On the Quickest Flow Problem in Dynamic Networks – A Parametric Min-Cost Flow Approach*

Maokai Lin † Patrick Jaillet ‡

October 2014

Abstract

We consider the quickest flow problem in dynamic networks with a single source s and a single sink t: given an amount of flow F, find the minimum time needed to send it from s to t, and the corresponding optimal flow over time.

We introduce new mathematical formulations and derive optimality conditions for the quickest flow problem. Based on the optimality conditions, we develop a new cost-scaling algorithm that leverages the parametric nature of the problem. The algorithm solves the quickest flow problem with integer arc costs in $O(nm \log(n^2/m) \log(nC))$ time, where n, m, and C are the number of nodes, arcs, and the maximum arc cost, respectively.

Our algorithm runs in the same time bound as the cost-scaling algorithm by Goldberg and Tarjan [10, 11] for solving the min-cost flow problem. This result shows for the first time that the quickest flow problem can be solved within the same time bound as one of the fastest algorithms for the min-cost flow problem. As a consequence, our algorithm will remain one of the fastest unless the quickest flow problem can be shown to be simpler than the min-cost flow problem.

1 Introduction

Static network flow models have been extensively studied and widely used in the past decades to formulate many real-world problems. Many applications, however, require a time dimension to be considered: instead of a picture of the network at one instant, we need to consider a video of the network flow over a period of time. These applications arise from various fields of studies, including transportation [14], evacuation planning [23, 12], job scheduling [2], electronic communication [5, 16], network routing [18, 3, 19], and parallel computing [15]. Such a requirement leads to the dynamic network flow model [8, 4, 13]. It is also called network flow over time [6].

A generic approach for solving optimization problems in dynamic networks is due to Ford and Fulkerson [8]. They showed that one can use the time-expansion technique to convert dynamic networks with discrete time horizon into static networks, then solve the problem using algorithms developed for static networks. This approach, however, leads to pseudo-polynomial algorithms because the number of nodes and arcs in time-expanded networks are normally proportional to the time horizon T, which is exponential in the input size $\log T$.

Although the general approach fails to provide polynomial-time algorithms for every problem in dynamic networks, there are efficient algorithms for some special problems. Ford and Fulkerson [7] showed that the *dynamic max-flow problem*, which is to send as much dynamic flow as possible through a single-source-single-sink network in a given time T, can be formulated as a min-cost flow problem in a slightly modified static network. As a result, the dynamic max-flow problem can be solved by various min-cost flow algorithms in polynomial time.

A closely related optimization problem in dynamic networks is the *quickest flow problem*, which is to find the minimum time T^* needed to send a given amount of dynamic flow F from a source node s to a sink node t. An immediate approach is to use binary search to find the minimum time T such that the maximum amount of dynamic flow that can be sent from s to t within time T exceeds F. This naive approach, however, does not automatically yield a fully polynomial-time algorithm. Burkard et al. [4] improved the naive approach and developed polynomial-time algorithms by using Newton's method and Megiddo's parametric search [17]. Their algorithms need to repeatedly call subroutines that solve the min-cost flow problem.

An open question remains: since the quickest flow problem is so similar to the dynamic max-flow problem, is it possible to develop an algorithm that solves the quickest flow problem within the same time bound as the min-cost flow problem? This would be impossible if we need to repeatedly solve the min-cost flow problem as a subroutine.

In this paper, we provide an affirmative answer to this question. We observe that the quickest flow problem is essentially a parametric min-cost flow problem.

*Accepted, ACM-SIAM Symposium on Discrete Algorithms, SODA 2015

†Operations Research Center, MIT, Cambridge, MA, 02139; lmk@csail.mit.edu

‡Department of Electrical Engineering and Computer Science, and Operations Research Center, MIT, Cambridge, MA, 02139; jaillet@mit.edu

By extending the cost-scaling algorithm introduced by Goldberg and Tarjan [10, 11], we design a new cost-scaling algorithm for the quickest flow problem that runs in the same time bound as Goldberg and Tarjan's algorithm. Because their algorithm remains one of the fastest for solving the min-cost flow problem in terms of the worst-case time bound, our result shows that the quickest flow problem can be solved within the same time bound as one of the best algorithms for the min-cost flow problem. Moreover, unless the quickest flow problem can be shown to be simpler than the min-cost flow problem, our algorithm will remain one of the fastest for solving the quickest flow problem.

In addition, our result shows that the preflow push algorithm framework is well suited for parametric extensions. Gallo et al. [9] showed that by extending the preflow push algorithm for the maximum flow problem, one could solve the parametric max-flow problem within the same time bound as of the maximum flow problem. Our result shows that a similar extension can be obtained for the quickest flow problem, which can be viewed as a parametric min-cost flow problem with a single source and a single sink.

1.1 Related Work The dynamic max-flow problem and the quickest flow problem are two fundamental optimization problems in dynamic networks. Consider a network $G = (V, E)$ with a single source s and a single sink t. In this network, each arc e is associated with an arc cost (also called transition time) τ_e and an arc capacity (also called maximum flow rate) u_e. The dynamic max-flow problem is to send as much flow as possible from s to t within a given time T, and the quickest flow problem is to find the minimum time T^* needed to send a given amount of flow F from s to t.

To solve the dynamic max-flow problem, Ford and Fulkerson [7] introduced a solution form called *temporally-repeated flow*. With any feasible static flow x in a network, we can obtain a temporally-repeated solution in the following way: Perform a flow decomposition on flow x to obtain a set of paths P and the corresponding flow $x(p)$ for each path $p \in P$. Send flow on each path $p \in P$ in a constant rate $x(p)$ from time 0 to $t(p) := \max\{T - \tau(p), 0\}$, where $\tau(p) := \sum_{e \in p} \tau_e$ is the cost of path p.

Ford and Fulkerson [7] showed that any temporally-repeated solution corresponding to a feasible static flow x is a feasible dynamic flow, and there exists an optimal solution to the dynamic max-flow problem that is in a temporally-repeated form. Using this result, they further showed that a dynamic max-flow problem can be formulated as a min-cost flow problem by adding an arc (t, s) with cost $\tau_{ts} = -T$ and infinite capacity to the original network. After solving this min-cost flow problem, one can transform the optimal static flow into a temporally-repeated flow, which is an optimal solution to the original dynamic max-flow problem.

The quickest flow problem is closely related to the dynamic max-flow problem. Burkard et al. [4] showed that the maximum amount of flow that can be sent through a network increases with time T. Thus, one can do a binary search over time T and solve a dynamic max-flow problem in each iteration, until the minimum time needed to send the given amount of flow F is found. Burkard et al. [4] further used Newton's method to improve the naive binary search and obtained a time bound of $O(\log(nU)\mathrm{MCF}(n, m))$, where n, m, and U are the number of nodes, arcs, and the maximum arc capacity, respectively, and $\mathrm{MCF}(n, m)$ is the time bound for solving one min-cost flow problem. They also showed that the quickest flow problem can be solved in strongly-polynomial time using Megiddo's parametric search [17].

1.2 Our Results In this paper, we consider the quickest flow problem in dynamic networks with a single source s and single sink t. Given a static flow x, we call a temporally-repeated flow converted from x a *temporally-repeated flow of* x. By transforming the formulation by Ford and Fulkerson [7], we show that the quickest flow problem can be formulated with the following fractional programming problem:

$$
\begin{aligned}
T^* = &\min \frac{F + \sum_{e \in E} \tau_e \cdot x_e}{v} \\
&\text{s.t.} \sum_{e \in \delta_w^+} x_e - \sum_{e \in \delta_w^-} x_e = \begin{cases} v & w = s \\ -v & w = t \\ 0 & \text{otherwise} \end{cases} \\
&0 \le x_e \le u_e \qquad \forall e \in E
\end{aligned} \tag{1.1}
$$

For any optimal solution x^* of (1.1), a temporally-repeated flow of x^* is an optimal dynamic flow for the quickest flow problem, and T^* is the shortest time needed to send F amount of flow from s to t.

By setting $\theta := 1/v$ and substituting x_e with $x'_e := x_e/v = x_e \cdot \theta$, we can linearize the objective function of (1.1) and obtain a linear programming problem (1.2):

$$
\begin{aligned}
T^* = &\min F \cdot \theta + \sum_{e \in E} \tau_e \cdot x_e \\
&\text{s.t.} \sum_{e \in \delta_w^+} x_e - \sum_{e \in \delta_w^-} x_e = \begin{cases} 1 & w = s \\ -1 & w = t \\ 0 & \text{otherwise} \end{cases} \\
&0 \le x_e \le u_e \cdot \theta \qquad \forall e \in E
\end{aligned} \tag{1.2}
$$

Let (θ^*, x^*) be an optimal solution to the linear program (1.2), then a temporally-repeated flow of the static

flow x^*/θ^* is an optimal solution to the quickest flow problem.

Observe that if we fix v in (1.1), it becomes a min-cost flow problem, with supply at s and demand at t both equal to v. Therefore, the quickest flow problem is essentially a parametric min-cost flow problem with respect to v. Using this observation, we derive the following optimality conditions for the quickest flow problem:

THEOREM 1.1. *A temporally-repeated flow of a feasible static flow x is optimal if and only if x is a min-cost flow with respect to its flow value v, and satisfies*

$$-d_{ts} \leq \frac{F + \sum_{e \in E} \tau_e \cdot x_e}{v} \leq d_{st} \tag{1.3}$$

where the flow value v is the amount of static flow sent from node s to t, and d_{ts} and d_{st} are the costs of the shortest t-s path and s-t path in the residual network of flow x, respectively.

Based on these optimality conditions, we introduce a new algorithm for solving the quickest flow problem with integer arc costs. Our algorithm adds an extra step to each scaling phase of the cost-scaling algorithm by Goldberg and Tarjan [11] for solving the min-cost flow problem, but still runs in $O(n^3 \log(nC))$ time. It can be further improved to run in $O(nm \log(n^2/m) \log(nC))$ time by applying the dynamic tree data structure developed by Sleator and Tarjan [21, 22]. Here n, m, and C are the number of nodes, arcs, and the maximum arc cost, respectively.

Organization: In Section 2, we show that the quickest flow problem can be formulated in terms of the fractional programming problem in (1.1) and derive optimality conditions based on this formulation. In Section 3, we introduce the new cost-scaling algorithm with analysis of its correctness and time complexity. We conclude the paper in Section 4 with some final remarks.

2 Mathematical Models and Optimality Conditions

2.1 Preliminaries

Network: $G = (V, E)$ is a network with a set of nodes V and arcs E. Let $n := |V|$ be the number of nodes, and $m := |E|$ be the number of arcs. We only consider networks with a single source $s \in V$ and a single sink $t \in V$. Each arc $e = (w, w') \in E$ is associated with a cost, or free-flow transition time $\tau_e \geq 0$, and a capacity, or maximum flow rate $u_e \geq 0$. In this paper, we assume τ_e and u_e do not change over time. We define $\delta_w^+ := \{(w, w') \in E, \forall w' \in V\}$ as the set of outgoing arcs from node w, and $\delta_w^- := \{(w', w) \in E, \forall w' \in V\}$ as the set of incoming arcs into node w.

Feasible Flow: If a static flow x satisfies the flow conservation conditions

$$\sum_{e \in \delta_w^+} x_e - \sum_{e \in \delta_w^-} x_e = \begin{cases} v & w = s \\ -v & w = t \\ 0 & \text{otherwise} \end{cases} \tag{2.4}$$

where $v \geq 0$, and the capacity constraints

$$0 \leq x_e \leq u_e \qquad e \in E \tag{2.5}$$

then we call x a feasible flow. We call v the flow value of x, and x a feasible flow with flow value v.

Preflow and Node Excess: A preflow $\hat{x}$ is a static flow that satisfies the capacity constraints (2.5) but not necessarily the flow conservation conditions (2.4). For a preflow $\hat{x}$, we define flow excess $e(w)$ on a node $w \in V$ as the sum of incoming flow minus the sum of outgoing flow. Formally, we have

$$e(w) := \sum_{e \in \delta_w^-} \hat{x}_e - \sum_{e \in \delta_w^+} \hat{x}_e \qquad \forall w \in V \tag{2.6}$$

In a preflow, $e(w) \geq 0$ for every $w \in V \backslash \{s, t\}$. In this paper, we will use x to denote a preflow in order to simplify notations. If x also satisfies the conservation constrains (2.4), we will emphasize that x is a *feasible* flow.

Flow Decomposition: It is well known (see Ahuja et al. [1], Sect. 3.5 for example) that for any feasible flow x, there exists a flow decomposition $\langle P, x(p) \rangle$, where P is a set of s-t-paths and cycles in G, and $x(p) \geq 0, \forall p \in P$ is the flow on the path or cycle p.

Residual Network: Residual networks are widely used for solving network flow problems. A residual network $G(x) = (V, E(x))$ with respect to a given flow x is defined as follows: Replace each arc $e = (w, w')$ in the original network G by a forward arc $e' = (w, w')$ and a backward arc $e'' = (w', w)$. The forward arc e' has a cost $\hat{\tau}_{e'} := \tau_e$ and residual capacity $r_{e'} := u_e - x_e$. The backward arc $e'' = (w', w)$ has a cost $\hat{\tau}_{e''} := -\tau_e$ and residual capacity $r_{e''} := x_e$. In this paper, we use $\hat{\tau}_e$ to indicate the cost of an arc e in a residual network. We have the following theorem in a residual network with respect to flow x:

THEOREM 2.1. (THEOREM 9.1, AHUJA ET AL. [1]) *If there does not exist any cycle with negative cost in the residual network $G(x)$, flow x is a min-cost flow.*

For further discussions of residual networks, we refer the reader to Ahuja et al. [1], Sect. 2.4.

Node Potentials and Reduced Cost: Node potentials arise from the dual formulation of the min-cost flow problem. They are the dual variables corresponding

to the flow conservation constraints (2.4). With a set of node potentials π, the reduced cost of arc $e = (w, w')$ in the residual network is defined as $c_e^\pi := \pi_{w'} + \hat{\tau}_e - \pi_w$, where $\hat{\tau}_e$ is the cost of arc e in the residual network $G(x)$.

Shortest Simple Path: A simple w-w' path is defined as a path from node w to w' without loop. We use $d_{st}(x)$ to denote the cost of a shortest simple s-t path in the residual network $G(x)$, and $d_{ts}(x)$ the cost of a shortest simple t-s path in $G(x)$. Such costs could be either positive or negative. Note that although the cost of a shortest path between two nodes could be minus infinity in a residual network with negative cycles, the cost of a shortest *simple* path is always bounded. In this paper, we use shorthands d_{st} and d_{ts} if the flow they correspond to is clear in the context.

Dynamic Flow (Flow over Time) A dynamic flow consists of a set of Lebesgue-integrable functions $f_e(t)$ that represents the rate of flow on each arc $e \in E$ over time t. The dynamic flow excess over time on node $w \in V$ is defined as:

$$g_w(t) := \sum_{e \in \delta_w^-} \int_0^{t-\tau_e} f_e(s)ds - \sum_{e \in \delta_w^+} \int_0^{t} f_e(s)ds \tag{2.7}$$

A feasible flow f must satisfy the capacity constraints $f_e(t) \le u_e, \forall e \in E, \forall t \ge 0$, and the node excess $g_w(t)$ must be non-negative at all time for all nodes. See Sect. 1 of Skutella [20] for a complete discussion of the definition of dynamic flows.

Temporally-Repeated Flow: First introduced by Ford and Fulkerson [7], temporally-repeated flow is a special type of dynamic flow. It turns any feasible static flow x into a feasible dynamic flow. For any feasible static flow x and a time horizon T, one can perform a flow decomposition on x and obtain a tuple $\langle P, x(p)\rangle$, where P is a set of paths and $x(p)$ is the flow on each path $p \in P$. A temporally-repeated flow sends flow over each path $p \in P$ at a constant rate $x(p)$ from time 0 to $t(p) := \max\{T - \tau(p), 0\}$, where $\tau(p) := \sum_{e \in p} \tau_e$ is the cost of path p. Mathematically, a temporally-repeated flow f corresponding to a flow decomposition $\langle P, x(p)\rangle$ of a feasible static flow x and time horizon T is defined as:

$$f_e(t) := \sum_{p \in P_e(t)} x(p),\ \forall e = (w, w') \in E, t \in [0, T) \tag{2.8}$$

where $P_e(t) := \{p \in P : e \in P \text{ and } \tau(p_{s,w}) \le t \text{ and } \tau(p_{w',t}) < T - t)\}$. In the definition, $\tau(p_{s,w})$ is the sum of costs of the arcs from node s to node w on path p, and $\tau(p_{w',t})$ is the sum of costs of the arcs from node w' to node t on path p. Again, see Sect. 1 of Skutella [20] for a complete discussion.

2.2 Formulations and Optimality Conditions Ford and Fulkerson [7] showed that for the dynamic max-flow problem, there exists an optimal solution in the temporally-repeated form. Therefore, a dynamic max-flow problem with time horizon T can be formulated as:

$$\begin{aligned} F^*(T) = \max\ & T \cdot v - \sum_{e \in E} \tau_e \cdot x_e \\ \text{s.t.}\ & \sum_{e \in \delta_w^+} x_e - \sum_{e \in \delta_w^-} x_e = \begin{cases} v & w = s \\ -v & w = t \\ 0 & \text{otherwise} \end{cases} \\ & 0 \le x_e \le u_e \qquad \forall e \in E \end{aligned} \tag{2.9}$$

LEMMA 2.1. (BURKARD ET AL. [4]) *$F^*(T)$ is non-decreasing as time T increases.*

Lemma 2.1 implies that finding an optimal solution to the quickest flow problem with a given amount of flow F is equivalent to finding the minimum time T^* such that $F^*(T^*) \ge F$. This observation is used to prove Theorem 2.2.

THEOREM 2.2. *The quickest flow problem can be formulated as the fractional programming problem (1.1) shown in Section 1.2.*

Proof. By Lemma 2.1, the quickest flow problem can be formulated as:

$$\begin{aligned} T^* = \min\ & T \\ \text{s.t.}\ & T \cdot v - \sum_{e \in E} \tau_e \cdot x_e \ge F \\ & \sum_{e \in \delta_w^+} x_e - \sum_{e \in \delta_w^-} x_e = \begin{cases} v & w = s \\ -v & w = t \\ 0 & \text{otherwise} \end{cases} \\ & 0 \le x_e \le u_e \qquad \forall e \in E \end{aligned} \tag{2.10}$$

In this formulation, T, x, and v are variables. The first constraint in (2.10) can be rearranged as:

$$T \ \ge\ \frac{F + \sum_{e \in E} \tau_e \cdot x_e}{v} \tag{2.11}$$

Note that $v \neq 0$. Because if $v = 0$, no flow will be sent, and the first constraint in (2.10) cannot hold. Since (2.11) is the only constraint for the variable T, we can move it to the objective function and obtain the fractional programming formulation (1.1). □

Note that by setting $\theta := 1/v$ and substituting x_e with $x'_e := x_e/v = x_e \cdot \theta$, we can linearize (1.1) and obtain the linear form (1.2).

In (1.1), if we consider v as a parameter, the objective function becomes a summation of the term F/v and $1/v$ times the optimal value of a min-cost flow problem. In order to use this property to prove the optimality conditions in Theorem 1.1, we first define the following function for the min-cost flow problem with flow value v:

$$
\begin{aligned}
g(v) = \min & \sum_{e \in E} \tau_e \cdot x_e \\
\text{s.t.} & \sum_{e \in \delta_w^+} x_e - \sum_{e \in \delta_w^-} x_e = \begin{cases} v & w = s \\ -v & w = t \\ 0 & \text{otherwise} \end{cases} \\
& 0 \le x_e \le u_e \qquad \forall e \in E
\end{aligned} \tag{2.12}
$$

We further define function $T^*(v)$ as:

$$
T^*(v) := \frac{F + g(v)}{v} \tag{2.13}
$$

Using this definition, we can express the optimal time T^* defined in (1.1) as:

$$
T^* = \min_{v > 0} T^*(v) \tag{2.14}
$$

We are now in a position to prove Theorem 1.1. We first show that the optimality condition (1.3) is satisfied if and only if v is a local minimum of function $T^*(v)$. We then show that $T^*(v)$ is unimodal, which implies that a local minimum of $T^*(v)$ is also a global minimum. Using this result, we finally establish the optimality conditions stated in Theorem 1.1.

LEMMA 2.2. *Flow value v is a local minimum for function $T^*(v)$ if and only if*

$$
-d_{ts} \le T^*(v) \le d_{st} \tag{2.15}
$$

Proof. See Appendix A. □

LEMMA 2.3. *Function $T^*(v)$ is unimodal.*

Proof. See Appendix A. □

With Lemma 2.2 and Lemma 2.3, we can now prove Theorem 1.1 given in Section 1.2.

Proof. [Proof of Theorem 1.1] From Lemma 2.2 and Lemma 2.3, we know that v^* is an optimal flow value for $T^*(v)$ if and only if $-d_{ts} \le T^*(v^*) \le d_{st}$.

Note that if x^* is a feasible flow with flow value v^*, then $g(v^*) = \sum_{e \in E} \tau_e \cdot x_e^*$ if and only if x^* is a min-cost flow. Hence, $T^*(v^*) = \left(F + \sum_{e \in E} \tau_e \cdot x_e^*\right)/v^*$ if and only if flow x^* is a min-cost flow with flow value v^*.

Joining the two equivalence relationships above, we conclude that a feasible flow x^* with flow value v^* is an optimal solution to the fractional programming formulation (1.1) if and only if 1) x^* is a min-cost flow, and 2) $-d_{ts} \le \left(F + \sum_{e \in E} \tau_e \cdot x_e^*\right)/v^* \le d_{st}$. Therefore a temporally-repeated flow of x^* is an optimal solution to the quickest flow problem if and only if the two optimality conditions above hold. □

3 Cost-Scaling Algorithm

In this section, we introduce a new cost-scaling algorithm for solving the quickest flow problem with integer arc costs. We prove the correctness of the algorithm by showing that when it terminates, the optimality conditions in Theorem 1.1 are met. For the time complexity, we show that the algorithm runs in $O(n^3 \log(nC))$ time, and can be further improved to run in $O(nm \log(n^2/m) \log(nC))$ time with the dynamic tree data structure introduced by Sleator and Tarjan [21, 22]. Here n, m, and C are the number of nodes, arcs, and the maximum arc cost, respectively.

Recall that there are two requirements in the optimality conditions: 1) Flow x must be a min-cost flow, and 2) the inequality $-d_{ts} \le \left(F + \sum_{e \in E} \tau_e x_e\right)/v \le d_{st}$ must be satisfied.

In order to find a min-cost flow, our algorithm follows the cost-scaling framework developed by Goldberg and Tarjan [10, 11]. For some $\epsilon > 0$, we allow the reduced cost c_e^π in the residual network $G(x)$ to be negative, but require that $c_e^\pi \ge -\epsilon$. A pair of flow x and node potentials π meeting this requirement is called ϵ-optimal. We start with a big ϵ and iteratively reduce ϵ by half and modify x and π to satisfy the ϵ-optimality. Such iterations continue until ϵ is so small that the sum of reduced costs through any cycle in $G(x)$ is greater than -1. Since we assume that all arc costs are integers, there must be no negative cycles in $G(x)$. Therefore x is a min-cost flow. We refer the reader to Goldberg and Tarjan [11] and Ahuja et al. [1], Sect. 10.3 for a more comprehensive discussion.

In order to have the ratio $r := \left(F + \sum_{e \in E} \tau_e x_e\right)/v$ between $-d_{ts}$ and d_{st} when the algorithm ends, we add an extra step to each scaling phase. It turns out that $\pi_s - \pi_t$ serves as a good approximation of both $-d_{ts}$ and d_{st}. Therefore, we push extra flow from s to t to guarantee that the upper bound of the gap between the ratio r and $\pi_s - \pi_t$ is reduced by half after each iteration. At the end of the scaling phases, such gap will be less than 1. Using the assumption that all arc costs are integers, we can obtain an optimal solution by solving at most one extra max-flow problem.

The complete algorithm is described from Algorithm 1 to 5. The main procedure is Algorithm 1. It has

a main loop with a sequence of ϵ-scaling phases and a final saturation subroutine. Each scaling phase consists of a *Refine* subroutine that modifies a 2ϵ-optimal flow into an ϵ-optimal flow, and a *Reduce-Gap* subroutine that closes the gap between the ratio r and $\pi_s - \pi_t$. In the subroutines, we call a node w active if its flow excess $e(w) > 0$. We call an arc in $G(x)$ admissible if its reduced cost $c_e^\pi < 0$. We define the admissible network of $G(x)$ as the sub-network consisting solely of admissible arcs.

Algorithm 1 : Cost Scaling

Set $\epsilon := C$, flow $x := 0$, node potentials $\pi := 0$
while $\epsilon \geq 1/(8n)$ **do**
  Refine(ϵ, x, π)
  Reduce-Gap(ϵ, x, π)
  $\epsilon := \epsilon/2$
end while
Saturate(x)
Return optimal flow x

Algorithm 2 : Refine(ϵ, x, π)

Set $x_e := 0$ for all $\{e \in E : c_e^\pi > 0\}$ and $x_e := u_e$ for all $\{e \in E : c_e^\pi < 0\}$
Compute flow excess $e(w)$ for all $w \in V$; Put all nodes in a list L
while there exists an active node **do**
  Choose the first active node w in L
  Push/Relabel(w)
end while

Throughout the cost-scaling algorithm, a key metric that we focus on is the gap between the ratio r and $\pi_s - \pi_t$. Define

$$\gamma(x, \pi) := \frac{F + \sum_{e \in E} \tau_e \cdot x_e}{v} - (\pi_s - \pi_t) \tag{3.16}$$

The main goal of the subroutine *Reduce-Gap* is to reduce the upper bound of the gap $\gamma(x, \pi)$ in each scaling phase, while maintaining a lower bound of $\gamma(x, \pi)$ in order to leave some buffer for future operations.

Subroutine *Reduce-Gap* consists of two levels of loops. The outer loop runs two main tasks before starting the inner loop: 1) Relabel the source node s if all admissible arcs outgoing from s are saturated, and 2) generate extra flow from node s. The inner loop then iteratively pushes the newly generated flow down to the sink. In order to avoid sending too much flow in one iteration, which might breach the lower bound of $\gamma(x, \pi)$, we do not relabel the source node s in the inner loop, and set the excess $e(s)$ to 0 when all admissible arcs outgoing from s are saturated.

Algorithm 3 : Reduce-Gap(ϵ, x, π)

Put all nodes except t in a list L
while $\left(F + \sum_{e \in E} \tau_e x_e\right)/v - (\pi_s - \pi_t) > 7n\epsilon$ **do**
  Relabel $\pi_s := \pi_s + \epsilon$ if there is no outgoing admissible arc from s in $G(x)$
  Set the excess of s to $e(s) := \left[\left(F + \sum_{e \in E} \tau_e x_e\right) - (\pi_s - \pi_t + 5n\epsilon) \cdot v\right]/6n\epsilon$
  while there exists an active node in L **do**
    Choose the first active node w in L
    if w is the source node s **then**
      while $e(s) > 0$ **and** $G(x)$ contains an admissible arc $e = (s, w')$ **do**
        Push $\delta := \min\{e(s), r_e\}$ units of flow from s to w'
      end while
      Set $e(s) := 0$
    else
      Push/Relabel(w)
    end if
  end while
end while

Algorithm 4 : Push/Relabel(w)

while $e(w) > 0$ **and** $G(x)$ contains a node w' such that arc $e = (w, w')$ is admissible **do**
  Push $\delta := \min\{e(w), r_e\}$ units of flow from w to w'
end while
if $e(w) > 0$ **then**
  Relabel $\pi_w := \pi_w + \epsilon$, and move w to the beginning of list L
end if

Algorithm 5 : Saturate(x)

Compute d_{st}, the cost of a shortest s-t path in $G(x)$
if $d_{st} < \left(F + \sum_{e \in E} \tau_e x_e\right)/v$ **then**
  Form a sub-network $G'(x)$ that only includes arcs that lie on some shortest s-t-path in $G(x)$
  Send maximum amount of flow from s to t in $G'(x)$
end if

As a summary, we prove the following lemmas before showing the correctness of the algorithm and analyzing its time complexity:

1. Lemma 3.1 shows that $\gamma(x,\pi)$ does not change too much after executing the subroutine *Refine*.
2. Lemma 3.2 shows that $\pi_s - \pi_t$ serves as a good approximation of $-d_{ts}$ and d_{st} during the execution of the subroutine *Reduce-Gap*.
3. Lemma 3.3 establishes the lower bound of $\gamma(x,\pi)$ in the subroutine *Reduce-Gap*.
4. Lemma 3.4 shows that the ratio r is non-increasing after each iteration of the outer loop in the subroutine *Reduce-Gap*.
5. Lemma 3.5 implies that node s is relabeled at least once in every two iterations of the outer loop in the subroutine *Reduce-Gap*.
6. Lemma 3.6 and Lemma 3.7 show that all nodes are relabeled at most $O(n)$ times in the subroutine *Reduce-Gap*.

In Lemma 3.1, we use $x^{2\epsilon}$ and $\pi^{2\epsilon}$ to denote the flow and node potentials when the 2ϵ-scaling phase ends, and x^{ϵ} and π^{ϵ} to denote the flow and node potentials when the subroutine *Refine* ends in the ϵ-scaling phase.

LEMMA 3.1. *If we have*

$$5n \cdot 2\epsilon \le \gamma(x^{2\epsilon}, \pi^{2\epsilon}) \le 7n \cdot 2\epsilon \tag{3.17}$$

when the 2ϵ*-scaling phase ends, we have*

$$5n\epsilon \le \gamma(x^{\epsilon}, \pi^{\epsilon}) \le 18n\epsilon \tag{3.18}$$

when the subroutine Refine *ends in the* ϵ*-scaling phase.*

Proof. See Appendix B. □

The following lemmas show how the range of $\gamma(x^{\epsilon}, \pi^{\epsilon})$ changes during the execution of subroutine *Reduce-Gap*. Because they all involve only x and π in the ϵ-scaling phase, we drop the superscript ϵ to simplify notations.

LEMMA 3.2. *Throughout the* ϵ*-scaling phase, we have:*

$$-d_{ts} - n\epsilon \le \pi_s - \pi_t \le d_{st} + n\epsilon \tag{3.19}$$

Proof. See Appendix B. □

In the following lemmas, we assume that x, v, and π are the flow, flow value, and node potentials when the inner loop of the subroutine *Reduce-Gap* starts, respectively, and x', v', and π' the flow, flow value, and node potentials when the the inner loop ends, respectively.

LEMMA 3.3. *When the inner loop of the subroutine* Reduce-Gap *ends, the gap* $\gamma(x',\pi') \ge 5n\epsilon$.

Proof. See Appendix B. □

Remark: A direct corollary of Lemma 3.3 is that when the subroutine *Reduce-Gap* ends, we have $5n\epsilon \le \gamma(x',\pi') \le 7n\epsilon$. Therefore, if the assumption that $5n \cdot 2\epsilon \le \gamma(x^{2\epsilon}, \pi^{2\epsilon}) \le 7n \cdot 2\epsilon$ in Lemma 3.1 holds when the 2ϵ-scaling phase ends, the inequality $5n \cdot \epsilon \le \gamma(x^{\epsilon}, \pi^{\epsilon}) \le 7n \cdot \epsilon$ will also hold when the ϵ-scaling phase ends.

LEMMA 3.4. *When the inner loop of the subroutine* Reduce-Gap *ends, we have*

$$\frac{F + \sum_{e\in E} \tau_e \cdot x'_e}{v'} \le \frac{F + \sum_{e\in E} \tau_e \cdot x_e}{v} \tag{3.20}$$

Proof. See Appendix B. □

LEMMA 3.5. *If the source node* s *is not relabeled in one iteration of the outer loop of the subroutine* Reduce-Gap*, either it will be relabeled in the next iteration, or we will have* $\gamma(x',\pi') \le 7n\epsilon$ *and the subroutine* Reduce-Gap *will terminate.*

Proof. See Appendix B. □

We are now in a position to prove the correctness of the algorithm and show that it runs in $O(n^3 \log(nC))$ time.

Correctness: Theorem 3.1 shows that the two optimality conditions in Theorem 1.1 are both satisfied when Algorithm 1 terminates. Therefore a temporally-repeated flow of the final static flow x is an optimal solution to the quickest flow problem.

THEOREM 3.1. *When Algorithm 1 terminates, flow* x *and node potentials* π *satisfy the optimality conditions: 1) Flow* x *is a min-cost flow with respect to its flow value* v*, and 2)* $-d_{ts} \le (F + \sum_{e\in E} \tau_e x_e)/v \le d_{st}$.

Proof. See Appendix B. □

Time complexity: There are $O(\log(nC))$ cost-scaling phases. Within each cost-scaling phase, we run the subroutine *Refine* once and *Reduce-Gap* once. The *Refine* subroutine is the same as the wave implementation of the cost-scaling algorithm for the min-cost flow problem. Therefore it runs in $O(n^3)$ time. If we can bound the time complexity of the *Reduce-Gap* subroutine by $O(n^3)$ as well, we can claim that the overall time complexity of the cost-scaling loop is bounded by $O(n^3 \log(nC))$.

To show that the subroutine *Reduce-Gap* indeed runs in $O(n^3)$ time, we first show that the source node s is relabeled at most $O(n)$ times. Based on this result, we then prove that other nodes are also relabeled at most $O(n)$ times. These two bounds lead to three key conclusions: 1) The *Reduce-Gap* subroutine performs at most $O(n^2)$ relabeling operations, 2) the number of saturating pushes is bounded by $O(nm)$, and 3) the number of non-saturating pushes is bounded by $O(n^3)$. Therefore the execution time of the subroutine *Reduce-Gap* is bounded by $O(n^3)$.

LEMMA 3.6. *The source node s is relabeled at most $11n$ times in the subroutine* Reduce-Gap.

Proof. First, consider the first cost-scaling phase. Because $\epsilon = C$, $\pi_s - \pi_t$ can increase from 0 to at most $n\epsilon$. Thus, node s can be relabeled at most n times in the first scaling phase.

For all the subsequent iterations, Lemma 3.1 shows that when the subroutine *Reduce-Gap* starts, we have $5n\epsilon \leq \gamma(x, \pi) \leq 18n\epsilon$. Because the subroutine *Reduce-Gap* ends when $\gamma(x, \pi) \leq 7n\epsilon$, $\gamma(x, \pi)$ decreases by at most $18n\epsilon - 7n\epsilon = 11n\epsilon$.

Lemma 3.4 shows that the ratio r does not increase, while each time s is relabeled, $\pi_s - \pi_t$ increases by ϵ. As a result, the gap $\gamma(x, \pi) = r - (\pi_s - \pi_t)$ decreases by at least ϵ each time s is relabeled. Hence, node s can be relabeled at most $11n$ times. □

LEMMA 3.7. *Any node $w \in V$ is relabeled at most $13n$ times in the subroutine* Reduce-Gap.

Proof. See Appendix B. □

THEOREM 3.2. *The time complexity of the cost-scaling Algorithm 1 is $O(n^3 \log(nC))$.*

Proof. From Lemma 3.6 and Lemma 3.7, we conclude that the total number of relabeling operations is bounded by $O(n^2)$. Further, because all nodes are relabeled at most $O(n)$ times, we can use the same analysis as in Goldberg and Tarjan [11] to obtain a $O(nm)$ bound for the number of saturating pushes.

For the number of non-saturating pushes, we note that if the subroutine *Reduce-Gap* does not relabel any node within n consecutive node examinations, there must be no active node. We will either start a new iteration of the outer loop of *Reduce-Gap*, or terminate the subroutine *Reduce-Gap* altogether. Lemma 3.5 implies that we must relabel node s at least once in two consecutive iterations of the outer loop. Thus, there is at least one node relabeling in $2n$ consecutive node examinations. Since the number of total relabeling operations is bounded by $O(n^2)$, the number of node examinations is bounded by $O(n^3)$. Because there is at most one non-saturating push per node examination, we obtain the bound $O(n^3)$ for the number of non-saturating pushes.

In conclusion, the time complexity of the cost-scaling loop is bounded by $O(n^3 \log(nC))$. The final subroutine *Saturate* might invoke a max-flow algorithm to saturate all shortest paths in the residual network. Since the time complexity of many well-known max-flow algorithms are bounded by $O(n^3 \log n)$, it is not a bottleneck of the algorithm. Thus, the overall time complexity of the cost-scaling algorithm for the quickest flow problem is $O(n^3 \log(nC))$. □

If we apply the dynamic tree data structure introduced by Sleator and Tarjan [21, 22], the time complexity of the cost-scaling Algorithm 1 can be further improved to $O(nm \log(n^2/m) \log(nC))$.

THEOREM 3.3. *The quickest flow problem can be solved in $O(nm \log(n^2/m) \log(nC))$ time.*

Proof. See Appendix B. □

4 Conclusions and Future Research

In this paper, we show that the quickest flow problem in dynamic networks can be formulated as a fractional programming problem or a linear programming problem. We then derive optimality conditions that lead to a new cost-scaling algorithm for the quickest flow problem. The new algorithm follows the cost-scaling framework designed to solve the min-cost flow problem. We add an extra step in each scaling phase to guarantee that the optimality conditions are met when the algorithm ends. We show that the algorithm runs in $O(n^3 \log(nC))$ time. We can further improve the time complexity to $O(nm \log(n^2/m) \log(nC))$ by applying the dynamic tree data structure.

The result is significant because it shows for the first time that the quickest flow problem can be solved in the same time bound as one of the fastest algorithms for the min-cost flow problem. It implies that the quickest flow problem might not be strictly harder than the min-cost flow problem if measured by time complexity. As a consequence, unless the quickest flow problem could be shown to be simpler than the min-cost flow problem, the dynamic tree implementation of our algorithm will remain one of the fastest algorithms for the quickest flow problem in terms of worst-case time complexity.

We observe that the quickest flow problem is essentially a parametric flow problem. This is evident from the fractional programming formulation (1.1). The reason that the cost-scaling algorithm can solve the quickest flow problem within the same time bound as the

min-cost flow problem might be attributed to this parametric nature of the problem. Gallo et al. [9] showed that the parametric max flow problem can be solved within the same time bound as the maximum flow problem using the preflow-push algorithm. In this paper, we show that the quickest flow problem, as a parametric min-cost flow problem, can also be solved within the same time bound as the min-cost flow problem. Similar to what Gallo et al. [9] observed, the preflow-push algorithm framework exhibits great elasticity for parametric extensions.

Our new formulations and algorithms for the quickest flow problem lead to several new research opportunities. An open question is whether one can modify the cost-scaling algorithm to obtain a strongly-polynomial algorithm. If this is possible, we expect the time bound of such an algorithm to be better than the strongly-polynomial algorithm introduced by Burkard et al. [4]. Secondly, our algorithm might be extended to solve more general parametric min-cost flow problems, such as determining the optimal flow value of any s-t min-cost flow that satisfies extra side constraints. Lastly, the cost-scaling approach introduced in this paper might inspire the development of new algorithms for a more general optimization problem in dynamic networks – the quickest transshipment problem. The quickest transshipment problem is similar to the quickest flow problem except it allows multiple source nodes and multiple sink nodes with their respective supplies and demands. Hoppe and Tardos [13] introduced the only polynomial-time algorithms for this problem. Similar to the approach used by Burkard et al. [4], their algorithms also search over time T to find an optimal solution. Our cost-scaling approach provides a different angle for looking at this problem, and new algorithms might be developed to reach better time bounds.

Acknowledgment

We thank Prof. James B. Orlin for the help to simplify the proofs for the optimality conditions.

References

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

[2] Luiz F Bittencourt and Edmundo RM Madeira. Towards the scheduling of multiple workflows on computational grids. *Journal of grid computing*, 8(3):419–441, 2010.

[3] Luiz F Bittencourt and Edmundo RM Madeira. Hcoc: a cost optimization algorithm for workflow scheduling in hybrid clouds. *Journal of Internet Services and Applications*, 2(3):207–227, 2011.

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

[5] YL Chen and YH Chin. The quickest path problem. *Computers & Operations Research*, 17(2):153–161, 1990.

[6] Lisa K Fleischer. Faster algorithms for the quickest transshipment problem. *SIAM journal on Optimization*, 12(1):18–35, 2001.

[7] Lester Randolph Ford and Delbert Ray Fulkerson. Constructing maximal dynamic flows from static flows. *Operations research*, 6(3):419–433, 1958.

[8] LR Ford and DR Fulkerson. *Flows in networks*, volume 1962. Princeton Princeton University Press, 1962.

[9] Giorgio Gallo, Michael D Grigoriadis, and Robert E Tarjan. A fast parametric maximum flow algorithm and applications. *SIAM Journal on Computing*, 18(1):30–55, 1989.

[10] Andrew Goldberg and Robert Tarjan. Solving minimum-cost flow problems by successive approximation. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 7–18. ACM, 1987.

[11] Andrew V Goldberg and Robert E Tarjan. Finding minimum-cost circulations by successive approximation. *Mathematics of Operations Research*, 15(3):430–466, 1990.

[12] Horst W Hamacher, Stephanie Heller, and Benjamin Rupp. Flow location (flowloc) problems: dynamic network flows and location models for evacuation planning. *Annals of Operations Research*, 207(1):161–180, 2013.

[13] Bruce Hoppe and Éva Tardos. The quickest transshipment problem. *Mathematics of Operations Research*, 25(1):36–62, 2000.

[14] Ekkehard Köhler, Rolf H Möhring, and Martin Skutella. Traffic networks and flows over time. In *Algorithmics of Large and Complex Networks*, pages 166–196. Springer, 2009.

[15] Kai Li and Shan-lin Yang. Non-identical parallel-machine scheduling research with minimizing total weighted completion times: Models, relaxations and algorithms. *Applied mathematical modelling*, 33(4):2145–2158, 2009.

[16] Yi-Kuei Lin. System reliability of a stochastic-flow network through two minimal paths under time threshold. *International journal of production economics*, 124(2):382–387, 2010.

[17] Nimrod Megiddo. Combinatorial optimization with rational objective functions. *Mathematics of Operations Research*, 4(4):414–424, 1979.

[18] Britta Peis, Martin Skutella, and Andreas Wiese. Packet routing: Complexity and algorithms. In *Approximation and Online Algorithms*, pages 217–228. Springer, 2010.

[19] Britta Peis and Andreas Wiese. Universal packet routing with arbitrary bandwidths and transit times. In *Integer Programming and Combinatoral Optimization*,

pages 362–375. Springer, 2011.
[20] Martin Skutella. An introduction to network flows over time. In *Research Trends in Combinatorial Optimization*, pages 451–482. Springer, 2009.
[21] Daniel D Sleator and Robert E Tarjan. A data structure for dynamic trees. In *Proceedings of the thirteenth annual ACM symposium on Theory of computing*, pages 114–122. ACM, 1981.
[22] Daniel D Sleator and Robert E Tarjan. A data structure for dynamic trees. *Journal of computer and system sciences*, 26(3):362–391, 1983.
[23] Hong Zheng, Yi-Chang Chiu, Pitu B Mirchandani, and Mark Hickman. Modeling of evacuation and background traffic for optimal zone-based vehicle evacuation strategy. *Transportation Research Record: Journal of the Transportation Research Board*, 2196(1):65–74, 2010.

A Proofs of Some of the Results in Section 2

Proof. [Proof of Lemma 2.2] By definition, flow value v is a local minimum if and only if for any arbitrarily small $\epsilon > 0$, we have:

$$T^*(v) = \frac{F + g(v)}{v} \leq \frac{F + g(v+\epsilon)}{v+\epsilon} = T^*(v+\epsilon) \tag{A.1}$$

and

$$T^*(v) = \frac{F + g(v)}{v} \leq \frac{F + g(v-\epsilon)}{v-\epsilon} = T^*(v-\epsilon) \tag{A.2}$$

Increasing a flow value v by a tiny amount ϵ in function $g(v)$ means to send an extra ϵ amount of flow from s to t while maintaining a min-cost flow. In order to achieve this goal, we need to send the extra flow through a shortest path in the residual network $G(x^*(v))$, where $x^*(v)$ is a min-cost flow with flow value v. The extra cost incurred by sending the extra ϵ amount of flow is $\epsilon \cdot d_{st}$. Therefore we have:

$$g(v+\epsilon) = g(v) + \epsilon \cdot d_{st} \tag{A.3}$$

Using (A.3), we can reduce (A.1) to:

$$\begin{aligned} &\frac{F + g(v)}{v} \leq \frac{F + g(v) + \epsilon \cdot d_{st}}{v+\epsilon} \\ \Rightarrow \quad &d_{st} \geq \frac{F + g(v)}{v} = T^*(v) \end{aligned} \tag{A.4}$$

Similarly, decreasing v by a tiny amount ϵ in $g(v)$ means to to pull ϵ amount of flow back from t to s. In order to keep the flow optimal, the extra flow needs to be pulled from a shortest t-s path in the residual network. Therefore the change of cost is $\epsilon \cdot d_{ts}$. Note that d_{ts} is negative. We have:

$$g(v-\epsilon) = g(v) - \epsilon \cdot (-d_{ts}) \tag{A.5}$$

Using (A.5), we can reduce (A.2) to:

$$\begin{aligned} &\frac{F + g(v)}{v} \leq \frac{F + g(v) - \epsilon \cdot (-d_{ts})}{v-\epsilon} \\ \Rightarrow \quad &-d_{ts} \leq \frac{F + g(v)}{v} = T^*(v) \end{aligned} \tag{A.6}$$

From (A.4) and (A.6), we conclude that v is a local minimum for $T^*(v)$ if and only if $-d_{ts} \leq T^*(v) \leq d_{st}$. □

Proof. [Proof of Lemma 2.3] Function $g(v)$ defines a linear programming problem with parameter v, and it is a minimization problem, therefore $g(v)$ is convex and piecewise linear with respect to v. As a result, we can express $g(v)$ as:

$$g(v) = \begin{cases} \lambda_0 \cdot v + \beta_0 & 0 = \alpha_0 \leq v < \alpha_1 \\ \lambda_1 \cdot v + \beta_1 & \alpha_1 \leq v < \alpha_2 \\ \quad \cdots & \\ \lambda_K \cdot v + \beta_K & \alpha_K \leq v < \infty \end{cases} \tag{A.7}$$

where K is the total number of break points, and α_i is the i'th break point. In (A.7), we have $\lambda_0 < \lambda_1 < \cdots < \lambda_K$ and $\beta_0 > \beta_1 > \cdots > \beta_K$.

Based on this expression, we have: $T^*(v) = (F + g(v))/v = \lambda_i + (F + \beta_i)/v$ for v between α_i and α_{i+1}.

Let k be the smallest index of the pieces such that $F + \beta_i < 0$. Mathematically, we define $k := \min_i\{i : F + \beta_i < 0\}$. Let k be $K+1$ if $F + \beta_i \geq 0$ for all i. Then for all $i < k$, we have $F + \beta_i \geq 0$. Consequently, $T^*(v) = \lambda_i + (F + \beta_i)/v$ is non-increasing on each piece i if $i < k$. On the other hand, for all $i \geq k$, we have $F + \beta_i < 0$. Therefore $T^*(v) = \lambda_i + (F + \beta_i)/v$ is monotone increasing on each piece i if $i \geq k$.

Because function $T^*(v)$ is continuous, we conclude that $T^*(v)$ is non-increasing when $v < \alpha_k$ and increasing when $v \geq \alpha_k$. Hence, $T^*(v)$ is unimodal. □

B Proofs of Some of the Results in Section 3

In order to prove Lemma 3.1, we first prove a technical lemma:

LEMMA B.1. *Let x be any ϵ-optimal flow with flow value v, and let x^* be a min-cost flow with the same flow value v. If the admissible network is acyclic, we have*

$$\sum_{e \in E} \tau_e x_e^* \leq \sum_{e \in E} \tau_e x_e \leq \sum_{e \in E} \tau_e x_e^* + n\epsilon \cdot v \tag{B.8}$$

Proof. The left inequality is by definition. For the right inequality, consider an ϵ-optimal flow x and node potentials π. Let A be the set of arcs in the original

network whose forward arc or backward arc in the residual network $G(x)$ is admissible. We alter the cost of each arc $e = (w, w') \in A$ in the following way: set $\tau'_e := \pi_w - \pi_{w'}, \forall e \in A$. For the arcs not in set A, we keep $\tau'_e = \tau_e$. Note that after this change, $c^{\pi}_{e'} \geq 0$ for every arc e' in $G(x)$. Therefore flow x is optimal in the network with the altered arc costs τ'_e. Mathematically, we have:

$$\sum_{e\in E} \tau'_e x_e \leq \sum_{e\in E} \tau'_e x^*_e \tag{B.9}$$

Due to the ϵ-optimality of x and π, we have $\pi_{w'} + \tau_e - \pi_w \geq -\epsilon$ and $\pi_w - \tau_e - \pi_{w'} \geq -\epsilon$ for all $e = (w, w') \in A$. Therefore we have

$$-\epsilon \leq \tau_e - \tau'_e \leq \epsilon \tag{B.10}$$

Note that x and x^* are both feasible flows with flow value v, so the flow on any arc cannot exceed v. That is,

$$0 \leq x_e \leq v, \;\; 0 \leq x^*_e \leq v, \;\; \forall e \in E \tag{B.11}$$

Combining all the inequalities above, we have

(B.12)

$$\begin{aligned}
\sum_{e\in E} \tau_e x_e &= \sum_{e\in A} (\tau'_e + (\tau_e - \tau'_e)) \cdot x_e + \sum_{e\notin A} \tau'_e \cdot x_e \\
&\leq \sum_{e\in E} \tau'_e \cdot x^*_e + \sum_{e\in A} (\tau_e - \tau'_e) \cdot x_e \qquad \text{by (B.9)} \\
&= \sum_{e\in A} (\tau_e + (\tau'_e - \tau_e)) \cdot x^*_e \\
&\qquad + \sum_{e\notin A} \tau_e \cdot x^*_e + \sum_{e\in A} (\tau_e - \tau'_e) \cdot x_e \\
&= \sum_{e\in E} \tau_e \cdot x^*_e + \sum_{e\in A} (\tau_e - \tau'_e) \cdot (x_e - x^*_e) \\
&\leq \sum_{e\in E} \tau_e \cdot x^*_e + \sum_{e\in A} \epsilon \cdot v \quad \text{by (B.10) and (B.11)}
\end{aligned}$$

To complete the proof, we use the assumption that the admissible network of the residual network $G(x)$ is acyclic. It implies that the number of arcs in set A cannot exceed $n - 1$. Combining this result with inequality (B.12), we obtain $\sum_{e\in E} \tau_e \cdot x_e \leq \sum_{e\in E} \tau_e x^*_e + n\epsilon \cdot v$, which is the right inequality of (B.8). □

Proof. [Proof of Lemma 3.1] We use $v^{2\epsilon}$ and v^{ϵ} to denote the flow value of $x^{2\epsilon}$ and x^{ϵ}, respectively. Because Subroutine *Refine* does not send extra flow from s to t, we have $v^{\epsilon} = v^{2\epsilon}$.

Let x^* be a min-cost flow with flow value v^{ϵ}. Because the Subroutine *Reduce-Gap* and *Refine* both use the push/relabel algorithm that yields admissible networks that are acyclic (see Corollary 5.6 in [11]), by Lemma B.1, we have:

$$\sum_{e\in E} \tau_e \cdot x^*_e \leq \sum_{e\in E} \tau_e \cdot x^{2\epsilon}_e \leq \sum_{e\in E} \tau_e \cdot x^*_e + n \cdot 2\epsilon \cdot v^{2\epsilon} \tag{B.13}$$

and

$$\sum_{e\in E} \tau_e \cdot x^*_e \leq \sum_{e\in E} \tau_e \cdot x^{\epsilon}_e \leq \sum_{e\in E} \tau_e \cdot x^*_e + n \cdot \epsilon \cdot v^{\epsilon} \tag{B.14}$$

Putting (B.13) and (B.14) together, we have:

(B.15)

$$\sum_{e\in E} \tau_e \cdot x^{\epsilon}_e - n\epsilon \cdot v^{\epsilon} \leq \sum_{e\in E} \tau_e \cdot x^{2\epsilon}_e \leq \sum_{e\in E} \tau_e \cdot x^{\epsilon}_e + 2n\epsilon \cdot v^{\epsilon}$$

Moreover, Goldberg and Tarjan [11] showed that Subroutine *Refine* could only increase each node's potential by at most $3n\epsilon$ (Lemma 5.8 in [11]). That is, $0 \leq \pi^{\epsilon}_s - \pi^{2\epsilon}_s \leq 3n\epsilon$ and $0 \leq \pi^{\epsilon}_t - \pi^{2\epsilon}_t \leq 3n\epsilon$. Therefore,

$$\pi^{2\epsilon}_s - \pi^{2\epsilon}_t - 3n\epsilon \leq \pi^{\epsilon}_s - \pi^{\epsilon}_t \leq \pi^{2\epsilon}_s - \pi^{2\epsilon}_t + 3n\epsilon \tag{B.16}$$

Using (3.17), (B.15) and (B.16), we have

(B.17)

$$\begin{aligned}
\pi^{\epsilon}_s - \pi^{\epsilon}_t &\geq \pi^{2\epsilon}_s - \pi^{2\epsilon}_t - 3n\epsilon \qquad \text{by (B.16)} \\
&\geq \frac{F + \sum_{e\in E} \tau_e \cdot x^{2\epsilon}_e}{v^{2\epsilon}} - 17n\epsilon \qquad \text{by (3.17)} \\
&\geq \frac{F + \sum_{e\in E} \tau_e \cdot x^{\epsilon}_e - n\epsilon \cdot v^{\epsilon}}{v^{\epsilon}} - 17n\epsilon \quad \text{by (B.15)} \\
&= \frac{F + \sum_{e\in E} \tau_e \cdot x^{\epsilon}_e}{v^{\epsilon}} - 18n\epsilon
\end{aligned}$$

and

(B.18)

$$\begin{aligned}
\pi^{\epsilon}_s - \pi^{\epsilon}_t &\leq \pi^{2\epsilon}_s - \pi^{2\epsilon}_t + 3n\epsilon \qquad \text{by (B.16)} \\
&\leq \frac{F + \sum_{e\in E} \tau_e \cdot x^{2\epsilon}_e}{v'^{2\epsilon}} - 7n\epsilon \qquad \text{by (3.17)} \\
&\leq \frac{F + \sum_{e\in E} \tau_e \cdot x^{\epsilon}_e + 2n\epsilon \cdot v^{\epsilon}}{v^{\epsilon}} - 7n\epsilon \quad \text{by (B.15)} \\
&= \frac{F + \sum_{e\in E} \tau_e \cdot x^{\epsilon}_e}{v^{\epsilon}} - 5n\epsilon
\end{aligned}$$

Combining (B.17), (B.18), and the definition of $\gamma(x, \pi)$ (3.16), we obtain (3.18). □

Proof. [Proof of Lemma 3.2] The pair of flow x and node potentials π is ϵ-optimal throughout the ϵ-scaling phase. Consider any shortest simple path p_{st} from s to t in the

residual network $G(x)$. We have

$$\begin{aligned}\sum_{e\in p_{st}} c_e^\pi &= \sum_{e=(w,w')\in p_{st}} (\pi_{w'} + \hat{\tau}_e - \pi_w) \\ &= \pi_t - \pi_s + \sum_{e\in p_{st}} \hat{\tau}_e \\ &= \pi_t - \pi_s + d_{st}\end{aligned}$$

Since $c_e^\pi \geq -\epsilon$, we have $\pi_t - \pi_s + d_{st} = \sum_{e\in P_{st}} c_e^\pi \geq -n\epsilon$. This is equivalent to the right inequality of (3.19).

Applying a similar argument on a shortest simple path p_{ts} from t to s in $G(x)$, we can obtain the left inequality of (3.19). □

We next consider the change of node potentials and the ratio $(F + \sum_{e\in E} \tau_e \cdot x_e)/v$ during the execution of the subroutine *Reduce-Gap*.

Let x, v, and π be the flow, flow value, and node potentials when the inner loop of the subroutine *Reduce-Gap* starts, and let x', v', and π' be the flow, flow value, and node potentials when the the inner loop ends. Let $\Delta v = v' - v$ be the change of the flow value. Note that because there is excess on node s when the inner loop starts, but no excess when it ends, we have $v' \geq v$.

In order to prove Lemma 3.3, Lemma 3.4, and Lemma 3.5, we first prove the following lemma:

LEMMA B.2. *When the inner loop of Subroutine* Reduce-Gap *ends, we have*

(B.19)
$$\frac{F + \sum_{e\in E} \tau_e \cdot x'_e}{v'} \geq \frac{F + \sum_{e\in E} \tau_e \cdot x_e + (\pi_s - \pi_t - n\epsilon)\cdot \Delta v}{v + \Delta v}$$

and

(B.20)
$$\frac{F + \sum_{e\in E} \tau_e \cdot x'_e}{v'} \leq \frac{F + \sum_{e\in E} \tau_e \cdot x_e + (\pi_s - \pi_t + n\epsilon)\cdot \Delta v}{v + \Delta v}$$

Proof. If we decompose the flow $x' - x$ in the residual network $G(x)$, we have a set P of feasible s-t paths in $G(x)$. Let $\tau(p)$ and $x(p)$ be the cost and flow on each path $p \in P$. Recall that $d_{st}(x)$ is the cost of a shortest simple s-t path in the residual network $G(x)$. By the right inequality of (3.19), we have

$$\tau(p) \geq d_{st}(x) \geq \pi_s - \pi_t - n\epsilon \tag{B.21}$$

Similarly, $d_{ts}(x')$ is the cost of a shortest simple t-s path in $G(x')$. Note that the reverse path of p is a feasible t-s path in the residual network $G(x')$, therefore we have $d_{ts}(x') \leq -\tau(p)$. Hence, by the left inequality of (3.19), we have

$$\tau(p) \leq -d_{ts}(x') \leq \pi'_s - \pi'_t + n\epsilon \tag{B.22}$$

Note that we do not relabel the source node s in the inner loop of Subroutine *Reduce-Gap*, and the sink node t is never relabeled. Therefore $\pi_s = \pi'_s$ and $\pi_t = \pi'_t$. Based on (B.21) and (B.22), we have

$$\begin{aligned}&\frac{F + \sum_{e\in E} \tau_e \cdot x'_e}{v'} \\ &\quad= \frac{F + \sum_{e\in E} \tau_e \cdot x_e + \sum_{p\in P} \tau(p)\cdot x(p)}{v + \Delta v} \\ &\quad\geq \frac{F + \sum_{e\in E} \tau_e \cdot x_e + (\pi_s - \pi_t - n\epsilon)\cdot \Delta v}{v + \Delta v}\end{aligned}$$

and

$$\begin{aligned}&\frac{F + \sum_{e\in E} \tau_e \cdot x'_e}{v'} \\ &\quad= \frac{F + \sum_{e\in E} \tau_e \cdot x_e + \sum_{p\in P} \tau(p)\cdot x(p)}{v + \Delta v} \\ &\quad\geq \frac{F + \sum_{e\in E} \tau_e \cdot x_e + (\pi'_s - \pi'_t + n\epsilon)\cdot \Delta v}{v + \Delta v}\end{aligned}$$

□

Proof. [Proof of Lemma 3.3] At the beginning of an iteration of the outer loop, we generate an extra amount of excess $\left[(F + \sum_{e\in E} \tau_e x_e) - (\pi_s - \pi_t + 5n\epsilon)\cdot v\right]/6n\epsilon$ at the source node s. No other excess is generated before the next iteration of the outer loop. Therefore,

$$\begin{aligned}&\Delta v \leq \frac{(F + \sum_{e\in E} \tau_e \cdot x_e) - (\pi_s - \pi_t + 5n\epsilon)\cdot v}{6n\epsilon} \\ \Rightarrow\quad &(\pi_s - \pi_t + 5n\epsilon)\cdot v + 6n\epsilon\cdot \Delta v \leq F + \sum_{e\in E} \tau_e \cdot x_e \\ \Rightarrow\quad &(\pi_s - \pi_t + 5n\epsilon)\cdot (v + \Delta v) \\ &\qquad\leq F + \sum_{e\in E} \tau_e \cdot x_e + (\pi_s - \pi_t - n\epsilon)\cdot \Delta v \\ \Rightarrow\quad &\pi_s - \pi_t + 5n\epsilon \\ &\qquad\leq \frac{F + \sum_{e\in E} \tau_e \cdot x_e + (\pi_s - \pi_t - n\epsilon)\cdot \Delta v}{v + \Delta v} \\ \Rightarrow\quad &\pi_s - \pi_t \leq \frac{F + \sum_{e\in E} \tau_e \cdot x'_e}{v'} - 5n\epsilon \quad \text{by (B.19)}\end{aligned}$$

Because we do not relabel node s in the inner loop of Subroutine *Reduce-Gap*, and the sink node t is never relabeled in Subroutine *Reduce-Gap*, we have $\pi_s = \pi'_s$, and $\pi_t = \pi'_t$. Therefore, we have $\gamma(x', \pi') \geq 5n\epsilon$. □

Proof. [Proof of Lemma 3.4] By Lemma 3.3, we have $\gamma(x,\pi) \geq 5n\epsilon$, or equivalently,

$$\pi_s - \pi_t \leq \frac{F + \sum_{e\in E} \tau_e \cdot x_e}{v} - 5n\epsilon \tag{B.23}$$

Using (B.20) and (B.23), we have

$$\begin{aligned}
&\frac{F + \sum_{e\in E} \tau_e \cdot x'_e}{v'} \\
&\quad \leq \frac{F + \sum_{e\in E} \tau_e \cdot x_e + (\pi_s - \pi_t + n\epsilon)\cdot \Delta v}{v + \Delta v} \\
&\quad \leq \frac{\left(F + \sum_{e\in E} \tau_e \cdot x_e\right) + \left(F + \sum_{e\in E} \tau_e \cdot x_e\right)\cdot \frac{\Delta v}{v}}{v + \Delta v} \qquad \text{by (B.23)} \\
&\quad = \frac{\left(F + \sum_{e\in E} \tau_e \cdot x_e\right)\cdot \left(1 + \frac{\Delta v}{v}\right)}{v + \Delta v} \\
&\quad = \frac{F + \sum_{e\in E} \tau_e \cdot x_e}{v}
\end{aligned}$$

□

Proof. [Proof of Lemma 3.5] According to Subroutine *Reduce-Gap*, if there is no outgoing admissible arc from s at the end of an iteration of the outer loop, s will be relabeled in the next iteration. Therefore, we only need to show that if there exists an outgoing admissible arc from s at the end of an iteration of the outer loop, we must have $\gamma(x',\pi') \leq 7n\epsilon$.

We first show that if there exists an outgoing admissible arc from s when an iteration ends, all the extra flow generated at node s at the beginning of the iteration has been sent to the sink. Mathematically, we will show that $\Delta v = [(F + \sum_{e\in E} \tau_e \cdot x_e) - (\pi_s - \pi_t + 5n\epsilon)\cdot v]/6n\epsilon$.

When s is the first active node in the list L, we keep pushing excess $e(s)$ to other nodes until either $e(s) = 0$ or the the residual network $G(x)$ does not contain any admissible arc outgoing from s. Note that any arc $e = (s,w)$ in $G(x)$ cannot become admissible again after it is saturated unless s is relabeled. Since there is still an admissible arc outgoing from s when the iteration ends, we must have $e(s) = 0$ every time we are done pushing excess from node s to other nodes. Therefore, setting $e(s) := 0$ after the pushes does not reduce the overall flow excess. Moreover, we do not lose any flow excess when performing push/relabel operations on other nodes. Thus, we can claim that all extra flow generated at the beginning of a iteration has been sent to the sink when the iteration ends.

With $\Delta v = [(F + \sum_{e\in E} \tau_e \cdot x_e) - (\pi_s - \pi_t + 5n\epsilon)\cdot v]/6n\epsilon$, we have:

$$\begin{aligned}
& (\pi_s - \pi_t + 5n\epsilon)\cdot v + 6n\epsilon \cdot \Delta v = F + \sum_{e\in E} \tau_e \cdot x_e \\
\Rightarrow \quad & (\pi_s - \pi_t + 7n\epsilon)\cdot (v + \Delta v) \\
& \qquad \geq F + \sum_{e\in E} \tau_e \cdot x_e + (\pi_s - \pi_t + n\epsilon)\cdot \Delta v \\
\Rightarrow \quad & \pi_s - \pi_t + 7n\epsilon \\
& \qquad \geq \frac{F + \sum_{e\in E} \tau_e \cdot x_e + (\pi_s - \pi_t + n\epsilon)\cdot \Delta v}{v + \Delta v} \\
\Rightarrow \quad & \pi_s - \pi_t \geq \frac{F + \sum_{e\in E} \tau_e \cdot x'_e}{v'} - 7n\epsilon \qquad \text{by (B.20)}
\end{aligned}$$

Because we do not relabel node s in the inner loop of Subroutine *Reduce-Gap*, and the sink node t is never relabeled in *Reduce-Gap*, we have $\pi_s = \pi'_s$, and $\pi_t = \pi'_t$. Therefore we have $\gamma(x',\pi') \leq 7n\epsilon$. □

Proof. [Proof of Theorem 3.1] The first condition is satisfied because when the algorithm terminates, any cycle C in the residual network has cost $\sum_{e\in C} c^\pi_e \geq -n\epsilon > -1/8$. Since all costs are integers, there does not exist any negative cycle in the residual network. By Theorem 2.1, the flow is a min-cost flow.

Now we show that the optimality condition (1.3) is satisfied when the final subroutine *Saturate* ends. Assume that the flow and node potentials are x and π respectively when the last scaling phase ends, and x^* and π^* are the flow and node potentials when the final subroutine *Saturate* ends. The termination condition of the subroutine *Reduce-Gap* and Lemma 3.3 indicate that when the last scaling phase ends, we have

$$\pi_s - \pi_t + 5n\epsilon \leq \frac{F + \sum_{e\in E} \tau_e x_e}{v} \leq \pi_s - \pi_t + 7n\epsilon \tag{B.24}$$

Using the left inequality of (3.19) in Lemma 3.2, we have

$$-d_{ts} - n\epsilon \leq \pi_s - \pi_t \leq \frac{F + \sum_{e\in E} \tau_e x_e}{v} - 5n\epsilon$$

Or equivalently,

$$-d_{ts} \leq \frac{F + \sum_{e\in E} \tau_e x_e}{v} - 4n\epsilon < \frac{F + \sum_{e\in E} \tau_e x_e}{v} \tag{B.25}$$

Therefore, the left inequality of the optimality condition (1.3) is satisfied. For the right inequality of (1.3), we use the right inequality of (3.19) in Lemma 3.2:

$$\frac{F + \sum_{e\in E} \tau_e x_e}{v} - 7n\epsilon \leq \pi_s - \pi_t \leq d_{st} + n\epsilon$$

Or equivalently,

$$\frac{F + \sum_{e \in E} \tau_e x_e}{v} \leq d_{st} + 8n\epsilon$$

When the scaling loop ends, we have $8n\epsilon < 1$. Therefore,

$$\frac{F + \sum_{e \in E} \tau_e x_e}{v} < d_{st} + 1 \tag{B.26}$$

If $\left(F + \sum_{e \in E} \tau_e x_e\right)/v \leq d_{st}$, the optimality condition (1.3) is met, and we do not need to execute the step of sending extra flow in the final subroutine *Saturate*.

On the other hand, if $(F + \sum_{e \in E} \tau_e x_e)/v > d_{st}$, the subroutine *Saturate* invokes a maximum flow algorithm to saturate all the paths with cost d_{st} in the residual network $G(x)$. Note that all the extra flow we send are through the shortest s-t paths whose costs are all $d_{st}(x)$ in the residual network $G(x)$. The reverse of these paths exist in the residual network $G(x^*)$. Therefore we have $-d_{ts}(x^*) = d_{st}(x)$, because otherwise we could find a negative cycle in the residual network $G(x)$, which contradicts the fact that x is a min-cost flow.

Assume the extra amount of flow sent from s to t is Δv, we have

$$\begin{aligned}\frac{F + \sum_{e \in E} \tau_e x_e^*}{v^*} &= \frac{F + \sum_{e \in E} \tau_e x_e + d_{st}(x) \cdot \Delta v}{v + \Delta v} \\ &\geq d_{st}(x) = -d_{ts}(x^*)\end{aligned}$$

On the other hand, d_{st} strictly increases when all the shortest paths are saturated. Since all costs are integer, we have $d_{st}(x^*) \geq d_{st}(x) + 1$. From (B.26), we have:

$$\begin{aligned}\frac{F + \sum_{e \in E} \tau_e x_e^*}{v^*} &= \frac{F + \sum_{e \in E} \tau_e x_e + d_{st}(x) \cdot \Delta v}{v + \Delta v} \\ &< d_{st}(x) + 1 \leq d_{st}(x^*)\end{aligned}$$

Now both the left and right inequalities of the optimality condition (1.3) are satisfied, and there is no negative cycle in the residual network. Therefore any temporally-repeated flow of the final solution x^* is an optimal solution to the quickest flow problem. □

Proof. [Proof of Lemma 3.7] The arguments we use here are similar to the ones used in Lemma 5.7 and 5.8 by Goldberg and Tarjan [11], where they showed that any node potential π_w could increase by at most $3n\epsilon$ in the Subroutine *Refine*.

Let x and π be the flow and node potentials when Subroutine *Reduce-Gap* starts in the ϵ-scaling phase, and x' and π' be the flow and node potentials at any step within Subroutine *Reduce-Gap*.

Consider any node w that is to be relabeled. According to Subroutine *Reduce-Gap*, w must have a positive flow access $e(w)$, and all the excess must have came from the source node s. If we perform a flow decomposition on flow $x' - x$ in the residual network $G(x)$, we can obtain a set of paths from s to t and from s to all the nodes with positive access, including w. This implies that there exists a path p_{sw} from s to w in the residual network $G(x)$ and its reverse path p'_{ws} from w to s in the residual network $G(x')$. Since both (x, π) and (x', π') are ϵ-optimal, we have:

$$\begin{aligned}-n\epsilon &\leq \sum_{e \in p_{sw}} c_e^{\pi} \\ &= \sum_{e=(w',w'') \in p_{sw}} (\pi_{w''} + \hat{\tau}_e - \pi_{w'}) \\ &= \pi_w - \pi_s + \sum_{e \in p_{sw}} \hat{\tau}_e\end{aligned} \tag{B.27}$$

and

$$\begin{aligned}-n\epsilon &\leq \sum_{e \in p'_{ws}} c_e^{\pi'} \\ &= \sum_{e=(w',w'') \in p'_{ws}} (\pi'_{w''} + \hat{\tau}_e - \pi'_{w'}) \\ &= \pi'_s - \pi'_w + \sum_{e \in p'_{ws}} \hat{\tau}_e\end{aligned} \tag{B.28}$$

where $\hat{\tau}_e$ is the cost of arc e in the residual networks.

Because path p'_{ws} is the reverse path of p_{sw}, we have $\sum_{e \in p'_{ws}} \hat{\tau}_e = -\sum_{e \in p_{sw}} \hat{\tau}_e$. Summing (B.27) and (B.28), we obtain

$$\pi'_w - \pi_w \leq \pi'_s - \pi_s + 2n\epsilon \leq 13n\epsilon \tag{B.29}$$

The last inequality holds because by Lemma 3.6, the source node s is relabeled at most $11n$ times, and each time it is relabeled, π_s increases by ϵ.

Since each relabeling increases the node potential of w by ϵ, we conclude that any node $w \in V$ is relabeled at most $13n$ times. □

Proof. [Proof of Theorem 3.3] We can apply the same analysis by Goldberg and Tarjan [11] here. Note that in their analysis, all computational costs are either attributed or amortized to the number of relabeling operations at each node in each scaling phase. Since we have shown that in our cost-scaling algorithm the number of relabeling operations on each node in each scaling phase is also bounded by $O(n)$, we can apply the same analysis and show that the computational time can be improved to $O(nm \log(n^2/m) \log(nC))$ by using the dynamic tree data structure. Since the analysis is the same as in [11], it is omitted here. □