Cost Scaling Quickest Flow Algorithms

Oliver Ren, Christopher Wang
oliveren@mit.edu, czw@mit.edu

December 18, 2017

Abstract

In this paper we consider two algorithms for the quickest flow problem with integer arc costs that follow the same cost scaling framework. The algorithm in [6] builds on a Goldberg-Tarjan Push-Relabel algorithm for minimum cost flow. The algorithm in [7] uses the same framework to build on Cancel-and-Tighten, a different Goldberg and Tarjan algorithm for minimum cost flow. In this paper, we compare and contrast how [6] and [7] use this framework to come up with quickest flow algorithms that have the same time complexity as the minimum cost flow algorithms they are based on.

1 Introduction

Consider the problem of evacuating people from a building, with layout given by a directed graph $G = (V, E)$. How should we model this problem? Flows of people do not move instantaneously across arcs, but instead require travel time. In the case of evacuation, we care not only about the volume of flow pushed through the network but also about the time it takes flow to move. Static flow networks have no notion of time, so they cannot capture this consideration. Instead, we turn to dynamic flow networks. In a dynamic flow network, each arc e has an associated cost τ_e, which is the time needed to travel along the arc. In this paper we will focus on algorithms for a particular dynamic flow problem: finding the *quickest flow*. We will define the *quickest flow* problem in terms of time horizon T and flow value F. Let the time horizon T be the time past which all flow has reached the sink. And, let the flow value F be the volume of total flow which passes through the network. Then for a flow value F, the objective of the quickest flow problem is to find the shortest time horizon T such that F flow can be pushed through the network in T time.

Ford and Fulkerson discuss dynamic flow networks in [3]. To solve dynamic flow problems with a discrete time horizon T, they use the method of time-expanded graphs. This involves creating multiple copies of the network graph, one for each instant in time. This approach has the advantage of turning dynamic flow problems into a static flow problems, which have been heavily studied. However, it does not give polynomial time algorithms.

In [6], Lin and Jaillet showed that the quickest flow problem with integer arc costs can be solved in the same $O\left(nm \log(n^2/m) \log(nC)\right)$ time bound as one of Goldberg and Tarjan's cost scaling minimum cost flow algorithms. Their approach made use of Goldberg and Tarjan's Push-and-Relabel [5] algorithm to achieve this

bound. But, their algorithm is weakly polynomial, because it depends on $\log(nC)$, where C is the maximum cost arc. Saho and Shigeno showed [7] that a strongly polynomial $O(nm^2(\log n)^2)$ bound is possible by making use of a different Goldberg and Tarjan cost scaling minimum cost flow algorithm: Cancel-and-Tighten [4]. For the remainder of the paper, we will differentiate between the two quickest flow algorithms [6][7] by referring to the Goldberg and Tarjan algorithm from which they are adapted.

The rest of the paper is organized as follows: In section (2), we establish notation and formally state the quickest flow problem. In section (3), we define the optimality conditions under which a minimum cost static flow can be easily converted to a quickest flow. In section (4), we describe the general framework that [6] and [7] both make use of, as well as the ways in which Push-Relabel and Cancel-and-Tighten are incorporated. In sections (5-6), we discuss the correctness and time complexity of each algorithm.

2 Background

2.1 Problem Description

Let the network $G = (V, E)$ be a directed graph with nodes V and arcs E. In this paper, we will only consider networks with a single source s and a single sink t. Each arc e has a capacity u_e and a cost τ_e. Because [7] and [6] focus on graphs with integer cost arcs, for the rest of the paper, it will be implicitly assumed that all the costs τ_e are integers. For each node w, we will define the set of arcs that flow into w as δ_w^- and the set of arcs that flow out of w as δ_w^+.

Definition 2.1 (Static Flow)**.** A static flow x does not have any notion of time: there is no cost τ_e per arc. Let x_e be the flow on arc e. In static flow networks, a flow x is called *feasible* if it satisfies flow conservation (1) and capacity constraints (2).

$$\sum_{e \in \delta_w^+} x_e - \sum_{e \in \delta_w^-} x_e = 0, \forall w \in V \setminus \{s, t\} \tag{1}$$

$$0 \leq x_e \leq u_e, \forall e \in E \tag{2}$$

For a feasible flow x, the value of the flow v is the sum of the flow which leaves s. That is, $v = \sum_{e \in \delta_s^+} x_e$. If only (2) is satisfied, but the excess (flow into a node minus flow out) for every node is non-negative, then x is called a *preflow*. If a node has non-negative excess, it is called an *active* node.

Definition 2.2 (Dynamic Flow)**.** A dynamic flow on G is given by a set of functions $f_e(t) : \mathbb{R}^+ \to \mathbb{R}$ that specify the rate of flow entering arc e at time t. Let $\rho_w(t)$ be the flow excess on a node $w \in V$ at time t. Then,

$$\rho_w(t) := \sum_{e \in \delta_w^-} \int_0^{t-\tau_e} f_e(s)ds - \sum_{e \in \delta_w^+} \int_0^t f_e(s)ds \tag{3}$$

The dynamic flow $f_e(t)$ is *feasible* if the excess flow $\rho_w(t)$ is non-negative for all $w \in V$ and the capacity constraints (4) are satisfied:

$$0 \leq x_e \leq u_e, \forall e \in E, t \geq 0 \tag{4}$$

The value F of a feasible dynamic flow is the volume of flow that leaves s.

Definition 2.3 (Temporally Repeated Flow)**.** A temporally repeated flow is a dynamic flow which is generated by a static s-t flow x. It is known that x can be decomposed into a set of paths $\mathcal{P}$ and cycles $\mathcal{C}$ [1]. Then, let x_P be the amount of flow in x on a path P. Given a feasible static flow x, a temporally repeated flow is constructed by pushing flow at a constant rate of x_P on path P for every $P \in \mathcal{P}$. The duration that we push flow on P is such that all flows can exit P by time T. That is, we should push flow from s until time $T - \sum_{e \in P} \tau_e$.

Quickest Flow

The *quickest flow* problem tries to find the shortest time horizon T needed to send a flow x of value F through the network. This is closely related to the *dynamic max flow* problem, the objective of which is to find the largest flow value F that can be sent in a given time horizon T. Ford and Fulkerson [3] were the first to show that the optimal solution to the *dynamic max flow* problem can be given as a temporally repeated flow, generated from a static flow x. The relationship between dynamic max flow and quickest flow is worth exploring as it can be exploited to show that both problems have an optimal solution that is a temporally repeated flow.

Given a static flow x with flow value v, the flow value $F(T)$ of a temporally repeated flow with time horizon T is given by:

$$F(T) = T \cdot v - \sum_{e \in E} \tau_e \cdot x_e. \tag{5}$$

The above expression comes from the definition of temporally repeated flows, namely, the fact that we push flow at a rate $x(P)$ along P for $T - \sum_{e \in P} \tau_e$ time.

Given Ford and Fulkerson's result, the problem of finding the dynamic max flow is the problem of finding a feasible flow x that gives $F^*(t)$, the maximum value of (5). As the time horizon T increases, the maximum flow $F^*(T)$ is non-decreasing. Burkard et al. showed this formally in [2]. So then, finding the *quickest flow* is the problem of finding a feasible flow x such that T is minimized, where the value of the dynamic flow must be at least F:

$$T \cdot v - \sum_{e \in E} \tau_e \cdot x_e \geq F \tag{6}$$

Rearranging, this becomes the problem of finding a feasible flow x that minimizes the time T:

$$\frac{F + \sum_{e \in E} \tau_e \cdot x_e}{v}. \tag{7}$$

Note that in the above, v and $\sum_{e \in E} \tau_e \cdot x_e$ are implicitly related by the fact that v is the value of the flow x. Moreover, the $\sum_{e \in E} \tau_e \cdot x_e$ term is the cost of the flow. If we are trying to minimize the above quantity, then, for a given v, the cost of the flow must be minimized. For a given flow value v, let us say that $g(v) := \min \sum_{e \in E} \tau_e \cdot x_e$, where x is a feasible flow of value v. Then, the problem of finding the quickest flow is the problem of finding the minimum $T(v)$, where

$$T(v) := \frac{F + g(v)}{v} \tag{8}$$

In other words, the quickest flow is generated by the static flow x^* if x^* gives $T^*(v)$ and

$$T^*(v) := \min_v T(v)$$

2.2 Network Concepts

Residual Network

Residual networks are defined as they were in class. Given a flow x, the residual network is $G(x) = (V, E(x))$, where $E(x) = E_F(x) \cup E_B(x)$. The set of forward arcs $E_F(x)$ is $\{e \in E | x_e < u_e\}$. The residual capacity $u_e(x)$ of $e \in E_F(x)$ is given by $u_e(x) = u_e - x_e$. The cost for a forward arc is still τ_e. The set of backward arcs $E_B(x)$ is $\{(w', w) \in E | e = (w, w') \in E, x_e \geq 0\}$. The residual capacity $u_e(x)$ of $e \in E_B(x)$ is given by $u_e(x) = x_e$. The cost of a backwards arc is $-\tau_e$.

Given a network G and flow x, d_{st} is defined as the minimum cost path from s to t in the residual network $G(x)$ that does not contain any cycles. d_{ts} is defined similarly. Notably, because d_{st} and d_{ts} by definition contain no cycles, they cannot be infinite in length.

Arcs with a negative reduced cost $\tau_e^\pi < 0$ are called *admissible*. The *admissible graph* is the subgraph of the residual network that only contains admissible arcs.

Node Potentials

Let π be a function for a network G that is defined for all nodes $w \in V$. Let the potential of w, π_w, be defined as the result of applying π on a node w. Then, the reduced cost τ_e^π of an arc (w, w') is defined as: $\tau_e^\pi := \tau_e - \pi_w + \pi_{w'}$. It is known that a minimum cost flow x is one in which there exists a potential function π for the residual graph $G(x)$ such that all the reduced costs are non-negative.

Subtracting flows

Given two static flows x and x', where $v(x) > v(x')$, define the flow $(x - x')$ as one where the flow across each arc is the difference between the flow across that arc in x and the flow across that arc in x'. More formally, for $e = (w, w') \in E$, $(x - x')_e = x_e - x'_e$. If this value is positive, then we assign $x_e - x'_e$ flow to the forward arc e. If it is negative, then we assign $x'_e - x_e$ flow along the backwards arc (w', w). Notably, in the residual graph $G(x)$, $(x - x')$ is a feasible flow. This flow can be decomposed into $\mathcal{P}(x')$ and $\mathcal{C}(x')$.

3 Optimality Conditions

Lin and Jaillet developed the cost scaling framework used in [6] and [7] after discovering a connection between a minimum cost static flow and a temporally repeated quickest flow [6].

Theorem 3.1. Let the static flow x generate the temporally repeated flow $\tilde{f}_e(t)$.

Then, $\tilde{f}_e(t)$ is a quickest flow iff

$$\text{(OC1)} \quad x \text{ is a min-cost flow} \tag{9}$$

$$\text{(OC2)} \quad \frac{F + \sum_{e \in E} \tau_e \cdot x_e}{v(x)} \geq -d_{st} \tag{10}$$

$$\text{(OC3)} \quad \frac{F + \sum_{e \in E} \tau_e \cdot x_e}{v(x)} \leq d_{st} \tag{11}$$

Proof. First, we show that the three conditions are necessary. Suppose that the static flow x with value v generates a quickest flow. Then, v and x give the minimum value of $T^*(v)$ as seen in (8). Then, x must be a minimum cost flow, because the numerator of $T(v)$ is the sum of F and the minimum cost of the flow with value v. If there were some lower cost flow x^*, then $T^*(v)$ would not be optimal. So (OC1) holds.

Now, we will show that (OC2) and (OC3) are necessary. The value $T^*(v)$ is a minimum if adding a small $|\epsilon| > 0$ amount of flow results in $T(v + \epsilon) \geq T^*(v)$. We show that this implies (OC2) by considering the case of adding positive ϵ flow. Consider $T(v + \epsilon)$ and the associated flow x' with flow value $v + \epsilon$. Then, x' is also a min cost flow, so it is composed of (1) some flow x with value v and (2) ϵ flow pushed along d_{st}, where d_{st} is the min cost path in the residual graph $G(x)$. Then, $g(v + \epsilon) = g(v) + \epsilon \cdot d_{st}$. So, because $T^*(v)$ is minimal,

$$\begin{aligned}
\frac{F + g(v)}{v} &\leq \frac{F + g(v) + \epsilon \cdot d_{st}}{v + \epsilon} \\
\Rightarrow \frac{(v + \epsilon)(F + g(v))}{v} - F + g(v) &\leq \epsilon \cdot d_{st} \\
\Rightarrow \frac{\epsilon \cdot (F + g(v))}{v} &\leq \epsilon \cdot d_{st} \\
\Rightarrow \frac{F + g(v)}{v} &\leq d_{st}
\end{aligned}$$

(OC3) can be proved similarly by considering the case when ϵ is negative.

Now, we show that the three conditions are sufficient. Suppose that the three conditions hold for some x with flow value $v(x)$. For the purposes of contradiction, assume that there exists a flow x^* with flow value $v(x^*)$ that satisfies the three conditions and

$$\frac{F + \sum_{e \in E} \tau_e \cdot x_e}{v(x)} > \frac{F + \sum_{e \in E} \tau_e \cdot x^*_e}{v(x^*)} \tag{12}$$

Now, we consider three cases. In the first case, if $v(x) = v(x^*)$, then $\sum_{e \in E} \tau_e \cdot x_e$ must not be a min-cost flow, which contradicts (OC1). Otherwise, if $v(x^*) > v(x)$, then recall that the flow $(x^* - x)$ can be decomposed into paths $\mathcal{P}(x)$, each of which carry flow $(x^* - x)_P$. Then,

$$\begin{aligned}
\sum_{P \in \mathcal{P}(x)} \left(\sum_{e \in P} \tau_e \right) (x^* - x)_P &\geq \sum_{P \in \mathcal{P}(x)} d_{st}(x)(x^* - x)_P \\
&= d_{st}(x)(v(x^*) - v(x))
\end{aligned}$$

But, for any path $P \in \mathcal{P}(x)$, if $(x^* - x)_P > 0$, then the reverse path $\overleftarrow{P}$ is a $t - s$ path in the residual graph. So

$$\begin{aligned}\sum_{P\in\mathcal{P}(x)} \left(\sum_{e\in P} \tau_e\right)(x^* - x)_P &= \sum_{P\in\mathcal{P}(x)} \left(-\sum_{e\in\overleftarrow{P}} \tau_e\right)(x^* - x)_P \\ &\leq \sum_{P\in\mathcal{P}(x)} -d_{ts}(x^*)(v(x^*) - v(x)) \\ &= -d_{ts}(x)(v(x^*) - v(x))\end{aligned}$$

Then, $d_{st}(x) \leq -d_{ts}(x^*)$. But then, from (OC2),

$$\begin{aligned}\frac{F + \sum_{e\in E} \tau_e \cdot x_e}{v(x)} &> \frac{F + \sum_{e\in E} \tau_e \cdot x_e^*}{v(x^*)} \\ &\geq -d_{ts}(x^*) \qquad \text{because } x^* \text{ satisfies (OC2)} \\ &\geq d_{ts}(x)\end{aligned}$$

But, this implies that $\frac{F+\sum_{e\in E} \tau_e \cdot x_e}{v(x)} > d_{ts}(x)$, which contradicts (OC3).

Using a similar argument, in the case when $v(x) > v(x^*)$, we find that $d_{st}(x) \leq -d_{ts}(x)$. Then, we can reach a contradiction of (12) by decomposing $(x - x^*)$ into a flow on a cycle $C \in \mathcal{C}(x^*)$. [7] □

4 Algorithm Descriptions

4.1 Framework

Now we describe the general cost scaling framework that both [6] and [7] make use of. The two papers each modify a cost scaling, minimum cost flow algorithm by adding a step to the scaling phase. This step ensures that when the algorithm finishes, it will have found a flow that satisfies the optimality conditions discussed in section (3). Over the course of the algorithms' run, the flows which the algorithms find do not satisfy the optimality conditions exactly. Rather, they will obey certain approximate optimality conditions, which we will detail in Section 5.1, that give rise to the optimality conditions for *quickest flow* when ϵ is small enough.

During each scaling phase, both of the *quickest flow* algorithms will find what is called an ϵ optimal flow. An ϵ optimal flow is one where the reduced cost of each arc e with respect to a potential function π is greater than $-\epsilon$:

$$\forall e \in E, \tau_e^\pi > -\epsilon$$

Both algorithms start by finding a minimum cost flow x with flow value $\frac{F}{3nC}$ where F is the flow value we are trying to get through the network as fast as possible and C is the maximum cost τ_e of an arc $e \in E$. Then by defining a potential function π which equals 0 for each arc, we can say that x is ϵ optimal for $\epsilon = C$ with respect to the potential function π, because each of the reduced costs will be greater than $-C$.

Each of the scaling phases reduces the value of ϵ by half in order to get closer and closer to the optimal flows. Because the costs of every arc is an integer, once we have $\epsilon < \frac{1}{n}$, we know that we have found a min cost flow.

Below, we list out the general framework of the two quickest flow algorithms in [6] and [7] using the general sub-procedure names *Refine*, *Reduce Gap*, and *Final Step*, which we will describe in detail immediately after.

$\epsilon \leftarrow C$
Find min cost flow x with $v(x) = \frac{F}{3n\epsilon}$
Set $\pi_w = 0$ for all $w \in V$
Modify π and x using *Reduce Gap*
while $\epsilon \geq 1/(8n)$ **do**
    $\tilde{\epsilon} \leftarrow \epsilon$
    while $\epsilon \geq \tilde{\epsilon}/2$ **do**
        Modify π, ϵ, and x using *Refine*
    end while
    Modify π and x using *Reduce Gap*
end while
Modify π and x using *Final Step*

4.2 Refine

The goal of the *Refine* step is to take a 2ϵ optimal flow and modify x and π so that we are left with a ϵ optimal flow. The two algorithms take somewhat similar different approaches to solving this problem.

Push-Relabel approaches this problem by starting with a 2ϵ optimal flow x with respect to potential function π, immediately cutting ϵ in half and then modifying x and π accordingly so that we are left with an ϵ optimal flow x' with respect to potential function π'.

Cancel-and-Tighten takes the approach of iteratively taking an ϵ optimal flow x and altering x and π to reduce ϵ by a factor of $(1 - \frac{1}{n})$ until we can go from a 2ϵ optimal flow to an ϵ optimal flow.

4.2.1 Push-Relabel

Suppose we start with a min cost flow x with respect to a potential function π for a 2ϵ optimal flow. In order to get an ϵ optimal flow, we will first create a preflow and then move the excess around until we get a circulation again. In our preflow, we will set the flow on any arc e to 0, if e has a positive reduced cost. We fully saturate the flow on any arc e with a negative reduced cost. If $\tau_e^\pi > 0$, then $x_e = 0$; if $\tau_e^\pi < 0$, then $x_e = u_e$.

Now, we will iterate through the active nodes w and run one of two procedures depending on the reduced cost of the arcs that come out of w, appropriately called *push* and *relabel*.

The goal of these procedures is to either move the excess flow away from s or relabel the potential of active nodes until there are no active nodes left. When there are no more active nodes, we will have an ϵ optimal flow because the total sum of the excess in the graph must be 0 as every arc (w, w') contributes positively to $\rho(w')$ and negatively to $\rho(w)$ and so $\sum_{w \in V} \rho(w)$ telescopes to 0.

We run *push* if there is an arc $e = (w, w')$ such that e is admissible. During *push*, we send flow with value equal to

$$\min\{\rho(w), u_e\}$$

where $\rho(w)$ is the excess at w from w to w'.

We run *relabel* otherwise, which means that there is no arc $e = (w, w')$ such that e is admissible. During *relabel*, we add ϵ to the potential of w, so

$$\pi_w := \pi_w + \epsilon.$$

This gives a ϵ optimal flow because *push* does not change reduced costs, and when we run *relabel* on a node w, arcs $e = (w', w)$ will have their reduced costs increased and arcs $e = (w, w')$ will have reduced costs $> -\epsilon$. This is true, because before the relabeling, the reduced cost of e was > 0 and so after the relabeling, the reduced cost will be $> -\epsilon$.

4.2.2 Cancel-and-Tighten

Suppose we are starting with an ϵ optimal flow x and a potential function π. First, we find another ϵ optimal flow by repeatedly sending flow along cycles C in the admissible graph. For each cycle C, we send a flow with value

$$\min\{u_x(e)|e \in C\}$$

until there are no more cycles in the admissible graph.

After doing this, we will be left with an acyclic admissible graph. This means that we can topologically sort all the nodes left in the admissible graph in order to give each node w a rank $h(w)$ which corresponds to its ordering in the topological sort. For all these nodes w, we will update their potential function π_w according to

$$\pi_w := \pi_w + \frac{h(w)\epsilon}{n}.$$

This will reduce our ϵ optimal flow to a $(1 - \frac{1}{n})\epsilon$ optimal flow. This follows immediately from the definition of reduced cost because if an arc $e = (w, w')$ is admissible, then $h(w) + 1 \leq h(w')$ and if $e = (w, w')$ is not admissible, then $\tau_e^\pi > 0$.

Then, we can go onto the next iteration of canceling cycles and tightening the potential function, but this time we set $\epsilon := (1 - \frac{1}{n})\epsilon$. Thus, as we iterate, we continue getting tighter and tighter optimal flows. In this way, we can reduce a 2ϵ optimal flow to a ϵ optimal flow.

4.3 Reduce Gap

The *Reduce Gap* step alters the flow x and the potential function π in order to ensure that $T(v)$ lies within the bounds given by the approximate optimality conditions. During the *Reduce Gap* step, we alternate between a *flow step*: sending flow from s to t along paths in the admissible graph, and a *potential step*: increasing the potential of nodes in the admissible graph reachable from s.

Let us call the flow at the beginning of the *Reduce Gap* step x. Suppose, we are at the start of a *flow step*. Let us label the current flow x'. We upper bound the amount of flow sent during a *flow step* with a function of x' and π that we call $\Delta(x', \pi)$. Now, we repeatedly find $s - t$ paths P in the admissible graph and push along P a flow of value

$$\min\left\{\min\{u_x(e))|e \in P\}, \Delta(x', \pi) - (v(x) - v(x'))\right\}$$

until the sum of the additional flow pushed equals $\Delta(x', \pi)$ or there are no more $s - t$ paths in the admissible graph.

During a *potential step*, the potential of any node w reachable from s, not including s, has its potential increased by ϵ. So, for all such w,

$$\pi_w := \pi_w + \epsilon.$$

The major differences between the two algorithms come from their respective formulas for $\Delta(x', p)$ and the point at which to stop alternating between *flow steps* and *potential steps*. These differences come about because the two algorithms have slightly different approximate optimality conditions, so they need their intermediate flows to fall within different bounds.

4.3.1 Push-Relabel

For Push-Relabel,

$$\Delta(x', \pi) := \frac{\left(F + \sum_{e \in E} \tau_e \cdot x_e\right) - (\pi_s - \pi_t + 5n\epsilon) \cdot v(x)}{6n\epsilon}.$$

And we stop the *Reduce Gap* step once

$$\frac{F + \sum_{e \in E} \tau_e \cdot x_e}{v(x)} - (\pi_s - \pi_t) \leq 7n\epsilon.$$

4.3.2 Cancel-and-Tighten

For Cancel-and-Tighten,

$$\Delta(x', \pi) := \frac{\left(F + \sum_{e \in E} \tau_e \cdot x_e\right) - (\pi_- \pi_t + (3n + 1)\epsilon) \cdot v(x)}{4n\epsilon}.$$

And we stop the *Reduce Gap* step once

$$\frac{F + \sum_{e \in E} \tau_e \cdot x_e}{v(x)} - (\pi_s - \pi_t) \leq (3n + 1)\epsilon.$$

4.4 Final Step

The procedure for *Final Step* is the same for both algorithms. *Final Step* makes one final modification to a min flow x that follows approximate optimality conditions. This results in a flow x^* that satisfies the actual optimality conditions, as defined in Section 3.

First, we need to compute d_{st}, which is the shortest path from s to t in the residual graph $G(x)$. Then, we will consider the subgraph $G_d(x)$ which consists of only arcs on a shortest path from s to t.

If $d_{st} < \frac{F + \sum_{e \in E} \tau_e \cdot x_e}{v(x)}$, then find a max $s - t$ flow in $G_d(x)$ and send the maximum flow from s to t to get the flow x^*. Otherwise, do nothing and set $x^* := x$. A temporally repeated flow of x^* will be a quickest flow.

5 Correctness Proofs

In this section, we will show that the optimality conditions from section (3) will be met at the end of both algorithms. We do this by showing that at the end of each Refine and Reduce-Gap step, certain approximate optimality conditions hold. And after the final step, these approxiate optimality conditions give rise to the optimality conditions defined in Section 3 that guarantee a min cost flow is also a quickest flow.

5.1 Approximate Optimality

As long as certain approximate optimality conditions are met, actual optimality will be achieved when the algorithm terminates. Let c_1, c_2 be functions of n that differ between Push-Relabel and Cancel-and-Tighten.

$$\begin{aligned}
&\text{(AOC1)} \quad \tau_e^\pi \geq -\epsilon, \forall e \in E(x) \\
&\text{(AOC2)} \quad \frac{F + \sum_{e \in E} \tau_e \cdot x_e}{v(x)} \geq \pi_s - \pi_t + c_1\epsilon \\
&\text{(AOC3)} \quad \frac{F + \sum_{e \in E} \tau_e \cdot x_e}{v(x)} \leq \pi_s - \pi_t + c_2\epsilon
\end{aligned}$$

For Push-Relabel, $c_1 := 5n, c_2 := 7n$ and for Cancel-and-Tighten, $c_1 := 3n, c_2 := 3n + 1$. Again, the difference in bounds is due to the different ways each algorithm uses the approximate optimality conditions.

A flow and potential that satisfy (AOC1) is called ϵ optimal. As mentioned in Section 4.1, if $\epsilon < \frac{1}{n}$ then we will have found a min-cost flow.

But why do (AOC2) and (AOC3) make sense as approximations for (OC2) and (OC3)? Let's examine (OC2) first. Assuming that (AOC1) holds, then

$$\begin{aligned}
-d_{ts} = -\sum_{e \in P_{ts}(x)} \tau_e &= -\left(\sum_{e \in P_{ts}(x)} \tau_e^\pi + \pi_s - \pi_t \right) \\
&\leq \pi_s - \pi_t + (n+1)\epsilon
\end{aligned}$$

It is immediately apparent that (AOC1) and (AOC2) imply that (OC2) holds since n is always positive, so $c_1\epsilon > (n+1)\epsilon$ for both values of c_1. Therefore, as long as both algorithms maintain (AOC1) and (AOC2) throughout, when the algorithms terminate with an $\epsilon < \frac{1}{n}$, both (OC1) and (OC2) will hold.

Similarly, we can show that

$$d_{st} \geq \pi_s - \pi_t - (n-1)\epsilon.$$

This will be used later on to show that as long as (AOC3) holds before we run *Final Step*, then after we run *Final Step*, we will have found a flow where (OC3) also holds, and so we will have found a min cost flow that can be temporally repeated to obtain a quickest flow. Specifically, this means that as long as (AOC3) holds after each *Reduce Gap* phase, in addition to maintaining (AOC1) and (AOC2), our algorithm will terminate with a quickest flow.

5.2 Refine

We showed in Sections 4.2.1 and 4.2.2 that after each *Refine* step, (AOC1) holds. In this section, we bound $T(V)$ and show that (AOC2) holds. In addition, we will maintain a looser upper bound than (AOC3) during the *Refine* step, but we will tighten the upper bound to (AOC3) during the *Reduce Gap* step. Intuitively, these proofs will rely on consequences of only running *push* and *relabel* on active nodes.

5.2.1 Push-Relabel

The refine step saturates all admissible arcs and then pushes flow and relabels potentials until all excess flow has been distributed. This procedure is the same as Goldberg and Tarjan's. Let x be the flow before some excess flow is distributed and x' be the flow afterwards. The goal of this step is to get a tighter bound on $T(v)$, which means that we need to show that $T(v)$ is growing since it needs to satisfy a higher lower bound. At the same time, we also need to show that $T(v)$ does not grow too fast since it also needs to satisfy a tighter upper bound.

By analyzing the flow in $x - x'$, whose value must be positive because we only send flow from active nodes along admissible arcs, we are able to bound $T(v)$ after *Refine* is over,

$$\pi_s - \pi_t + 5n\epsilon \leq T(v) \leq \pi_s - \pi_t + 18n\epsilon.$$

The proof of this bound involves using the acyclic nature of the admissible graph after each *Refine* step and the fact, as proven by Golberg and Tarjan, that each node's potential can increase by at most $3n\epsilon$ after *Refine* to bound how much $T(v)$ can change after *Refine*. [5] The full proof, which can be found in the paper by Lin and Jaillet, uses these facts to bound the value of the new flow x' between functions of the value of the old flow x in order to bound the value of $\pi_{s'} - \pi_{t'}$ using $\pi_s - \pi_t$ and thus bound $T(v)$. [6]

5.2.2 Cancel-and-Tighten

The proof of how (AOC1) is satisfied is shown above in Section 4.2.2. We will show here that the other two approximate conditions are satisfied. We want to show that at the end of the Refine step, (AOC2) holds and (AOC3) is close to being satisfied. Say that we begin the refine step with an ϵ optimal flow and end with an ϵ' optimal flow. As mentioned above, each cancel step will reduce ϵ by a factor of $(1 - \frac{1}{n})$, so $(1 - \frac{1}{n})^k \epsilon = \epsilon'$. At the lth iteration of the Cancel step, we modify π_w for all nodes $w \in V$ by at most $(1 - \frac{1}{n})^l \epsilon$. So, we have

$$\begin{aligned} |(\pi_s - \pi_t) - (\pi'_s - \pi'_t)| &\leq \sum_{l=1}^{k} (1 - 1/n)^l \epsilon_0 \\ &= (n-1)(\epsilon_0 - \epsilon). \end{aligned}$$

Then, $v(x) = v(x')$, so $(x - x_0)$ can be decomposed into flows on cycles in $\mathcal{C}(x_0)$.

$$\begin{aligned} \sum_{e \in E(x)} \tau_e^{\pi}(x - x_0)(e) &\geq \sum_{e \in E^-(x)} \tau_e^{\pi}(x - x_0)(e) \\ &\geq -\epsilon_0(n-1)v(x). \end{aligned}$$

Then,

$$\frac{F + \sum_{e \in E} \tau_e \cdot x_e}{v(x)} > \frac{F + \sum_{e \in E} \tau_e \cdot x_e - n\epsilon x_e}{v(x)} = \frac{F + \sum_{e \in E} \tau_e \cdot x_e - n\epsilon v(x)}{v(x)}.$$

5.3 Reduce Gap

Our goal in this section is to show that after *Reduce Gap*, we will be able to tighten the upper bound guarantee after *Refine* so that (AOC3) holds. However, we need to be careful that we do not invalidate (AOC1) or (AOC2), so we also show in this section that (AOC1) and (AOC2) also hold when *Reduce Gap* finishes. The correctness proofs in this section rely on the specific formulas chosen for $\Delta(x', \pi)$. Those formulas were based off how far away our particular values of $T(v)$ were from the bounds that we need in order to satisfy (AOC2) and (AOC3). And so, they are used to prevent us from going outside our bounds during *Reduce Gap.*

5.3.1 Push-Relabel

Based on the stopping condition for *Reduce Gap*, when *Reduce Gap* is finished, (AOC3) holds immediately. However, we do need to show that *Reduce Gap* actually terminates, and then show that after it terminates, (AOC1) and (AOC2) also hold.

This is why our specific formula for $\Delta(x', \pi)$ was chosen. By bounding how much flow we can send from s to t, we can bound how much $T(v)$ changes.

Because we know that the flow sent during a *flow step* of *Reduce Gap* cannot exceed $\Delta(x', \pi)$ for the flow x' and potential π prior to the *flow step*, we will be able to upper bound the value of $\pi_s - \pi_t$,

$$\pi_s - \pi_t \leq T(v') - 5n\epsilon$$

where v' is the value of the new flow after the *flow step*. This is easily done by upper bounding the change in flow value and rearranging the variables in $\Delta(x', \pi)$ so that $\pi_s - \pi_t$ is on the left hand of the $\leq$ inequality. [6]

During *Reduce Gap*, the sink node t will never have its potential increased because it is never an active node, and the sink node s purposefully also never has its potential increased, thus π'_s and π'_t are the same throughout all the potential changes, and so this upper bound for $\pi_s - \pi_t$ holds after *Reduce Gap* is terminated.

(AOC1) holds after *Reduce Gap* because if a node w is reachable from s during the *potential step*, then for any arc (w, w'), if w' is reachable then e is admissible and so τ_e^π doesn't change. Otherwise, e is non-admissible and so $\tau_e^\pi > 0$. For any arc $e = (w', w)$, just by definition τ_e^π cannot decrease since all potentials are increased by the same amount, ϵ, during a *potential step.*

5.3.2 Cancel-and-Tighten

In [7], the approximate optimality conditions (AOC1) and (AOC2) are always satisfied at the end of the refine step. Then, the purpose of the reduce gap step is to ensure that (AOC3) can be satisfied while maintaining (AOC1) and (AOC2). Recall that for Cancel-and-Tighten, we push flow along admissible paths until (AOC3) is

satisfied. So, will the procedure terminate? Using a lot of algebra, Saho and Shigeno show that each iteration of the procedure increases $\pi_s - \pi_t$ by ϵ, so progress will definitely be made, until $\pi_s - \pi_t \geq T(v)$, at which point (AOC3) will be satisfied. The proofs that (AOC1) and (AOC2) are maintained are essentially the same as for Push-Relabel except that for (AOC2), we get a tighter bound because our stopping condition is different.

5.4 Final Step

Now that we've gotten ϵ really small, we know that there are no negative cycles in the residual graph, so it is a min cost flow and (OC1) must be satisfied. Again, this is because $\epsilon < 1/n$ and the flow is ϵ optimal. Thus every cycle will have cost > -1. But because the costs are integers, this implies that there are no negative cycles in the residual graph.

Finally, (AOC1) and (AOC2) is satisfied, so (OC2) must be satisfied. However, (AOC3) being satisfied does not imply that (OC3) is satisfied because (OC3) gives a tighter bound than (AOC3). If it happens that (OC3) is satisfied, then we are done. Otherwise, let $G_s(x)$ denote the subgraph induced by all arcs of the residual graph which are on a shortest $s - t$ path. We can satisfy (OC3) by finding a maximum flow y in the $G_s(x)$ and updating x by pushing the maximum flow y from s to t. Let us call the updated flow x^*.

Because we augment only using shortest $s-t$ paths, we know that x^* also satisfies (OC1) and so is a min cost flow. Now, because there cannot be negative cycles in x, we know that $d_{st}(x) = -d_{ts}(x^*)$. Thus, because the flow in x^* is greater than in x, we get that (OC2) holds. (AOC3) in conjunction with $\epsilon < \frac{1}{8n}$, which is the stopping condition for the full algorithm, means that $T(v) - d_{st} < 1$. Because of integer costs, we know that $d_{st}(x) + 1 \leq d_{st}(x^*)$ since the shortest path must increase when all shortest paths in x are saturated in *Final Step*. This then gives us that (OC3) holds. Thus, we have that all three optimality conditions (OC1), (OC2), and (OC3) hold and so x^* can be temporally repeated to get a quickest flow.

6 Runtime Analyses

Note that the biggest difference between Goldberg and Tarjan's Push-Relabel algorithm and Lin and Jaillet's algorithm was the addition of the *Reduce Gap* step during every iteration of the scaling phase. The same thing can be said for Goldberg and Tarjan's Cancel-and-Tighten algorithm and Saho and Shigeno's algorithm.

Now for both algorithms, preprocessing and *Final Step* involve running at most one max flow iteration, and so their overall runtimes will be dominated by their cost scaling phases.

Thus, if their *Reduce Gap* can have the same runtime as the *Refine* step for their respective cost scaling min cost flow algorithm, then the runtimes of the two quickest flow algorithms will have the same time complexity as the cost scaling min cost flow algorithm they were modeled after.

Both papers use the dynamic tree data structure introduced by Sleator and Tarjan as black boxes to improve the runtime of their algorithms, so we will do the same throughout our runtime analysis. [8]

6.1 Push-Relabel

There are a total of $O(\log(nC))$ cost scaling phases and each phase contains one iteration of *Refine* and one iteration of *Reduce Gap*. An analysis from the original Goldberg and Tarjan paper [5] shows that each iteration of *Refine* is bounded by $O(n^3)$.

By bounding the number of times each node, in particular the source node s, can participate in a *potential step* by $O(n)$, we are able to bound the number of saturating flows calls pushed during *flow steps* by $O(mn)$ and non-saturating flows pushed during *flow steps* by $O(n^3)$ by considering how many consecutive *Push* calls can be made before there are no more active nodes. Thus, the runtime of a single *Reduce Gap* iteration is $O(n^3)$. The full details of the derivation can be found in the Lin and Jaillet paper. [6]

Thus, the runtime of the Push-Relabel quickest flow algorithm is the same as the Push-Relabel min cost flow algorithm, which had its $O(n^3 \log nC)$ runtime improved using dynamic trees to get the weakly polynomial runtime

$$O\left(nm \log\left(\frac{n^2}{m}\right) \log(nC)\right). \text{ [5]}$$

6.2 Cancel-and-Tighten

Similarly, the runtime of the *Relabel* phase using dynamic trees is $O(nm \log(n))$. The analysis can be found in the Golberg and Tarjan Cancel-and-Tighten paper. [4]

Naively, the runtime per iteration of *Reduce Gap* in Cancel-and-Tighten would be the same as in Push-Relabel because the procedure is almost identical between the two. However, an analysis that can be found in the original paper using fixed arcs - essentially arcs whose reduce costs will not change as ϵ gets smaller - takes advantage of the tighter approximate optimality conditions for *Reduce Gap* to get a better runtime. Again, dynamic trees are used to get the strongly polynomial runtime of

$$O(nm^2 \log^2 n). \text{ [6][4]}$$

7 Conclusion

The two algorithms described in this paper can serve as a template to convert future cost scaling min cost flow algorithms into quickest flow algorithms with the same time complexity. Because the quickest flow problem is equivalent to solving the min cost flow problem with added optimality conditions, as long as we can alter the cost scaling, min cost flow algorithms to maintain certain approximate optimality conditions throughout the algorithm that are equivalent to the real optimality conditions when we scale our costs down to a small enough ϵ, we will have found a quickest flow. As these two algorithms showed, this may require the use of a *Reduce Gap* step with algorithmic-specific constants to maintain the bounds during each scaling phase. But as long as this *Reduce Gap* has the same runtime per iteration as the *Refine* steps in the original cost scaling min cost flow algorithms, we will have successfully taken the min cost flow algorithm and turned it into a quickest flow algorithm with the same runtime. Not only does this framework show that dynamic quickest flow problems are no more complicated than min cost flow problems, it also

gives us a malleable way to take an algorithm for one problem and convert it to an algorithm that solves the other.

8 Acknowledgments

We would like to thank Professor Karger for teaching this course. We also would like to thank Andy Wei and Rogers Epstein for their part in reviewing and improving our paper.

References

[1] Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. Network flows: theory, algorithms, and applications. 1993.

[2] Rainer E Burkard, Karin Dlaska, and Bettina Klinz. The quickest flow problem. *Zeitschrift für Operations Research*, 37(1):31–58, 1993.

[3] Lester Randolph Ford Jr and Delbert Ray Fulkerson. *Flows in networks*. Princeton university press, 2015.

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

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

[6] Maokai Lin and Patrick Jaillet. On the quickest flow problem in dynamic networks: a parametric min-cost flow approach. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1343–1356. Society for Industrial and Applied Mathematics, 2015.

[7] Masahide Saho and Maiko Shigeno. Cancel-and-tighten algorithm for quickest flow problems. *Networks*, 69(2):179–188, 2017.

[8] Daniel D. Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, June 1983.