A FAST MAX FLOW ALGORITHM

James B. Orlin
MIT, jorlin@mit.edu;
Xiaoyue Gong
MIT, xygong@mit.edu

ABSTRACT. In 2013, Orlin proved that the max flow problem could be solved in $O(nm)$ time. His algorithm ran in $O(nm + m^{1.94})$ time, which was the fastest for graphs with fewer than $n^{1.06}$ arcs. If the graph was not sufficiently sparse, the fastest running time was an algorithm due to King, Rao, and Tarjan. We describe a new variant of the excess scaling algorithm for the max flow problem whose running time strictly dominates the running time of the algorithm by King et al. Moreover, for graphs in which $m = O(n \log n)$, the running time of our algorithm dominates that of King et al. by a factor of $O(\log \log n)$.

1. INTRODUCTION

Network flow problems form an important class of optimization problems within operations research and computer science. Within that class, the max flow problem, has been widely investigated since the seminal research of Ford and Fulkerson [7] in the 1950s. A discussion of algorithms and applications for the max flow problem can be found in [1]. We consider the max flow problem on a directed graph with n nodes, m arcs, and integer valued arc capacities (possibly infinite), in which the largest finite capacity is bounded by U. The fastest strongly polynomial time algorithms are due to Orlin [16] and King et al. [14]. The running time for Orlin's algorithm is $O(nm + m^{31/16} \log^2 n)$, which is $O(nm)$ when $m < n^{1.06}$. On networks with $O(n)$ arcs, the algorithm can be further improved to run $O(\frac{n^2}{\log n})$ time. The running time for the algorithm by King et al. [14] is $O(nm \log_\beta n)$ where $\beta = 2 + \lfloor \frac{m}{n \log n} \rfloor$. The running time is $O(nm)$ whenever $m = \Omega(n^{1+\epsilon})$ for any fixed $\epsilon > 0$. Their algorithm is based upon the randomized algorithm of Cheriyan and Hagerup [4], which runs in $O(nm + n^2 \log^2 n)$ expected time.

Orlin's algorithm is based on using a fast weakly polynomial time algorithm. The fastest weakly polynomial time algorithms are those of Goldberg and Rao [9] with running time $O(\min\{m^{1/2}, n^{2/3}\} m \log \frac{n^2}{m} \log U)$, and Lee and Sidford [12] with running time $\tilde{O}(mn^{1/2} \log^2 U)$.

Here we present a new fast max flow algorithm that runs in $O(kn^2 + nm \log_k n)$ time, where the parameter k can be selected by the user. Assuming that $n \geq 4$ and $m \geq n$ one can optimize the asymptotic running time by choosing $k = \lceil \frac{\log n}{\log \log n} + \frac{m}{n} \rceil$. This leads to a running time of

$$O \left(\frac{nm \log n}{\log \log n + \log \frac{m}{n}} \right).$$

For all values of n and m, the running time of our algorithm dominates the running time of King et al. [14]. When $m \leq n \log n$, our algorithm is faster than the algorithm of [14] by a factor of $\log \log n$.

Our algorithm is based on the stack-scaling algorithm of Ahuja et al. [3], while also drawing upon the idea of "special pushes" in the paper by [15]. The analysis of special pushes in this paper is similar to that in the paper by Orlin [16].

The contributions of this paper may be described as follows.

1. We present a simple variant of the stack-scaling algorithm of [3] in which there are no stacks. We refer to the revised algorithm as the *Large-Medium Excess-Scaling (LMES) Algorithm.*

2. We give a new (and simpler) proof of Orlin's Contraction Lemma [16], which was used to develop an $O(nm)$ max flow algorithm.

3. In the modified version of the LMES algorithm (called the Enhanced LMES Algorithm), we permit node excesses that are slightly negative. When the negative excess of a node v reaches a threshold value, then node v is added to the "flow-return forest," which is a data structure designed for the Enhanced LMES. Within $O(\log_k n)$ additional scaling phases, flow is sent to node v, after which $e(v) \geq 0$.

4. Our algorithm achieves its improved running time without relying on the dynamic tree data structure of [18]. It is an open question as to whether a further speed-up is possible with the use of dynamic trees.

It is possible that the flow-return forest may be of use in other algorithms for the max flow problem or closely related problems.

2. Preliminaries

Let $G = (N, A)$ be a directed graph with node set N and arc set A. Let $n = |N|$, and $m = |A|$. Each arc $(i, j) \in A$ is associated with a non-negative integer *capacity* u_{ij}. The assumption of finiteness of capacities is without loss of generality. For networks with no infinite capacity path from source to sink, one can replace any infinite capacity by nU, where U is the largest finite capacity of an arc.

There is a unique *source* node $s \in N$ and a unique *sink* node $t \in N$. For convenience of exposition, we will assume that there is at most one arc from i to j and that for each arc $(i, j) \in A$, (j, i) is also in A, possibly with $u_{ji} = 0$. We refer to $u_{ij} + u_{ji}$ as the *bi-capacity* of arc (i, j). By symmetry, it is also the bi-capacity of arc (j, i).

For each node $i \in A$, we let $A^+(i)$ denote the set of arcs directed out of node i. We let $A^-(i)$ denote the set of arcs directed into node i. We let $A(i) = A^+(i) \cup A^-(i)$. We assume that there is a preprocessing step during which the arcs of $A^+(i)$ and $A^-(i)$ are ordered in non-increasing order of their bi-capacity. The preprocessing step takes $O(m \log n)$ steps and is not a bottleneck operation of our max flow algorithm. This assumption is of use later in this paper when we discuss the Enhanced LMES Algorithm.

Suppose that α, β and γ are all integers. When we write $\alpha \equiv \beta \pmod{\gamma}$, we mean that there is some integer τ such that $\alpha = \beta + \tau\gamma$. When we write $\alpha = \mathrm{Mod}(\beta, \gamma)$, we mean that α is the non-negative remainder obtained by dividing β by γ.

All logarithms in the paper are base two unless an explicit base is given.

A *feasible flow* is a function $x : A \to \mathbf{R}$ that satisfies the flow bound constraint $0 \leq x_{ij} \leq u_{ij}$, and the mass balance constraints

$$(2.1) \qquad \sum_{(j,i)\in A^-(i)} x_{ji} - \sum_{(i,j)\in A^+(i)} x_{ij} = 0, \forall i \in N - \{s, t\}.$$

The value $|x|$ of a flow x is the net flow into the sink; that is, $|x| = \sum_{(i,t)\in \bar{A}(t)} x_{it}$. The *maximum flow problem* is to determine a feasible flow with maximum flow value.

Given a flow x, the *residual capacity* $r_{ij}(x)$ of an arc $(i, j) \in A$ is $r_{ij}(x) = u_{ij} + x_{ji} - x_{ij}$. It can be interpreted as the maximum additional flow that can be sent from node i to node j using the arcs (i, j) and (j, i). Under the assumption that at most one of x_{ij} and x_{ji} is positive, $r_{ij}(x) + r_{ji}(x) = u_{ij} + u_{ji}$. Often, we will express the residual capacity of (i, j) more briefly as r_{ij}.

We refer to the network consisting of the arcs with positive residual capacities as the *residual network*, which we denote as $G(x)$.

Our algorithm uses preflows, which were introduced by Karzanov [13]. In a preflow x, for each node $i \in N \setminus \{s, t\}$, the *excess* of node i is defined to be

$$e_x(i) = \sum_{(j,i)\in A^-(i)} x_{ji} - \sum_{(i,j)\in A^+(i)} x_{ij}.$$

In a preflow x, the conservation of flow constraint is replaced by the following constraint on excesses.

$$(2.2) \qquad e_x(i) \geq 0, \forall i \in N - \{s\}.$$

Often, we will write the excess of node i more briefly as $e(i)$.

For a preflow x, the value $|x|$ is the net flow into the sink node t. A maximum s-t preflow is a preflow x that maximizes the amount of flow arriving at node t. Any preflow can be transformed into a flow by utilizing flow decomposition to return the flow from nodes with excess to the source node s. (See, for example, [1].) The transformation of a preflow into a flow does not alter the amount of flow arriving at the sink. This establishes the following well known lemma.

Lemma 2.3. *The maximum value of an s-t preflow is equal to the maximum value of an s-t flow.*

We now make an additional assumption. We assume that for any node $j \neq s$ or t, there are arcs (j, s) and (t, j) in A with $u_{js} = u_{tj} = U$. Adding these arcs is without loss of generality. If x^* is an optimal flow, then x^* can be efficiently transformed into another optimum flow x' such that for all $j \in N$, $x'_{js} = x'_{tj} = 0$. To transform x^* into x', first express x^* using flow decomposition (See [1].) Then eliminate all flows around cycles.

The advantage of introducing arcs directed into node s is that any node j with $d(j) = n + 1$ can send all of its excess to node s via the admissible arc (j, s). Accordingly, $d(j) \leq n + 1$ throughout the algorithm. The advantage of introducing arcs directed is that these arcs needed in the initialization of the Enhanced LMES Algorithm, which is described in Subsection 11.3.

Our algorithm is based on the stack-scaling algorithm of [3], which in turn is based in the push/relabel algorithm of Goldberg and Tarjan [10]. We review the push/relabel algorithm next.

3. Review of the generic Push-relabel algorithm

In this section, we review the generic push-relabel algorithm of Goldberg and Tarjan [10]. Push-relabel algorithms begin with an initial preflow x in which each arc out of node s is saturated. That is,

$$x_{si} = \begin{cases} u_{si} & \text{for each arc } (s, i) \in A^+(s) \\ 0 & \text{otherwise.} \end{cases}$$

For each node $i \in N$, the algorithm maintains a *distance label* $d(i)$, where $d(i)$ is a non-negative integer. The distance labels are said to be *valid* if they satisfy the following invariant.

Invariant 1. *(Validity)* $d(t) = 0$; for all arcs $(i, j) \in A$, if $r_{ij} > 0$, then $d(i) \leq d(j) + 1$.

The generic push-relabel algorithm begins with the following distance labels: $d(s) = n$, and $d(i) = 0$ for $i \neq s$. Throughout the rest of the generic algorithm, $d(t) = 0$, and $d(s) = n$.

The *length of a directed path* in $G(x)$ is the number of arcs on the path. The distance labels $d(\,)$ provide lower bounds on the minimum length of a path to node t, as stated in the following lemma proved in [10].

Lemma 3.1. *Suppose that the distance labels are valid with respect to a preflow x. Then for each $j \in N$, $d(j)$ is a lower bound on the length of the shortest path in $G(x)$ from j to t.*

An arc (i, j) is *admissible* if $d(i) = d(j) + 1$, and *inadmissible* otherwise. The generic push-relabel algorithm maintains a preflow, and all pushes are on admissible arcs.

We now describe the push-relabel algorithm in more detail.

If $e(i) > 0$, we refer to node i as *active*. The presence of active nodes in a push-relabel algorithm indicates that the solution is not yet a feasible flow. The algorithm moves excess flow from active nodes through admissible arcs towards the sink. Excess flow at node j that cannot be moved to the sink is returned to the source s when $d(j) = n + 1$. The algorithm terminates

with a maximum flow when the network contains no active node, i.e., when the preflow is also a flow. The generic push-relabel algorithm consists of repeating the following two steps until no node is active.

$Push(i, j)$
  Applicability: node i is active and arc (i, j) is admissible.
  Action: send $\min\{e(i), r_{ij}\}$ units of flow in arc (i, j).

$Relabel(i)$
  Applicability: node i is active and no arc of $A^+(i)$ is admissible.
  Action: replace $d(i)$ by $d(i) + 1$.
    (An alternative is to replace $d(i)$ by $1 + \min\{d(j) : \text{there is an admissible arc } (i, j)\}$.)

In the push operation, a push of r_{ij} units of flow is called *saturating*. Otherwise, it is called *non-saturating*. A push of $e(i)$ units of flow is called *emptying*.

The following results are either proved in [10] or are implicit in that paper.

Lemma 3.2. *The generic push-relabel algorithm has the following properties:*

  (1) At each iteration, $0 \le d(i) \le n + 1$ for each node $i \in N$.
  (2) A distance label never decreases, and it increases at most $n+1$ times. The total number of relabel operations is less than n^2.
  (3) The total number of saturating pushes is less than nm.
  (4) The total number of pushes is $O(n^2 m)$.
  (5) The algorithm terminates with a maximum flow.
  (6) One can implement the push-relabel algorithm to find a maximum flow in $O(n^2 m)$ time.

The running time of many implementations of the push-relabel algorithm are the same as the total number of pushes. To achieve these running times, one needs an efficient method of selecting admissible arcs. Goldberg and Tarjan [10] provided an efficient method that is sometimes referred to as the "Current Arc" method. In seeking an admissible arc directed out of node i, the algorithm scans the arc list $A^+(i)$ in sequential order starting with its first arc. The scan terminates when an admissible arc (i, j) is found. Then the algorithm sets CurrentArc(i) to (i, j), and (i, j) becomes the starting point for the next scan of $A^+(i)$. If the end of $A^+(i)$ is reached without identifying an admissible arc, then there is no admissible arc in $A^+(i)$. The push-relabel algorithm then increases $d(i)$ by 1 (a relabel step), and sets CurrentArc(i) to be the first arc of $A^+(i)$.

There are a variety of efficient implementations of the generic push-relabel algorithm. Efficiencies can be obtained by use of Sleator and Tarjan's [18] dynamic tree data structure and/or by a modification of the rule for selecting a node for a push/relabel. The $O(nm \log_k n + kn^2)$ max flow algorithm developed in this paper is based on the stack-scaling algorithm of Ahuja et al. [3], which is described in the next section. The stack-scaling algorithm is based on Ahuja and Orlin's excess-scaling algorithm ([2] and [1]).

4. Large-Medium Excess Scaling Algorithm (LMES)

In this section, we provide a modified version of the stack-scaling algorithm of [3]. We refer to this modified algorithm, which no longer relies on stacks, as the Large-Medium Excess Scaling (LMES) Algorithm.

In general, a scaling algorithm for the max flow problem solves the max flow problem as a sequence of scaling phases. Associated with each phase is a scaling parameter Δ, which remains constant during each phase and is decreased following the phase.

For the LMES Algorithm, the initial value of Δ is the least power of 2 that exceeds U. At each iteration of the Δ-scaling phase for the LMES Algorithm, the algorithm maintains a preflow in which the excesses satisfy the following property.

(4.1)   *For each $i \in N \backslash \{s, t\}$, $0 \le e(i) \le \Delta$.*

The Δ-scaling phase ends with the following property: for each $i \in N \backslash \{s, t\}$, $0 \le e(i) < \frac{\Delta}{k}$.

After the scaling phase terminates, the scaling parameter is divided by the *scaling factor k*, and the next scaling phase begins. The parameter k is a power of 2 that is selected by the user. The algorithm continues until $\Delta < 1$, at which point the algorithm terminates with a maximum flow. The total number of scaling phases is $O(\log_k U)$.

We express most of our algorithmic running times and bounds using the parameter k. In theorems that give the running time for the LMES and the Enhanced LMES, we will specify a value of k that optimizes the worst case asymptotic running time.

In the Δ-scaling phase, Equation 4.1 requires that $e(i) \le \Delta$ for each node i. To maintain this inequality, we modify the push operation.

LMES-Push(i, j)
   Applicability: Node i is active and arc (i, j) is admissible.
   Action: send $\delta = \min\{e(i), r_{ij}, \Delta - e(j)\}$ units of flow in arc (i, j).

In the Δ-scaling phase, a node $i \in N \backslash \{s, t\}$ is called a *large-excess node* if $e(i) \ge \frac{\Delta}{2}$. We let *LargeSet* denote the subset of nodes in $N \backslash \{s, t\}$ with large excess. Node i is called a *medium excess node* if $\frac{\Delta}{k} \le e(i) < \frac{\Delta}{2}$. We let *MediumSet* denote the subset of nodes in $N \backslash \{s, t\}$ with medium excess.

The procedure *LMES-Select-Node*, described below, selects a large or medium excess node i.

$i := $ LMES-Select-Node
   Applicability: there is a node with medium or large excess.
   Action: if LargeSet $\ne \emptyset$, then $i := \operatorname{argmin}\{d(j) : j \in \text{LargeSet}\}$;
      else, $i := \operatorname{argmax}\{d(j) : j \in \text{MediumSet}\}$.

The phase ends when there are no large or medium excess nodes. Otherwise, the algorithm selects a medium or large excess node i using the above procedure. If there is an admissible arc (i, j), then the algorithm next runs procedure *LMES-Push(i, j)*. If there is no admissible arc, the algorithm runs *Relabel(i)*. Subsequently, the algorithm returns to selecting a large or medium excess node.

The following theorem is stated and proved in [3].

Theorem 4.2. *The LMES Algorithm terminates after $O(\log_k U)$ scaling phases with a maximum flow from source to sink. Its running time is $O(nm + kn^2 + n^2 \log_k U)$.*

In order to balance the terms in the running time and optimize the overall running time, Ahuja et al. [3] chose k to be the least power of 2 that exceeds $2 + \frac{\log U}{\log \log U}$. For this value of k, the running time of the LMES Algorithm is $O(nm + n^2 \frac{\log U}{\log \log U})$.

5. The running time analysis of the LMES Algorithm

In this section, we will analyze the running time of the LMES Algorithm and prove Theorem 4.2. The proof here is based on the proof in [3]. We include the proof here for two reasons. First, the LMES Algorithm is different from the stack scaling algorithm, most notably because the LMES Algorithm does not use stacks. Second, the potential function arguments here are the starting point for the analysis of the Enhanced LMES Algorithm, whose pseudo-code is first presented in Section 8.

5.1. **The number of large pushes.** The proof of Theorem 4.2 relies on two potential function arguments. The first potential function is Φ_1.

$$\Phi_1 := \sum_{i \in N \setminus \{s,t\}} \frac{e(i)d(i)}{\Delta}.$$

Ahuja and Orlin [2] relied on Φ_1 in establishing the running time of the excess scaling algorithm. The excess-scaling algorithm is the special case of the LMES Algorithm in which there are no medium pushes, and $k = 2$. The running time of the excess-scaling algorithm is $O(nm + n^2 \log U)$.

Ahuja et al. [3] repurposed the potential function as part of the proof of the running time of the stack scaling algorithm.

Lemma 5.1. *During the Δ-scaling phase of the LMES Algorithm, the total amount of flow sent in pushes is less than $2n^2\Delta$.*

Proof. Each push of δ units of flow during the Δ-scaling phase decreases Φ_1 by $\frac{\delta}{\Delta}$. To prove the lemma, we will prove that the total decrease in Φ_1 (that is, the sum of all of the decreases in Φ_1) during the Δ-scaling phase is at most $2n^2$.

The total decrease in Φ_1 during the Δ-scaling phase is bounded above by the value of Φ_1 at the beginning of the Δ-scaling phase plus the total increase in Φ_1 during the Δ-scaling phase.

The initial value of Φ_1 is less than $(n-2)(n+1)$ because the sum is over at most $n-2$ nodes, and for each node i, $e(i) < \Delta$ and $d(i) \leq n+1$. To bound the total increase in Φ_1 during the scaling phase, we observe that the only way for Φ_1 to increase during the phase is when there is a distance relabel. For each node $i \in N$, an increase of $d(i)$ by 1 leads to an increase in Φ_1 by at most 1. Since $d(i) \leq n+1$, the total increase in Φ_1 caused by increases in distance labels is less than $(n-2)(n+1)$ over all phases.

Thus, the total amount pushed during the Δ-scaling phase is at most Δ times the initial value of Φ_1 plus Δ times the total increase of Φ_1, which is less than $2n^2\Delta$. □

By Lemma 3.2, the number of saturating pushes is $O(nm)$. We are interested in bounding the number of non-saturating pushes.

We say that a push of δ units of flow in (i,j) is a *large push* during the Δ-scaling phase if it is non-saturating and if $\delta \geq \frac{\Delta}{2}$. We say that the push in (i,j) is a *medium push* if it is non-saturating and if $\frac{\Delta}{k} \leq \delta < \frac{\Delta}{2}$.

The following is an immediate consequence of Lemma 5.1.

Corollary 5.2. *The total number of large pushes during the Δ-scaling phase is at most $4n^2$. The total number of large pushes over all scaling phases of the LMES Algorithm is at most $4n^2 \log_k U$.*

In the next subsection, we will bound the number of medium pushes. But before doing so, we provide some insight as to why the use of medium pushes leads to a speed-up of the excess-scaling algorithm.

If there were only large pushes, the algorithm would be equivalent to the Ahuja-Orlin excess-scaling algorithm [2], in which $k = 2$. In that case, the Δ- scaling phase would terminate with excesses bounded above by $\frac{\Delta}{2}$. The increase in Φ_1 because of the beginning of each scaling phase is $O(n^2)$ per scaling phase and $O(n^2 \log U)$ over all scaling phases. The increase in Φ_1 due to all relabels is $O(n^2)$. The running time of the excess scaling algorithm $O(nm + n^2 + n^2 \log U)$. One question that this analysis raised is the following: is there a way to modify the excess scaling algorithm so that the $n^2 \log U$ term in the running time is decreased. It would be OK if the n^2 term in the running time increased provided that it didn't increase much. The LMES Algorithm answered this question in the positive.

In the LMES algorithm, the medium pushes are of size proportional to $\frac{\Delta}{k}$. At the end of the Δ-scaling phase, the excesses are all less than $\frac{\Delta}{k}$, which permits an increase in the scaling factor from 2 to k. Increasing the scale factor improves the running time by a factor of $O(\log k)$

provided that there are not too many medium pushes in total. Selecting medium pushes from nodes with maximum distance labels is sufficient for establishing the bound on medium pushes proved in the next subsection.

5.2. The number of medium pushes.
We now use a different potential function argument to bound the number of medium pushes. We state the bounds as a lemma.

Lemma 5.3. *The total number of medium pushes in the LMES Algorithm is $O(kn^2 + n^2 \log_k U)$.*

Proof. We use a potential function Φ_2 of [3], which is based on parameters ℓ and P. At the beginning of a scaling phase, $\ell = n + 1$. Subsequently, ℓ is the minimum distance label from which there has been a medium push during the phase. At the beginning of the scaling phase, $P = \emptyset$. Subsequently,

$P := \{i \in N : d(i) > \ell\} \cup \{i \in N : d(i) = \ell, \text{ and there was a medium push from } i\}$.

We observe that whenever there is a medium push from any node v, then $v \in P$ except possibly for the first medium push from node v. For any scaling phase, the number of first medium pushes from nodes of N is at most $n - 2$. To bound the remaining number of medium pushes, we rely on the following potential function developed in [3].

The potential function Φ_2 is defined as follows.

$$\Phi_2 := \sum_{j \in P} e(j) \cdot \frac{d(j) - \ell + 1}{\Delta}.$$

Note that $\Phi_2 = 0$ at the beginning of a scaling phase.

Every medium push from a node in P leads to a decrease in Φ_2 by at least $\frac{1}{k}$. Thus, the total number of medium (non-first) pushes in the Δ-scaling phase is at most k times the total decrease in Φ_2 during the phase. Because $\Phi_2 = 0$ at the start of the scaling phase, the total decrease in Φ_2 during the phase is bounded by the total increase in Φ_2 during the phase.

There are three ways in which Φ_2 can increase during a scaling phase:

(1) A node's distance label increases.
(2) ℓ decreases.
(3) A node enters P.

We first consider the impact of relabels on Φ_2. At an iteration at which a node i increases its distance label by 1, Φ_2 increases by $\frac{e(i)}{\Delta}$, which is at most 1, except possibly in the case in which $d(i) = \ell$ and $i \notin P$. In this latter case, Φ_2 increases by $\frac{2e(i)}{\Delta}$, which is at most 2. Thus the increase in Φ_2 over all phases due to distance increases is at most $2(n-2)(n+1)$.

We next consider the impacts from decreases in the parameter ℓ. Suppose that ℓ is decreased by 1. Let P be defined immediately prior to the decrease in ℓ. For each node $i \in P$, $d(i) - \ell$ increases by 1, and Φ_2 increases by at most 1. The total increase in Φ_2 per unit decrease in ℓ is at most $|P|$, which is at most $n - 3$. The increase in Φ_2 caused by all of the decreases in ℓ is less than n^2.

Finally, we consider the increases in Φ_2 that occur when nodes enter P. If a node i with $d(i) = \ell$ enters P, then Φ_2 increases by at most 1. If a node i enters P because of a decrease of ℓ by 1, then Φ_2 increases by at most 2. (We consider ℓ as decreasing by 1 unit at a time.) Thus, the increase in Φ_2 caused by nodes entering P is less than $2n$.

By the above, the total increase in Φ_2 over all scaling phases is $O(n^2 + \frac{n^2 \log_k U}{k})$, and thus the total number of medium pushes over all scaling phases is $O(kn^2 + n^2 \log_k U)$. $\square$

To complete the proof of Theorem 4.2, we need to bound the time for the selection of medium and large excess nodes. We first state it as a lemma.

Lemma 5.4. *The total time for selecting nodes is $O(\#\text{of pushes} + \#\text{of relabels} + n \log_k U)$.*

Proof. In order to carry out *LMES-Select-Node* efficiently, the algorithm maintains the following data structures. These data structures differ from those in [3] because the selection rule in LMES is different from the rule in the stack-scaling algorithm.

- MSet(i) = linked list of medium excess nodes with distance label i for $0 \leq i \leq n+1$.
- LSet(i) = linked list of large excess nodes with distance label i for $0 \leq i \leq n+1$.
- MaxML = maximum distance label of a node that has medium or large excess. If there is no medium or large excess node, then the scaling phase terminates.
- MinL = minimum distance label of a node with large excess. If there is no large excess node, then MinL = -1.
- NextL(i) for $0 \leq i \leq n-1$. If LSet(i) $\neq \emptyset$, then NextL(i) = $\min\{j : j > i$ and LSet(j) $\neq \emptyset\}$. If there is no such j or if LSet(i) = $\emptyset$, then NextL(i) = 0.

We select a node in $O(1)$ time as follows: if MinL $\neq 0$, we select the first node in LSet(MinL); else, if MaxML $\neq 0$, we select the first node in MSet(MaxML); if MaxML = 0, the phase ends.

The total time to initialize the data structures is $O(n)$ per scaling phase. We note that MSet( ) and LSet( ) can each be updated in $O(1)$ time following a push/relabel. To establish the running time of Lemma 5.4, we will show how to carry out all of the updates of MaxML, MinL, and NextL( ) in time proportional to the number of pushes plus the number of relabels.

We first consider increases in MaxML. MaxML can increase following a relabel of a medium or large excess node. This update takes $O(1)$ time per relabel and $O(n^2)$ over all scaling phases.

We now consider decreases in MaxML. MaxML can decrease by at most 1 following a medium or large push. However, following a saturating push from node v, MaxML can decrease by much more than 1. If MaxML decreases from a value i, the algorithm updates MaxML by iteratively checking MSet(j) and LSet(j) for $j = i-1, i-2, \ldots$, until it finds a non-empty set. (If all sets are empty, the phase terminates.) The time to update MaxML when MaxML is decreasing is proportional to the amount of decrease. The total decrease in MaxML during a scaling phase is bounded from above by the total increase in MaxML plus n. Thus, the total decrease in MaxML over all scaling phases is bounded by $n^2 + n \log_k U$. The time to carry out all updates MaxML over all scaling phases is $O(n^2 + n \log_k U)$.

We next consider updates to MinL and NextL( ). There are three operations (and four cases) in which NextL( ) or MinL needs to be updated.

- The beginning of a new scaling phase.
- A medium or large push in (v, w) causes node w to have large excess.
- A large push in (v, w) eliminates the large excess from node v.
- There is a relabel of a large excess node v.

At the beginning of a scaling phase, the time to initialize NextL( ) and MinL is $O(n)$. For each of the three remaining cases, NextL( ) and MinL can be updated in $O(1)$ steps. Thus, the total time to update NextL( ) and MinL is $O(\# \text{ of pushes} + n \log_k U)$. This completes the proof of Lemma 5.4 as well as the proof of Theorem 4.2.   □

We observe that we cannot use an amortized analysis similar to the analysis of MaxML when analyzing MinL because one cannot provide simple bounds on the increase or decrease in MinL over a phase. If a medium push from level j creates a large excess node, then MinL changes from -1 to $j-1$. If a saturating push from a node at level j results in LargeSet becoming empty, then MinL changes from j to -1. For this reason, we introduced the array NextL( ).

6. The parameters Q, ϵ, and M

We will employ a number of parameters in the remainder of this paper. We first define parameters Q, ϵ, and M.

We let $Q = \lceil \log_k 4n \rceil$. We let $\epsilon = k^{-Q}$. We let $M = k^{2Q}$.

The following are true:

(1) $\frac{1}{4nk} < \epsilon \leq \frac{1}{4n}$.
(2) Q is the least integer for which $k^{-Q} \leq \frac{1}{4n}$.
(3) $M = \epsilon^{-2}$.

(4) $16n^2 \leq M < 16k^2M^2$.
(5) If Δ is the scaling parameter for a scaling phase, then Q scaling phases later, the scaling parameter is $\epsilon\Delta$.

In subsequent sections, many of the parameters will be expressed as powers of ϵ or expressed in terms of multiples of M or Q.

7. An overview of the Enhanced LMES Algorithm

In this section, we present concepts and procedures that improve the running time from $O(nm + kn^2 + n^2 \log_k U)$ to $O(kn^2 + nm \log_k n)$. The $O(kn^2)$ term is due to the increases in Φ_2 from distance relabels. That term also occurs in the Enhanced LMES Algorithm. We do not consider that term further in this section. Instead, we focus on the $O(n^2 \log_k U)$ term, which arises in the analysis of both potential functions.

(1) **Increases in Φ_1 at the beginning of phases.** Consider each node i. When the scaling parameter is decreased by a factor k at the beginning of a scaling phase, the contribution $\frac{e(i)d(i)}{\Delta}$ increases by a factor of k. This leads to an increase in Φ_1 by $O(n)$. Over $log_k U$ scaling phases, the contribution to the increase in Φ_1 is $O(n \log_k U)$ for each node $i \in N$, which accounts for $O(n \log_k U)$ large pushes. Summing over all nodes, this accounts for $O(n^2 \log_k U)$ large pushes.

(2) **Increases in Φ_2 due to decreases in ℓ.** Consider each node i. For each unit decrease in the parameter ℓ, each node in P leads to an increase in Φ_2 by $O(\frac{1}{k})$. Thus, for each node i, reductions in ℓ lead to an increase in Φ_2 of $O(\frac{n}{k})$ during the scaling phase, accounting for $O(n)$ medium pushes. Node i accounts for $O(n \log_k U)$ medium pushes over all scaling phases. Summing over all nodes, this accounts for $O(n^2 \log_k U)$ medium pushes.

Any further discussion of increases in Φ_1 in this section refers to increases due to the beginning of phases. Any further discussion of increases in Φ_2 in this section, refers to increases caused by decreases in ℓ.

In the Enhanced LMES Algorithm, we reduce this increase of Φ_1 to $O(nm \log_k n)$. We reduce this increase in Φ_2 to $O(\frac{nm \log_k n}{k})$. These improvements reduce the number of large and medium pushes to $O(kn^2 + nm \log_k n)$ in total.

We will rely on various concepts, procedures, and data structures to achieve the improved running time. They are:

(1) the partition of arcs into small, medium, and large arcs;
(2) abundant arcs and contractions;
(3) special nodes; and
(4) the flow-return forest.

In the next four subsections, we provide an overview of how these four ideas lead to an improved strongly polynomial running time.

7.1. **Small, medium, and large arcs.** We say that arc (i, j) is *small* at the Δ-scaling phase if $u_{ij} + u_{ji} < \epsilon^5\Delta$. We say that the arc (i, j) is *medium* at the Δ-scaling phase if $\epsilon^5\Delta \leq u_{ij} + u_{ji} < 2M\Delta$. We say that the arc is arc (i, j) is *large* at the Δ-scaling phase if $u_{ij} + u_{ji} \geq 2M\Delta$. We sometimes refer to the arc as small, medium, or large without reference to Δ if the scaling parameter is clear from context.

The small arcs do not contribute much to the excess of a node, and accordingly do not contribute much to the increases in Φ_1 or Φ_2 described above. With respect to the analysis of the potential functions, we can (essentially) ignore flows in small arcs.

We now consider medium arcs. Each arc (i, j) is a medium arc for $O(\log_k n)$ scaling phases. Let $\Psi = \{\Delta : \Delta$ is a parameter for a scaling phase$\}$. For a given scaling parameter $\Delta \in \Psi$, let $N_{med}(\Delta)$ denote the set of nodes that are incident to at least one medium arc during the Δ-scaling phase. Accordingly,

$$\sum_{\Delta \in \Psi} |N_{med}(\Delta)| = O(m \log_k n).$$

Moreover, the contribution to the increases in Φ_1 and Φ_2 of nodes that are incident to a medium arc are $O(nm \log_k n)$ and $O(\frac{nm \log_k n}{k})$ respectively. Thus, over all scaling phases, nodes that are incident to medium arcs account for $O(nm \log_k n)$ large and medium pushes.

What remains is to analyze the contribution to increases in Φ_1 and Φ_2 from nodes that are not incident to any medium arc. But prior to focusing on nodes that are not incident to medium arcs, we discuss abundant arcs and contractions.

7.2. **Abundant arcs and contraction.** We say that an arc (i, j) is abundant at the Δ-scaling phase if (i, j) is large and if $r_{ij} \geq M\Delta$, where r_{ij} is the residual capacity at the beginning of the scaling phase. Once an arc becomes abundant, it remains abundant. We state this result as the following corollary to Lemma 14.6, which is the counterpart of Lemma 5.1 for the Enhanced LMES.

Corollary 14.7. *If (i, j) is abundant at the beginning of the Δ-scaling phase of the Enhanced LMES Algorithm, then for every scaling parameter $\Delta' \leq \Delta$ the following are true.*

  (1) *(i, j) is abundant in the Δ'-scaling phase and*
  (2) *$r_{ij} > 0$ throughout the Δ'-scaling phase.*

The importance of abundant arcs is that they can be used to transform the max flow problem into an equivalent problem of smaller size. In particular, if there is a directed cycle C of abundant arcs, then one can contract the cycle C into a single node called a *merged node* and run the Enhanced LMES algorithm on the contracted graph. As each abundant cycle is discovered, it is contracted into a merged node. The Enhanced LMES Algorithm continues from where it left off on the contracted graph. Eventually, the algorithm determines a maximum flow in some contracted version of the original network. Then the maximum flow in this contracted graph is converted into a maximum flow in the original graph by expanding the merged nodes in the reverse order in which they were contracted. We describe the contraction in Section 10. We describe the expansion in the appendix in Section A.

Contraction, in and of itself, does not even lead to a strongly polynomial time algorithm. We illustrate this fact by the following example. Suppose that the node set $N = \{s, 1, 2, t\}$. Suppose that there are arcs $(s, 1)$ and $(1, t)$ with a capacity of $U = k^\alpha$ for some very large integer α. There are also arcs $(s, 2)$ and $(2, t)$ with a capacity of 1. After initialization, $e(1) = k^\alpha$ and $e(2) = 1$. The initial scaling parameter is $\Delta = U$. However, there is no push from node 2 until $\Delta = 1$. If each scaling parameter is obtained by dividing by k, then the number of scaling phases would be α, which is not strongly polynomial.

The algorithm fails to be strongly polynomial in this example because of the scaling phases in which there was no push. We refer to a scaling phase as *useful* if there is at least one push during the scaling phase or if the flow-return forest (discussed in Subsection 7.4) is non-empty. Otherwise, it is *useless*. Our algorithm avoids useless scaling phases by modifying the rule for choosing the next scaling parameter. We will establish the following result.

Corollary 14.3. *The number of scaling phases in the enhanced LMES is $O(m \log_k n)$.*

7.3. **Special nodes.** The enhanced LMES will partition the nodes in $N \setminus \{s, t\}$ into nodes that are "special" and nodes that are "non-special". (Actually, we will further partition the non-special nodes into two parts, but for now it suffices to consider special and non-special nodes.) We will define the term "special" later in this subsection. For now, we provide two properties of special nodes and one property of non-special nodes.

  (1) If node i is special, then $e(i) < 1.5\epsilon^4 \Delta$ at the beginning of the scaling phase.
  (2) If node i is special, then $e(i) < 1.5\epsilon^4 \Delta$ whenever ℓ is reduced.

(3) If node i is not special, then within $O(\log_k n)$ scaling phases, i will be incident to a medium arc, or i will be a node of an abundant cycle that is contracted.

The first two properties imply that each special node contributes much less than $\frac{1}{n}$ per scaling phase to the increase of Φ_1 and Φ_2. Since there are $O(m \log_k n)$ scaling phases in total, special nodes account for $O(nm \log_k n)$ large and medium pushes.

The third property implies that the total number of nodes that are not special is $O(m \log_k n)$ over all scaling phases. Thus, nodes that are not special will account for $o(nm \log_k n)$ large and medium pushes.

To summarize, non-special nodes account for $O(kn^2 + nm \log_k n)$ medium and large pushes. And special nodes also account for $O(kn^2 + nm \log_k n)$ medium and large pushes. Thus, the number of medium and large pushes is $O(kn^2 + nm \log_k n)$. This (apparently) leads to the running time claimed in our paper. However, we have not yet mentioned an important issue that arises because of special nodes. In order to ensure that Properties 1 and 2 are satisfied by each special node, we permit the excess for a special node i to be negative, provided that $e(i) \geq -1.5\epsilon^4\Delta$. This small difference in what is permitted makes a huge difference in the analysis. The generic push/relabel algorithm terminates with a maximum flow because every excess is non-negative at termination. We needed to adapt our algorithm so that every excess is 0 at termination. This required new methods for sending flow to nodes with negative excess.

In the next subsection, we will describe a new data structure that returns flow to nodes with negative excess. Using this data structure, excesses are always at least $-\epsilon\Delta$, and all excesses are nonnegative at termination. But first, we define what it means for a node to be special. We also explain why it makes sense to permit negative excesses for special nodes.

Recall that (i, j) is abundant if (i, j) is large and $r_{ij} \geq M\Delta$. If (i, j) is abundant, and if (j, i) is not abundant, then arc (j, i) is called *anti-abundant*. If (i, j) and (j, i) are both abundant, then arc (i, j) is called *bi-abundant*. A bi-abundant arc is an abundant cycle with two arcs. In general, the algorithm contracts any bi-abundant arc. (There is an exception discussed in the section on the flow-return forest.)

During the Δ-scaling phase of the Enhanced LMES Algorithm, we require that the residual capacity of each anti-abundant arc is a multiple of $\frac{\Delta}{k}$. In Section 11.2, we explain how this invariant can be satisfied throughout the Enhanced LMES with only a constant increase in running time.

For a given preflow x and a given node v, we define the Δ-*imbalance* of v with respect to x to be:
$$\text{IMB}(v, x, \Delta) = e_x(v) + \sum_{(j,v) \in Anti(\Delta)} r_{jv}(x) - \sum_{(v,j) \in Anti(\Delta)} r_{vj}(x).$$

We define $\text{IMB}(v, \Delta)$ to be $\text{IMB}(v, x, \Delta)$, where x is the initial preflow at the beginning of the Δ-scaling phase.

Let us for the time being ignore small capacity arcs. (Assume that there are no small arcs incident to node v.) Because residual capacities of anti-abundant arcs are multiples of $\frac{\Delta}{k}$,

$$\text{IMB}(v, x, \Delta) \equiv e_x(v) \left(\text{mod } \frac{\Delta}{k} \right).$$

A node v is called a *special node* at the Δ-scaling phase if v is not incident to any medium arcs and if $|\text{IMB}(v, \Delta)| \leq \epsilon^4\Delta$.

If v is special, then every medium or large push from v will be a multiple of $\frac{\Delta}{k}$ units of flow. Our algorithm analysis relies on special nodes having an excess close to 0 at the beginning of scaling phases and when ℓ is reduced. This works well for a special node v if $0 \leq \text{IMB}(v, \Delta) \leq \epsilon^4\Delta$. When v is no longer medium or large, then $0 \leq e(v) \leq \epsilon^4\Delta$.

Consider what happens if $-\epsilon^4\Delta < \text{IMB}(v, \Delta) < 0$. Suppose that node v is not permitted to have a negative excess. When v is no longer medium or large, then $\frac{\Delta}{k} - \epsilon^4\Delta < e(v) < \frac{\Delta}{k}$. In this scenario, a special node would contribute significantly to the increase of the potential functions.

So, our Enhanced LMES Algorithm permits pushing flow from a special node v when $e(v)$ is close to but slightly less than $\frac{\Delta}{k}$. This can result in a very small negative excess. To ensure that flow is returned to v prior to termination, the algorithm uses a data structure that we call "the flow-return forest."

7.4. **The flow-return network.** Actually, a special node v can have negative excess provided that $e(v) \geq -1.5\epsilon^4\Delta$. The procedures of the LMES Algorithm are not enough to guarantee that flow will be sent to node v. Accordingly, we make two additional changes in the LMES Algorithm. The first change is the requirement of a "buffer" excess at each non-special node. In particular, if node w is not special, then we "require" that $e(w) \geq \epsilon\Delta$. If w is non-special and if $e(w) \geq \epsilon\Delta$, we say that w is *normal*. If w is non-special and if $e(w) < \epsilon\Delta$, we say that w is *violating*. A node w becomes violating at the beginning of the first scaling phase after w stops being special.

When a node v becomes violating, it is added to a data structure that we call the "flow-return forest" (FRF), which we usually denoted as F. The flow-return forest F has the following properties.

(1) F is a forest; that is, it has no cycles.
(2) Each component of F has a root node. For each $w \in F$, $Root(F, w)$ denotes the root node of w in F.
(3) For each node $w \in F$, there is a directed path of abundant arcs in F from $Root(F, w)$ to w. This implies that F is a directed forest, with all arcs directed away from root nodes.
(4) An arc (i, j) is called *FRF-eligible* if (i, j) is abundant and $d(j) \leq d(i) + 1$. Every arc of the flow-return forest is FRF-eligible.
(5) If a violating node v is added to F in the Δ-scaling phase, then Flow-Needed(v) is set to $\frac{\epsilon^2\Delta}{k}$. Suppose that $\Delta' = \epsilon^2\Delta$. If v is still violating at the Δ'-scaling phase, then $\frac{\epsilon^2\Delta}{k}$ units of flow will be sent from $Root(F, v)$ to v in F. (Note that the reversal of the path P from $Root(F, v)$ to v will have residual capacity after flow is sent in P. Condition (4) on distance labels is needed so that every arc of the reversal of P is valid.)
(6) Each root vertex $i \in F$ is a normal node. A root vertex i maintains additional excess called $Reserve(i)$, where

$$Reserve(i) = \sum_{v \in Desc(i,F)} NeededFlow(v),$$

where $Desc(i, F)$ denotes the descendants of node i in F. The reserve requirements imply that $e(i) \geq \epsilon\Delta + Reserve(i)$.
(7) Every leaf node of F is violating. (Non-violating leaf nodes are deleted from F).
(8) If $v \in F$ at the Δ-scaling phase, then $e(v) \geq -\frac{\epsilon\Delta}{2k}$.
(9) If $v \in F$ is violating at the Δ-scaling phase and if Flow-Needed$(v) = \frac{\Delta}{k}$, then $\frac{\Delta}{k}$ units of flow are sent from $Root(F, v)$ to v in F. Subsequent to the flow being sent, $e(v) \geq \frac{\Delta}{2k}$.

In ensuring that the above properties are all satisfied by the flow-return forest, we also need to address several complexities that arise. We briefly describe them here.

(1) The algorithm needs to be able to add any violating node to the flow-return forest while satisfying all of the above conditions. The FRF-eligible Path Lemma of Section 12 states the following. For any violating node v, there is a path of FRF-eligible arcs from a normal node to node v. By relying on this lemma, we are always able to add violating nodes to the flow-return forest. Using a breadth first search algorithm, the running time is $O(m)$ to add a violating node to F.
(2) The number of violating nodes added to the flow-return forest is $O(m)$. In order to achieve a total running time of $O(kn^2 + m \log_k n)$, we cannot afford to use the $O(m)$ breath first search algorithm to add each violating node v. Instead, we use a depth first search algorithm that runs in $O(n)$ time.
(3) If node $j \in F$ and if j is relabeled, then the FRF-eligible arc directed into node j might stop being eligible. We bypass this difficulty by ensuring that no node of F is relabeled.

(4) The reserve of root nodes might exceed Δ. We bypass this difficulty by not including the reserve of root nodes when evaluating the potential functions. For each root node v, the term included in the evaluation of the potential functions is $e(v) - Reserve(v)$.

(5) We need to ensure that $e(v) - Reserve(v) \leq \Delta + k\epsilon\Delta$. To guarantee this property, we sometimes send additional flow from root nodes in the procedure *FRF-Delete*.

8. ANNOTATED PSEUDO-CODE

Here we present a very high level pseudo-code for the Enhanced LMES Algorithm. We also present the pseudo-code of the scaling phase. Following the pseudo-code is an annotation explaining of the steps and where the pseudo-code of procedures are located.

Enhanced-LMES Algorithm
01. **begin**
02.   *Initialize-Enhanced-LMES*
03.   **while** $e(v) \neq 0$ for some $v \neq s$ or t, **do** *ScalingPhase*(Δ, F);
04.   expand the contracted cycles, obtaining a max flow;
05. **end**

Line 02. The initialization is very similar to the initialization of the LMES Algorithm except that each node other than s and t is guaranteed to have an excess of at least $\epsilon\Delta$ after initialization. Details are given in Subsection 11.3.

Line 03. *ScalingPhase*( ) is the main procedure. A high level view is presented below.

Procedure ScalingPhase(Δ, F)
01. **begin**
02.   **while** there is a newly violating node $v \notin F$, **do** *FRF-Add*(v, Δ, F);
03.   **while** there is a violating node $v \in F$ with $FlowNeeded(v) = \frac{\Delta}{k}$,
–   **do** *FRF-Pull*(v, F, Δ);
04.   *FRF-Recursive-Delete-and-Merge*(Δ, F);
05.   **while** there is a medium or large excess node **do**
06. $v :=$ *Enhanced-LMES-Select-Node*;
07.   **if** $v \in F$, **then do** *FRF-Push*(v, Δ, F);
08.   **else if** $v \notin F$ **then do** *Enhanced-LMES-Push/Relabel*(v, Δ);
09.   *FRF-Recursive-Delete-and-Merge*(Δ, F);
10.   **endwhile**
11.   *Get-Next-Scaling-Parameter*(Δ);
12. **end**

Line 02. A newly violating node v is a node such that $e(v) < \epsilon\Delta$ at the beginning of the scaling phase, and such that node v was special in the previous scaling phase.

Line 02. *FRF-Add*(v, Δ, F) appends node v as a leaf of F (and possibly adds other nodes to F) in such a way that the path from $Root(F, v)$ to v consists of FRF-eligible arcs. The pseudo-code for the procedure *FRF-Add* is given in Subsection 12.5.

Line 03. If v is violating and if $FlowNeeded(v) = \frac{\Delta}{k}$, then $\frac{\Delta}{k}$ units of flow are sent from $Root(F, v)$ to v along arcs in F. Subsequent to sending this flow, node v is no longer violating. The pseudo-code for the procedure *FRF-Pull* is given in Subsection 12.5.

Line 04. If any leaf node of F is non-violating, it is deleted from F. Any bi-abundant arc with at most one endpoint in F is contracted. The pseudo-code for the procedure *FRF-Recursive-Delete-and-Merge* is given in Subsection 13.1. It relies on the procedure *FRF-Delete*, which is

described in Subsection 12.5.

Line 05. A special node v has *medium excess* (resp., *large excess*) if $e(v) \geq \frac{\Delta}{k} - 1.5\epsilon^4\Delta$ (resp., $e(v) \geq \frac{\Delta}{2} - 1.5\epsilon^4\Delta$). A normal node v has *medium excess* (resp., *large excess*) if $e(v) \geq \frac{\Delta}{k} + \epsilon\Delta$ (resp., $e(v) \geq \frac{\Delta}{2} + \epsilon\Delta$).

Line 07. *FRF-Push*(v, Δ, F) sends $\frac{\Delta}{k}$ units of flow in arcs of F from v to a violating node w of F. The node w is a leaf node that is a descendant of node v. After the push, node w becomes non-violating. The pseudo-code for the procedure *FRF-Push* is given in Subsection 12.5.

Line 08. The Procedure *Enhanced-LMES-Push/Relabel*(v, Δ) guarantees that the following invariant is always satisfied: if $r_{ij} \leq r_{ji}$, and if (i,j) is not bi-abundant, then r_{ij} is a multiple of $\frac{\Delta}{k}$. Because this invariant is satisfied, it is also true that the residual capacity of each anti-abundant arc is a multiple of $\frac{\Delta}{k}$. The pseudo-code for the procedure *Enhanced-LMES-Push/Relabel* is given in Subsection 11.4.

Line 11. Typically, the scaling parameter Δ is replaced by $\frac{\Delta}{k}$. An exception is when there would be no push for any of the next $2Q$ scaling phases and where $F = \emptyset$. In this latter case, the scaling parameter is chosen so that in the next scaling phase one of the following two events takes place (i) there is a push, or (ii) a node becomes violating and F becomes non-empty. The pseudo-code for the procedure *Get-Next-Scaling-Parameter* is given in Subsection 13.1.

9. Abundant arcs and contractions

We say that an arc (i,j) is *abundant* in the Δ-scaling phase of the Enhanced LMES if $r_{ij} \geq M\Delta = 16n^2\Delta$ at the beginning of the scaling phase. Abundant arcs are guaranteed to have positive residual capacity throughout the remainder of the execution of the algorithm. A common approach for transforming a weakly polynomial flow algorithm into a strongly polynomial time algorithm involves the contraction of directed cycles of abundant arcs.

Lemma 5.1 states that the total amount of flow pushed in a scaling phase of the LMES Algorithm is at most $2n^2\Delta$. We will show in Subsection 14.5 that the total amount of flow pushed in the Δ-scaling phase of the enhanced LMES Algorithm is less than $5n^2\Delta$. As a corollary, one can show that an arc that is abundant in the Δ-scaling phase is also abundant in the next scaling phase, and hence in all subsequent scaling phases. To see why, note assume that $r_{ij} \geq M\Delta$ at the beginning of the Δ-scaling phase. At the end of the phase, $r_{ij} \geq M\Delta - 5n^2\Delta \geq \frac{M\Delta}{2}$. If Δ' denotes the scaling parameter at the next scaling phase, then $\Delta' \leq \frac{\Delta}{2}$ and $r_{ij} > M\Delta'$, and (i,j) remains abundant.

An *abundant cycle* is a directed cycle of abundant arcs. When the algorithm identifies an abundant cycle W, it contracts the cycle into a single node that we refer to as a *merged node*. The contraction of W reduces the number of nodes in the network by $|W| - 1$. Since each abundant cycle has at least two nodes, the number of contractions is at most $n - 1$. The number of merged nodes created over all scaling phases is at most $n - 1$. The total number of arcs in the contracted abundant cycles is at most $2n - 2$.

Each of the $n-1$ contractions can be carried out in $O(m)$ time. Thus, the total time to carry out the contractions is $O(nm)$, which is not a bottleneck.

After contracting an abundant cycle, the Enhanced LMES algorithm continues on the contracted network starting from where it left off. Eventually, the algorithm identifies a maximum flow, possibly after additional contractions of abundant cycles. Then the algorithm expands the merged nodes into abundant cycles and recovers a maximum flow in the original network. We will explain how to expand pseudo-nodes and recover the maximum flow in Subsection A in the Appendix. The expansion of the contracted cycles is not a bottleneck operation.

The contraction of abundant cycles is an approach based on the seminal work by Tardos [19] for transforming weakly polynomial time algorithms into strongly polynomial algorithms. Orlin

[16] contracted abundant cycles in his $O(nm)$ max flow algorithm. In addition, contraction of cycles was an important aspect of Goldberg and Rao's [9] weakly polynomial max flow algorithm.

Of special importance to the algorithm is the contraction of bi-abundant arcs. Also of importance are the reversals of abundant arcs. If (j, i) is abundant and if (i, j) is not abundant, then we say that arc (i, j) is *anti-abundant*. We will explain how the algorithm efficiently maintains lists of abundant, anti-abundant, and bi-abundant arcs in Section B of the appendix.

9.1. **Distance labels following a contraction.** In the generic push/relabel algorithm, the running time analysis relies on distance labels never decreasing. However, when we introduce contractions, distance labels might need to decrease. This is illustrated in Figure 1.

Figure 1 shows a subset of nodes before and after a contraction. Unless $d(1)$ is decreased following the contraction (or $d(6)$ is increased), there is no way of assigning a distance label to node w so that the distance labels are valid.

In our Enhanced LMES Algorithm, we permit distance labels to decrease subsequent to the contraction of an abundant cycle W.

We assume that prior to the contraction of W, $d(v)$ is the minimum length of a path v to t. We can make the assumption true by relabeling the nodes in $O(m)$ time prior to the contraction.

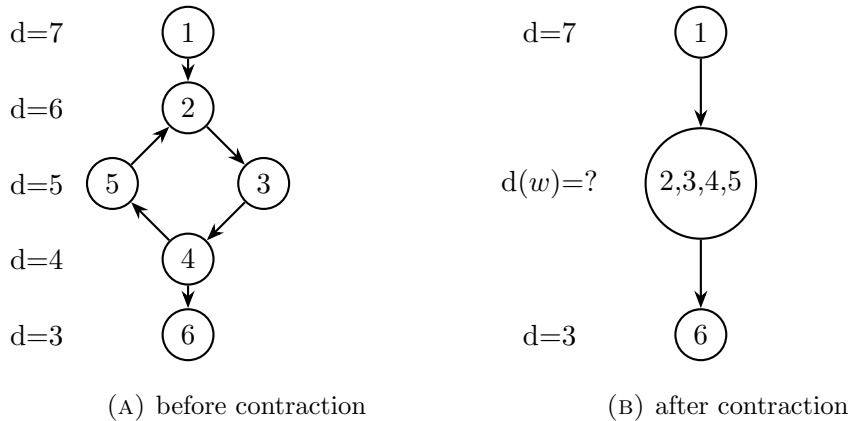(A) before contraction   (B) after contraction

FIGURE 1. Part of a graph before and after contraction

After the contraction of an abundant cycle, the algorithm lets $d(s) = |N^c|$, and lets $d(v)$ be the minimum length of a path v to t in $G^c(x)$. If there is no path from node v to node t, then $d(v) = d(s) + 1$. Then the current arc of each node is set to the first arc of its arc list, and the current node of each merged node is set to the first node in its node list. The time to update the distance labels is $O(m)$ per contraction and $O(nm)$ in total.

Even though the distance label of a node v may decrease following a contraction, the total number of increases in $d(v)$ is still bounded above by $n + 1$.

Lemma 9.1. *For each node v, the total number of increases in $d(v)$ over all iterations of the LMES with contraction is at most $n + 1$.*

Proof. As the algorithm proceeds, let n' denote the number of nodes in the (contracted) network. For each node v, let $d'(v) = d(v) - (n - n')$. Prior to any contractions in the network, $n' = n$, and $d'(v) = d(v)$. At every iteration, $d'(s) = n' + (n - n') = n$.

Note that $d'(v)$ increases by 1 whenever v is relabeled, and $d'(v)$ does not decrease following a contraction. Moreover, $d'(v) \leq d'(s) + 1 = n + 1$. Thus the number of relabels of node v is at most $n + 1$. $\square$

9.2. **Large excesses following a contraction.** Suppose cycle W is contracted, resulting in merged node w. After the contraction, $e(w)$ is the sum of the excesses of nodes of W. After the

contraction, $e(w)$ can be nearly as large as $|W| \times \Delta$. The running time analysis for the LMES algorithm relied on excesses not exceeding Δ. We can circumvent that difficulty as follows. We let $N(\Delta)$ denote the nodes (including merged nodes) at the beginning of the Δ-scaling phase. The potential functions Φ_1 and Φ_2 include summations over the nodes in $N(\Delta)$ at the beginning of the phase. Suppose that a merged node w is obtained during the phase by contracting a cycle W. Rather than replacing the nodes of W by w in the summation for the potential functions, we continue to sum over the nodes of $N(\Delta)$. For each node $v \in W$, we define $e(v)$ to be $\frac{e(w)}{|W|}$. And we define $d(v)$ to be $d(w)$. Upon contraction of the cycle W, the potential functions Φ_1 and Φ_2 do not increase. Moreover, $e(v)$ does not exceed Δ during the phase.

For further details, see Theorem 14.10.

10. The contraction lemma and its implications

In this section, we state and prove the Contraction Lemma, which provides conditions that guarantee the existence of a bi-abundant arc within $O(\log_k n)$ scaling phases. The Contraction Lemma is a variation of a lemma proved in Orlin [16] as part of an $O(nm)$ max flow algorithm. The proof here is new. It is simpler than the proof in [16].

10.1. **Small, medium, and large arcs.** Let $G(\Delta) = (N(\Delta), A(\Delta))$ denote the (contracted) network at the beginning of the Δ-scaling phase. Recall that an arc is called *small*, *medium*, or *large* according as $u_{ij} + u_{ji} < \epsilon^5 \Delta$ or $\epsilon^5 \Delta \leq u_{ij} + u_{ji} < 2M\Delta$ or $u_{ij} + u_{ji} \geq 2M\Delta$. Let Small($\Delta$), Medium($\Delta$), and Large($\Delta$) denote the sets of small, medium, and large arcs of the Δ-scaling phase.

If (i, j) is large at the Δ-scaling phase, then (i, j) must be abundant or anti-abundant. Let Abundant(Δ) and Anti(Δ) denote the subset of arcs of Large(Δ) that are abundant and anti-abundant arcs at the beginning of the Δ-scaling phase. The following two relationships follow directly from the definitions.

- $A(\Delta) = \text{Small}(\Delta) \cup \text{Medium}(\Delta) \cup \text{Large}(\Delta)$.
- $\text{Large}(\Delta) = \text{Abundant}(\Delta) \cup \text{Anti}(\Delta)$.

We say that there is an *occurrence* of an arc (i, j) (resp., node v) at the Δ-scaling phase if $(i, j) \in A(\Delta)$ (resp., $v \in N(\Delta)$).

Each arc in A can remain medium for at most $O(\log_k n)$ scaling phases. We express this fact in terms of the total number of occurrences of medium arcs.

Lemma 10.1. *The total number of occurrences of medium arcs over all scaling phases is $O(m \log_k n)$.*

10.2. **Node imbalances and the contraction lemma.** For any preflow x defined at or after the Δ-scaling phase, the Δ-*imbalance* of node v with respect to x is:

$$\text{IMB}(v, x, \Delta) = e_x(v) + \sum_{(j,v) \in Anti(\Delta)} r_{jv}(x) - \sum_{(v,j) \in Anti(\Delta)} r_{vj}(x).$$

We let $\text{IMB}(v, \Delta)$ refer to $\text{IMB}(v, x, \Delta)$ where x is the initial preflow of the Δ-scaling phase.

If y and x are preflows, then $y - x$ denotes the difference. Our first result is a simplification of the formula for $\text{IMB}(v, y - x, \Delta)$ in the case that x and y are both preflows obtained at or after the Δ-scaling phase.

Lemma 10.2. *Suppose that preflow x is obtained by the algorithm at the Δ-scaling phase. Suppose that y is a preflow obtained at the Δ'-scaling phase for $\Delta' \leq \Delta$. Suppose further that node v is not incident to a bi-abundant arcs at either of these two scaling phases. Then*

$$(10.3) \qquad IMB(v, y - x, \Delta) = e_{y-x}(v) - \sum_{(v,j) \in Large(\Delta)} r_{vj}(y - x).$$

In addition, if node v is not incident to any medium arcs at the Δ-scaling phase, then the following three equalities and inequality are valid.

$$(10.4) \qquad\qquad e_{y-x}(v) = \sum_{(v,j)\in Small(\Delta)\cup Large(\Delta)} r_{vj}(y-x).$$

$$(10.5) \qquad\qquad IMB(v,y-x,\Delta) = \sum_{(v,j)\in Small(\Delta)} r_{vj}(y-x)$$

$$(10.6) \qquad |IMB(v,y-x,\Delta)| \leq \sum_{(v,j)\in Small(\Delta)} (u_{vj}+u_{jv}) \quad \leq 2n\epsilon^5\Delta < .5\epsilon^4\Delta.$$

Proof. For each arc $(i,j) \in A$, $r_{ij}(y-x) = -r_{ji}(y-x)$. By applying this observation to arcs $(j,v) \in Anti(\Delta)$, one obtains Equation 10.3.

We next prove that Equation 10.4 is valid. If there are no medium arcs incident to node v at the Δ-scaling phase, then $A(\Delta) = Small(\Delta) \cup Large(\Delta)$. In sending δ units of flow in any arc (i,j), r_{ij} decreases by δ, and r_{ji} increases by δ. To keep track of changes in flow incident to node v, it suffices to consider the change in residual capacity of arcs leaving node v. Therefore, Equation 10.4 is valid. Equality 10.5 follows from equations 10.3 and 10.4. Inequality 10.6 follows directly from equation 10.5. □

We say that node i is α-*balanced* in the Δ-scaling phase if $|IMB(v,\Delta)| \leq \alpha$. We say that i is α-*imbalanced* if $|IMB(v,\Delta)| > \alpha$.

Lemma 10.7. (Contraction Lemma). *Suppose that node v is not incident to any medium arcs at the Δ-scaling phase and that v is $(\epsilon^4\Delta)$-imbalanced at the beginning of the Δ-scaling phase. Then node v will become incident to a bi-abundant arc within $7Q$ scaling phases.*

Proof. Let x denote the preflow at the beginning of the Δ-scaling phase. Thus $|IMB(v,x,\Delta)| > \epsilon^4\Delta$. Let $\Delta' = \epsilon^7\Delta$ be the scaling parameter $7Q$ scaling phases subsequently. Let y be the preflow at the beginning of the Δ'-scaling phase. Then

$$|IMB(v,y,\Delta)| \geq |IMB(v,x,\Delta)| - |IMB(v,y-x,\Delta)|$$
$$\geq \epsilon^4\Delta - 2n\epsilon^5\Delta \geq .5\epsilon^4\Delta.$$

The second inequality follows from Lemma 10.2. Note that $e_y(v) < \Delta'$. We now consider two cases according as $IMB(v,y,\Delta)$ is positive or negative. If $IMB(v,y,\Delta) > 0$, then

$$IMB(v,y,\Delta) = e_y(v) + \sum_{(j,v)\in Anti(\Delta)} r_{jv}(y) - \sum_{(v,j)\in Anti(\Delta)} r_{vj}(y) > .5\epsilon^4\Delta > nM\Delta'.$$

In this case, there is an arc $(j,v) \in Anti(\Delta)$ such that $r_{jv}(y) > M\Delta'$, in which case arc (j,v) is bi-abundant at the Δ'-scaling phase. If, instead, $IMB(v,y,\Delta) < 0$, then

$$IMB(v,y,\Delta) = e_y(v) + \sum_{(j,v)\in Anti(\Delta)} r_{jv}(y) - \sum_{(v,j)\in Anti(\Delta)} r_{vj}(y) < -.5\epsilon^4\Delta < -nM\Delta'.$$

In this case, there is an arc $(v,j) \in Anti(\Delta)$ such that $r_{vj}(y) > M\Delta'$, in which case arc (v,j) is bi-abundant at the Δ'-scaling phase. This completes the proof. □

11. Special, Violating, and Normal nodes.

In this section, we review the three types of nodes: special nodes, violating nodes, and normal nodes. We describe properties and invariants satisfied by these nodes. We also provide the procedures *Initialize*, *Select*, and *Push* for the Enhanced LMES Algorithm.

11.1. **Three flavors of nodes.** The Enhanced LMES Algorithm partitions the nodes of $N(\Delta)\backslash\{s,t\}$ into three different flavors (types). For each node v, we also define the modified excess $\hat{e}(v,\Delta)$, which replaces $e(\ )$ in the procedures within the Enhanced LMES Algorithm.

Nodes that are special at the Δ-scaling phase.
     A node v is a *special node* during the Δ-scaling phase if it satisfies the following.

(1) Node v is not incident to any medium arcs, and
(2) $|\mathrm{IMB}(v,\Delta)| \leq \epsilon^4\Delta$.

     For each special node $v \in N(\Delta)$, we define $\hat{e}(v,\Delta) = e(v) + 1.5\epsilon^4\Delta$. The following are properties of nodes that are special at the Δ-scaling phase.

S1. $\mathrm{IMB}(v,\Delta) \leq \epsilon^4\Delta$. (By definition.)
S2. v is not incident to any medium arcs. (By definition.)
S3. If an anti-abundant arc a is incident to v, then for each preflow x during the Δ-scaling phase, $r_a(x) \equiv 0 \left(\mathrm{mod}\ \frac{\Delta}{k}\right)$. (See Lemma 11.1.)
S4. $e(v) \geq -1.5\epsilon^4\Delta$, and thus $\hat{e}(v) \geq 0$. (See Lemma 11.1.)
S5. If $\hat{e}(v) > 3\epsilon^4$, then $\hat{e}(v) \geq \Delta/k$. Equivalently, if $\hat{e}(v) > 3\epsilon^4$, then v will either have medium or large excess. (See Lemma 11.1.)

Nodes that are normal at the Δ-scaling phase.
     We will "require" every non-special node to maintain an excess of at least $\epsilon\Delta$ during the Δ-scaling phase. This small amount of buffer is needed later when we add nodes to the flow-return forest.
     A node v is called a *normal node* during the Δ-scaling phase if $e(v) \geq \epsilon\Delta$.
     For each normal node $v \in N(\Delta)$, we define $\hat{e}(v,\Delta) = e(v) - \epsilon\Delta$. The following are properties of nodes that are normal during the Δ-scaling phase.

N1. If node v is normal during the Δ-scaling phase, then $\hat{e}(v,\Delta) \geq 0$. (By definition.)
N2. If node v is normal during the Δ-scaling phase, then $\hat{e}(v,\Delta) \leq \Delta + (k-1)\epsilon\Delta$. (See Lemma 14.5.)
N3. The total number of occurrences of normal nodes is $O(m\log_k n)$. (See Lemma 14.1.)

Nodes that are violating at the Δ-scaling phase.
     A node v is a *violating node* at the Δ-scaling phase if $e(v) < \epsilon\Delta$ and v is not special. For each violating node $v \in N(\Delta)$, we define $\hat{e}(v,\Delta) = e(v) - \epsilon\Delta$. A node is called *newly violating* at the Δ-scaling phase if it is violating at the Δ-scaling phase but not violating at the previous phase.
     The following are properties of nodes that are violating at the Δ-scaling phase. Several of these properties mention a data structure called the flow-return forest, which is described in Section 12.

V1. $\hat{e}(v,\Delta) < 0$. (By definition.)
V2. If v is newly violating at the Δ-scaling phase, then v is added to the flow-return forest at that phase. (Refer to the Procedure *FRF-Add* in Section 12.)
V3. If v is violating at the Δ-scaling phase but not newly violating, then v is already in the flow-return forest at the beginning of the Δ-scaling phase. Node v will be deleted from the flow-return forest at the iteration at which v is non-violating and v is a leaf of the flow-return forest.
V4. If v is violating at the Δ-scaling phase, then v will become non-violating within $2Q$ scaling phases. (See Lemma 12.7.)
V5. If v is violating, then $\hat{e}(v) \geq -\epsilon\Delta$. (Corollary 12.9.)
V6. The total number of occurrences of newly violating nodes is $O(m\log_k n)$. (See Lemma 14.1.)

     We have defined $\hat{e}(v,\Delta)$ for special, violating and normal nodes. In Section 12, we will modify our definition of $\hat{e}(v,\Delta)$ for root nodes of the flow-return forest.

11.2. **A new flow invariant.** In order to ensure that Property S3 is satisfied for special nodes, we will require that an even stronger (more restrictive) invariant is satisfied.

Invariant 2. *If $r_{ij} < r_{ji}$, then $r_{ij} \equiv 0 \left(\text{mod } \frac{\Delta}{k}\right)$.*

We will adjust our rules for pushing later in this section. The modified rules ensure that Invariant 2 is satisfied at all iterations. If (i, j) is an anti-abundant arc, then $r_{ij} < r_{ji}$. By Invariant 2, $r_{ij} \equiv 0 \left(\text{mod } \frac{\Delta}{k}\right)$. In addition, the following lemma is true.

Lemma 11.1. *Suppose that node v is special throughout the Δ-scaling phase. Then for each preflow y during the phase, $0 \leq Mod\left(\hat{e}_y(v), \frac{\Delta}{k}\right) \leq 3\epsilon^4\Delta$.*

Proof. Let x denote the initial preflow of the phase, and let y be any other preflow during the phase. Then

(1) $\hat{e}_y(v) = e_y(v) + 1.5\epsilon^4\Delta$.
(2) $e_y(v) = e_x(v) - e_{x-y}(v)$.
(3) $e_x(v) \equiv \text{IMB}(v, \Delta)(\text{mod } \frac{\Delta}{k})$.
(4) $-\epsilon^4\Delta \leq \text{IMB}(v, \Delta) \leq \epsilon^4\Delta$ (because v is special).

Adding (1) to (4) and combining terms yields the following:

$$\epsilon^4\Delta \leq \text{Mod}\left(\hat{e}_y(v) + e_{x-y}(v) + .5\epsilon^4\Delta, \frac{\Delta}{k}\right) \leq 3\epsilon^4\Delta.$$

.

Lemma 10.2 implies that

$$0 \leq \text{Mod}\left(e_{x-y}(v) + .5\epsilon^4\Delta, \frac{\Delta}{k}\right) \leq \epsilon^4\Delta.$$

Combining the above inequalities completes the proof. $\qquad\square$

Using the above notation, we observe the following, which we state as a lemma.

Lemma 11.2. *Suppose that v is normal in the Δ-scaling phase as well as in the previous scaling phase. Then $\hat{e}(v, \Delta) \geq (k - 1)\epsilon\Delta$ at the beginning of the Δ-scaling phase.*

Proof. At the end of the $k\Delta$-scaling phase and also at the beginning of the Δ-scaling phase, $e(v) \geq \epsilon(k\Delta)$. Therefore, at the beginning of the Δ-scaling phase, $\hat{e}(v, \Delta) = e(v, \Delta) - \epsilon\Delta \geq (k - 1)\epsilon\Delta$. $\qquad\square$

11.3. **Initialization of the Enhanced LMES Algorithm.** The Initialization procedure of the Enhanced LMES Algorithm finds an initial preflow satisfying the following.

(1) All arcs in $A^+(s)$ are saturated.
(2) $d(s) = n$; for $i \neq s$, $d(i) = 0$.
(3) For any arc (i, j) with $r_{ij} < r_{ji}$, $r_{ij} \equiv 0 \left(\text{mod } \frac{\Delta}{k}\right)$.
(4) For all $v \in N$, $\hat{e}(v) \geq 0$.

Initialize-Enhanced-LMES
01. **begin**
02.   **for each** arc $(i, j) \in A$, $x_{ij} := 0$;
03.   **for each** node $i \in N$, $d(i) := 0$;
04.   **for each** arc $(s, i) \in A^+(s)$ send r_{si} units of flow in (s, i);
05. $d(s) := n$;
06. $\Delta := \max\{e(i) : i \in N\}$;
07.   **for each** arc (i, j) with $r_{ij} < r_{ji}$, send $\text{Mod}\left(r_{ij}, \frac{\Delta}{k}\right)$ units of flow in (i, j);
08.   **for each** node $i \in N\backslash\{s, t\}$ with $e(i) < \epsilon\Delta$, send $\frac{\Delta}{k}$ units of flow in arc (t, i);
09. **end**

Steps 07 and 08 guarantee that invariant 2 is satisfied. Step 08 guarantees that each node $v \neq s$ or t is normal and $\hat{e}(v) \geq 0$. Step 08 accomplishes this by permitting flows in the arcs directed out of node t.

11.4. Large and medium excess nodes, and a new push procedure.

The Enhanced LMES Algorithm bases the definition of large and medium excess nodes using $\hat{e}$ rather than e. Node v has *large excess* at the Δ-scaling phase if $\hat{e}(v, \Delta) \geq \frac{\Delta}{2}$. Node v has *medium excess* at the Δ-scaling phase if $\frac{\Delta}{k} \leq \hat{e}(v, \Delta) < \frac{\Delta}{2}$.

During the Δ-scaling phase, LargeSet and MediumSet are the sets of large and medium excess nodes. To identify MediumSet and LargeSet, one needs to evaluate $\hat{e}$, which requires that one needs to know whether a node is special, which requires that one can identify when arcs are anti-abundant. We discuss efficient implementation of these terms in Subsection B in the appendix.

The rule for selecting active nodes is the same for the Enhanced LMES Algorithm as for the LMES Algorithm with the exception that LargeSet and MediumSet are defined in terms of $\hat{e}$ rather than e.

Enhanced-LMES-Select-Node

    Applicability: there is a node with medium or large excess.
    Action: if LargeSet $\neq \emptyset$, then let $i = \operatorname{argmin}\{d(j) : j \in \text{LargeSet}\}$;
        else, $i = \operatorname{argmax}\{d(j) : j \in \text{MediumSet}\}$.

As before, the phase ends when there are no medium or large excess nodes.

The procedure for pushing flow in an arc is given below. It ensures that Invariant 2 is always satisfied. The procedure is called when node i is selected for pushing and node i is not a node of the flow-return forest. If i is a node of the flow-return forest, then the procedure *FRF-Push(i)* is called instead, as described in Section 12.

Enhanced-LMES-Push(i, j, Δ)

Applicability: node i is active, arc (i, j) is admissible, and node i is not a node of the flow-return forest.

Action: δ units of flow is sent in arc (i, j) so as to satisfy Invariant 2.

01. **begin**
02.   **if** $\hat{e}(i) \geq \frac{\Delta}{2}$, **then** $D := \frac{\Delta}{2}$;
03.   **else** $D := \hat{e}(i) - \text{Mod}\left(\hat{e}(i), \frac{\Delta}{k}\right)$;
04.   **if** $r_{ij} < D$, **then** $\delta := r_{ij}$;
05.   **else if** $r_{ij} - r_{ji} \geq 2D$, **then** $\delta := D$;
06.   **else if** $r_{ij} < r_{ji}$, **then** $\delta := D$;
07.   **else if** $0 \leq r_{ij} - r_{ji} < 2D$, **then** $\delta := \frac{(r_{ij} - r_{ji})}{2}$;
08.   **else if** $r_{ij} = r_{ji}$, **then** $\delta := D - \text{Mod}\left(r_{ij}, \frac{\Delta}{k}\right)$
09.   send δ units of flow in arc (i, j);
10. **end**

In Steps 02 and 03, D is chosen to be the largest integral multiple of $\frac{\Delta}{k}$ that is at most $\min\{\hat{e}(i), \frac{\Delta}{2}\}$.

Suppose that Invariant 2 is true prior to running *Enhanced-LMES-Push*(i, j, Δ). We claim that it will remain true subsequent to the push. Let r'_{ij} denote the residual capacity before the push. Let r_{ij} denote the residual capacity of (i, j) after the push.

  (1) If δ is chosen as in Step 04, then $r_{ij} = 0$.
  (2) If δ is chosen as in Step 05, then $r'_{ji} < r_{ji} < r_{ij} < r'_{ij}$. In this case, $r'_{ji} \equiv 0 \pmod{\frac{\Delta}{k}}$, and $r_{ji} \equiv 0 \pmod{\frac{\Delta}{k}}$.

(3) If δ is chosen as in Step 06, then $r'_{ij} \equiv 0 \pmod{\frac{\Delta}{k}}$, and $r_{ij} \equiv 0 \pmod{\frac{\Delta}{k}}$.

(4) If δ is chosen as in Step 07, then $r_{ij} = r_{ji}$.

(5) If δ is chosen as in Step 08, then $r_{ij} < r'_{ij} = r'_{ji} < r_{ji}$ and $r_{ij} \equiv 0 \pmod{\frac{\Delta}{k}}$.

A push of δ units is called *large* at the Δ-scaling phase if $\delta = \frac{\Delta}{2}$. The push is called *medium* if it is not large, and $\delta = D$; that is, δ is selected in line 05 or line 06. In this case, δ is chosen the largest integral multiple of $\frac{\Delta}{k}$ that is at most $\hat{e}(i)$. Subsequent to the push, $\hat{e}(i) < \frac{\Delta}{k}$.

If δ is chosen in line 04 or 07 or 08, then the push is neither large nor medium. For fixed arc (i, j) and for fixed value of $d(i)$, each of these choices for δ can occur at most once. Since each node is relabeled at most n times, the following lemma is true.

Lemma 11.3. *The number of pushes in the Enhanced LMES Algorithm that are neither large nor medium is $O(nm)$.*

12. The flow-return forest

When a node v becomes newly violating, then $\hat{e}(v, \Delta)$ becomes negative, and node v is added to a new data structure that we call the *flow-return forest* (FRF). Within $2Q$ additional scaling phases, flow is sent to node v, after which it is no longer violating. In this section, we describe the FRF. But first, we bound the number of occurrences of nodes that become newly violating.

Lemma 12.1. *The total number of occurrences of newly violating nodes is $O(m)$.*

Proof. A node v can become violating if v is a merged node that is newly created by contracting an abundant cycle, or v stops being special. The number of occurrences of newly merged nodes is less than n. We now consider a violating node v that was special at the previous scaling phase. Either (1) an arc incident to v becomes newly medium, or (2) $|\text{IMB}(v, \Delta)| > \epsilon^4 \Delta$. There are at most $2m$ nodes that can become newly violating because of newly medium arcs. (Each arc becomes medium at most once. Each newly medium arc might cause both endpoints to become non-special.)

Suppose instead that v becomes newly violating at the Δ-scaling phase because of (2) and assume that v is not incident to any medium arcs. Then $|\text{IMB}(v, \Delta)| > \epsilon^4 \Delta$. We claim that v cannot become special again. To see why, let Δ' be a scaling parameter at some phase after the Δ-scaling phase. Let y denote the preflow at the beginning of the Δ'-scaling phase. By Lemma 10.2, $|\text{IMB}(v, y, \Delta) - \text{IMB}(v, \Delta)| < .5\epsilon^4 \Delta$. It follows that $\text{IMB}(v, y, \Delta) > .5\epsilon^4 \Delta \geq 2\epsilon^4 \Delta'$, and node v is not special at the Δ'-scaling phase.

Thus, the total number of times that (2) can occur without (1) is less than n. This completes the proof. □

12.1. **Nodes and arcs of the FRF.** Let F denote the flow-return forest. For each node $v \in F$, let $Root(v, F)$ denote the root of node v. We let $Roots(F)$ denote the set of root nodes of F.

An arc (i, j) is called *FRF-eligible* if (i, j) is abundant and $d(j) \leq d(i) + 1$. We require that every arc of the flow-return forest is FRF-eligible. We also require that for each node $v \in F$, the path from $Root(v, F)$ to node v is an a directed path. Since each arc of F is FRF-eligible, the path from $Root(v, F)$ to node v is abundant, and the reversal of the path satisfies the distance validity conditions.

In addition, every leaf node of F is a violating node. When a leaf node of F becomes non-violating, it is deleted from F.

For each node $i \in F$, let $Desc(i, F)$ denote the descendants of i in F.

12.2. **NeededFlow( ) and Reserve( ).** The algorithm maintains an array called $NeededFlow(\,)$ and another array called $Reserve(\,)$. When a newly violating node v is added to F at the Δ-scaling phase, $NeededFlow(v)$ is set to $\frac{\epsilon^2 \Delta}{k}$. For each non-violating node $i \in F$, $NeededFlow(i) = 0$. After $2Q$ additional scaling phases, the scaling parameter is $\Delta' = \epsilon^2 \Delta$. If node v is still violating at the Δ'-scaling phase, then $\frac{\Delta'}{k}$ units of flow are sent on the path in F from $Root(v, F)$ to node v. Subsequently, v is non-violating.

In order to ensure that $Root(v, F)$ has sufficient excess, the algorithm maintains an array $Reserve(\,)$ that satisfies the following two conditions. For each node $i \in Roots(F)$:

$$(12.2) \qquad Reserve(i) = \sum_{v \in Desc(i,F)} NeededFlow(v).$$

$$(12.3) \qquad \text{If } i \in Roots(F), \text{ then } e(i) \geq Reserve(i) + \epsilon\Delta.$$

Invariant 12.2 is easily maintained as follows: if node v is added to F at the Δ-scaling phase then, $\text{NeededFlow}(v) := \frac{\epsilon^2\Delta}{k}$, and $Reserve(Root(v, F))$ is incremented by $\frac{\epsilon^2\Delta}{k}$. Immediately prior to deleting a node v from F, $Reserve(Root(v, F))$ is decremented by $NeededFlow(v)$, after which $NeededFlow(v)$ is set to 0.

We modify our definition of of $\hat{e}$ for nodes of $Roots(F)$ as follows: If $i \in Roots(F)$, then $\hat{e}(i) = e(i) - Reserve(i) - \epsilon\Delta$. Then 12.3 is equivalent to:

$$(12.4) \qquad \text{If } i \in Roots(F), \text{ then } \hat{e}(i) \geq 0.$$

We still consider all root nodes to be normal.

For every newly violating node v for which $i = Root(v, F)$, $Reserve(i)$ is incremented by $\frac{\epsilon^2\Delta}{k}$, which causes $\hat{e}(i)$ to be decremented by the same amount. The maximum increase in reserve for node i (and corresponding decrease of $\hat{e}(i)$) at the beginning of the Δ-scaling phase is $\frac{n\epsilon^2\Delta}{k}$, which is less than $\epsilon\Delta$. By Lemma 11.2, $\hat{e}(i) \geq (k-1)\epsilon\Delta$ at the beginning of the scaling phase. Therefore, $\hat{e}(i) \geq (k-2)\epsilon\Delta$ after all additions to F at the Δ-scaling phase.

In the next subsection, we prove that it is always possible to add a newly violating node to F by appending a path of FRF-eligible arcs to F.

12.3. **Adding nodes to the FRF.** For each newly violating node v in the Δ-scaling phase, the algorithm determines an FRF-eligible path W from w to v where $w = t$ or w is normal. Then F is replaced by $F \cup W$. The following lemma guarantees that such a path W exists.

Lemma 12.5. (The FRF-eligible Path Lemma.) *Suppose that v is a newly violating node at the start of the Δ-scaling phase. Then there is a directed path of FRF-eligible arcs from some node w to node v, where $w = t$ or else w is a normal node.*

Proof. Let $S = \{i \in N(\Delta) : \text{there is an FRF-eligible path from } i \text{ to } v\}$. We want to show that $t \in S$ or there is a normal node in S. We suppose that this is not the case, and we will derive a contradiction. Since S contains no normal nodes, it contains no root nodes of F. Accordingly, S cannot contain any node of F. It follows that S consists of special nodes and newly violating nodes. (Any violating node that is not newly violating is in F.) Thus, for every node $i \in S$, $|e(i)| < \epsilon^3\Delta$.

In the following, we say that a push in (i, j) is *special* at the Δ'-scaling phase if node i is special at the Δ'-scaling phase and if $e(i) < \epsilon\Delta'$ after the push.

Let (i_1, i_2) be the most recent push from a node of S, and suppose that the push took place in the Δ'-scaling phase. Let $\bar{e}(\,)$ denote the vector of excesses immediately subsequent to the push, and let $e(\,)$ denote the vector of excesses at the beginning of the Δ-scaling phase.

The push in (i_1, i_2) was special since $\bar{e}(i_1, \Delta') \leq e(i_1, \Delta) < \epsilon^3\Delta < \epsilon^3\Delta'$. Moreover, the amount of flow pushed in (i_1, i_2) was at least $\frac{\Delta'}{k}$. After the push, node i_2 became a medium or large excess node. (This is true regardless of whether i_2 was special or normal.) Therefore, there was a subsequent push from i_2 at the Δ'-scaling phase.

Since the push in (i_1, i_2) was special, arc (i_1, i_2) was large during the Δ'-scaling phase. (It could not have been small, and there were no medium arcs incident to i_1.)

The following are all true:
  (1) $|\bar{e}(i_1)| < 1.5\epsilon^4\Delta'$.
  (2) $\bar{e}(i_2) > \frac{\Delta'}{k} - \epsilon\Delta'$.

(3) There was no medium or large push into node i_1 subsequent to the push in (i_1, i_2). (There was no subsequent push from node i_1 and $e(i_1) < \epsilon\Delta$.)
(4) Node $i_2 \notin S$. (The last push from a node in S was in (i_1, i_2), and there was a subsequent push from i_2.)
(5) Arc (i_1, i_2) is abundant. (Otherwise, arc (i_1, i_2) is anti-abundant, which implies that arc (i_2, i_1) is FRF-eligible, which implies that $i_2 \in S$.)

We now consider the sum of the imbalances of nodes of S.

$$\text{IMB}(S, \Delta) = \sum_{i \in S} \text{IMB}(i, \Delta) = \sum_{i \in S} e(S) + \sum_{(i,w) \in Anti(\Delta): i \in S, w \notin S} r_{iw} - \sum_{(w,i) \in Anti(\Delta): i \in S, w \notin S} r_{wi}.$$

Note that $|\text{IMB}(i, \Delta)| < \epsilon^3\Delta$ for each $i \in S$, and $|e(i)| < 1.5\epsilon\Delta$ for each $i \in S$. Therefore, $|\text{IMB}(S, \Delta) - e(S)| < 2|S|\epsilon\Delta \leq .5\Delta$.

By (5) above, (i_2, i_1) is anti-abundant and directed into S. There was no push in (i_2, i_1) after the push in (i_1, i_2). Therefore, $r_{i_2, i_1} \geq \frac{\Delta'}{k} \geq \Delta$. Since $|\text{IMB}(S, \Delta) - e(S)| < .5\Delta$, there must be some anti-abundant arc (i_3, i_4) directed out of S with $r_{i_3, i_4} > \frac{\Delta}{2m}$. It follows that (i_4, i_3) is FRF-eligible. But this also implies that $i_4 \in S$, contradicting that (i_3, i_4) is directed out of S. In all cases we have derived a contradiction. Thus S contains node t or a normal node. This completes the proof. $\qquad\square$

12.4. Properties of nodes of the flow-return forest.
The following are properties of the nodes of the flow-return forest. After each property, we indicate why it is true or where we will show it to be true.

FRF 1. Every leaf node of F is violating. (Non-violating leaf nodes are deleted from F.)
FRF 2. Every node of $Roots(F)\backslash\{t\}$ is normal. (The procedure *FRF-Add* only adds normal root nodes.)
FRF 3. If v is added to F at the Δ-scaling phase, then $FlowNeeded(v) = \frac{\epsilon^2\Delta}{k}$. (Specified in the procedure *FRF-Add*.)
FRF 4. If $i \in Roots(F)$, then $Reserve(i) = \sum_{v \in Desc(i,F)} FlowNeeded(v)$. (The equality is preserved when running the procedure *FRF-Add*.)
FRF 5. If $i \in Roots(F)$ at the Δ-scaling phase, then $\hat{e}(i) = e(i) - \epsilon\Delta - Reserve(i) \geq 0$. (See Lemma 12.5.)
FRF 6. For each normal node $i \in F$, $\hat{e}(i) \leq \Delta + (k-1)\epsilon\Delta$. (See Lemma 14.5.)
FRF 7. If an active medium or large excess node i of F is selected for pushing at the Δ-scaling phase, then $\frac{\Delta}{k}$ units of flow are sent from i to a violating node $v \in Desc(i, F)$. (Specified in the procedure *FRF-Push*.)
FRF 8. The number of scaling phases in which the flow-return forest is non-empty is $O(m \log_k n)$. (See Corollary 12.8.)

12.5. Operations on the FRF.
In this subsection, we describe the operations that are performed on the FRF. We also indicate how the operations are implemented and provide their total running time. We describe all of the procedures in terms of their "Applicability" and "Action." The procedures *FRF-Add* and *FRF-Delete* have additional details that are provided in their pseudo-codes.

Procedure FRF-Initialize(F)
Applicability: Initialization of the algorithm.
Action: This procedure sets F to be the null forest.

Procedure FRF-Push(v, F, Δ)
Applicability: Node $v \in F$ has medium or large excess and v is not a leaf node of F.
Action: The procedure finds a violating leaf node $i \in Desc(v, F)$. Then $\frac{\Delta}{k}$ units of flow are sent
in the path from v to i in F.

Procedure FRF-Pull(v, F, Δ)
Applicability: Node v is violating at the beginning of the Δ-scaling phase and one of the follow-
ing two conditions is satisfied. (1) Node v is still violating and $FlowNeeded(v) = \frac{\Delta}{k}$, or (2) v is
a leaf node of F that is no longer violating and about to be deleted, and $\hat{e}(Root(v, F), \Delta) \geq \frac{\Delta}{k}$.
Action: $\frac{\Delta}{k}$ units of flow are sent on the path in F from $Root(v, F)$ to node v.

Procedure FRF-Add(v, F, Δ)
Applicability: Node v is newly violating at the Δ-scaling phase.
Action: This procedure calls Procedure $FRF\text{-}Reverse\text{-}DFS(v, F, \Delta)$, which either produces an
FRF-eligible directed cycle W or else it produces an FRF-eligible path W. In the former case,
the cycle W is contracted and $FRF\text{-}Add(v, F, \Delta)$ is executed on the contracted graph. In the
latter case, F is replaced by $F \cup W$, and the arrays $NeededFlow(\,)$ and $Reserve(\,)$ are updated.

W := FRF-Reverse-DFS(v, F, Δ)
Applicability: $FRF\text{-}Add(v, F, \Delta)$ has just been called.
Action: Let $Y = F \cup \{t\} \cup \{i \notin F : i \text{ is normal.}\}$. The algorithm finds an FRF-eligible cycle W
such that $W \cap F = \emptyset$ or it finds an FRF-eligible path W from a node $w \in Y$ to node v such
that $(W \setminus w) \cap F = \emptyset$.

Procedure FRF-Delete(v, F, Δ)
Applicability: Node v is a leaf node of F and $\hat{e}(v, \Delta) \geq 0$.
Action: $FlowNeeded(v)$ and $Reserve(Root(v, F))$ are both updated. If $\hat{e}(Root(v, F), \Delta) \geq \frac{\Delta}{k}$
and if v was a violating node at the beginning of the Δ-scaling phase, then $\frac{\Delta}{k}$ units of flow are
sent from $Root(v, F)$ to v. Node v is then deleted from F.

We now provide additional details for the procedures *FRF-Add*, *FRF-Reverse-DFS*, and *FRF-
delete* .

Procedure FRF-Add(v, F, Δ)
01. **begin**
02. $W := FRF\text{-}Reverse\text{-}DFS(v, F, \Delta)$;
03.   **if** W is a cycle, **then do**
04.   contract the cycle W;
05.   run $FRF\text{-}Add(v, F, \Delta)$ on the contracted network;
06.   **else continue**
07.   let j be the first node of W;
08.   **if** $j \notin F$, **then** $w := j$ and $Reserve(w) := 0$; **else** $w := Root(j, F)$;
09. $F := F \cup W$;
10.   **for every** node $i \in W$ **do**
11. $Root(i, F) := w$;
12.   **if** i is newly violating, **then do**
13. $NeededFlow(i) := \epsilon^2 \Delta$;
14. $Reserve(w) := Reserve(w) + \epsilon^2 \Delta$;
15. **end**

In order to run *FRF-Reverse-DFS*, one needs to be able to identify FRF-eligible arcs for the
depth first search. For every node i, we maintain an array called *FRF-eligible*(i), which consists
of all FRF-eligible arcs directed into node i. When procedure *FRF-Reverse-DFS* is run, it scans

at most the first arc of FRF-$eligible(i)$ for each i. Each call of FRF-$Reverse$-DFS takes $O(n)$ time.

The algorithm also maintains an array called $Abundant(i)$, which consists of the Abundant arcs directed into node i. The array $Abundant(\;)$ is needed so that FRF-$eligible(\;)$ can be efficiently maintained. $Abundant(\;)$ is updated whenever an arc becomes large, and whenever an anti-abundant arc becomes bi-abundant. (See Section B of the appendix for more details.) FRF-$eligible(\;)$ is updated whenever $Abundant(\;)$ is updated and whenever a node is relabeled. The total time to update $Abundant(\;)$ and FRF-$eligible(\;)$ is proportional to the number of pushes plus the number of scaling phases.

We now consider the procedure FRF-$delete$. The unusual feature of this procedure is the sending of additional flow to nodes prior to their being deleted from F. Let $V(\Delta)$ denotes the subset of nodes v of $N(\Delta)$ such that $0 < NeededFlow(v) < \Delta/k$. That is, node v is violating at the beginning of the Δ-scaling phase, and v is not scheduled for a "pull" during the Δ-scaling phase.

Procedure FRF-Delete(v, F, Δ)
01. **begin**
02. $w := Root(v, F)$;
03. $\delta := \text{NeededFlow}(v)$;
04. $\text{NeededFlow}(v) := 0$;
05. $Reserve(w) := Reserve(w) - \delta$;
06.   **if** $w \in V(\Delta)$ and $\hat{e}(w, \Delta) \geq \frac{\Delta}{k}$, **then** call Procedure FRF-$Pull(v, F, \Delta)$;
07.   delete node v from F;
08. **end**

Suppose that a leaf node v is deleted during the Δ-scaling phase. Let $i = Root(v, F)$, and suppose that $\hat{e}(i) \geq \frac{\Delta}{k}$ immediately prior to the deletion of node v from F. We claim that under these circumstances, the deletion of v from F does not increase the modified excess of node i. We consider three cases corresponding to different values of $NeededFlow(v)$ at the beginning of the Δ-scaling phase: (1) $NeededFlow(v) = 0$, (2) $NeededFlow(v) = \frac{\Delta}{k}$, and (3) $0 < NeededFlow(v) < \frac{\Delta}{k}$.

In Case (1), $Reserve(i)$ is not altered by the deletion of v, and thus the modified excess of node i is not changed. In Case (2), FRF-$Pull(v, F)$ is called at the beginning of the Δ-scaling phase. This leads to a decrease in the modified excess of node i by $\frac{\Delta}{k}$. When node v is later deleted, then $Reserve(i)$ is decreased by $\frac{\Delta}{k}$ in Step 05, thus increasing the modified excess of node i by $\frac{\Delta}{k}$. The increase and decrease cancel each other. In Case (3), the modified excess of i will increase by less than $\frac{\Delta}{k}$ in Step 05. In Step 06, $\frac{\Delta}{k}$ units of flow will be sent from i to v, which reduces the modified excess of i by $\frac{\Delta}{k}$. The decrease exceeds the increase. Thus, in all three cases, the deletion of node v does not lead to an increase in the modified excess of node i.

Theorem 12.6. *Consider the procedures FRF-ADD, FRF-Reverse-DFS, FRF-Delete, FRF-Push, and FRF-Pull. The bounds given in Table 1 are correct. The total time taken by all of these procedures is $O(nm)$.*

Proof. In the proof of Lemma 12.1, we showed that there are at most $2m + n$ newly violating nodes added to F over all scaling phases.

FRF-ADD is run when a node becomes newly violating and when FRF-$Reverse$-DFS creates a contraction. There are at most n newly violating nodes in a scaling phase, and at most $2m + n$ violating nodes over all scaling phases. There are fewer than n contractions. The time for each execution of FRF-ADD is $O(n)$.

FRF-$Reverse$-DFS is run whenever FRF-ADD is run. Each time that it is run takes $O(n)$ time.

Operation	Max number of times run per scaling phase	Running time	Max total number of times run	Total running time
FRF-Add	*n*	$O(n)$	$2m + n$	$O(nm)$
FRF-Reverse-*DFS*	*n*	$O(n)$	$2m + 2n$	$O(nm)$
FRF-Delete	*2n*	$O(1)$	$nm + n^2$	$O(nm)$
FRF-Push	*n*	$O(n)$	$2m + n$	$O(nm)$
FRF-Pull	*2n*	$O(n)$	$4m + 2n$	$O(nm)$

TABLE 1. FRF-Procedures.

When *FRF-Delete* is called, the running time is $O(1)$, ignoring the time for *FRF-Pull*, which is accounted for elsewhere. The number of times that *FRF-Delete* is called is the total number of nodes that are added to the flow-return forest, which is at most $n(m + n)$.

Whenever *FRF-Push* is called, a violating node becomes non-violating. This can happen at most $2m + n$ times in total. The running time of procedure *FRF-Push* is $O(n)$.

For each violating node v, *FRF-Pull* can be called at most twice. It can be called once if Needed-Flow$(v) = \frac{\Delta}{k}$. It can be called once during *FRF-Delete*(v, F) if $0 <$ Needed-Flow$(v) < \frac{\Delta}{k}$. Thus, *FRF-Pull* can be called at most $4m + 2n$ times. The running time of procedure *FRF-Pull* is $O(n)$. □

12.6. **The number of scaling phases in which F is non-empty.** In this subsection, we prove that there are $O(m \log_k n)$ scaling phases in which F is non-empty. We also show that a violating node becomes non-violating prior to having excess as low as $-\epsilon\Delta$.

Lemma 12.7. *Each node is violating for at most $2Q + 1$ phases of the Enhanced LMES Algorithm.*

Proof. Suppose that node v is newly violating in the Δ'-scaling phase. Then, $FlowNeeded(v) = \frac{\epsilon^2\Delta'}{k}$. We will show that v becomes non-violating within $2Q$ additional scaling phases.

Now consider Line 03 of the procedure *ScalingPhase*. If v is still violating at the Δ-scaling phase and if $FlowNeeded(v) = \frac{\Delta}{k}$, then $\Delta = \epsilon^2\Delta'$. There has been exactly $2Q$ phases since node v was added to F. Then the algorithm runs *FRF-Pull*(v, F, Δ), after which v is no longer violating. □

Corollary 12.8. *The number of scaling phases in which $F \neq \emptyset$ is $O(m \log_k n)$.*

Proof. If $F \neq \emptyset$, then F has a violating node. By Lemma 12.1, there are $O(m)$ newly violating nodes over all scaling phases. By Lemma 12.7, each violating node is in F for at most $2Q + 1$ scaling phases. Thus the number of scaling phases in which $F \neq \emptyset$ is $O(m \log_k n)$. □

Corollary 12.9. *Suppose that node v is violating at the beginning of the Δ-scaling phase. Then $-\epsilon\Delta < e(v, \Delta) < \epsilon\Delta$.*

Proof. For node v to be violating, $e(v, \Delta) < \epsilon\Delta$. We need to prove that $e(v, \Delta) > -\epsilon\Delta$. Suppose that v became newly violating at the Δ'-scaling phase when the preflow was x. Suppose v becomes non-violating at the Δ-scaling phase when the preflow is y. Then $e_y(v) \geq e_x(v)$ since there is no push from v in the iterations leading up to preflow y.

By Lemma 12.7, $\Delta \geq \epsilon^2\Delta'$. At the end of the $(k\Delta')$-scaling phase (the phase prior to the Δ'-scaling phase), node v was special. Thus $e_y(v) \geq e_x(v) \geq -1.5\epsilon^4(k\Delta') > -\epsilon^3\Delta' \geq -\epsilon\Delta$. □

13. The Enhanced LMES max flow algorithm

In this section, we give the pseudo-code for the remaining procedures of the Enhanced LMES Algorithm. We also prove the correctness of the Enhanced LMES Algorithm and prove that the running time is $O(kn^2 + nm \log_k n)$.

13.1. **The Pseudo-code.** We have already provided the pseudo-code for the Enhanced-LMES Algorithm including the main procedure, which is called *ScalingPhase*. Here, we provide the pseudo-code the remaining procedures: *Enhanced-LMES-Push/Relabel*, *FRF-Recursive-Delete-and-Merge*, and *Get-Next-Scaling-Parameter*.

Enhanced-LMES-Push/Relabel(v, Δ)
01. **begin**
02.   **if** there is no admissible arc in $A^+(v)$, **then** $d(v) := d(v) + 1$;
03.   **else** select an admissible arc $(i, j) \in A^+(v)$;
04.   *Enhanced-LMES-Push*(i, j, Δ)
05. **end**

We next describe Procedure *FRF-Recursive-Delete-and-Merge*. An arc (i, j) with distinct endpoints is called *mergeable* if (i, j) is doubly abundant and at most one endpoint of (i, j) is in F. The Enhanced LMES Algorithm contracts mergeable bi-abundant arcs. It does not contract a bi-abundant arc if both endpoints of the arc are in F. (This restriction ensures that F remains a forest.) If both endpoints are in F, then the algorithm defers contraction until one or both of the endpoints are deleted from F.

When a node v is deleted from F, the algorithm checks whether v is incident to a bi-abundant arc. If so, the arc is contracted. When v is deleted from F, the algorithm also checks the parent of v in F. If the parent node has become a leaf node that is non-violating, then the procedure deletes the parent node.

Procedure FRF-Recursive-Delete-and-Merge(F, Δ)
01. **begin**
02.   **while** there is a mergeable arc or a non-violating leaf node of F **do**
03.   **if** there is a non-violating leaf node v of F **then** *FRF-Delete*(v, F, Δ)
04.   **else** contract a mergeable arc;
05.   **endwhile**
06. **end**

We now describe the procedure that chooses the subsequent scaling parameter. In general, the scaling parameter after the Δ-scaling phase is $\frac{\Delta}{k}$. However, this could lead to a long number of scaling phases that are useless; that is, scaling phases with no push and such that $F = \emptyset$. The procedure *Get-Next-Scaling-Parameter* avoids long sequences of useless phases, while also maintaining Invariant 2.

Suppose that the current scaling parameter is Δ. Suppose further that the next $2Q + 1$ scaling phases are going to be useless. Then the scaling parameter is set to a value Γ so that the Γ-scaling phase is useful and so that $\Gamma \leq \frac{\Delta}{Mk}$.

Procedure Get-Next-Scaling-Parameter(Δ)
01. **begin**
02. $\Gamma_1 := \max\{e(v) : e(v) > 0\}$;
03. $\Gamma_2 := \max\{-\epsilon^3 e(v) : e(v) < 0\}$;
04. $\Gamma := \max\{\Gamma_1, \Gamma_2\}$;
05.   **if** $\Gamma < \frac{\Delta}{kM}$, **then** $\Delta := \Gamma$; **else,** $\Delta := \frac{\Delta}{k}$;
06. **end**

Suppose that Δ is replaced by Γ in Step 05. We claim that Invariant 2 is satisfied at the beginning of the Γ-scaling phase . Suppose that $r_{ij} > 0$ and $r_{ji} > 0$ at the end of the Δ-scaling phase. Then it is also true that $r_{ij} \geq \frac{\Delta}{k}$, and $r_{ji} \geq \frac{\Delta}{k}$. Then $r_{ij} \geq M\Gamma$ and $r_{ji} \geq M\Gamma$, and (i,j) is bi-abundant at the Γ-scaling phase. This choice of Γ ensures that Invariant 2 remains satisfied.

14. The Enhanced LMES Algorithm runs in $O(kn^2 + nm\log_k n)$ time

14.1. The main theorem and its supporting lemmas.
In this subsection, we state the main theorem as well as the lemmas that are used in establishing the theorem. In subsequent subsections, we prove the lemmas and the main theorem.

Theorem 14.10. *The Enhanced LMES Algorithm finds the maximum flow in $O(kn^2+nm\log_k n)$ time. Let k be the smallest power of 2 that is at least $\max\{\log\log n, \frac{m}{n}, 4\}$. Then the running time is*

$$O\left(\frac{nm\log n}{\log\log n + \log\frac{m}{n}}\right).$$

Lemma 14.1. *The total number of occurrences of normal and violating nodes in the Enhanced LMES Algorithm is $O(m\log_k n)$.*

Corollary 14.3. *The number of scaling phases of the Enhanced LMES Algorithm is $O(m\log_k n)$.*

Corollary 14.4. *The Enhanced LMES Algorithm terminates with a maximum flow.*

Lemma 14.5. *If $v \in N(\Delta)$, then throughout the Δ-scaling phase of the Enhanced LMES Algorithm, $\hat{e}(v) < (1 + (k-1)\epsilon)\Delta$.*

Lemma 14.6. *The total amount of flow pushed in the Δ-scaling phase of the Enhanced LMES Algorithm is at most $5n^2\Delta$.*

Corollary 14.7. *If (i,j) is abundant at the beginning of the Δ-scaling phase of the Enhanced LMES, then for every scaling parameter $\Delta' \leq \Delta$ the following are true.*
  (1) (i,j) is abundant in the Δ'-scaling phase and
  (2) $r_{ij} > 0$ throughout the Δ'-scaling phase.

Lemma 14.8. *The total number of medium and large pushes in the Enhanced LMES Algorithm is $O(kn^2 + nm\log_k n)$.*

14.2. The number of normal and violating nodes.

Lemma 14.1. *The total number of occurrences of normal and violating nodes in the Enhanced LMES Algorithm is $O(m\log_k n)$.*

Proof. If a node v is normal, then (by the Contraction Lemma), node v will become incident to a medium arc or a bi-abundant arc within $7Q$ scaling phases. Thus the number of occurrences of normal node that are not incident to a bi-abundant arc is $O(m\log_k n)$.

We now claim that there are $O(m\log_k n)$ occurrences of normal nodes that are incident to a bi-abundant arc. Each of these nodes must be in F; otherwise the bi-abundant arc would be contracted. If $v \in F$ is normal at the Δ-scaling phase, then $e(v) \geq \epsilon\Delta$, and node v will have medium or large excess within Q scaling phases, after which there will be an FRF-push from node v. By Theorem 12.6, the total number of FRF-pushes is $O(m)$. Therefore, there are $O(m\log_k n)$ occurrences of normal nodes that are incident to bi-abundant arcs.

Finally, we consider occurrences of violating nodes, all of which are added to F when they become newly violating. By Lemma 12.1, at most $2m + n$ newly violating nodes were added to

F. After being added to F, a violating node will receive flow within $2Q$ scaling phases, at which point it will become normal. Thus, there are $O(m \log_k n)$ occurrences of violating nodes. $\square$

14.3. **The number of scaling phases.** We say that a scaling phase is *useful* if there is at least one push during the scaling phase.

Lemma 14.2. *The number of useful scaling phases of the Enhanced LMES Algorithm is $O(m \log_k n)$.*

Proof. By Corollary 12.8, the number of scaling phases in which $F \neq \emptyset$ is $O(m \log_k n)$. By Lemma 14.1, the number of phases in which there is a normal node is $O(m \log_k n)$. If there is no normal node, and if $F = \emptyset$, then there is no push during the scaling phase, and the scaling phase is useless. $\square$

Corollary 14.3. *The number of scaling phases of the Enhanced LMES Algorithm is $O(m \log_k n)$.*

Proof. We need to consider scaling phases in which $F = \emptyset$ and no node is normal. This implies that every node must be special except for s and t. We claim that there will be a new normal node or a new violating node within $2Q + 2$ scaling phases. Let $\Gamma = \max\{\Gamma_1, \Gamma_2\}$, as selected in Step 04 of *Get-Next-Scaling-Parameter*. Consider first the case that $\Gamma < \frac{\Delta}{kM}$. In this case, the next scaling parameter is Γ. If $\Gamma = \Gamma_1$, then there will be a normal node at the Γ-scaling phase. If $\Gamma = \Gamma_2$, then there will be violating node at the Γ-scaling phase, after which F becomes non-empty. Therefore, the total number of scaling phases is $O(m \log_k n)$.

Now consider the case that $\Gamma > \frac{\Delta}{kM}$. In this case, we continue to divide scaling parameters by k until there is a normal node or a violating node. This will occur within $2Q + 2$ scaling phases, and which point the scaling parameter is Δ' and $\Delta' < \frac{\Delta}{kM}$.

The number of newly normal nodes is bounded at most n plus the number of newly special nodes, which is $O(m)$. The number of newly violating nodes is at most the number of newly special nodes, which is $O(m)$. Thus, the number of useless phases is $O(m \log_k n)$. $\square$

Corollary 14.4. *The Enhanced LMES Algorithm terminates with a maximum flow.*

Proof. The algorithm must terminate since there are a bounded number of scaling phases. The termination criterion is that there are no nodes with excess, which is the optimality criterion for the generic push-relabel algorithm. The flow is optimum in the contracted graph. Subsequently, the contracted abundant cycles are expanded, resulting in an optimal flow in the original graph. $\square$

14.4. **An upper bound on $\hat{e}(v, \Delta)$.** Let $N(\Delta)$ be the set of nodes (including merged nodes) at the beginning of the Δ-scaling phase. In this section, we show that for all nodes $v \in N(\Delta)$, $\hat{e}(v, \Delta) \leq \Delta + (k-1)\epsilon\Delta$. This upper bound would be obvious except for the impact of contractions on node excesses.

Suppose now that w is a merged node that is created during the Δ-scaling phase by the contraction of an abundant cycle W. When w is created, $e(w) = \sum_{v \in W} e(v)$.

It is possible that $e(w)$ is proportional to $|W|\Delta$, which violates the upper bound on excess that is needed for the complexity argument. The assumption that $e(v) = O(\Delta)$ is needed to bound the increase in potential functions for relabels of nodes. We bypass this difficulty by evaluating potential functions in the Δ-scaling phase by summing over nodes of $N(\Delta)$ and not including the merged nodes obtained by contractions during the Δ-scaling phase.

Suppose that w is created by the merging of one or more abundant cycles during the Δ-scaling phase. Let W denote the nodes of $N(\Delta)$ that were nodes of any of the cycles that were merged into node w.

For a given preflow x, for all $v \in W$, we define $\hat{e}_x(v, \Delta)$ as follows:

(1) If $v \notin Roots$, then $\hat{e}_x(v) = \frac{e_x(w) - \epsilon\Delta}{|W|}$.
(2) If $v \in Roots$, then $\hat{e}_x(v) = \frac{e_x(w) - \epsilon\Delta - Reserve(v)}{|W|}$.

Lemma 14.5. *If $v \in N(\Delta)$, then throughout the Δ-scaling phase of the Enhanced LMES Algorithm, $\hat{e}(v, \Delta) < (1 + (k-1)\epsilon)\Delta$.*

Proof. The operations that can lead to an increase in $\hat{e}(v)$ are the following: (1) a push into node v, (2) the beginning of a scaling phase, and (3) a contraction including node v or a node that v has been merged into.

(1) If there is a push into node v, then $\hat{e}(v, \Delta) < \frac{\Delta}{2}$ prior to the push, and $\hat{e}(v, \Delta) < \Delta$ after the push.

(2) At the end of the previous scaling phase (i.e, the $(k\Delta)$-scaling phase), $\hat{e}(v, k\Delta) = e(v) - \epsilon k\Delta < \frac{(k\Delta)}{k} = \Delta$. At the beginning of the Δ-scaling phase, $\hat{e}(v, \Delta) = e(v) - \epsilon\Delta < \Delta + (k-1)\epsilon\Delta$.

(3) Suppose that node w is a contracted node that is formed in the Δ-scaling phase. Let W denote the set of nodes of $N(\Delta)$ that are contracted into node w. For each node $v \in W$, let $\bar{e}(v, \Delta)$ denote the modified excess prior to contraction, and let $\hat{e}(v, \Delta)$ denote the modified excess after contraction. Assume inductively that for each node $v \in W$, $\bar{e}(v, \Delta) < (1 + (k-1)\epsilon)\Delta$. Then

$$\hat{e}(v, \Delta) < \sum_{v \in W} \frac{\hat{e}(v, \Delta)}{|W|} < (1 + (k-1)\epsilon)\Delta.$$

This completes the proof. □

14.5. **Bounding the total flow sent in a scaling phase.** In this subsection, we bound the total flow in a scaling phase. In order to bound the total flow sent in a scaling phase, we replace the potential function Φ_1 used in Section 5 with a similar potential function $\Phi_3(\Delta)$ defined next. Recall that $N(\Delta)$ is the set of nodes (both original and merged) at the beginning of the Δ-scaling phase. Let $R(\Delta)$ denote the set of root nodes of F at the beginning of the Δ-scaling phase.

$$\Phi_3(\Delta) = \sum_{v \in N(\Delta)\setminus R(\Delta)} \frac{e(v)d(v)}{\Delta} + \sum_{v \in R(\Delta)} \frac{(e(v) - Reserve(v)) \cdot d(v)}{\Delta}.$$

Lemma 14.6. *The total amount of flow pushed in any arc in the Δ-scaling phase of the Enhanced LMES Algorithm is at most $5n^2\Delta$.*

Proof. The flows in the Δ-scaling phase can be (1) pushes in admissible arcs or (2) flow sent on paths of F in the procedures *FRF-Push* and *FRF-Pull*. We refer to the latter as FRF-flows.

We first consider the FRF-flows. In the proof of Theorem 12.6, *FRF-Delete* can be called at most $2n$ times in a scaling phase. Accordingly, *FRF-Push* can be called at most $2n$ times during a scaling phase. And *FRF-Pull* can be called at most $4n$ times during a scaling phase. Thus the total amount of flow sent in *FRF-Push* and *FRF-Pull* per scaling phase is less than $\frac{6n\Delta}{k}$, which is less than $1.5n\Delta$.

We now consider flow sent in admissible arcs. Each push of δ units in an admissible arc reduces $\Phi_3(\Delta)$ by $\frac{\delta}{\Delta}$. Therefore, the total flow in the Δ-scaling phase divided by Δ is at most the total decrease in $\Phi_3(\Delta)$, which is at most the initial value of $\Phi_3(\Delta)$ plus the total increase in $\Phi_3(\Delta)$ during the scaling phase.

The initial value of $\Phi_3(\Delta)$ is less than $(n+1)(n-2)(1 + (k-1)\epsilon) < n^2$. The increase in $\Phi_3(\Delta)$ due to relabels is less than $(n+1)(n-2)(1 + (k-1)\epsilon) < n^2$.

Contractions do not lead to an increase in $\Phi_3(\Delta)$. However, calls of *FRF-Push* and *FRF-Pull* may lead to an increase in $\Phi_3(\Delta)$. In addition, decreases in $Reserve(v)$ for $v \in R(\Delta)$ lead to an increase in $e(v) - Reserve(v)$, and thus to an increase in $\Phi_3(\Delta)$.

When sending flow from node i to node v in F, it is possible that $d(i) < d(v)$, and that $\Phi_3(\Delta)$ will increase. When *FRF-Push* or *FRF-Pull* is called, that maximum possible increase in $\Phi_3(\Delta)$ is at most $\frac{n}{k}$. The total increase in $\Phi_3(\Delta)$ due to FRF flows is at most $6n \cdot \frac{n}{k}$, which is at most $1.5n^2$.

Reserve( ) is decreased only when a previously violating node becomes non-violating and is deleted from F. Each decrease in *Reserve*( ) leads to an increase in $\Phi_3(\Delta)$ by at most $\frac{n}{k}$. This

occurs at most $2n$ times in a scaling phase. The total increase in $\Phi_3(\Delta)$ due to decreases in $Reserve(\,)$ is at most $2n \cdot \frac{n}{k}$, which is at most $.5n^2$.

Thus, the total increase in $\Phi_3(\Delta)$ during the Δ-scaling phase is less than $3n^2$. The initial value of $\Phi_3(\Delta)$ plus the total increase in $\Phi_3(\Delta)$ during the Δ-scaling phase is less than $4n^2$. And the total amount of flow sent in the Δ-scaling phase is pushes in any arc (i,j) is most $4n^2\Delta$. If we include the contribution of FRF-pushes as well, then the total amount of flow sent in any arc (i,j) is most $5n^2\Delta$. □

Corollary 14.7. *If (i,j) is abundant at the beginning of the Δ-scaling phase of the Enhanced LMES, then for every scaling parameter $\Delta' \leq \Delta$ the following are true.*
  (1) *(i,j) is abundant in the Δ'-scaling phase and*
  (2) *$r_{ij} > 0$ throughout the Δ'-scaling phase.*

Proof. If (i,j) is abundant at the beginning of the Δ-scaling phase, then $r_{ij} \geq M\Delta$ at the beginning of the phase. By Lemma 14.6, $r_{ij} \geq (M - 5n^2)\Delta > .5M\Delta$ at the end of the Δ-scaling phase. It follows that (i,j) is abundant at the beginning of the next scaling phase. □

14.6. **The total number of pushes.** In this subsection, we bound the total number of pushes.

Lemma 14.8. *The total number of medium and large pushes in the Enhanced LMES Algorithm is $O(kn^2 + nm\log_k n)$.*

Proof. Let Ψ be the set of distinct scaling parameters. By Lemma 14.3, $|\Psi| = O(m\log_k n)$.

We define Φ_4 as a modification of Φ_2 in Section 5. The parameters P and ℓ are defined in the same manner as for Φ_2 except that P is restricted to nodes of $N(\Delta)$ and does not include any nodes merged at the Δ-scaling phase.

$$\Phi_4(\Delta) := \sum_{j \in P \setminus R(\Delta)} e(j) \cdot \frac{d(j) - \ell + 1}{\Delta} + \sum_{j \in P \cap R(\Delta)} (e(j) - Reserve(j)) \cdot \frac{d(j) - \ell + 1}{\Delta}.$$

For each $\Delta \in \Psi$, let $S(\Delta)$ denote the subset of nodes of $N(\Delta)$ that are special at the beginning of the Δ-scaling phase. The observations in Lemma 14.9 are straightforward and will be used in the proof of Lemma 14.8

Lemma 14.9. *The following statements are all true.*
Fact 1. If $v \in S(\Delta)$, then $e(v) \leq \epsilon^4\Delta$ at the beginning of the Δ-scaling phase.
Fact 2. If $v \in S(\Delta)$, then v remains special throughout the Δ-scaling phase.
Fact 3. If $v \in S(\Delta)$ and if $v \in P$ at an iteration at which ℓ is reduced, then $e(v) \leq 1.5\epsilon^4\Delta$ at that iteration.
Fact 4. $\sum_{\Delta \in \Psi} |N(\Delta) \setminus S(\Delta)| = O(m\log_k n)$.

We now bound the number of large pushes using Φ_3. The number of large pushes is at most 2 times the total increase in Φ_3 over all scaling phases. The value of Φ_3 at the beginning of the Δ-scaling phase is at most

$$\sum_{v \in N(\Delta) \setminus S(\Delta)} \frac{d(v)e(v)}{\Delta} < (n+1) \cdot |N(\Delta) \setminus S(\Delta)|.$$

Thus, the sum of all increases in Φ_3 at the beginning of scaling phases is less than

$$\sum_{\Delta \in \Psi} (n+1)|N(\Delta) \setminus S(\Delta)| = O(nm\log_k n).$$

The sum of the increases in Φ_3 over all scaling phases due to relabels is $O(n^2)$.

Finally, we consider increases in Φ_3 due to pushes in F and decreases in $Reserve(v)$. Since there are at most $O(m)$ violating nodes in total, there are $O(m)$ FRF-flows and decreases in

Reserve( ). Each of these operations can increase Φ_3 by at most $\frac{n}{k}$. The total increase in Φ_3 due to FRF operations is $O(\frac{nm}{k})$.

Taking into account all increases in Φ_3, we conclude that the number of large pushes is $O(n^2 + nm \log_k n)$.

We now bound the number of medium pushes using Φ_4. Recall that the initial value of $\Phi_4(\Delta)$ is 0 in each scaling phase.

Every medium push decreases Φ_4 by at least $\frac{1}{k}$ except for the first medium push from a node. Since there are $O(m \log_k n)$ scaling phases and there are at most n nodes per phase, there are $O(nm \log_k n)$ first medium pushes from nodes. (It is perhaps surprising that the first medium pushes from nodes is a bottleneck operation.)

The remaining number of medium pushes is at most k times the total increase in Φ_4 over all scaling phases. The potential function Φ_4 can increase in the following situations. (1) there is a relabel of a node, (2) ℓ decreases, (3) a node is added to P, (4) there is an operation of *FRF-Push* or *FRF-Pull*, or (5) there is an operation of *FRF-delete* in which *Reserve*( ) is decreased.

The total contribution to the increase in Φ_4 due to relabels is at most n^2. The contribution due to FRF operations (i.e., (4) and (5)) is the same as the corresponding contribution to Φ_3, which is $O(\frac{nm}{k})$. We now consider decreases to ℓ and additions to P.

By Lemma 14.9, special nodes cause an increase in Φ_4 of less than 1 over all scaling phase.

Finally, we analyze the contributions of non-special nodes when ℓ decreases and when a node is added to P. For each decrease of ℓ by 1, the contribution of each node $v \in N(\Delta) \backslash S(\Delta)$ is at most $\frac{1}{k}$ to the increase in Φ_4. Thus, the total contribution of $\Phi_4(\Delta)$ due to all decreases in ℓ during the phase is at most $\frac{n}{k} |N(\Delta) \backslash S(\Delta)|$. The increase in Φ_4 due to all decreases in ℓ over all phases is less than

$$\sum_{\Delta \in \Psi} \frac{n}{k} \cdot |N(\Delta) \backslash S(\Delta)| = O\left(\frac{nm \log_k n}{k}\right).$$

Finally, we consider the increases in Φ_4 when normal nodes are added to P. A node v can be added to P in one of three ways: (i) node v is relabeled, (ii) the parameter ℓ is reduced, and (iii) there is a medium push from node v. We have already accounted for increases due to (i) and (ii) in our analysis. We now consider (iii). If a normal node v is added to P following a medium push, then $e(v) \leq \epsilon \Delta + \frac{\Delta}{k}$ following the push. This leads to an increase in Φ_4 of $O(\frac{1}{k})$. Summing this increase over all normal nodes in all phases, we obtain an increase of $O(\frac{m \log_k n}{k})$.

We conclude that the total increases in Φ_4 over all scaling phases is $O(n^2 + \frac{nm \log_k n}{k})$. This shows that the number of medium pushes over all scaling phases is $O(kn^2 + nm \log_k n)$. $\square$

14.7. The main theorem.

Theorem 14.10. *The Enhanced LMES Algorithm finds the maximum flow in $O(kn^2 + nm \log_k n)$ time. Let k be the smallest power of 2 that is at least $\max\{\log \log n, \frac{m}{n}, 4\}$. Then the running time is*

$$O\left(\frac{nm \log n}{\log \log n + \log \frac{m}{n}}\right).$$

Proof. The optimality of the algorithm was established in Corollary 14.4. By Lemma 14.8, the number of medium and large pushes is $O(kn^2 + nm \log_k n)$. The time for contractions is $O(nm)$. The time for all operations in the flow-return forest is $O(nm)$. The time to update all arrays and lists at the beginning of a scaling phase is $O(n)$ per scaling phase and $O(nm \log_k n)$ in total.

The time for all other updates of arrays and lists is $O(nm + \# \text{ of pushes})$. We conclude that the total running time is $O(nm \log_k n)$. If we choose k to be the least power of 2 that is at least $\max\{\log \log n, \frac{m}{n}, 4\}$, we obtain the optimized running time. $\square$

15. Summary and conclusions

In this paper, we have developed Enhanced LMES Algorithm, which is a strongly polynomial time algorithm for the max flow problem with a running time of

$$O\left(\frac{nm\log n}{\log\log n + \log\frac{m}{n}}\right).$$

In the case that $\frac{m}{n} \geq n^{1/16}$, our algorithm dominates Orlin's [16] algorithm, which runs in $O(nm+m^{31/16}\log^2 n)$ time. Our algorithm strictly dominates the best other strongly polynomial time algorithm for the max flow problem, which is due to King et al [14]. In the case that $m = O(n\log n)$, the running time of our algorithm is faster than the algorithm in [14] by a factor of $\log\log n$.

The LMES Algorithm is based on the Stack Scaling Algorithm of of Ahuja et al. [3]. The Enhanced LMES Algorithm exploits the Contraction Lemma of [16] in order to improve the running time.

In order to use the Contraction Lemma to decrease the running time, we introduced several innovations in this paper. We first made a technical change to the amount of flow pushed so that anti-abundant arcs had residual capacities that were multiples of $\frac{\Delta}{k}$. More significantly, we permitted a push to result in a slightly negative excess under restricted conditions. We then introduced a data structure called the flow-return forest, which ensured that the excess of a node would never become "too negative." The flow-return forest is potentially of value for other max flow algorithms or possibly for minimum cost flow algorithms.

We explored whether we could obtain an improved running time using dynamic trees. For problems in which m is close to n, there might possibly be a small improvement in running time. However, we did not see how to implement dynamic trees so as to improve the running time. It is an open question whether such a speed-up is possible.

Appendix A. Expanding contracted cycles

After a max flow in the contracted network is obtained, the contracted cycles are expanded in the reverse order in which they were contracted. Subsequently, the flow in the expanded network is a maximum flow. The time for expansion of these cycles is $O(n^2)$. We now describe how to expand a contracted cycle W.

Let $G^c = (N^c, A^c)$ denote the contracted network. Let W be the cycle that was most recently contracted to obtain the network G^c. We will show how to expand the optimal flow for G^c. A node in N^c that was obtained by contracting a cycle is referred to as a *merged node*. The other nodes of N^c were also in N. They are referred to as *original nodes*.

Let x denote the preflow in the network G^c immediately prior to its contraction of the abundant cycle W. At the time that W was contracted, $r_{ij}(x) \geq M\Delta$ for each $(i,j) \in W$. Let y denote the optimal flow in the contracted network immediately prior to expanding W. Thus y is not defined for arcs of W. Let (S,T) denote the minimum cut in the contracted network.

The minimum cut in the network obtained by expanding W is (S',T') obtained as follows. If $w \in S$, then $S' = (S \setminus w) \cup W$, and $T' = T$. Otherwise, $T' = (T \setminus w) \cup W$, and $S' = S$. The capacity of (S,T) is equal to the capacity of (S',T').

We now want to modify y to obtain a maximum s-t flow. It suffices to modify y on arcs of W so that the resulting flow is feasible. We first extend y so that it is defined on arcs of W as follows. For each arc $(i,j) \in A$ with $i \in W$ and $j \in W$, we let $y_{ij} = x_{ij}$. By Lemma 14.8,

(A.1)
$$\sum_{i \in W} |e_y(i)| < 5n^2\Delta.$$

We next show how to obtain a flow y' on the arcs of W is *feasible* so that $y + y'$ is feasible for the network obtained from G^c by expanding W. That is, for all $v \in W$, $e_{y+y'}(v) = 0$, and $y' \geq 0$. (We ignore arc capacities for now.) Choose a vector y' defined on the arcs of W that minimizes $\sum_{(i,j) \in W} |y'_{ij}|$ and so that $y + y'$ is feasible as restricted to nodes of W. Then $y + y'$ is feasible for N^c. Moreover, for each $(i, j) \in W$, $0 \leq y'_{ij} < M\Delta$.

The time to compute y' is $O(|W|^2)$ since one can restrict attention to spanning tree flows (basic flows), and there are only $O(|W|)$ spanning subtrees of W. The time to expand a single cycle in this manner is $O(|W|^2)$. The time to expand all cycles is $O(n^2)$ since there are $O(n)$ arcs in all of the cycles that were contracted. We note that the running time for expanding cycles can be improved further, but we do not do so here since it is not a bottleneck of our algorithm.

Appendix B. Evaluating $\hat{e}(v, \Delta)$ and $\text{IMB}(v, x, \Delta)$ in $O(1)$ time

In order to evaluate $\hat{e}(v, \Delta)$, we need to identify whether a node is normal or special. To identify whether a node is normal or special, the algorithm maintains $\text{IMB}(v, x, \Delta)$ for each node v and for each preflow x. $\text{IMB}(v, x, \Delta)$ is updated at the following times: (1) when there is a push in an arc, (2) when an arc becomes anti-abundant, and (3) when an arc stops being anti-abundant and becomes bi-abundant. In addition, to determine whether a node is special, we need to determine whether the node is incident to a medium arc. For this, we need to know (4) when an arc transitions from small to medium, and (5) when an arc transitions from medium to large. (We also update data structures following a contraction. This is easily accomplished in the $O(m)$ time that the algorithm spends on each contraction.)

Updating IMB after a push is straightforward. We next deal with (2), (4) and (5). Recall that for an arc to become anti-abundant, the arc must first become large. For each node i, we store nodes in $A^+(i)$ in non- increasing order of bi-capacity. At each scaling phase, we maintain a pointer to the first medium arc of $A^+(i)$ as well to the first small arc of $A^+(i)$. For each node i the total time to update these two pointers over all scaling phases is $O(|A^+(i)| + \#$ of scaling phases$)$. When we update the pointers, we can determine when an arc becomes medium and when an arc becomes large.

Finally, we deal with (3). There are two ways that an arc (i, j) can transition from anti-abundant to bi-abundant. Suppose that the algorithm is in the Δ-scaling phase for some $\Delta \in \Psi$. For each subsequent scaling phase $\Delta' \in \Psi$, we let Bi-Abundant(Δ') denote the "bucket" of arcs that will become bi-abundant at the Δ'-scaling phase, assuming that there is no further change in flow in the arc. We store the buckets for the next $2Q + 1$ scaling phases, which is sufficient. Whenever there is a push in a large arc (i, j), we check whether (i, j) has become bi-abundant. If not, we update the bucket containing (i, j). When an arc changes from one bucket to another it shifts at most 2 buckets to the left or right. (It is also possible for an arc with residual capacity 0 to gain residual capacity. In this case, the arc is stored in the last or second to last bucket; (the last bucket contains arcs that will be abundant after $2Q + 1$ additional phases.) In any case, the updating of buckets following a push takes $O(1)$ steps. Moreover, the updating of buckets does not require the operation of division nor the operation of taking logs.

We summarize our results in the following lemma.

Lemma B.1. *At each iteration of the algorithm, for each node $v \in N(\Delta)$ and for each preflow x, $IMB(x, v, \Delta)$ and $\hat{e}(v, \Delta)$ can each be evaluated in $O(1)$ time.*

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows. Theory, Algorithms, and Applications.* Prentice Hall, Englewood Cliffs, NJ, 1993.

[2] R. K. Ahuja and J. B. Orlin. A fast and simple algorithm for the maximum flow problem. *Operations Research*, 37:748–759, 1989.

[3] R. K. Ahuja, J. B. Orlin, and R. E. Tarjan. Improved time bounds for the maximum flow problem. *SIAM Journal on Computing*, 18:939–954, 1989.

[4] J. Cheriyan and T. Hagerup. A randomized maximum-flow algorithm. 30th Annual Symposium on Foundations of Computer Science, 118–123, 1989.

[5] E. A. Dinic Algorithm for solution of a problem of maximum flow in networks with power estimation.. *Soviet Mathematics Doklady*, 11:1277–280, 1970.

[6] J.Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *JACM*, 19:248–264, 1972.

[7] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[8] H. N. Gabow Scaling algorithms for network problems. 24th Annual Symposium on Foundations of Computer Science, 248–258, 1983.

[9] A. V. Goldberg and S. Rao. Beyond the flow decomposition barrier. *Journal of the ACM*, 45:783–797, 1998.

[10] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum-flow problem. *J. ACM*, 35:921–940, 1988.

[11] D. S. Hochbaum. The pseudoflow algorithm: a new algorithm for the maximum-flow problem. *Operations Research*, 56:992–1009, 2008.

[12] Y. T. Lee and A. Sidford, Path finding methods for linear programming: solving linear programs in $\tilde{O}(\sqrt{rank})$ iterations and faster algorithms for maximum flow 55th Annual Symposium on Foundations of Computer Science 424–433, 2014.

[13] A. V. Karzanov. Determining the max flow in a network by the method of preflows. *Soviet Mathematics Doklady*, 15:435–437, 1974..

[14] V. King, S. Rao, and R. Tarjan. A faster deterministic maximum flow algorithm. *J. Algorithms*, 23:447–474, 1994.

[15] J. B. Orlin. A faster strongly polynomial minimum cost flow algorithm. *Operations Research*, 41:338–350, 1993.

[16] J. B. Orlin. Max flows in O(nm) Time, or Better Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing, 765–774, 2013.

[17] Y.Shiloach and U. Vishkin. An $O(n^2 \log n)$ parallel max flow algorithm. *Journal of Algorithms*, 3:128–146, 1982.

[18] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *J. Computer and System Sciences*, 24:362–391, 1983.

[19] E Tardos. A strongly polynomial algorithm to solve combinatorial linear programs. *Operations Research*, 34:250–256, 1986.