A FAST MAX FLOW ALGORITHM

Xiaoyue Gong
MIT, xygong@mit.edu
James B. Orlin
MIT, jorlin@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 very sparse graphs. 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, Rao, and Tarjan.

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 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 [9] and King et al. [7]. The running time for Orlin's algorithm is $O(nm + m^{31/16}\log^2 n)$, which is the fastest running time when $m < n^{1.06}$. Its running time can be sped up to $O(\frac{n^2}{\log n})$ on networks with $O(n)$ arcs. The running time for the algorithm by King et al. is $O(nm \log_\beta n)$, where $\beta = \frac{m}{n\log n}$. 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 the algorithm of Goldberg and Rao [5], which is the fastest weakly polynomial time algorithm. Their running time is $O(\min\{m^{1/2}, n^{2/3}\}m \log \frac{n^2}{m} \log U)$, where U is an upper bound on the largest finite capacity of an arc.

Here we present a max flow algorithm whose running time (slightly) dominates that of King et al. The running time of our algorithm is $O(nm \log_k n)$, where $k = 1 + \frac{m}{n}$. Our algorithm is based on the stack-scaling algorithm by Ahuja et al. [3], while drawing upon the idea of "special pushes" in the paper by Orlin [8]. And the analysis of special pushes is similar to that in the paper by Orlin [9].

The contributions of the paper may be described as follows.

(1) We present a simpler variant of the stack-scaling algorithm of Ahuja et al. [3] in which there are no stacks. We refer to the revised algorithm as the *large-medium excess-scaling algorithm.*
(2) We show how to adjust the amount of flow pushed in arcs so that the LM excess-scaling algorithm runs in $O(nm \log_k n)$ provided that excesses of nodes are never between $\frac{\Delta}{2} - \frac{\Delta}{m^4}$ and $\frac{\Delta}{2}$, where Δ is the parameter associated with a scaling phase.
(3) We modify the LM excess-scaling algorithm to permit node excesses that are slightly negative. In order to guarantee that the excesses do not become too negative relative to Δ, we create a new data structure called the "flow-return forest." Prior to the scaling phase in which the excess of node $e(i) < -\frac{\Delta}{4n}$, $\frac{\Delta}{k}$ units of flow will be sent from the root(i) to node i in arcs of the flow return forest.

Date: July 5, 2018. Revised August 1, 2018.

Although the flow-return forest was created specifically for the LM excess-scaling algorithm, it is possible that the flow-return forest may be of use in other algorithms for the max flow problem.

One aspect that differentiates our algorithm from the other fastest algorithms for the maximum flow problem is that our algorithm does not use the dynamic tree data structure of Sleator and Tarjan [10]. In fact, we do not see how to speed up our algorithm using dynamic trees. It is an open question as to whether such a speed-up is possible.

All proofs of lemmas, corollaries, and theorems are deferred to the appendix.

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 or infinite-valued *capacity* u_{ij}. Assume without loss of generality there are no multiple arcs from i to j. There is a unique *source* node $s \in N$ and a unique *sink* node $t \in N$.

Without loss of generality, we can assume that for each arc $(i, j) \in A$, (j, i) is also in A with u_{ji} possibly being 0. We also assume that for every node $i \neq s, t$, (s, i), (i, s), (i, t), and (t, i) are in A with $u_{t,i} = u_{i,s} = \infty$. If an optimal flow included an arc (i, s) or (t, i), we could obtain an optimal flow without these arcs by expressing the flow using flow decomposition and then eliminating any directed cycles.

We assume $m \geq 2n$, $U \geq 4$, and m, n are powers of 2. We let $k = \frac{m}{n}$. 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

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

The *maximum flow* problem is to determine a feasible flow that maximizes the amount of flow sent to node t.

Given a flow x, the *residual capacity* r_{ij} of any arc $(i, j) \in A$ is the maximum additional flow that can be sent from node i to node j using the arcs (i, j) and (j, i). The *residual capacity* of any arc $(i, j) \in A$ is $r_{ij} = u_{ij} + x_{ji} - x_{ij}$. We refer to the network $G(x)$ consisting of the arcs with positive residual capacities as *the residual network.*

The arcs (i, j) and (j, i) are referred to as a *pair of anti-parallel arcs.* We refer to (j, i) as the *reversal* of (i, j), and refer to (i, j) as the reversal of (j, i).

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

$$\sum_{\{j:(j,i)\in A\}} x_{ji} - \sum_{\{j:(i,j)\in A\}} x_{ij} \geq 0, \forall i \in N - \{s, t\}. \tag{2.2}$$

We assume that $x_{ji} = 0$ whenever $x_{ij} > 0$. Accordingly, $x_{ij} = \max\{0, u_{ij} - r_{ij}\}$.

A *preflow-push algorithm* maintains a preflow at each intermediate stage. For any given preflow, the *excess* of each internal node $i \in N$ is

$$e(i) = \sum_{\{j:(j,i)\in A\}} x_{ji} - \sum_{\{j:(i,j)\in A\}} x_{ij},$$

which is nonnegative.

A *distance function* is a mapping from N into $\mathbf{Z}^+ \cup \{0\}$. A distance function d is *valid* if

$$\begin{cases} d(t) = 0 \\ d(i) \leq d(j) + 1, & \text{for each arc } (i, j) \text{ such that } r_{ij} > 0. \end{cases}$$

We refer to $d(i)$ as the *distance label* of a node i.

We require distance functions to remain valid. We state this as an invariant:

Invariant 1. *(Validity)* $\forall (i, j) \in A$, *if* $r_{ij} > 0$, *then* $d(i) \leq d(j) + 1$.

3. Review of Generic Preflow Push Algorithm

Goldberg and Tarjan [6] introduced the preflow-push method for solving the maximum-flow problem. We review the method in this section. Preflow push algorithms begin with a preprocess step:

```
procedure preprocess;
begin
  $x := 0$;
  $d(j) := 0$ for all nodes $j \in N$;
  $d(s) := n$;
  $x_{s,j} := u_{s,j}$ for all arcs $(s,j) \in A$;
end;
```

Whenever x or r is modified, we assume that e, x and r are all updated so as to be consistent. Subsequent to the preprocess step and throughout the rest of the algorithm, $d(t)$ remains at a value of 0, and $d(s)$ remains at a value of n. Because there is an arc (j,i) with $u_{js} = \infty$, it follows that $d(j) \leq n+1$ throughout the algorithm.

The following lemma was proved by Goldberg and Tarjan [6].

Lemma 3.1. *Suppose that the distance labels are valid with respect to preflow x. Then $d(j)$ 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 preflow push algorithm maintains a preflow, and all pushes are on admissible arcs. In special circumstances, our algorithm will permit some negative excesses, and we will permit flows to be sent on inadmissible arcs.

We now describe the preflow push algorithm in more detail.

If $e(i) > 0$, we refer to node i as *active.* The presence of active nodes in a preflow-push algorithm indicates that the solution is not a feasible flow. The algorithm moves flow from active nodes through unsaturated arcs towards the sink, along paths estimated to contain as few arcs as possible. Excess flow at node j that cannot be moved to the sink is returned to the source after the distance label at node j increases to $n+1$. The algorithm terminates with a flow when the network contains no active node.

The fundamental operation of the preflow push algorithms is called "push/relabel." Informally, a push of δ in (i,j) sends δ units of flow from node i to node j. This is achieved by increasing x_{ij} or decreasing x_{ji} by a total of δ units. Equivalently, r_{ij} decreases by δ, and r_{ji} increases by δ. The operation push/relabel is called after an active node i is selected. Here is the pseudo-code.

```
procedure push/relabel($i$);
begin
  if the network contains an admissible arc $(i,j)$, then
    push $\delta := \min\{e(i), r_{ij}\}$ units of flow from node $i$ to node $j$;
  else replace $d(i)$ by $\min\{d(j)+1 : (i,j) \in A(i)$ and $r_{ij} > 0\}$;
  if $d(i) = n+1$ then push $\delta = e(i)$ from node $i$ to node $s$;
end;
```

In the case where $\delta = r_{ij}$, we say that the push in (i,j) is *saturating.* The generic preflow push algorithm is as follows:

```
procedure preflow-push;
begin
  preprocess;
  while the network contains an active node do
  begin
    select an active node $i$;
    push/relabel($i$);
  end;
end;
```

The following results are either proved in [6] or are implicit in that paper.

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

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

4. Large-Medium Excess Scaling Algorithm(LMES)

In this section we provide a modified version of the stack-scaling algorithm of Ahuja, Orlin and Tarjan [3]. The stack-scaling is, in turn, an enhancement of the excess-scaling algorithm [2],[1]. Our variant of the stack scaling algorithm does not use stacks in the selection of nodes for pushing. We refer to our variant as the Large-Medium Excess Scaling (LMES) Algorithm.

LMES uses an integer scaling factor k. We assume $k \geq 2$, and k is a power of 2. For the weakly polynomial algorithm, we will choose $k = 1 + \lfloor \frac{\log U}{\log \log U} \rfloor$. For the strongly polynomial algorithm presented later, we will choose $k = \frac{m}{n}$. We assume $k \geq 2$, and k is a power of 2. Recall that $m \geq 2n$, and m and n are both powers of 2.

In the Δ-scaling phase, a node i is a *large excess* node if $e(i) \geq \frac{\Delta}{2}$. A node i is a *medium excess* node if $\frac{\Delta}{k} \leq e(i) < \frac{\Delta}{2}$. A push of δ units is called *large* if $\delta \geq \frac{\Delta}{2}$. A push of δ units is called *medium* if $\frac{\Delta}{2k} \leq \delta < \frac{\Delta}{2}$. A push of δ units is called *small* if $\delta < \frac{\Delta}{2k}$.

The algorithm consists of scaling phases each with a parameter Δ. At the end of a Δ-scaling phase, $e(i) < \frac{\Delta}{k}$ for all nodes i, at which point Δ is replaced by $\frac{\Delta}{k}$, and a new scaling phase is begun. For the weakly polynomial algorithm, we begin the algorithm with Δ equal to the smallest power of k such that $\Delta > U$. Our strongly polynomial algorithm will begin with $\Delta = U + 1$.

For LMES, a phase consists of push/relabel steps using the following node selection rule.

LMES selection step: if there is a large excess node, then select a large excess node with the lowest distance label; else, if there is a medium excess node, select a medium excess node with the highest label; else, the phase ends and Δ is replaced by $\frac{\Delta}{k}$.

We apply the push/relabel step to each node selected. At the termination of the scaling phase in which $\Delta = k$, $e(i) < 1$ for each node i. Given that all arc flows are integer, the algorithm will have obtained a maximum flow. The algorithm terminates after at most $\log_k U + 1$ phases.

We state a few useful results (with proofs in the appendix):

Lemma 4.1. *The total amount of flow pushed in arcs by LMES during the Δ-scaling phase is at most $2n^2\Delta$.*

We obtain the following corollary.

Corollary 4.2. *If $r_{ij} \geq 4n^2\Delta$ at some iteration of the Δ-scaling phase, then (i, j) has positive residual capacity at all subsequent iterations.*

5. Abundant Arcs and Contraction

An arc (i, j) is called "abundant" in a scaling phase if r_{ij} is guaranteed to be positive in all subsequent scaling phases. A common approach for transforming a weakly polynomial flow algorithm into a strongly polynomial time algorithm involves the contraction of abundant arcs.

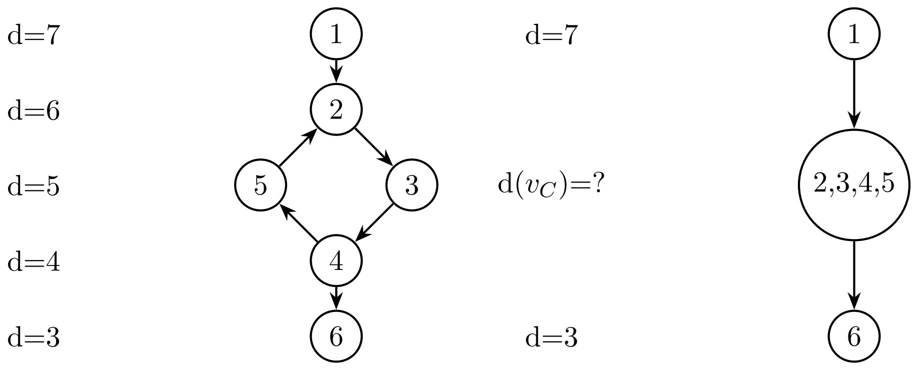
Corollary 4.2 applies to the LMES algorithm of the previous section, implying that an arc becomes abundant when $r_{ij} \geq 4n^2\Delta$. We let $M = 4m^2$. (We use M in several contexts and needed to choose a value greater than $4n^2$.) We say an arc (i, j) is *abundant* at the Δ-scaling phase if $r_{ij} \geq M\Delta$; arc (i, j) is called *anti-abundant* if (j, i) is abundant and $0 < r_{ij} < M\Delta$; arc (i, j) is called *doubly abundant* if (i, j) and (j, i) are both abundant.

An *abundant cycle* is a directed cycle in which every arc is abundant. Our algorithm will contract an abundant cycle C by replacing the nodes of C by a single pseudo-node, which we label here as v_C. We will only contract abundant cycles with distance labels less than n. We can ignore all nodes with distance label $n + 1$ since they have no excess.

Contraction is an essential aspect of our improved running time. However, contraction creates some difficulties with our analysis that we need to address:

(1) When we contract an abundant cycle C, there may be no way of setting the distance label of the contracted pseudo-node so that distance labels remain valid.
(2) The contracted pseudo-node might have an excess that exceeds Δ, thus violating the condition that $e_{max} \leq \Delta$.

Figure 1 shows that there may be no way of assigning a distance label to node v_C so that the distance labels are valid.



(A) before contraction (B) after contraction

Figure 1. Part of a graph before and after contraction

We now describe how we perform contraction so as to deal with both difficulties. We say that an original node j is *contained in a pseudo-node* v_C if node j was in the cycle C that was contracted to obtain v_C or if j was in a pseudo-node in the cycle C.

Modification of distance labels. Suppose an abundant cycle C has been contracted into a single pseudo-node. If $t \in C$, then the pseudo-node retains the label of t. Otherwise, we label the pseudo-node as v_C. Let N_c denote the nodes and pseudo-nodes of the contracted network. We recompute all distance labels following a contraction so that:

(1) If there is a path in $G(x)$ from j to t, then $d(j)$ is the minimum length of such a path.
(2) If there is no path in $G(x)$ from j to t, then $d(j) = n + 1$. Any excess at j is sent to s.
(3) For each original node i that is contained in a pseudo-node v_C, we let $d(i) = d(v_C)$. That is, both pseudo-nodes and the original nodes contained in them have an associated distance label.

Lemma 5.1. *The total number of distance label increases in the generic preflow push algorithms with contraction is at most $n^2 + n$.*

Excesses that exceed Δ. Our analysis of the running time of the LMES scaling algorithm relies on two potential functions Φ_1 and Φ_2. Φ_1 is defined and analyzed in the proof of Lemma 4.1 in the Appendix. The potential function Φ_2 is described and analyzed in the next section. The analysis of these potential functions relies on excesses being bounded by Δ. We circumvent this difficulty in the contracted network by carrying out the analysis Φ_1 and Φ_2 with respect to the original nodes. With that in mind, we define the *mean excess* of v_C as $\frac{e(v_C)}{|S|}$. For each original node j is contained in a pseudo-node, we define $e(j)$ to be the mean excess of the pseudo-node that contains j.

Lemma 5.2. *Suppose that v_C is a pseudo-node that does not contain s or t. At the Δ-scaling phase, for every original node j in v_C, $e(j) \leq \Delta$.*

6. An Additional Invariant

We introduce an additional modification for the LMES algorithm, which we call Invariant 2.

Invariant 2. *If $r_{ij} < r_{ji}$, and if $d(i) \geq 1$, then r_{ij} is a multiple of $\frac{\Delta}{k}$.*

When we push flow in an arc (i, j), we push the greatest amount δ of flow so that $\delta \leq min\{e(i), r_{ij}, \frac{\Delta}{2}\}$ and so that Invariant 2 is satisfied. In order to maintain Invariant 2, there are three situations in which the amount δ pushed in arc (i, j) might be small, even though the selected node i must have medium or large excess.

(1) $r_{ij} < \frac{\Delta}{k}$. (In this case, $\delta = r_{ij}$).
(2) $\frac{\Delta}{k} \leq e(i) < r_{ji} - r_{ij}$. (In this case, $\delta = \frac{(r_{ij} - r_{ji})}{2}$. After the push, $r_{ij} = r_{ji}$. Note: if $e(i) \geq r_{ji} - r_{ij} \geq \frac{\Delta}{k}$, then sending $r_{ji} - r_{ij}$ units of flow will satisfy the invariant.)
(3) $\frac{\Delta}{k} \leq e(i) \leq \frac{2\Delta}{k}$, $r_{ij} = r_{ji}$, and r_{ij} is not a multiple of $\frac{\Delta}{k}$. (In this case, $\delta = r_{ij} \bmod \frac{\Delta}{k}$. After the push, r_{ij} is a multiple of $\frac{\Delta}{k}$).

For each node i, when $d(i) = q$ for any fixed integer q, there can be at most one occurrence of each of the above three possibilities. Therefore, for fixed distance label of node i, there will be at most three small pushes in arc (i, j). The following is thus true for the LMES algorithm.

Lemma 6.1. *The number of small pushes in LMES is $O(nm)$.*

We need to be careful to maintain Invariant 2 following a contraction. When contracting a cycle C, the pseudo-node v_C, may become incident to parallel arcs. That is, there may be multiple arcs directed from (resp., to) node v_C to (resp., from) node j. If we were to aggregate arcs from node v_C to node j into a single arc (i, j), the aggregated arc may possibly violate Invariant 2. Here, we avoid the difficulty in a simple manner. We do not aggregate arcs from i to j unless each arc has $r_{ij} = 0$ or each of them has $r_{ji} = 0$. Instead, we (temporarily) permit multiple arcs directed from a pseudo-node to a node or pseudo-node. We can merge the arcs when $d(i)$ or $d(j)$ is relabeled. Henceforth, for notational convenience, we will ignore the possibility of multiple arcs from node i to node j.

7. Analysis of LMESC

The number of large pushes can be bounded based on Lemma 4.1. A large push will send $\frac{\Delta}{2}$ units of flow. By Lemma 4.1, the number of large pushes per scaling phase is $O(n^2)$. The number of large pushes over all scaling phases is $O(n^2 \log_k U)$.

Our analysis of the number of medium pushes is essentially the same as in [3]. We review it here in part because our algorithm here is different (because we do not rely on stacks) and in part because the analysis will be needed again for our strongly polynomial time algorithm. We define the potential function Φ_2, which is based on parameters ℓ and P. At the beginning of a scaling phase, we let $\ell = n$, $P = \emptyset$. Subsequently,

$$\ell := \min\{j : \text{there was a medium push from some node at distance level } j \text{ during the phase}\}.$$

$P := \{i \in N\backslash\{s,t\} : d(i) > \ell\} \cup \{i : d(i) = \ell,\, e(i) < \frac{\Delta}{k}$ and there was a medium push from $i\}$.

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

We next bound the number of medium pushes. We first partition the medium pushes into two groups, (1) the first medium push(es) from a node, and (2) every other medium push. We assume that we push from a selected medium excess node i until $e(i) < \frac{\Delta}{k}$ or until i is relabeled, whichever comes first. We refer to this sequence of pushes as the "first pushes from node i." During the first pushes from node i, there is at most one push in any arc (i,j). (We push as much flow as possible while satisfying Invariant 2.) If there is flow pushed in a second arc, then the flow sent in the first arc must have been saturating. Thus, there is at most one medium non-saturating push from node i during the first pushes from node i. There are at most n medium non-saturating pushes per scaling phase that are due to first pushes from nodes.

Every medium push in the second group is from a node in P. We now bound this number based on the following observations.

(1) Each medium push from a node in P decreases Φ_2 by at least $\frac{1}{k}$.
(2) At the beginning of a scaling phase, $\Phi_2 = 0$.
(3) There are three ways in which Φ_2 can increase during a scaling phase:
   (a) A node enters P.
   (b) A node's distance label increases.
   (c) ℓ decreases.

Because of (1) and (2) above, the number of medium pushes in a scaling phase is bounded from above by $k\times$ the total increase in Φ_2 during the scaling phase. The increase in Φ_2 from a node being added to P is at most $\frac{1}{k}$ per node and at most $\frac{n}{k}$ per scaling phase. The increase in Φ_2 from a node having its distance label increased by 1 is at most 1. As summed over all nodes in all scaling phases, the increase in Φ_2 due to distance increases is $O(n^2)$. Thus, the distance increases can account for as many as $O(kn^2)$ medium pushes. Finally, when ℓ is decreased by 1, every node of P except for the node with the most recent medium push has a distance label greater than ℓ. Each of these nodes contributes at most $\frac{1}{k}$ to the increase in Φ_2. Thus, each decrease in ℓ by 1 can lead to a total increase in Φ_2 by at most $\frac{n}{k}$. As ℓ decreases from $n+1$ to its minimum value, the total increase in Φ_2 is $O(\frac{n^2}{k})$. The number of pushes accounted for in this manner is $O(n^2)$ per scaling phase. If we now take into account all three operations that could lead to increases in Φ_2, the number of medium pushes over all scaling phases is $O(n^2 \log_k U + n^2 k)$.

By Lemma 6.1, the total number of small pushes over all scaling phases is $O(nm)$. As stated above, the total number of large pushes is $O(n^2 \log_k U)$. Therefore, the total number of pushes is $O((nm) + n^2 \log_k U + n^2 k)$. By choosing $k = \frac{\log U}{\log\log U}$, the number of pushes is bounded from above at $O(nm + \frac{n^2 \log U}{\log\log U})$. Moreover, the algorithm can be implemented to run in $O(nm + \frac{n^2 \log U}{\log\log U})$ time.

Our goal here is to modify the LM excess scaling algorithm so that it runs in $O(nm \log_k n)$ time. We first note that the time to contract strongly abundant cycles will not be a bottleneck.

Lemma 7.1. *The total time to contract nodes of G is $O(nm)$.*

We next provide the lemma that is most fundamental to our improved algorithm. We refer to the lemma as the *contraction lemma*. Based on the contraction lemma, we will show that the LM excess scaling algorithm would run in $O(nm \log_k n)$ time if the excesses at the end of scaling phases satisfied a simple property that depends on the residual capacities and original capacities.

At a given scaling phase, we partition the arcs into three subsets as follows: (Recall that $u_{ij} + u_{ji} = r_{ij} + r_{ji}$.) We say that (i,j) is a *small capacity* arc if $u_{ij} + u_{ji} < \frac{\Delta}{M^2}$; (i,j) is a *medium capacity* arc if $\frac{\Delta}{M^2} \le u_{ij} + u_{ji} < 2M\Delta$; (i,j) is a *large capacity* arc if $u_{ij} + u_{ji} \ge 2M\Delta$. If (i,j) has large capacity, then (i,j) or (j,i) or both are abundant.

In the following, $a \mod b$ denotes the non-negative remainder obtained by dividing a by b. In the following lemma, we assume that $|e(j)| \le 2\Delta$ at the Δ-scaling phase. Up till now, we have not considered negative excesses, and we have not considered the possibility of excesses exceeding Δ, but we shall do so starting in the next section.

Lemma 7.2. ***(Contraction Lemma)*** *Suppose that there are no medium capacity arcs incident to node j and that one (or both) of the following two conditions are true:*

(1) $\frac{\Delta}{M^2} \le e(j) \mod \frac{\Delta}{k} \le \frac{\Delta}{k} - \frac{\Delta}{M^2}$.

(2) $e(j) \ge \frac{\Delta}{M^2}$ *and there are no anti-abundant arcs with positive residual capacity directed out of node j.*

Then node j will become incident to a medium capacity arc within $O(\log_k n)$ scaling phases or it will be contracted within $O(\log_k n)$ scaling phases.

We refer to a node i as *quasi-medium* if node i is incident to a medium capacity arc or if i will become incident to a medium capacity arc within $O(\log_k n)$ scaling phases. Because each arc is medium for $O(\log_k n)$ scaling phases, and each arc is incident to exactly two nodes, the number of quasi-medium nodes summed over all scaling phases is $O(m \log_k n)$.

We refer to a node i as being *quasi-contracted* if node i will become contracted within $O(log_k n)$ scaling phases. The number of quasi-contracted nodes over all scaling phases is $O(n \log_k n)$.

The following theorem is a direct consequence of the contraction lemma.

Theorem 7.3. *Suppose that the Large-Medium Excess Scaling Algorithm with Contraction is run. Suppose that at each iteration, each node $j \ne s, t$ satisfies the following condition.*

$$\frac{\Delta}{M^2} \le e(j) \mod \frac{\Delta}{k} \le \frac{\Delta}{k} - \frac{\Delta}{M^2}$$

.

Then the total number of quasi-medium and quasi-contracted nodes over all scaling phases is $O(m \log_k n)$.

We can use this theorem to prove a bound on the number of medium and large pushes under restricted conditions. It is this theorem that motivates the use of "special pushes" that permit excesses to become negative, and the creation of a data structure for restoring positive excesses to nodes.

Theorem 7.4. *Suppose that the Large-Medium Excess Scaling Algorithm with Contraction is run. Suppose further that whenever a node i is incident to no medium capacity arcs, then $e(i) \mod \frac{\Delta}{k} < \frac{\Delta}{k} - \frac{\Delta}{M^2}$. Then the number of scaling phases in which there is a push is $O(m \log_k n)$. And the total number of medium and large pushes over all scaling phases is $O(nm \log_k n)$.*

In the next section, we will show how to handle nodes that are not quasi-medium with excess greater than $\frac{\Delta}{k} - \frac{\Delta}{M^2}$. For these nodes, we will (usually) permit $\frac{\Delta}{k}$ units of flow to be pushed from them, even if this results in a (slightly) negative excess. Our algorithm uses a new data structure called the "flow return forest" that will restore flow to these negative excess nodes so as to guarantee that no excess is ever less than $-\frac{\Delta}{k}$. Similar to other preflow push algorithms, the algorithm terminates when all excesses are 0.

After showing that there are at most $O(nm \log_k n)$ medium and large pushes, we still need to modify the algorithm to bound the time spent in scaling phases with no pushes. Actually, we will modify the algorithm so that the number of scaling phases is bounded at $O(m \log_k n)$. We can then conclude that the total running time is $O(nm \log_k n)$.

8. Large-Medium Excess Scaling with Contraction and Flow-Return Forest

In this section we will describe "special pushes" and the "flow-return forest." In the first four subsections, we describe conditions under which special pushes are allowed. We will also introduce a data structure consisting of arcs that are eligible to be return-arcs. In these subsections, we will assume that $e(i) > -\frac{\Delta}{nM}$. In the following subsection, we will introduce the "flow-return forest," which we will denote as F. The flow-return forest is not required by the algorithm until a node excess is less than $-\frac{\Delta}{nM}$.

8.1. **Large and medium excess nodes.** We will introduce modifications into our max flow algorithm. Our modified algorithm differs from the LM excess-scaling algorithm with contraction in several ways.

(1) We modify the definitions of "large excess" and "medium excess" as well as the selection rule in Push/Relabel.
(2) We require $e(i)$ to be at least $\frac{\Delta}{2n}$ with some exceptions. These exceptions include the possibility that a node will have negative excess.
(3) We develop a data structure called the flow-return forest for sending flow to nodes with sufficiently large negative excess.

We first modify the definitions of medium and large excess nodes.

(1) If $e(i) \geq \frac{\Delta}{2} + \frac{\Delta}{2n}$, then node i has *large excess.*
(2) A node i is said to have *medium excess* if $\frac{\Delta}{k} + \frac{\Delta}{2n} \leq e(i) < \frac{\Delta}{2} + \frac{\Delta}{2n}$.

Invariant 3. *Suppose that node i has been selected for pushing. If node i is a large or medium excess node, then subsequent to the push* $e(i) \geq \frac{\Delta}{2n}$.

By Invariant 3, most nodes will have an excess that is at least $\frac{\Delta}{2n}$. However, we will permit nodes to have less excess as well. If a node i has $e(i) < \frac{\Delta}{2n}$, it will be called *special.* It can only become special following a "a special push", which we will describe soon.

We may consider the lower bound of $\frac{\Delta}{2n}$ on excess as a "buffer". Our algorithm will take advantage of this extra excess at nodes when we construct the flow-return forest.

8.2. **Eligible arcs and the special push conditions.** An arc (j, i) is said to be *eligible* for node i if (j, i) is abundant and if $d(i) \geq d(j) + 1$. For example, if $d(i) \leq 1$, then (t, i) is eligible. (Recall that $r_{ti} = \infty$).

We let *eligible*(i) denote a node j for which (j, i) is eligible, assuming that there is an eligible arc for node i. Otherwise, *eligible*$(j) = \emptyset$. We maintain the array *eligible*$(\cdot)$ throughout the execution of the algorithm. Once there is an admissible arc directed into node j, we determine *eligible*(j). We update *eligible*(j) whenever the distance label of node j is modified. The total time to maintain the array *eligible* is $O(nm)$.

We will permit a node i with $e(i) < \frac{\Delta}{k} + \frac{\Delta}{2n}$ to be selected for pushing if it satisfies the following three conditions, which we call the *special push conditions.*

(1) $\frac{\Delta}{k} - \frac{\Delta}{M^2} \leq e(i) < \frac{\Delta}{k} + \frac{\Delta}{2n}$.
(2) Node i is not incident to a medium capacity arc, and
(3) *eligible*$(i) \neq \emptyset$.

We will add a fourth condition in the next subsection. But first, we will state a lemma.

Lemma 8.1. *Suppose that node i satisfies the first two conditions for a special push, and suppose that eligible*$(i) = \emptyset$. *Then node i will be contracted in* $O(\log_k n)$ *scaling phases.*

8.3. **Special nodes and the ERA Graph.** If a node i satisfies the special push conditions, we will permit a push of $\frac{\Delta}{k}$ from node i. After the push, node i becomes special. In the case that $e(i) < 0$ after the push, we say that node i is *deficient.*

After a special push from node i, we let ERA(i) := eligible(i). ERA is an abbreviation of "eligible to be a return arc." The arc $(\text{ERA}(i), i)$ is referred to as the *ERA arc* for node i. There is an ERA arc for each special node.

The *ERA graph* consists of all of the ERA arcs and its incident nodes. Without further restrictions on special pushes, it would be possible for the ERA graph to contain a directed cycle. However, we will forbid the possibility of directed cycles in the ERA graph.

Invariant 4. The ERA graph is a forest.

In order to ensure that the ERA graph is a forest, we add a fourth special push condition.

(4) There is no path in the ERA graph from node i to node eligible(i).

If there is a path P from node i to node eligible(i), then we contract the abundant cycle consisting of P and $(eligible(i), i)$. We only check on condition (4) if the first three special push conditions are satisfied.

Because of (4), a contraction will never create a pseudo-node that is special, as implied by the next lemma. This lemma is needed to ensure that every special node or pseudo-node has an ERA arc. It would be possible for a special pseudo-node to have no ERA arc if the pseudo-node were created by contracting an abundant cycle of special nodes and pseudo-nodes.

Lemma 8.2. *Suppose that node i satisfies the first three conditions for a special push, and suppose that there is an abundant path P from node i to node eligible(i). Let v_C be the pseudo-node created upon contracting the cycle C consisting of P and $(eligible(i), i)$. Suppose further that every node of C has excess at least $-\frac{n\Delta}{M}$. Then $e(v_C) > \frac{\Delta}{2n}$.*

Because of (4), the ERA graph contains no cycles, and each special node has an ERA arc directed into it. Each node of the ERA graph is special except for its root nodes. Node i is a *root node of the ERA graph* if it is non-special, and if $i = ERA(j)$ for some special node j.

Computing distance labels following a contraction. We have previously pointed out that distance labels are recomputed following a contraction. In recomputing distance labels, we want to ensure that ERA arcs remain eligible. Instead of computing distance labels with respect to the residual network $G(x)$, we instead compute distance labels with respect to $G(x) \cup H$, where H is the reversal of the ERA graph. That is, if $i = \text{ERA}(j)$, then $(j, i) \in H$. In this manner, ERA arcs remain eligible.

Note that all of the arcs in $G(x) \cup H$ are valid. Therefore, Lemma 3.1 is correct with respect to $G(x) \cup H$. That is the distance $d(j)$ is a lower bound on the distance from j to t in $G(x) \cup H$. Accordingly, Lemma 5.1, which bounds the total number of distance relabels, remains true even if we recompute distances with respect to $G(x) \cup H$ rather than $G(x)$.

8.4. Selecting nodes for push/relabel.

In this subsection, we describe how to select a node for pushing. Let *Large* and *Medium* denote the sets of large and medium nodes at a given iteration. Let *PartSpecial* denote the nodes satisfying the first three conditions for special pushes.

The procedure Select/Push/Relabel describes selection and pushing. It prematurely includes how to push if the selected node is contained in the Flow-return forest F. We write the lines of pseudo-code here so that we don't need to repeat the algorithm later on. For now, the reader can ignore the lines "**if** $i \in F$, **then** $Push(F, i)$." These lines will make sense after the flow-return forest is described in detail. The reader can review them again at that time.

The procedure Push/Relabel is the same as in Section 4, except that Invariant 3 requires excesses to be at least $\frac{\Delta}{2n}$ following a push from a large or medium node.

```
Procedure Select/Push/Relabel
begin
  if $Large \neq \emptyset$, then
    $i :=$ argmin$\{d(i) : i \in Large\}$;
    if $i \in F$, then $Push(F, i)$; else Push/Relabel($i$);
  else if $Medium \cup PartSpecial \neq \emptyset$, then
    $i :=$ argmax$\{d(i) : i \in Medium \cup PartSpecial\}$;
    if $i \in F$, then $Push(F, i)$;
    else if $i \in PartSpecial$ and if there is a path $P$
          in the admissible graph from $i$ to ERA($i$), then
```

```
        $C := P$, (ERA($i$), $i$);
        contract abundant cycle $C$
    else Push/Relabel($i$);
end
```

If a node i satisfying the four conditions for a special push is selected, and if there is an admissible arc, then exactly $\frac{\Delta}{k}$ are pushed from node i. If there is no admissible arc out of i, then node i is relabeled. If $eligible(i) = \emptyset$ following the label, then node i no longer satisfies the conditions for a special push, and is it is "de-selected."

For the purposes of analysis, special pushes are treated the same as medium pushes. Both push an amount of flow that is at least $\frac{\Delta}{k}$.

8.5. **A preview of the flow-return forest.** A node i with $e(i) < 0$ is called *deficient.* A node i with $e(i) < -\frac{\Delta}{nM}$ is called *strongly deficient.*

A deficient node i will eventually become strongly deficient if no flow is pushed to it. After node i becomes strongly deficient, it is added to the flow-return forest F, and we identify another node of F as root(i). At the time that node i is added to F, the amount of excess reserved for node i at root(i) is $\gamma(i) = \frac{\Delta}{k^Q}$, where Q is the least integer such that $k^Q \geq 4n^2$.

After $Q - 1$ additional scaling phases, the scaling parameter is $\Delta' = \frac{\Delta}{k^{Q-1}} = k\gamma(i)$. At this scaling phase, if node i is still a node of F, then $\gamma(i) = \frac{\Delta'}{k}$ units of flow are sent from root(i) to i, after which i will be deleted from F. After sending the flow to node i, the excess at i is at least $\frac{\Delta}{2n}$. (We will later describe situations when the scaling parameter at the subsequent scaling phase is not $\frac{\Delta}{k}$. However, when $F \neq \emptyset$, the scaling parameter at the next scaling phase will always be $\frac{\Delta}{k}$.)

For every root node $i \in F$, we let $\Gamma(i) = \sum_{j:root(j)=i} \gamma(j)$.

If i is added to F at the Δ-scaling phase, then $\gamma(i) \leq \frac{\Delta}{4n^2}$. This means that the total amount of flow reserved because of additions of nodes to F at the Δ-scaling phase is less than $\frac{\Delta}{4n}$. It is for this reason that we have required all non-special nodes to have an excess of at least $\frac{\Delta}{2n}$.

The flow-return forest maintains the following properties:

(1) If a node enters F as a leaf node, it does not leave F until i is no longer special. This can only occur after a push into node i.
(2) For every node $i \in F$, there is an abundant path from root(i) to i in F consisting of eligible arcs.
(3) For every root node $i \in F$, $e(i) \geq \Gamma(i) + \frac{\Delta}{2n}$ at the beginning of the Δ-scaling phase, and $e(i) \geq \Gamma(i) + \frac{\Delta}{4n}$ throughout the Δ-scaling phase. ($e(i) - \Gamma(i)$ can decrease after the beginning of a phase if another node j is added to F such that $i = root(j)$.)
(4) No node of F is contracted while still in F. If a node i of F is incident to a doubly abundant arc, then the contraction of the doubly abundant arc is deferred until after i is deleted from F.
(5) No node of F is relabeled while in F, except for when all nodes are relabeled following a contraction. (If a node $i \in F$ is selected for pushing, then flow is sent from a node i to a leaf node of F. If $i \in F$, the procedure Push/Relabel is not called.)

8.6. **The Flow-Return Forest.** In this section, we describe the operations permitted on the flow-return forest. We also show how the flow-return forest can be used to guarantee that flow is sent to strongly deficient nodes prior to their excess reaching $-\Delta/k$. Each strongly deficient node will receive flow and become non-deficient within $O(log_k n)$ scaling phases after being added to F.

Adding leaf nodes to the flow-return forest

The flow-return forest F is a data structure on which we will perform several operations. Initially, $F = \emptyset$. At the scaling phase at which a node i becomes strongly deficient, we add node i to the forest as a leaf node. We describe this procedure now. We use $root(i)$ to denote the root of node i in the flow-return forest. We use ERA-root(i) to denote the root of node i in the ERA graph.

```
Procedure Add$(F, i)$
begin
  $j :=$ ERA-root$(i)$;
  let $P$ be the path in the ERA graph from node $j$ to node $i$;
  if $j = t$, then
    send $\frac{\Delta}{k}$ units of flow from $t$ to $i$ in $P$ and quit;
  else continue
  if $j \notin F$, then $root(i) := j$; else, $root(i) := root(j)$;
  for all $w \in P$, $root(w) := root(i)$;
  $F := F \cup P$;
  let $SD(P)$ be the set of strongly deficient nodes in $P \setminus F'$;
  for each $w \in SD(P)$ do $\gamma(w) := \frac{\Delta}{k^Q}$;
  $\Gamma(root(i)) = \sum_{w:root(w)=Root} \gamma(w)$;
end
```

The above procedure describes how nodes are added to F. We observe i is strongly deficient when added to F, and every internal node of P is special when added to F.

We now describe conditions under which a node may be deleted from F.

(1) If node i was added to F when it was strongly deficient, or if i became strongly deficient while in F, then i is deleted from F if (i) $e(i) \geq \frac{\Delta}{k}$, and (ii) node i is a leaf node of F.
(2) If node i was not strongly deficient when added to F and if it did not become strongly deficient while in F, then i is deleted from F as soon as it becomes a leaf node of F.

Suppose that node i becomes strongly deficient at the Δ-scaling phase; that is, $e(i) \leq -\frac{\Delta}{nM}$. We will add node i to F, as described below. And we set $\gamma(i) = \frac{\Delta}{k^Q}$. At the beginning of each subsequent scaling phase, we will look at node i. We delete node i from F if it is not special. And if the scaling parameter is $k\gamma(i)$, we will send $\gamma(i)$ units of flow from $root(i)$ to node i in F. Node i will no longer be special after the push, and will be deleted from F. We maintain the invariant that $e(i) - \Gamma(i) \geq \frac{\Delta}{4n}$ throughout the scaling phase so that there is sufficient flow to send from each root node. The following procedure returns the reserve capacity for the root node of a node i that enters the forest.

Operations on the flow-return forest.
The following are six operations that can be carried out on the flow-return forest F.

- *CreateNull*(F). This operation creates a null flow-return graph.
- *Add*(F, i). This operation adds a strongly deficient node i as a leaf of F. It is described in detail above.
- *Pull*(F, i). This operation is called when $\gamma(i) = \frac{\Delta}{k}$, and that $i \in F$. Then $\gamma(i)$ units of flow are sent from $root(i)$ to i along arcs of F. Each call of *Pull*(F, j) is immediately followed by calls to *UpdateReserve*(F) and *UpdateNodes*(F) (see below).
- *Push*(F, j). This operation assumes that node $j \in F$ is selected by the algorithm for pushing. Thus, node j has large excess or medium excess, or it satisfies the conditions for a push. The procedure then selects a leaf node of F that is a descendent i of node j in F. (Such a node is guaranteed to exist and be strongly deficient.) Then $\frac{\Delta}{k}$ units of flow are pushed from j to i in F. Each call of *Push*(F, j) is immediately followed by calls to *UpdateReserve*(F) and *UpdateNodes*(F).
- *UpdateReserve*(F). This operation is called in either of the following conditions: (i) there is a node $i \in F$ with $e(i) \geq \frac{\Delta}{k}$ and $\gamma(i) > 0$. (This will occur after a push into node i and prior to $\gamma(i)$ being updated); (ii) there is a strongly deficient node $i \in F$ with $\gamma(i) = 0$. (This can occur if a node $i \in F$ became strongly deficient at the beginning

of the scaling phase and prior to $\gamma(i)$ being set.) In either case, $\gamma(i)$ is updated, as is $\Gamma(root(i))$.

- $UpdateNodes(F)$. This operation is called if there is a leaf node of F that satisfies conditions for it to be deleted. If so, the operation deletes the leaf node from F. The process continues until no leaf node of F can be deleted. (Possibly, every node of F will be deleted.)

We note that each of these six operations to F can be implemented so that they run in $O(n)$ time each time they are called. Every Pull operation and Push operation transforms a leaf node of F into a node that is no longer special and which will be deleted from F.

By the Contraction Lemma, the number of times that some node can become strongly deficient is $O(m)$ over all scaling phases. (A strongly deficient node will either become incident to a (new) medium capacity arc within $O(\log_k n)$ scaling phases or it will be contracted in $O(\log_k n)$ scaling phases.) This implies that the number of times that some node can be transformed from being a strongly deficient node into a non-special node is also $O(m)$. The operations of having positive excess and being deleted from F each takes $O(n)$ steps. Since each of the forest operations can be carried out in $O(n)$ time, the total time spent on operations for the flow-return-forest is $O(nm)$.

8.7. Modifications Introduced Because of the Flow-Return Forest.

We now show how to integrate the operations on the flow-return forest into our previously described algorithm.

We have already integrated the operation "$Push(F, i)$" into"Select/Push/Relabel" as discussed earlier in this section. Each call of $Push(F, i)$ is followed directly by the procedures *UpdateReserve* and *UpdateNodes*. There are two other changes that we introduce at the beginning of a scaling phase dealing with adding and deleting nodes from F.

(1) For all $i \in F$, if $\gamma(i) = \frac{\Delta}{k}$, do Pull$(F, i)$.
(2) For all nodes i that have just become strongly deficient, if $i \notin F$, then $Add(F, i)$.

If we carry out these changes, the number of pushes will be $O(nm \log_k n)$, as stated and proved in the next section. However, there is still one remaining difficulty to take care of. It is possible that there would be an exponentially large number of scaling phases with no pushes. Our final modification of the algorithm is the change in scaling parameter under restricted conditions.

Change of scaling parameter. At the beginning of a scaling phase, we first check to see if there are any nodes with non-zero excess. If not, the current flow is optimal in the contracted graph. We would then expand all of the contracted nodes, as in Section 10.7 of Ahuja et al. [1], obtaining an optimal flow for the original problem.

If there is some node with non-zero excess, we check to see if the following conditions are satisfied: (i) there are no medium capacity arcs, (ii) for each node i, $-\frac{\Delta}{M^2} \le e(i) \le \frac{\Delta}{M}$. If these two conditions are satisfied, we replace the scaling parameter by $\min\{\Delta_1, \Delta_2\}$, where $\Delta_1 = \max\{\frac{e(i)}{k} : e(i) \ge 0\}$, and $\Delta_2 = \max\{\frac{-e(i)}{kM} : e(i) < 0\}$. Then contract each pair of doubly abundant arcs. If the two conditions are not satisfied, we replace Δ by $\frac{\Delta}{k}$.

We refer to this procedure as "UpdateScalingParameter."

Lemma 8.3. *Suppose that UpdateScalingParameter is called at the beginning of each scaling phase. Then Invariant 2 is satisfied following any change in scaling parameter. The total number of scaling phases is $O(m \log_k n)$.*

9. Analysis of LMESCF and conclusion

In this section, we show that the LM excess scaling algorithm with the flow-return forest and contraction determines a maximum flow in $O(nm \log_k n)$ time.

We first state a lemma that bounds the number of non-special nodes at the beginning of scaling phases. After, we state a lemma that bounds the number of times that a strongly deficient node can enter F. Then we prove our main result on the running time.

Lemma 9.1. *The total number of non-special nodes at the beginning of all scaling phases is $O(nm \log_k n)$.*

Lemma 9.2. *The number of times that a strongly deficient node enters F is $O(m)$.*

Theorem 9.3. *The LM excess scaling algorithm with the flow-return forest and contraction determines a maximum flow in $O(nm \log_k n)$ time.*

In this paper, we have shown how to modify the stack-scaling algorithm of Ahuja et al. into a strongly polynomial time algorithm whose running time dominates previous algorithms except for that of Orlin [9]. Orlin's algorithm dominates our algorithm for sufficiently sparse cases. Our algorithm dominate's Orlin's algorithm if $\frac{m}{n} \geq n^{1/16}$.

We have introduced three algorithmic innovations in this paper. The first is to eliminate stacks from the stack-scaling algorithm. The second is to ensure that anti-abundant arcs have residual capacities that are multiples of Δ/k. (We actually required that Invariant 2 is satisfied, which is more restrictive. But it would have sufficed to restrict attention to anti-abundant arcs.) Both of these innovations are relatively simple. The third innovation was to permit special pushes that could result in negative excess at nodes, and then to use a new data structure called the flow-return forest to return flow to these nodes. Much of this paper has been concerned with special pushes and the flow return forest.

Perhaps there will be other flow algorithms for which the flow-return forest will be a useful data structure. We have explored the possibility for the wave scaling algorithm of Ahuja et al, but without success.

We also explored whether we could obtain an even better running time using dynamic trees. However, we did not see how to successfully implement dynamic trees without violating Invariant 2. As such, we were unable to achieve any speedup using the dynamic tree data structure. It is an open question whether such a speed-up is possible.

Appendix A. Proofs for Theorems, Lemmas, and Corollaries

A.1. **Lemma 4.1:** The total amount of flow pushed in arcs by the excess scaling algorithm during the Δ-scaling phase is at most $2n^2\Delta$.

Proof. This relies on a potential function Φ_1 from [2].

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

We may ignore any node with distance label n or greater. (If $d(i) > n$, then $d(i) = n+1$ and i has no excess.) The initial value of Φ_1 is at most n^2 because $e(i) < \Delta$ and $d(i) < n$ for each node i with excess. Whenever there is a push of δ units, Φ_1 decreases by $\frac{\delta}{\Delta}$. The total decrease in Φ_1 during the Δ-scaling phase is bounded by the initial value of Φ_1 plus the total increase in Φ_1. We now show that the total increase in Φ_1 during the Δ-scaling phase is bounded above by n^2. The only way for Φ_1 to increase is when there is a distance relabel. The maximum increase in distance labels (ignoring increases to $n+1$) is n^2 over all scaling phases.

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

A.2. **Lemma 4.2:** If $r_{ij} \geq 4n^2\Delta$ at some iteration of the Δ-scaling phase, then (i,j) has positive residual capacity at all subsequent iterations.

Proof. This follows from the fact that the total amount of flow pushed in (i,j) in the Δ-scaling phase and all subsequent scaling phases is bounded above by $2n^2(\Delta + \frac{\Delta}{2} + \frac{\Delta}{4} + \cdots) = 4n^2\Delta$. □

A.3. **Lemma 5.1:** The total number of distance label increases in the generic preflow push algorithms with contraction is at most $n^2 + n$.

Proof. Let $G' = (N', A')$ denote the contracted graph at a given iteration. If there is a path from node (or pseudo-node) v to node t, then $d(v) < |N'|$. At a given iteration, let $DistInc(j)$ denote an upper bound on the total number of subsequent distance increases of node j, assuming no subsequent contractions. Initially, $DistInc(j) < n+1$. Each increase in $d(j)$ by 1 leads to a decrease of $DistInc(j)$ by 1.

Now consider the effect when a set C of nodes is contracted. Distance labels may decrease by as much as $|C|-1$, which would lead to an increase in $DistInc(j)$ by $|C|-1$. At the same time, the number of nodes decreases by at least $|C|-1$, which would lead to a decrease of $DistInc(j)$ by at least $|C|-1$. Therefore, $DistInc(j)$ either stays the same or decreases following the contraction. This implies that the number of distance increases throughout the algorithm is less than $n+1$ per node and less than n^2+n in total. □

A.4. **Lemma 5.2:** Suppose that v_C is a pseudo-node that does not contain s or t. At the Δ-scaling phase, for every original node j in v_C, $e(j) \leq \Delta$.

Proof. If the excess of every original node is at most Δ, then the mean excess of any contracted pseudo-node must be at most Δ. □

A.5. **Lemma 7.1:** The total time to contract nodes of G is $O(nm)$.

Proof. There can be at most n contractions over the entire algorithm. Each contraction takes $O(m)$ time to evaluate. We can recompute distance labels following each contraction in $O(m)$ time using a backwards breadth first search starting at node t. □

A.6. **Contraction lemma:** Suppose that there are no medium capacity arcs incident to node j and that one (or both) of the following two conditions are true:

(1) $\frac{\Delta}{M^2} \le e(j) \mod \frac{\Delta}{k} \le \frac{\Delta}{k} - \frac{\Delta}{M^2}$.
(2) $e(j) \ge \frac{\Delta}{M^2}$ and there are no anti-abundant arcs with positive residual capacity directed out of node j.

Then node j will become incident to a medium capacity arc within $O(\log_k n)$ scaling phases or it will be contracted within $O(\log_k n)$ scaling phases.

Proof. Let Δ' to be the first scaling parameter after the Δ-scaling phase such that $\Delta' \le \frac{\Delta}{nM^3}$. The parameter Δ' will be the scaling parameter within $O(\log_k n)$ scaling phases. If node j has become incident to a medium capacity arc at or prior to the Δ'-scaling phase, there is nothing to prove. Suppose instead that node j has not become incident to a medium capacity arc. We will show that node j must be incident to a doubly abundant arc at the Δ'-scaling phase. Given that doubly abundant arcs are contracted, this contradiction will prove the lemma.

Let $D(a,b) = \min\{(a \mod b), (-a \mod b)\}$. Thus $D(a,b)$ is the minimum distance from a to an integer multiple of b.

Let $e(j)$ (resp., $e'(j)$) denote the excess of node j at the beginning of the Δ-scaling phase (resp., Δ'-scaling phase). Let r and r' denote the corresponding residual capacities. Let $\delta = e(j) - e'(j)$. We express δ as $\delta_l + \delta_s$, where the value δ_l is due to changes in flow in large capacity arcs incident to node j and δ_s is due to changes in flow in small capacity arcs. Note that $\delta_s < \frac{n\Delta'}{M^2}$.

We now consider (1), which is equivalent to saying that $D(e(j), \frac{\Delta}{k}) \ge \frac{\Delta}{M^2}$. Then

$$D(\delta_l, \tfrac{\Delta}{k}) \ge D(e(j), \tfrac{\Delta}{k}) - D(e'(j), \tfrac{\Delta}{k}) - \delta_s > \frac{\Delta}{M^2} - 2\Delta' \ge nM\Delta' - 2\Delta' > (n-1)M\Delta'.$$

It follows that there must be an arc (j,i) for which $D(r_{ji} - r'_{ji}, \frac{\Delta}{k}) > M\Delta'$. Since $r_{ij} - r'_{ij} = r'_{ji} - r_{ji}$, it follows that $D(r_{ij} - r'_{ij}, \frac{\Delta}{k}) > M\Delta'$. We claim that arc (i,j) is doubly abundant. We now consider the case that (i,j) was abundant at the Δ-scaling phase, and thus (j,i) was anti-abundant. The case that (j,i) was abundant can be proved analogously.

If $r_{ij} > r'_{ij}$, then $r'_{ji} = (r_{ij} - r'_{ij}) + r_{ji} > M\Delta'$. In this case, (j,i) has become abundant, and (i,j) is doubly abundant. Suppose instead that $r_{ij} < r'_{ij}$ and thus $r_{ji} > r'_{ji}$ By Invariant 2, $r_{ji} = 0 \mod \frac{\Delta}{k}$. Therefore, $D(r_{ji} - r'_{ji}, \frac{\Delta}{k}) = D(r'_{ji}, \frac{\Delta}{k}) > M\Delta'$. So, in this case, (j,i) must be abundant. We conclude that (i,j) becomes doubly abundant if (1) above is satisfied.

We now consider case (2); that is, we assume that there are no anti-abundant arcs directed out of node j. As we proved above in (1), $e(j) - e'(j) = \delta_l + \delta_s$, and $\delta_l > (n-1)M\Delta'$. Similarly, there must be an arc (j,i) for which $D(r_{ji} - r'_{ji}, \frac{\Delta}{k}) > M\Delta'$. In this case, (j,i) must be abundant at the Δ-scaling phase (and also the Δ'-scaling phase), and $r'_{ji} \le r_{ji} - M\Delta'$. It follows that $r'_{ij} \ge M\Delta'$, and (i,j) is abundant at the Δ'-scaling phase. Thus (j,i) has become doubly abundant, which is what we wanted to prove.

We conclude that j will be incident to a medium capacity arc or it will be contracted within $O(\log n)$ phases. □

A.7. **Theorem 7.4:** Suppose that the Large-Medium Excess Scaling Algorithm with Contraction is run. Suppose further that whenever a node i is incident to no medium capacity arcs, then $e(i) \mod \frac{\Delta}{k} < \frac{\Delta}{k} - \frac{\Delta}{M^2}$. Then the number of scaling phases in which there is a push is $O(m \log_k n)$. And the total number of medium and large pushes over all scaling phases is $O(nm \log_k n)$.

Proof. The primary difference between the assumptions in this theorem and the previous theorem is that here we permit a node i that is not incident to any medium capacity arcs to have an excess $e(i) \mod \frac{\Delta}{k} < \frac{\Delta}{M^2}$.

We first bound the number of scaling phases with pushes. We claim that there is at least one quasi-medium or quasi-contracted node (as defined in the proof of the Contraction Lemma) per scaling phase with a push. Therefore, there are $O(m \log_k n)$ scaling phases. We see why as follows. Let i be the node with the first push in the Δ-scaling phase. Let $e(i)$ denote its excess at the beginning of the Δ-scaling phase, which is also its excess at the end of the previous scaling phase. Let $\Delta' = k\Delta$ be the scaling parameter at the previous scaling phase. By assumption, $e(i) \mod \frac{\Delta'}{k} < \frac{\Delta'}{k} - \frac{\Delta'}{M^2}$. Because the first push is from node i at the Δ-scaling phase, $e(i) \geq \frac{\Delta}{k} = \frac{\Delta'}{k^2}$. These two inequalities imply that $D(e(i), \frac{\Delta'}{k}) > \frac{\Delta'}{M^2}$. It follows from the Contraction Lemma that node i is either quasi-medium or quasi-contracted at the Δ'-scaling phase (and also at the Δ-scaling phase.)

We next bound the number of large and medium pushes. Let $n_1(j)$ be the number of quasi-medium and quasi-contracted nodes at the j-th scaling phase. Let $n_2(j) = n - n_1(j)$ be the number of remaining nodes at the j-th scaling phase. Let Δ_j denote the scaling parameter. Let $e_j(i)$ denote the excess at node i at the beginning of the j-th scaling phase.

We first analyze the number of large pushes using the potential function Φ_1. The increase in Φ_1 due to distance label increases is $O(n^2)$ over all scaling phases. We now bound the increase in Φ_1 due to the start of the j-th scaling phase. This increase is at most $\Phi_1(j)$, which is defined at the beginning of the j-th scaling phase as:

$$\Phi_1(j) = \sum_{i \in N} \frac{e_j(i) d_j(i)}{\Delta_j} \leq n \sum_{i \in N} \frac{e_j(i)}{\Delta_j} \leq (n \times n_1(j)) + (n \times \frac{n_2(j)}{M^2}) < n \times n_1(j) + 1.$$

The increase in Φ at the beginning of all scaling phases with at least one push, is at most the sum of $\Phi_1(j)$, which is $O(nm \log_k n)$.

We now analyze the number of medium pushes. We will refer to a node i as "peculiar" if (1) $e_j(i) < \frac{\Delta_j}{M^2}$ at the beginning of the scaling phase and (2) node i is neither quasi-medium nor quasi-contracted.

We use the following fact: if node i is peculiar, then every push into or out of node i is either a multiple of $\frac{\Delta_j}{k}$ or it is a push in a small capacity arc. Suppose that the excess at node i is less than $\frac{\Delta_j}{k}$ at some iteration during the phase. Taking into account the possibility of as many as $n-1$ pushes into node i from small capacity arcs, we conclude that $e(i) \leq \frac{(n+1)\Delta_j}{M^2}$.

Now let us analyze Φ_2. At the start of a scaling phase, $\Phi_2 = 0$. Every medium push reduces Φ_2 by at least $\frac{1}{k}$. So, it suffices to consider increases in Φ_2. These increases occur when (1) distance labels increase, or (2) ℓ decreases, or (3) a node is added to P. The increases in Φ_2 due to distance relabels is $O(n^2) = O(\frac{nm}{k})$.

We now consider the impact on Φ_2 of decreases in ℓ. The contribution of each peculiar node to the increase in Φ_2 at each decrease of ℓ is at most $\frac{n+1}{M^2}$, and thus the total contribution of peculiar nodes to the increase in Φ_2 over the phase is less than 1. The contribution by each other node to the increase in Φ_2 at each decrease of ℓ is at most $\frac{1}{k}$. Thus, the total increase in Φ_2 over a scaling phase due to decreases in ℓ is at most $O(\frac{n(j) \times n}{k})$. Summing over all phases, the total increase in Φ_2 due to decreases in ℓ is $O(\frac{nm \log_k n}{k})$.

We now consider the increases in Φ_2 over a specific scaling phase due to the times that a node is added to P. If a node i is added to P following a relabel of node i, we account for the increase in Φ_2 by charging it the increase to the relabel, which we have already bounded. So, we now consider cases in which i is added to P when $d(i) = \ell$. At the time that node i is added to P, $e(i) < \frac{\Delta}{k}$. The increase in Φ_2 is less than $\frac{1}{k}$. The total increases in Φ_2 due to additions to P is $O(\frac{n}{k})$ per scaling phase, and thus at most $O(\frac{nm \log_k n}{k})$ over all scaling phases in which there is a push.

Finally, we consider the effect of contractions. The increase in Φ_1 and Φ_2 following contractions can all be attributed to increases because of relabels of nodes, and this has already been accounted for.

Therefore, the total increase in Φ_2 over all phases due to decreases in ℓ is $O(\frac{nm \log_k n}{k})$. This implies that the total number of medium pushes over all phases is $O(nm \log_k n)$. □

A.8. **Lemma 8.1:** Suppose that node i satisfies the first two conditions for a special push, and that $\mathit{eligible}(i) = \emptyset$. Then node i will be contracted in $O(\log_k n)$ scaling phases.

Proof. We assume that $eligible(i) = \emptyset$. This implies that node i is not incident to an anti-abundant arc (i, j) with $r_{ij} > 0$. (The arc (j, i) would be eligible.) By the Contraction Lemma, node i will be contracted within $O(\log_k n)$ scaling phases. □

A.9. **Lemma 8.2:** Suppose that node i satisfies the first three conditions for a special push, and suppose that there is an abundant path P from node i to node $eligible(i)$. Let v_C be the pseudo-node created upon contracting the cycle C consisting of P and $(eligible(i), i)$. Suppose further that every node of C has excess at least $-\frac{n\Delta}{M}$. Then $e(v_C) > \frac{\Delta}{2n}$.

Proof. Prior to the contraction, $e(i) \geq \frac{\Delta}{k} - \frac{\Delta}{M^2}$. Every other node of the cycle has excess at least $-\frac{n\Delta}{M}$ The result follows. □

A.10. **Lemma 8.3.** Suppose that UpdateScalingParameter is called at the beginning of each scaling phase. Then Invariant 2 is satisfied following any change in scaling parameter. The total number of scaling phases is $O(m \log_k n)$.

Proof. Let Δ be the scaling parameter prior to calling UpdateScalingParameter. Suppose that $\Delta' = \min\{\Delta_1, \Delta_2\}$ is the scaling parameter after the update. We note that $\frac{\Delta}{k} \geq M\Delta'$. This is because $\Delta' \leq \Delta_1 = max\{e(i) : \frac{e(i)}{k} \geq 0\} \leq \frac{\Delta}{kM}$, and $\Delta' \leq \Delta_2 = max\{\frac{-e(i)}{kM} : e(i) \leq 0\} \leq \frac{\Delta}{kM}$.

To see that Invariant 2 remains satisfied, consider an arc (i, j) such that $0 < r_{ij} < r_{ji}$. We claim that (i, j) will become doubly abundant (and contracted) after the change in scaling parameter. By Invariant 2, $r_{ji} > r_{ij} \geq \frac{\Delta}{k}$. Because $\frac{\Delta}{k} \geq M\Delta'$, it follows that $r_{ji} > r_{ij} \geq M\Delta'$. Thus, in the Δ'-scaling phase, arc (i, j) is doubly abundant.

We now bound the number of scaling phases. The number of scaling phases in which there is a non-special node, medium capacity arc, or a node in F is $O(m \log_k n)$. Consider the remaining phases. If there is a node i with $\frac{\Delta}{kM^2} < |e(i)| < \frac{\Delta}{k}$, then within $O(\log_k n)$ phases the node will become strongly deficient or non-special. Thus this occurs $O(m \log_k n)$times. Finally, we consider the case in which $|e(i)| \leq \frac{\Delta}{kM^2}$. In this case, the procedure UpdateScalingParameter modifies the scaling parameter. After the update, $\frac{\Delta'}{M} < |e(i)| < \frac{\Delta'}{k}$, which reduces to the previous case. This completes the proof. □

A.11. **Lemma 9.1:** The total number of non-special nodes at the end of all scaling phases is $O(nm \log_k n)$.

Proof. The number of quasi-medium nodes and quasi-contracted nodes is $O(nm \log_k n)$. We claim that every non-special node is quasi-medium or quasi-contracted.

Consider a non-special node i at the end of a scaling phase. Thus $\frac{\Delta}{2n} \leq e(i) < \frac{\Delta}{k} + \frac{\Delta}{2n}$. If node i is incident to a medium arc, then it is quasi-medium. If node i has no incident anti-abundant arcs with positive residual capacity, then i is quasi-contracted. So, suppose that neither of these two conditions is satisfied. In that case, node i satisfies conditions (2) and (3) of the special push conditions. In addition, we have assumed that at the end of the phase, $e(i) < \frac{\Delta}{k} + \frac{\Delta}{2n}$. If it is also true that $e(i) \geq \frac{\Delta}{k} - \frac{\Delta}{M^2}$, then node i satisfies the first three conditions of the special push conditions, and there would be either a special push or a contraction. So, we may assume that $\frac{\Delta}{2n} \leq e(i) < \frac{\Delta}{k} - \frac{\Delta}{M^2}$. But, in this case, $D(e(i), \frac{\Delta}{k}) > \frac{\Delta}{M^2}$. Therefore, node i must be quasi-contracted, completing the proof. □

A.12. **Lemma 9.2:** The number of times that a strongly deficient node enters F is $O(m)$.

Proof. When we refer to a strongly deficient node of F, we consider nodes that were strongly deficient when added to F as well as nodes that became strongly deficient after being added to F. We will show that the number of times a strongly deficient node leaves F is $O(m)$. This is equivalent to the statement of the lemma.

If a strongly deficient node $i \in F$ is incident to a medium capacity arc (i, j) when it entered F or when it left F, we will "charge" the leaving of F to one the medium capacity arcs incident to node i. We claim that each arc (i, j) can be charged at most twice in this manner, once for node i and once for node j. We see why as follows. Prior to node i reentering F, node i must become special, and this must be preceded by a special push from node i. In order to satisfy the special push conditions, arc (i,j) must have become large capacity. Since arc (i,j) has become large capacity prior to node i reentering F, (i,j) cannot be charged when node i leaves F the next time.

We now consider nodes in F that were not incident to a medium capacity arc. Suppose that i was strongly deficient when it entered F and it was not incident to a medium capacity arc when it entered or left F. When node i leaves F, $\frac{\Delta}{2n} \leq e(i) < \frac{\Delta}{2k}$. By the Contraction Lemma, node i will become incident to a medium capacity arc in less than $6 \log_k m$ scaling phases, or else it will become incident to a doubly abundant arc and it will be contracted. In the former case, we also charge the leaving of node i from F to an (i,j) that becomes medium capacity. In the latter case, we charge the leaving of F to the contraction. We claim that each arc (i,j) can be charged at most 12 times in this manner, six times for node i and six times for node j. And the deletion of node i from F can be charged at most six times to an upcoming contraction.

If node i reenters F, it must first become special and deficient, and then become strongly deficient. It takes at least $\log_k \frac{M}{n}$ scaling phases from the time a node first becomes deficient (where $e(i) \geq -\frac{\Delta}{M^2}$) to the time it becomes strongly deficient. This number is greater than $\log_k m$. If node i reenters F six more times without being incident to a medium capacity arc, then node i will be incident to a doubly abundant arc prior to it having the opportunity to reenter F again. So, each arc and each contraction is charged at most six times, completing the proof. □

A.13. **Theorem 9.3.** The LM excess scaling algorithm with the flow-return forest and contraction determines a maximum flow in $O(nm \log_k n)$ time.

Proof. By Lemma 8.3, the algorithm terminates in $O(m \log_k n)$ scaling phases. The only way for the algorithm to terminate is when all excesses are 0. At this point, the flow is feasible and optimal in the contracted graph. This is identified in the procedure "Update Scaling Parameter." Then the contracted graph is expanded, and an optimal flow for the original network is obtained.

To show that the running time is $O(nm \log_k n)$, we consider the various types of operations of the algorithm.

(1) Small pushes. The number of small pushes is $O(nm)$ by Lemma 6.1.
(2) Medium and large pushes. We will prove below that the number of medium and small pushes $O(nm \log_k n)$.
(3) Selection of nodes for pushing. The time is proportional to the number of pushes.
(4) Contractions. This takes $O(nm)$ time by Lemma 7.1.
(5) Operations on the flow-return forest. These operations take $O(nm)$ time in total, as stated at the end of Subsection 8.6.
(6) Scaling phases in which there are no pushes. The total time spent on these phases is $O(n)$ per phase and $O(nm \log_k n)$ in total. The $O(n)$ time per phase is for checking whether the scaling parameter needs to be adjusted. This could be sped up if it were the bottleneck.

We now bound the number of medium and large pushes by modifying the potential function arguments given in Section 7. The modifications are required because of the flow-return forest.

Let R be the set of root nodes of the flow-return forest F. We define the *adjusted excess* $\hat{e}$ of nodes as follows: (i) if $i \in R$, then $\hat{e}(i) = e(i) - \Gamma(i)$; (ii) if $i \in F \setminus R$, then $\hat{e}(i) = e(i) + \gamma(i)$, (iii) for all other nodes i, $\hat{e}(i) = e(i)$.

The modified potential functions Φ_1' and Φ_2' can be obtained from Φ_1 and Φ_2 by replacing node excesses by their adjusted excesses. That is,

$$\Phi_1' := \sum_{i \in N} \frac{\hat{e}(i) d(i)}{\Delta}.$$

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

Our analysis of Φ'_1 and Φ'_2 will rely on our previous analysis of Φ_1 and Φ_2 in the proof of Theorem 7.4. We first bound the number of large pushes by Φ'_1.

In the proof of Theorem 7.4, we have already established that the increase in Φ'_1 due to relabels and due to the excesses at the beginning of scaling phases is $O(nm \log_k n)$ except for nodes i such that (1) i is neither quasi-medium nor quasi-contracted and (2) $e(i) \mod \frac{\Delta}{k} > \frac{\Delta}{k} - \frac{\Delta}{M^2}$. Any such node i satisfies the first three conditions of a special push. At the end of the scaling phase, either node i is part of a contracted abundant cycle or there is a special push from node i and $e(i) < 0$. In either case, node i does not contribute to Φ'_1 at the next scaling phase, nor at any subsequent scaling phase until it is either quasi-medium or quasi-contracted.

The remaining contributions to the increase in Φ'_1 are due to operations on the flow-return forest, which we will return to below.

We now bound the number of medium (and special) pushes. Our analysis proceeds as in our analysis of Φ_1. By Theorem 7.4, we have already proved the result except for nodes that are neither quasi-medium nor quasi-contracted and such that $e(i) \mod \frac{\Delta}{k} > \frac{\Delta}{k} - \frac{\Delta}{M^2}$. As above, such nodes are contracted or their excess becomes negative. As such, these nodes do not contribute to an increase in Φ'_2 when distance labels are increased, or when P increases in size, or when ℓ decreases. The increases in Φ'_2 accounted for so far are $O(\frac{nm \log_k n}{k})$ over all scaling phases. These would account for $O(nm \log_k n)$ medium pushes. The remaining contributions to the increase in Φ'_2 are due to operations on the flow-return forest.

We are left with the impact from operations involving the flow-return forest.

Each push or pull sends $\frac{\Delta}{k}$ units of flow from one node i of F to a leaf node j of F. It is possible that $d(j) > d(i)$ because arcs of F are not required to be admissible. Thus, Φ'_1 and Φ'_2 can both increase, but by less than $\frac{n}{k}$. It is also possible for Φ'_1 and Φ'_2 to increase by less than $\frac{n}{k}$ when γ and Γ are modified following a push. Since the total number of operations on F is bounded by m, the total increase in Φ'_1 and Φ'_2 due to operations on F is at most $O(\frac{nm}{k})$. This accounts for an additional $O(\frac{nm}{k})$ large pushes, and an additional $O(nm)$ medium pushes. This completes the proof. □

References

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

[2] R.K. Ahuja and J.B. Orlin. A fast and simple algorithm for the maximum flow problem. *Operations Research*, 37(5):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(5):939–954, 1989.

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

[5] A.V. Goldberg and S. Rao. Beyond the flow decomposition barrier. *J. ACM*, 45(5):783–797, September 1998.

[6] A.V. Goldberg and R.E. Tarjan. A new approach to the maximum-flow problem. *J. ACM*, 35(4):921–940, October 1988.

[7] V. King, S. Rao, and R.E. Tarjan. A faster deterministic maximum flow algorithm. In *Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '92, pages 157–164, Philadelphia, PA, USA, 1992. Society for Industrial and Applied Mathematics.

[8] J.B. Orlin. A faster strongly polynomial minimum cost flow algorithm. *Operations Research*, 41(2):338–350, 1993.

[9] J.B. Orlin. Max flows in O(nm) time, or better. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, STOC '13, pages 765–774, New York, NY, USA, 2013. ACM.

[10] D.D. Sleator and R.E. Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362 – 391, 1983.