify that the problem in question is feasible. To do this we first need to check if the sum of all the supplies (positive values) and demands (negative values) equals zero.

$$\sum_{\{i \in N\}} s(i) = 0$$

Next, we need to verify that a feasible flow exists. To do this we need to convert the problem into a max flow problem by adding an $'s'$ node and a $'t'$ node with arcs going to each supply and demand node respectively. We will set the capacities of these arcs to the supply/demand values of those nodes.

Now, we will verify we can find a valid max flow using a max flow algorithm like Edmonds-Karp (an implementation of the Ford-Fulkerson augmenting path algorithm). If we find a valid max flow, we will use that as our initial feasible solution to start the Cycle Cancelling Algorithm.

Solving and Optimality Condition

while the residual network G(x) contains a negative cost cycle:

1. Build a residual graph based on the initial feasible solution.
2. Use an algorithm (options discussed in complexity section) to detect a negative cost cycle W in the residual graph of the initial feasible solution. If no negative cost cycle is found **end algorithm.** (The current solution is the optimal solution.)
3. If a negative cost cycle W is found, set $\delta = min\{r_{ij} : (i,j) \in W\}$; (sets the amount of units to augment the path to the lowest remaining capacity along that path)
4. Augment δ units of flow along all arcs in W to "cancel" the negative cycle and update residual graph $G(x)$; (The updated $G(x)$ becomes the residual graph used for step 1 of the next iteration.)

Optimality:
"Theorem 9.1 (Negative Cycle Optimality Conditions). A feasible solution x* is an optimal solution of the minimum cost flow problem if and only if it satisfies the negative cycle optimality conditions: namely, the residual network G(x*) contains no negative cost (directed) cycle." **(Ahuja et al., 1993)**

What this means is if there is no negative cost cycle in the residual network, an optimal solution has been reached. This is because the existence of a negative cost cycle implies that an alternate path with a lower cost still exists and/or has not been fully utilized yet. Therefore if we have achieved maximum flow

and there are no negative cycles, there is no better path to choose and we have found the optimal solution. □

Complexity

The complexity of the Cycle Cancelling algorithm will depend on the choices of subsystem algorithms. There are several components we will look at to find the order of the overall complexity but for a specific number it will end up being based on the algorithm choices made for the max-flow algorithm and the negative cycle location algorithm.

Initially we add arcs in place of supply/demand values. In the worst case, where all nodes have non-zero supply/demand values, we add n arcs. $\mathbf{O(n)}$
Of note, this also increases our arc count m to $m + n$ and our node count n to $n + 2$.

Then we run a max-flow algorithm to solve for the initial feasible solution. Here are a few options with varied complexity to use as a starting point:

1. $\mathbf{O(nm^2)}$ Edmonds-Karp (Better for low density graphs)
2. $\mathbf{O(mF)}$ (where F is the value of the max flow) Ford-Fulkerson (better if we know there is a small max flow)
3. $\mathbf{O(n^3)}$ preflow-push (using the FIFO selection rule, best for extremely dense graphs)

Next, we will be running an algorithm to find a negative cost cycle. This algorithm will run one time for each iteration, up to the maximum iteration count of **mCU** due to the integral data. Here are two options included in **(Ahuja et al., 1993)** which have different running times.

1. $\mathbf{\Theta(n^3)}$ Floyd-Warshall
2. $\mathbf{O(mn)}$ FIFO label-correcting algorithm for shortest path (Chapter 5.4 (Ahuja et al., 1993))

As an example, if we choose Edmonds-Karp, the FIFO label-correcting algorithm for shortest path, and let mCU represent the maximum number of iterations needed to clear all negative cycles, we would be in the order of $O(n + nm^2 + mCUmn) \subseteq \mathbf{O(CUnm^2)}$

We can see that while the choice of max flow algorithm can affect the complexity, it is likely that the iteration count will force the negative cost cycle finding algorithm term to dominate the complexity function. For larger problems with specific structure, it can be useful to research a variety of options for these algorithms and find ones that suit your particular problem well.

Code Outline

As there are multiple algorithms used as a subroutine for the cycle cancelling algorithm, it lends itself well to object oriented programming languages. We can create a "Cycle-Cancelling" class with methods for each step. This will allow the user to select which algorithms they prefer to use for the max-flow and negative cycle detection subroutines. Having a default algorithm for each will allow less knowledgeable users, as well as those who don't need that level of granularity to solve problems using Cycle Cancelling.

The first method would convert the problem to a max-flow problem by taking the supply/demand values and making them into arc capacities leading to dedicated supply and demand nodes. We can write methods for the initial max-flow calculation and for each algorithm you may want to use to solve it. We will also write a method to detect negative cycles by calling a specified algorithm and again another for each algorithm you want to have available for this part. This structure makes it easy to adapt and add to. The class can also have a solve() method which calls these methods and runs until an optimal solution is found.

The other advantage of coding it this way is you could build an algorithm selector which calculates the optimal algorithm to use based on the size/nature of the problem data using the complexity for each algorithm. There is a psudocode example shown at the end of this file which is also on Github at https://github.com/pgmath/Cycle-Cancelling-Algorithm

Sources

Ahuja, R. K., Magnanti, T. L., & Orlin, J. B. (1993). *Network flows: Theory, algorithms, and applications.* Prentice-Hall.

Korte, B., & Vygen, J. (2018). *Combinatorial optimization: Theory and algorithms.* Springer.

```
[ ]  1 def FICO_preflow_push(mf_problem_data)
     2     # uses the FICO Preflow push algorithm to solve the max flow problem
     3     return init_feasible_solution
     4 def FIFO_label_correcting(mf_problem_data)
     5     # uses the FICO label correcting algorithm to find negative cycles
     6     if neg_cost_cycle=true:
     7             return neg_cost_cycle
     8         else return opt_sol
```

```
 1 class CycleCancelling(object):
 2     def __init__(self, problem_data):
 3             #initializes the object
 4         return
 5     def maxflow_converter(self, problem_data):
 6         # converts the problem into a max flow problem
 7         return mf_problem_data
 8     def maxflow_solver(self, mf_problem_data, mf_algorithm=FICO_preflow_push):
 9         # outputs the initial feasible solution from the max flow problem
10         return init_feasible_solution
11     def resgraph_update(self, init_feasible_solution):
12         # generates an array to represent the residual graph with the
13         # initial feasible solution as the input
14         return res_graph_array
15     def neg_cost_cycle_finder(self, res_graph_array, ng_cost_algorithm=FIFO_label_correcting)
16         # finds a negative cycle in the residual graph and outputs it
17         # if no negative cost cycle is found, output current solution as optimal
18         if neg_cost_cycle=true:
19             return neg_cost_cycle
20         else return opt_sol
21     def flow_augmentation(self, neg_cost_cycle, res_graph_array)
22         # augments flow in the graph and sets the res_graph_array to the new
23         # augmented values
24         res_graph_array = augmented_graph
25         return res_graph_array
26     def solve(self, problem data, mf_algorithm=FICO_preflow_push, ng_cost_algorithm=FIFO_label_correcting)
27         # solved problem with selected options
28         return opt_sol
```