Beyond the Flow Decomposition Barrier

ANDREW V. GOLDBERG AND SATISH RAO

NEC Research Institute, Princeton, New Jersey

Abstract. We introduce a new approach to the maximum flow problem. This approach is based on assigning arc lengths based on the residual flow value and the residual arc capacities. Our approach leads to an $O(\min(n^{2/3}, m^{1/2})m \log(n^2/m) \log U)$ time bound for a network with n vertices, m arcs, and integral arc capacities in the range $[1, \ldots, U]$. This is a fundamental improvement over the previous time bounds. We also improve bounds for the Gomory–Hu tree problem, the parametric flow problem, and the approximate s-t cut problem.

Categories and Subject Descriptors: F.2.2, [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems; G.2.2 [**Discrete Mathematics**]: Graph Theory

General Terms: Algorithms

Additional Key Words and Phrases: Combinatorial optimization, maximum flows

1. *Introduction*

The maximum flow problem and its dual, the minimum cut problem, are classical combinatorial problems with a wide variety of scientific and engineering applications. The maximum flow problem and related flow and cut problems have been studied intensively for over three decades.

The network simplex method of Dantzig [1951] for the transportation problem solves the maximum flow problem as a natural special case. Soon thereafter, Ford and Fulkerson [1956] developed the augmenting path method for the maximum flow problem. A natural variant of this method, the shortest augmenting path algorithm, was shown to be polynomial by Dinitz [1970] and Edmonds and Karp [1972]. Capacity scaling, developed by Edmonds and Karp [1972] and Dinitz [1973], also gives polynomial algorithms for the problem. (See also Gabow [1985].) Classical books [Adel'son-Vel'ski et al. 1975; Ford and Fulkerson 1962] describe earlier work in detail.

Most efficient algorithms for the maximum flow problem are based on the blocking flow and the push-relabel methods. The first blocking flow algorithm was developed by Dinitz [1970] in the framework of the augmenting path approach. Karzanov [1974] was the first to state the finding of a blocking flow as

Authors' address: NEC Research Institute, 4 Independence Drive, Princeton, NJ 08540; e-mail: [avg, satish]@research.nj.nec.com.

a separate problem and to suggest the use of preflows to solve it. The push-relabel method, implicit in Goldberg's algorithm [1985], was fully developed by Goldberg and Tarjan [1988].

The shortest augmenting path algorithm, the blocking flow method, and the push-relabel method use a concept of distance. To talk about distances, one has to define arc lengths. All these approaches use the unit length function: the length of every residual arc is defined to be one. Edmonds and Karp [1972] discussed the use of general length functions in the context of maximum flows. Wallacher and Zimmermann [1991] and Wallacher [1991] study length functions in the context of the minimum-cost flow problem. However, prior to our work, the bounds obtained using general length functions were no better than the bounds using the unit length function.

In this paper, we extend the blocking flow method to more general length functions and obtain substationally better time bounds. We study a *binary length function* that assigns zero length to large capacity arcs and unit length to small capacity arcs. A novel feature of our length function is that it is *adaptive*: we define the length threshold relative to an estimate on the residual flow value. Adaptivity is crucial for the time bound improvements.

Table I gives the history of the maximum flow time bounds. We denote the number of vertices in the input network by n and the number of arcs by m. For polynomial algorithms, we assume that arc capacities are integers in the range $[1, \dots, U]$. Strongly polynomial algorithms do not need this assumption. We use the *similarity assumption* of Gabow [1985], $\log U = O(\log n)$, to compare polynomial and strongly polynomial algorithms. By a *ballpark* bound we mean an O^* bound under the similarity assumption.[1] For a randomized algorithm running in $O(t)$ expected time, we denote its running time by $E(t)$.

The $\Omega(nm)$ bound is a natural barrier for maximum flow algorithms. In a path decomposition of a flow, the total path length is $\Theta(nm)$ in the worst case. This implies an $\Omega(nm)$ lower bound on algorithms which output explicit flow decomposition and on algorithms which augment flow one path at a time and, for each augmenting path, one arc at a time. This lower bound does not apply to algorithms that work with preflows or use data structures like dynamic trees [Sleator and Tarjan 1983] to manipulate flows. However, in spite of numerous attempts, no previous algorithm achieves this lower bound in general. For dense graphs, the $O(n^3/\log n)$ algorithm of Cheriyan et al. [1990] beats this lower bound, but only by using word operations on $(\log n)$-bit integers. The ballpark bound of nm was achieved by Dinitz [1973].

In the unit capacity case, the total decomposition path length is $O(m)$ and $o(nm)$ bounds have been known for a long time: Karzanov [1973] and independently Even and Tarjan [1975] have shown that Dinitz's algorithm [Dinic 1970] runs in $O(\min(n^{2/3}, m^{1/2})m)$ time on unit capacity networks with no parallel arcs. (We define $\Lambda = \min(n^{2/3}, m^{1/2})$; this term will appear quite often.) This bound is based on the $O(\Lambda)$ bound on the number of blocking flow computations of Dinitz's algorithm on such networks. This result suggests a possibility of an $o(nm)$ bound for general networks. However, for over two decades, no algorithm improved upon the nm ballpark bound.

[1] $O^*(f(n)) = O((\log n)^{O(1)}f(n))$.

TABLE I. HISTORY OF MAXIMUM FLOW BOUNDS.

#	Year	Discoverer(s)	Bounds		Reference
			Exact	Ballpark	
1	1951	Dantzig	$O(n^2mU)$	n^2mU	[Dantzig 1951]
2	1955	Ford & Fulkerson	$O(nmU)$	nmU	[Ford and Fulkerson 1956]
3	1970	Dinitz Edmonds & Karp	$O(nm^2)$	nm^2	[Dinitz 1970] [Edmonds and Karp 1972]
4	1970	Dinitz	$O(n^2m)$	n^2m	[Dinitz 1970]
5	1972	Edmonds & Karp Dinitz	$O(m^2 \log U)$	m^2	[Edmonds and Karp 1972] [Dinitz 1973]
6	1973	Dinitz Gabow	$O(nm \log U)$	nm	[Dinitz 1973] [Gabow 1985]
7	1974	Karzanov	$O(n^3)$		[Karzanov 1974]
8	1977	Cherkassky	$O(n^2\sqrt{m})$		[Cherkassky 1977]
9	1980	Galil & Naamad	$O(nm \log^2 n)$		[Galil and Naamad 1980]
10	1983	Sleator & Tarjan	$O(nm \log n)$		[Sleator and Tarjan 1983]
11	1986	Goldberg & Tarjan	$O(nm \log(n^2/m))$		[Goldberg and Tarjan 1988]
12	1987	Ahuja & Orlin	$O(nm + n^2 \log U)$		[Ahuja and Orlin 1989]
13	1987	Ahuja et al.	$O(nm \log(n\sqrt{\log U}/(m + 2))$		[Ahuja et al. 1989]
14	1989	Cheriyan & Hagerup	$E(nm + n^2 \log^2 n)$		[Cheriyan and Hagerup 1995]
15	1990	Cheriyan et al.	$O(n^3/\log n)$		[Cheriyan et al. 1996]
16	1990	Alon	$O(nm + n^{8/3} \log n)$		[Alon 1990]
17	1992	King et al.	$O(nm + n^{2+\epsilon})$		[King et al. 1992]
18	1993	Phillips & Westbrook	$O(nm(\log_{m/n} n + \log^{2+\epsilon} n))$		[Phillips and Westbrook 1993]
19	1994	King et al.	$O(nm \log_{m/(n \log n)} n)$		[King et al. 1994]
20	1998	Goldberg & Rao	$O(m^{3/2} \log(n^2/m)\log U)$ $O(n^{2/3}m \log(n^2/m)\log U)$	$m^{3/2}$ $n^{2/3}m$	this paper this paper

NOTE: We list only the bounds which in some way improve the previous bounds. We state the original publication date in a publicly available conference proceedings, technical report, or journal, but cite the most complete publications. In the case of independent discoveries, we give the date of the first one.

We achieve a Λm ballpark bound for general networks. Each iteration of our algorithm is dominated by a blocking flow computation on an acyclic graph. Using the blocking flow algorithm of Goldberg and Tarjan [1988], we get an $O(\Lambda m \log(n^2/m)\log U)$ bound. This result breaks the nm barrier by a polynomial factor. Consider sparse networks with $m = O(n)$, which come up in practical applications, (e.g., [Middleton 1995; Ogielski 1986; Roy and Cox 1998]). On these networks, our bound is better than the best previous bounds by a factor of $\Omega(\sqrt{n}/\log U)$.

In addition to direct applications, maximum flow algorithms are used as subroutines in other algorithms. Our results imply better bounds for those algorithms dominated by maximum flow computations. For example, a Gomory–Hu tree [1961] can be constructed in $n - 1$ maximum flow computations [Gomory and Hu 1961; Gusfield 1990], so we improve the corresponding time bound from the ballpark of n^2m to Λnm. For parametric flows, the combination of our results and those of Gallo et al. [1989] gives an $O(\Lambda m \log(n^2/m)\log^2 U)$ time bound.

Our bounds can be improved if an approximate solution is sufficient. We can find a flow and a cut, such that the cut capacity is within a factor of $(1 + \epsilon)$ of the flow value, in $O(\Lambda m \log(n^2/m)\log(m/\epsilon))$ time. Note that this time bound does not depend on U.

In the special case of undirected graphs, Benczúr and Karger [1996] show that a cut with capacity of at most $(1 + \epsilon)$ of optimal can be constructed in $E^*(n^2/\epsilon^2)$ time using nonuniform sampling. The combination of our results and their nonuniform sampling technique gives $E^*(\min(\log U, \log(1/\epsilon))n^{5/3}/\epsilon^2 + m)$ and $E^*(\min(\log U, \log(1/\epsilon))n^{3/2}/\epsilon^3 + m)$ bounds. For example, if ϵ is a constant, we get an $E^*(n^{3/2} + m)$ time bound.

This paper is organized as follows: We start by giving basic definitions and background results in Section 2. In Section 3, we introduce length functions for maximum flow computations and prove fundamental theorems about them. We study a binary length function and introduce the binary blocking flow algorithm in Section 4. Section 5 gives technical details of the algorithm. We analyze the algorithm in Section 6. In Section 7, we study approximation algorithms for finding maximum flows and minimum cuts. Section 8 discusses more general length functions. Section 9 contains concluding remarks.

2. *Background*

A *flow network* consists of a directed graph $G = (V, E)$, a *source* vertex s, a *sink* vertex t, and an integral *capacity function* u: $E \rightarrow \{1, \ldots, U\}$.[2] We refer to $u(a)$ as the capacity of the arc a.

We assume the adjacency list representation of the input graph. We also assume that each arc has a pointer to the reverse arc. We refer to elements of V in the input graph as *vertices*. By *nodes* we mean sets of vertices that arise from graph contraction operations. Note that a graph can have vertices and nodes. We use the term, *node*, unless we know that the element corresponds to a single vertex of the input graph.

We denote the set of real numbers by $\mathbf{R}$ and the set of nonnegative real numbers by $\mathbf{R}^+$.

A flow in a flow network is a function f: $E \rightarrow \{1, \ldots, U\}$ where for each arc, the *capacity constraint* $f(a) \leq u(a)$ holds and for each vertex $j \in V - \{s, t\}$, the

[2] Unless mentioned otherwise, we work with integral capacities and flows.

conservation constraint $\Sigma_{(j,k)} f(j, k) - \Sigma_{(i,j)} f(i, j) = 0$ holds. The value of the flow is $|f| = \Sigma_{(j,t)} f(j, t)$. We refer to $f(a)$ as the *flow* on arc a.

Without loss of generality, we assume that the graph has no parallel arcs. This allows us to uniquely specify an arc by its endpoints. If an arc (i, j) is in the graph but the reverse arc (j, i) is not, we add (j, i) to the graph and define $u(j, i) = 0$. For an arc a, a^R denotes the reverse arc. Without loss of generality, we assume that either $f(a) = 0$ or $f(a^R) = 0$.

The residual capacity of an arc (i, j), $u_f(i, j)$, with respect to flow f is defined to be $u(i, j) - f(i, j) + f(j, i)$. We say that (i, j) is a *residual arc* if it has strictly positive residual capacity, and denote the set of residual arcs by E_f. The *residual graph* is the graph (V, E_f). A *residual flow* is the difference between an optimal flow f^* and the current flow f: if $f^*(a) \geq f(a)$, then the residual flow on a is $f^*(a) - f(a)$, and otherwise the residual flow on a^R is $f(a) - f^*(a)$.

A *blocking flow* in a directed flow network is a flow f where every directed s-t path contains an arc with zero residual capacity. The problem of finding blocking flows in layered networks has been introduced by Dinitz [1970] in the context of maximum flow algorithms. For more general acyclic networks, the problem was studied by Goldberg and Tarjan [1990] in the context of minimum cost flows. Many, but not all, algorithms for layered networks work on acyclic networks and achieve the same time bounds. Our maximum flow algorithms need a subroutine for finding blocking flows in acyclic networks. The fastest known algorithm for the problem runs in $O(m \log(n^2/m))$ time [Goldberg and Tarjan 1995].

3. *Length Functions*

By a *length function*, we mean a function $\ell: A \rightarrow \mathbf{R}^+$. We say that a function $d: V \rightarrow \mathbf{R}^+$ is a *distance labeling* with respect to ℓ if $d(t) = 0$ and for every arc $(i, j) \in A$, the *reduced cost* of (i, j), defined by $\ell_d(i, j) = \ell(i, j) + d(j) - d(i)$, is nonnegative. Since $d(t) = 0$ and a sum of reduced costs of arcs on a path telescopes, for a path Γ from i to t we have $\ell(\Gamma) = d(i) + \ell_d(\Gamma) \geq d(i)$. Given a length function ℓ on arcs and a vertex i, we define $d_\ell(i)$ to be the distance with respect to ℓ from i to t in G. Note that d_ℓ is a distance labeling. In fact, it is the "biggest" distance labeling: for any distance labeling d, $d \leq d_\ell$.

A length function gives an upper bound on the residual flow value. A residual arc a has "width" $u_f(a)$, length $\ell(a)$, and *volume* $vol_{f,\ell}(a) = u_f(a)\ell(a)$ if $a \in E_f$ and 0, otherwise. The total volume of the network is defined by $Vol_{f,\ell} = \Sigma_a u_f(a)\ell(a)$. Since each unit of residual flow takes at least $d(s)$ units of volume, we have the following result:

LEMMA 3.1. *Given a flow f, a length function ℓ, and a distance labeling d such that $d(s) > 0$, the residual flow value is at most $Vol_{f,\ell}/d(s)$.*

Note that $d_\ell(s)$ yields the best upper bound in this context.

Given a flow f, a length function ℓ, and a distance labeling d, a residual arc (i, j) is *admissible* if $d(i) > d(j)$ or $d(i) = d(j)$ and $\ell(i, j) = 0$.[3] We denote the set of all admissible arcs by $A(f, \ell, d)$ and the induced admissible graph by $(V, A(f, \ell, d))$.

[3] For the binary length function, these conditions are equivalent to $d(i) = d(j) + \ell(i, j)$.

By allowing our length function to take on zero values on large capacity arcs, we keep $Vol_{f,\ell}$ small and get better bounds on the residual flow value than one gets using the unit length function. Zero length arcs, however, cause several technical problems which we need to overcome.

4. *Binary Blocking Flow Algorithm*

In this section, we introduce the *binary blocking flow algorithm* based on a binary length function. This length function is zero on arcs with large residual capacity and one on arcs with small residual capacity. The notions of small and large depend on the current residual flow value. We maintain an upper bound F on the residual flow value[4] and update F every time our value estimate improves by a factor of two (one can use a different constant factor here). We use $\Sigma_{(s,w)}\, u(s, w) \leq nU$ as the initial estimate of F. Since the capacities and flows are integral, as soon as F becomes less than one the algorithm terminates. A *phase* is the computation done by the algorithm between two consecutive updates of F. This implies an $O(\log(nU))$ bound on the number of phases. A phase consists of *update steps*.

The current value of F determines the value of the parameter Δ. Intuitively, Δ is the amount of flow we are trying to get to the sink in one update step of the algorithm. We maintain the following property:

Δ-*invariant.* *An update step changes flow through every vertex by at most* Δ.

The binary length function for the phase is based on the following definition:

$$\ell(a) = \begin{cases} 0 & \text{if } u_f(a) \geq 3\Delta \\ 1 & \text{otherwise.} \end{cases} \tag{1}$$

Length of some arcs, however, is modified so that the s to t distance increases after an augmentation of a blocking flow. We say that an arc (i, j) is *special* if $2\Delta \leq u_f(i, j) < 3\Delta$, $d(i) = d(j)$, and $u_f(j, i) \geq 3\Delta$. We define a new length function $\bar{\ell}$ that is equal to zero on special arcs and equal to ℓ on all other arcs. Note that this does not change the distances: $d_\ell = d_{\bar{\ell}}$.

At the beginning of each update step, we compute d_ℓ and $\bar{\ell}$. Note that the admissible graph can have cycles of zero-length arcs. We deal with this problem by contracting strongly connected components of the graph induced by zero-length arcs. If we do not change flow through any contracted vertex by more than Δ, a new flow in the contracted graph corresponds to a feasible flow in the original graph.

We use an algorithm for acyclic graphs which has the following property: it either computes a blocking flow or computes a flow of value Δ.[5] Since the contracted admissible graph is acyclic, this property implies the Δ-invariant. In Section 5, we show how to extend flow found in the contracted graph to the original graph.

Analysis of the blocking flow method is based on the fact that the source-to-sink distance increases at every iteration. For the binary blocking flow algorithm,

[4] If we know the residual flow value, we can use it instead of an upper bound. In this case, the algorithm becomes simpler.

[5] Later we show that any blocking flow algorithm for acyclic graphs can be used.

we prove a generalization of this result. Let f and ℓ be the flow and the length function immediately before an augmentation of f by a flow in $A(f, \ell, d_\ell)$ and let f' and ℓ' be the flow and the length function immediately after. Note that ℓ is modified on the special arcs and ℓ' is not. Then the following lemma holds.

LEMMA 4.1. *d_ℓ is a distance labeling with respect to ℓ'.*

PROOF. We need to show that for any arc (i, j), $d_\ell(j) + \ell'(i, j) \geq d_\ell(i)$. We know $d_\ell(j) + \ell(i, j) \geq d_\ell(i)$, and $\ell'(i, j) < \ell(i, j)$ only if (i, j) is not admissible for ℓ. In this case, $d_\ell(j) \geq d_\ell(i)$, thus $d_\ell(j) + \ell'(i, j) \geq d_\ell(i)$. □

COROLLARY 4.2. $d_\ell \leq d_{\ell'}$.

Our handling of the special arcs ensures the following key result.

THEOREM 4.3. *Suppose $f' - f$ is a blocking flow in $A(f, \ell, d_\ell)$. Then $d_\ell(s) < d_{\ell'}(s)$.*

PROOF. By Lemma 4.1, d_ℓ is a distance labeling with respect to ℓ'. Thus, the reduced costs $c(i, j) = d_\ell(j) + \ell'(i, j) - d_\ell(i)$ are nonnegative for all $(i, j) \in E_f$. Let Γ be a path from s to t in $G_{f'}$. (If there is no such path, the lemma is trivial.) Since $\ell'(\Gamma) = d_\ell(s) + c(\Gamma)$, it is enough to show that at least one arc (i, j) on Γ has $c(i, j) > 0$.

By the definitions of f' and blocking flow, Γ must contain an arc (i, j) such that $(i, j) \notin A(f, \ell, d_\ell)$. We claim that $c(i, j) > 0$. Since $(i, j) \notin A(f, \ell, d_\ell)$ and $(i, j) \in E_{f'}$, $d_\ell(i) \leq d_\ell(j)$. Thus, if $c(i, j) = 0$, then $d_\ell(i) = d_\ell(j)$ and $\ell'(i, j) = 0$. By the fact that $(i, j) \notin A(f, \ell, d_\ell)$, $\ell(i, j) > 0 = \ell'(i, j)$. This can only happen if $(j, i) \in A(f, \ell, d_\ell)$ and $\ell(j, i) = 0$. But then (i, j) must be a special arc by the Δ-invariant, contradicting the choice of (i, j). □

Finally, we need to describe how the algorithm updates F. This can be done in several ways. For example, one can use Lemma 3.1 and, when $Vol_{f,\ell}/(d_\ell(s) \leq F/2$, terminate the phase and set F to $Vol_{f,\ell}/d_\ell(s)$. We use a slightly different approach. We maintain a *current cut* (S, T) such that $u_f(S, T) = F$. When we find a cut (S', T') such that $u_f(S', T') \leq F/2$, we stop the phase, and set (S, T) to (S', T') and F to $u_f(S', T')$.

Maintaining the current cut has its advantages; in particular, it allows us to modify the algorithm to find approximately maximum flows. The resulting algorithm maintains a primal solution (flow) and a dual solution (cut), and a phase reduces the gap between the values of the two solutions.

Initially, $(\{s\}, V - \{s\})$ is the current cut. To update the cut we need the following definitions. For $k = 1, \ldots, d_\ell(s)$, let $S_k = \{v \in V: d(v) \geq k\}$ and let $T_k = V - S_k$. Note that for $k = 1, \ldots, d_\ell(s)$, (S_k, T_k) is an s-t cut. We call these cuts *canonical.* We can compute the capacity of every canonical cut in $O(m)$ time (see Section 5). Let $(\bar{S}, \bar{T})$ be a canonical cut with the smallest residual capacity. At the beginning of each update step, the algorithm examines these cuts and terminates the phase if $u_f(\bar{S}, \bar{T}) \leq F/2$.

Next we give a detailed description of the main loop of a phase. The goal of a phase is to decrease F by at least a factor of two. At the beginning of every iteration of a phase, we have ℓ and d_ℓ. An iteration of a phase works as follows.

(1) Compute $(\bar{S}, \bar{T})$. If $u_f(\bar{S}, \bar{T}) \leq F/2$, terminate the phase, update F and the current cut.

(2) Compute $\bar{\ell}$.
(3) Contract strongly connected components of $A(f, \bar{\ell}, d_{\bar{\ell}})$ induced by zero length arcs.
(4) In the resulting graph, find a blocking flow or a flow of value Δ.
(5) Augment the current flow by the flow found in the previous step and extend the resulting flow to the input graph.
(6) Compute ℓ and d_ℓ.

Correctness of this implementation is immediate.

5. *Bits and Pieces*

In this section, we show how to use any blocking flow algorithm for acyclic graphs in step (4) of our algorithm, and describe the details of graph contraction, reconstructing flow in the original graph, computing d_ℓ, and computing $(\bar{S}, \bar{T})$.

We can use any blocking flow algorithm for acyclic graphs as follows. Run the blocking flow algorithm to the end and let X be the value of the resulting flow. If $X > \Delta$, return extra flow to the source as follows. Place $X - \Delta$ units of excess at the sink. Process vertices in a reverse topological order; for the current vertex, reduce flow on the incoming arcs to eliminate its excess. This postprocessing takes $O(m)$ time.

We need to contract the strongly connected components at the beginning of a phase, and undo the contractions at the end of the phase. Each vertex participates in at most one contract operation before these operations are undone. This allows the use of simple data structures.

We represent a set S_r of contracted vertices by creating a *representative node r* that points to a linked list of the vertices in S_r. Each vertex in S_r points to r. Initially, all vertices are marked as uncontracted. To contract a set S of vertices, we create a node r and form a linked list of vertices in the set. The node r points to the linked list, and each vertex in the list points to r. The total contraction work during a phase is $O(n)$. It is easy to see that this representation of the contracted graph allows us to work with the contracted graph with a constant factor overhead compared to the adjacency list implementation. To undo the contraction operations, we mark each vertex as uncontracted.

With our representation of the contracted graph, the blocking flow computation on this graph changes flow on some arcs in the original graph. The flow conservation constraints hold at the nodes of the contracted graph but not at the vertices of the original graph. In other words, we need to route flow inside the strongly connected components. We compute for each vertex in the original graph its flow *balance* (i.e., the sum of the flow entering a vertex minus the sum of the flow leaving the vertex.) By the Δ-invariant, the absolute values of the total positive and total negative balance for a component are at most Δ each.

Consider a component not containing the source or the sink. The sum of the balances over the vertices in a single strongly connected component not containing the source and sink is zero since this component corresponds to a node in the blocking flow computation. We choose an arbitrary vertex in each strongly connected component as its root. Then we form an in-tree and an out-tree of this component. We route all of the positive balances to the root using the in-tree, and we route the resulting flow excess from the root to the negative balances using the out-tree. The resulting flow does not violate any arc capacity since each arc in either tree has residual capacity of at least 2Δ and at most Δ flow is routed to and from the root using such an arc.

Similar procedure applies to the source (sink) component, except the negative (positive) balance is less than the positive (negative) balance and we choose the source (sink) as the root of the trees.

Remark. The above procedure requires strongly connected component arc capacities to be at least 2Δ to route the total of Δ flow through. Since we also need a slack of Δ for the special arcs, we used 3Δ in (1). If we use the wheels within wheels data structure of Knuth [1974] and the disjoint union data structure [Tarjan 1975], we can route Δ units of flow through strongly connected graphs with arc capacities of Δ or more. In a graph with n vertices and m arcs, this takes $O(m\alpha(m, n))$ time.[6] This is more natural and allows the use of 2Δ instead of 3Δ in (1), but the resulting algorithm for routing flow through the components is more complicated.

Since the arc lengths are 0 and 1, we can compute d_ℓ in linear time, for example, using Dial's algorithm [Dial 1969].

We compute $u_f(S_k, T_k)$ for $k = 1, \ldots, d_\ell(s)$ as follows. For every k, we initialize $u_f(S_k, T_k)$ to zero. Then, we examine all arcs (i, j). If $d(i) > d(j)$, we increase $u_f(S_i, T_i)$ by $u_f(i, j)$. This procedure takes $O(m)$ time.

6. *Time Bounds*

In this section, we derive bounds on the binary blocking flow algorithms. Consider any 0–1 length function. Let M be the maximum residual capacity of a length one arc and let $(\bar{S}, \bar{T})$ be as defined in the previous section. We have the following variant of Lemma 3.1.

LEMMA 6.1. *For a* 0–1 *length function d_ℓ on the residual graph,*

$$u_f(\bar{S}, \bar{T}) \leq \frac{m}{d_\ell(s)} M.$$

PROOF. Since ℓ is a 0–1 function, each arc crosses at most one of the canonical cuts and for every arc a crossing such a cut, $\ell(a) = 1$. The observation that the total capacity of length one arcs is at most mM completes the proof. □

For dense graphs, the following lemma gives better bounds.

LEMMA 6.2. *For a* 0–1 *length function d_ℓ on the residual graph,*

$$u_f(\bar{S}, \bar{T}) \leq \left(\frac{2n}{d_\ell(s)}\right)^2 M.$$

PROOF. Let V_k be the set of vertices at distance k from t. Since $\Sigma_{k=0}^{d_\ell(s)} |V_k| = n$, there are at most $d_\ell(s)/2$ values of k where $|V_k| > 2n/d_\ell(s)$ vertices. Thus, at least $\lceil d_\ell(s)/2 \rceil + 1$ of the $d_\ell(s) + 1$ values of k have $|V_k| \leq 2n/d_\ell(s)$. By the pigeonhole principle there is a j such that $|V_j| \leq 2n/d_\ell(s)$ and $|V_{j+1}| \leq$

[6] α is the inverse Ackerman's function.

$2n/d_\ell(s)$. Since all arcs from V_{j+1} to V_j have length one, $u_f(S_j, T_j) \leq (2n/d(s))^2 M$. □

To get the desired bounds, we set $\Delta = \min(\lceil F/m^{1/2} \rceil, \lceil F/n^{2/3} \rceil)$.

LEMMA 6.3. *For the binary blocking flow algorithm,*

—for $\Delta = \lceil F/m^{1/2} \rceil$, *a phase terminates in* $O(\sqrt{m})$ *update steps;*

—for $\Delta = \lceil F/n^{2/3} \rceil$, *a phase terminates in* $O(n^{2/3})$ *update steps.*

PROOF. In the first case, the number of update steps which increase $|f|$ by Δ or more is $O(m^{1/2})$ since the residual flow is $O(\Delta m^{1/2})$. Other iterations increase $d_\ell(s)$ by at least one. Using Lemma 6.1 and the fact that $M \leq 3\Delta$, we conclude that after $6\lceil m^{1/2} \rceil$ such update steps, $d_\ell(s) \geq 6m^{1/2}$ and

$$u_f(\bar{S}, \bar{T}) \leq \frac{3\Delta m}{d_\ell(s)} \leq \frac{3Fm}{6m^{1/2}m^{1/2}} = \frac{F}{2}.$$

Therefore, the phase terminates in $O(m^{1/2})$ update steps.

In the second case, the number of update steps which increase $|f|$ by Δ or more is $O(n^{2/3})$ since the residual flow is $O(\Delta n^{2/3})$. Other iterations increase $d_\ell(s)$ by at least one. Using Lemma 6.2 and the fact that $M \leq 3\Delta$, we conclude that after $5\lceil n^{2/3} \rceil$ of such update steps, $d_\ell(s) \geq 5n^{2/3}$ and

$$u_f(\bar{S}, \bar{T}) \leq \left(\frac{2n}{d_\ell(s)}\right)^2 3\Delta \leq \frac{12F}{25} < F/2.$$

Therefore, the phase terminates in $O(n^{2/3})$ update steps. □

Note that each update step is dominated by a blocking flow computation. Using an $O(m \log(n^2/m))$ blocking flow algorithm, we get our main result:

THEOREM 6.4. *The maximum flow problem can be solved in* $O(\Lambda m \log(n^2/m) \log U)$ *time.*

PROOF. Consider the phases from the first time $\Delta \leq U$ to the first time $\Delta = 1$. The number of such phases is $O(\log U)$ by definition of the phase.

Next we account for the update steps during the time $\Delta > U$. Throughout these update steps, all arc lengths are one, and every update step increases $d_\ell(s)$. Thus, after at most Λ such steps, $F \leq \Lambda U$ by Lemmas 6.1 and 6.2 and the definition of U.

Finally, when $\Delta = 1$, $F \leq \Lambda$, and the algorithm terminates in $O(\Lambda)$ update steps. □

Our results imply improved parallel bounds as well. Using the blocking flow algorithm of Vishkin [1992], we get the following parallel bound.

THEOREM 6.5. *The maximum flow problem can be solved in* $O(\Lambda n \log(n) \log U)$ *time on a PRAM with* $O(n^2)$ *memory and* $O(n)$ *processors.*

7. *Approximation Results*

The fact that our algorithms maintain a monotone bound on the residual flow value allows us to speed up the algorithms for the approximate maximum flow case. In particular, we can prove the following result.

THEOREM 7.1. *A flow and a cut such that the cut capacity is within a factor of* $(1 + \epsilon)$ *of the flow value can be found in* $O(\Lambda m \log(n^2/m) \log(m/\epsilon))$ *time.*

PROOF. Since F is an upper bound on the residual flow value, $|f| + F \geq |f^*|$. Therefore

$$|f|\left(1 + \frac{F}{|f|}\right) \geq |f^*|.$$

We show how to quickly obtain initial f and F such that $|f| \geq F/m$. Since $|f|$ is monotone and each phase reduces F by a factor of two, the theorem follows.

Consider a bottleneck shortest path in G with respect to the length function $1/u$. We can find such a path in $O^*(m)$ time (see, e.g., Tarjan [1983]). Let δ be the smallest arc capacity on the path. We set the initial flow f to be δ at the path arcs and zero everywhere else. Consider the set S of vertices reachable from s via paths of arcs with capacity greater than δ. We can find this set in $O(m)$ time. This set defines an s-t cut of capacity at most $m\delta$. Thus, we set $F = m\delta$. $\square$

For the rest of this section, we consider the special case of undirected graphs. For this case, Benczúr and Karger [1996] introduce a nonuniform random sampling technique that has the following properties.

LEMMA 7.2. [BENCZÚR AND KARGER 1996]. *Given a graph* G *and an error parameter* ϵ, *in* $O(m \log^3 n)$ *time, we can construct a graph* G' *such that with high probability,* G' *has the following properties:*

—*G' has* $O((n \log n)/\epsilon^2)$ *edges and*

—*the value of every cut in* G' *is* $(1 \pm \epsilon)$ *times the value of the corresponding cut in* G.

They use this lemma in combination with the maximum flow algorithm of Goldberg and Tarjan [1988] to get an $E^*(n^2/\epsilon^2)$ algorithm to find a cut with capacity at most $(1 + \epsilon)$ times the minimum cut capacity. We get an improvement using the algorithm in this paper instead of that of Goldberg and Tarjan [1988]. We can get a slightly different bound using Theorem 7.1. We set $\epsilon' = \epsilon/2$ and do the sampling with the parameter ϵ'. We also use ϵ' instead of ϵ in our approximate flow algorithm. We summarize these approximation results as follows:

THEOREM 7.3. *A cut with capacity at most* $(1 + \epsilon)$ *of the minimum cut capacity can be found in* $E^*(\min(\log U, \log(1/\epsilon))n^{5/3}/\epsilon^2 + m)$ *or* $E^*(\min(\log U, \log(1/\epsilon)) n^{3/2}/\epsilon^3 + m)$ *time.*

8. *Generalizations*

In this section, we extend the conditions that the length function need to satisfy so that an augmentation by a blocking flow in the admissible graph increases the distance between the source and the sink. This generalization identifies important length function properties and may be useful for development of more general or more practical results.

Consider an augmentation of a flow by a flow in the admissible graph. Let f and f' be the flow before and after the augmentation, respectively, and let ℓ and ℓ' be the length function before and after the augmentation, respectively. Note that we perform distance computations only on residual graphs. We can define length of arcs with zero residual capacity to be infinity.

We say that the tuple (f, ℓ, f', ℓ') is *proper* if for all $a \in A$

(1) $\ell(a) > \ell'(a) \notin A(f, \ell, d_\ell)$ & $a^R \in A(f, \ell, d_\ell)$,
(2) $\ell'(a) = 0 \Rightarrow \ell(a) = 0$ or $\ell(a^R) > 0$.

One can easily verify that the binary length function modified at the special arcs falls into this framework. Note that the second condition implies that if $u_f(a) > 0$, then $\ell'(a) > 0$.

The following results are generalizations of Lemma 4.1, Corollary 4.2, and Theorem 4.3. The proofs are similar.

LEMMA 8.1. *If (f, ℓ, f', ℓ') is proper, then d_ℓ is a distance labeling with respect to ℓ'.*

COROLLARY 8.2. *If (f, ℓ, f', ℓ') is proper, then $d_\ell \le d_{\ell'}$.*

THEOREM 8.3. *Suppose s and t are not contracted together in $A(f, \ell, d_\ell)$, $f' - f$ is a blocking flow in $A(f, \ell, d_\ell)$, and (f, ℓ, f', ℓ') is proper. Then $d_\ell(s) < d_{\ell'}(s)$.*

One can use results of this section to analyze algorithms based on a wide class of length functions. For example, one can use

$$\ell(a) = \left\lfloor \frac{3\Delta}{u_f(a)} \right\rfloor$$

instead of the binary length function to obtain the same time bounds.

9. *Concluding Remarks*

The best previous maximum flow implementations are based on the push-relabel method. These implementations are the result of intensive experimental research; (see, for example, Anderson and Setubal [1993], Cherkassky and Goldberg [1995], Derigs and Meier [1989], Goldberg [1994], and Nguyen and Venkateswaran [1993]). It would be interesting to see if the ideas introduced in this paper lead to practical improvements.

An interesting open problem is if the flow decomposition bound can be improved upon by a strongly polynomial algorithm. For example, is there an algorithm running in $O^*(\Lambda m)$ time?

Another open problem is to extend our results and to obtain better bounds for the minimum-cost flow problem. The results of Wallacher [1991] and Wallacher and Zimmerman [1991] may be relevant here.

Our ballpark bounds for directed capacitated flows cannot be improved without improving bounds for directed unit capacity flows. Recent improvements for the undirected unit capacity case [Goldberg and Rao 1997; Karger 1997] suggest a possibility of improving the directed case as well, although the techniques of Goldberg and Rao [1997] and Karger [1997] do not seem to apply directly.

ACKNOWLEDGMENTS. We would like to thank Bob Tarjan for insightful discussions, helpful comments, and for bringing the wheels within wheels data structure to our attention. We are grateful to Efim Dinitz and Alexander Karzanov for comments on a draft of this paper. Finally, we would like to thank Cliff Stein for bringing to our attention the fact that general length functions have been discussed in the Edmonds–Karp paper.

REFERENCES

ADEL'SON-VEL'SKI, G. M., DINIC, E. A., AND KARZANOV, A. V. 1975. *Flow Algorithms*. Nauka, Moscow, CIS (in Russian).

AHUJA, R. K., AND ORLIN, J. B. 1989. A fast and simple algorithm for the maximum flow problem. *Oper. Res. 37*, 748–759.

AHUJA, R. K., ORLIN, J. B., AND TARJAN, R. E. 1989. Improved time bounds for the maximum flow problem. *SIAM J. Comput. 18*, 939–954.

ALON, N. 1990. Generating pseudo-random permutations and maximum flow algorithms. *Inf. Proc. Lett. 35*, 201–204.

ANDERSON, R. J., AND SETUBAL, J. C. 1993. Goldberg's algorithm for the maximum flow in perspective: A computational study. In *Network Flows and Matching: First DIMACS Implementation Challenge*, D. S. Johnson and C. C. McGeoch, eds. American Mathematics Society, Providence, R.I., pp. 1–18.

BENCZÚR, A. A., AND KARGER, D. R. 1996. Approximating s–t minimum cuts in $\tilde{O}(n^2)$ time. In *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing* (Philadelphia, Pa., May 22–24). ACM, New York. pp. 47–55.

CHERIYAN, J., HAGERUP, T., AND MEHLHORN, K. 1996. $O(n^3)$-time maximum-flow algorithm. *SIAM J. Comput. 45*, 1144–1170.

CHERKASSKY, R. V. 1977. Algorithm for construction of maximal flows in networks with complexity of $O(V^2\sqrt{E})$ operations. *Math. Meth. Sol. Economic. Prob. 7*, 112–125 (in Russian).

CHERKASSKY, B. V., AND GOLDBERG, A. V. 1995. On implementing push-relabel method for the maximum flow problem. In *Proceedings of the 4th International Programming and Combinatorial Optimization Conference.* Lecture Notes in Computer Science. Springer-Verlag, New York, pp. 157–171.

DANTZIG, G. B. 1951. Application of the simplex method to a transportation problem. In *Activity Analysis and Production and Allocation.* Wiley, New York, pp. 359–373.

DERIGS, U., AND MEIER, W. 1989. Implementing Goldberg's Max-Flow algorithm—A computational investigation. *ZOR–Meth. Mod. Oper. Res. 33*, 383–403.

DIAL, R. B. 1969. Algorithm 360: Shortest-path forest with topological ordering. *Commun. ACM 12*, 11 (Nov.), 631–633.

DINIC, E. A. 1970. Algorithm for solution of a problem of maximum flow in networks with power estimation. *Soviet Math. Dokl. 11*, 1277–1280.

DINIC, E. A. 1973. Metod porazryadnogo sokrashcheniya nevyazok i transportnye zadachi. In *Issledovaniya po Diskretnoi Matematike.* Nauka, Moskva, CIS (in Russian; Title translation: Excess Scaling and Transportation Problems), pp. 46–57.

EDMONDS, J., AND KARP, R. M. 1972. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM 19*, 2 (Apr.), 248–264.

EVEN, S., AND TARJAN, R. E. 1975. Network flow and testing graph connectivity. *SIAM J. Comput. 4*, 507–518.

FORD, JR., L. R., AND FULKERSON, D. R. 1956. Maximal flow through a network. *Canad. J. Math. 8*, 399–404.

FORD, JR., L. R., AND FULKERSON, D. R. 1962. *Flows in Networks.* Princeton Univ. Press, Princeton, N.J.

GABOW, H. N. 1985. Scaling algorithms for network problems. *J. Comput. Syst. Sci. 31*, 148–168.

GALIL, Z., AND NAAMAD, A. 1980. An $O(EV \log^2 V)$ algorithm for the maximal flow problem. *J. Comput. Syst. Sci. 21*, 203–217.

GALLO, G., GRIGORIADIS, M. D., AND TARJAN, R. E. 1989. A fast parametric maximum flow algorithm. *SIAM J. Comput. 18*, 30–55.

GOLDBERG, A. V. 1985. A new Max-Flow algorithm. Tech Rep. MIT/LCS/TM-291. Laboratory for Computer Science, MIT, Cambridge, Mass.

GOLDBERG, A. V. 1987. Efficient graph algorithms for sequential and parallel computers. Ph.D. dissertation. MIT, Cambridge, Mass. (Also available as Tech Rep. TR-374, Laboratory for Computer Science, MIT, Cambridge, Mass., 1987.)

GOLDBERG, A. V., AND RAO, S. 1997. Flows in undirected unit capacity networks. In *Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science.* IEEE Computer Society Press, Los Alamitos, Calif., pp. 32–35.

GOLDBERG, A. V., AND TARJAN, R. E. 1988. A new approach to the maximum-flow problem. *J. ACM 35*, 4 (Oct.), 921–940.

GOLDBERG, A. V., AND TARJAN, R. E. 1990. Finding minimum-cost circulations by successive approximation. *Math. Oper. Res. 15*, 430–466.

GOMORY, R. E., AND HU, T. C. 1961. Multi-terminal network flows. *J. SIAM 9*, 551–570.

GUSFIELD, D. 1990. Very simple methods for all pairs networks flow analysis. *SIAM J. Comput. 19*, 143–155.

KARGER, D. R. 1997. Using random sampling to find maximum flows in uncapacitated undirected graphs. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing* (El Paso, Tex., May 4–6). ACM, New York, pp. 240–249.

KARZANOV, A. V. 1973. O nakhozhdenii maksimal'nogo potoka v setyakh spetsial'nogo vida i nekotorykh prilozheniyakh. In *Matematicheskie Voprosy Upravleniya Proizvodstvom*, vol. 5. Moscow State University Press, Moscow, CIS (in Russian; title translation: On finding maximum flows in networks with special structure and some applications).

KARZANOV, A. V. 1974. Determining the maximal flow in a network by the method of preflows. *Soviet Math. Dokl. 15*, 434–437.

KING, V., RAO, S., AND TARJAN, R. 1992. A faster deterministic maximum flow algorithm. In *Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms* (Orlando, Fla., Jan. 27–29). ACM, New York, pp. 157–164.

KING, V., RAO, S., AND TARJAN, R. 1994. A faster deterministic maximum flow algorithm. *J. Algorithms 17*, 447–474.

KNUTH, D. E. 1974. Wheels within wheels. *J. Combinat. Theory 16*, 42–46.

MIDDLETON, A. A. 1995. Numerical results for the ground-state interface in random medium. *Phys. Rev. E 52*, R3337–R3340.

NGUYEN, Q. C., AND VENKATESWARAN, V. 1993. Implementations of Goldberg–Tarjan maximum flow algorithm. In *Network Flows and Matching: First DIMACS Implementation Challenge.* D. S. Johnson and C. C. McGeoch, eds. American Mathematical Society, Providence, R.I., pp. 19–42.

OGIELSKI, A. T. 1986. Integer optimization and zero-temperature fixed point in Ising random-field systems. *Phys. Rev. Lett. 57*, 1251–1254.

PHILLIPS, S., AND WESTBROOK, J. 1993. Online load balancing and network flow. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing* (San Diego, Calif., May 16–18). ACM, New York, pp. 402–411.

ROY, S., AND COX, I. 1998. A maximum flow formulation of the *N*-camera Stereo Correspondence problem. In *Proceedings of the International Conference on Computer Vision* (Bombay, India, Jan.). IEEE Computer Society Press, Los Alamitos, Calif., pp. 492–499.

SLEATOR, D. D., AND TARJAN, R. E. 1983. A data structure for dynamic trees. *J. Comput. Syst. Sci. 26*, 362–391.

TARJAN, R. E. 1975. Efficiency of a good but not linear set union algorithm. *J. ACM 22*, 2 (Apr.), 215–225.

TARJAN, R. E. 1983. *Data Structures and Network Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, Pa.

VISHKIN, U. 1992. A parallel blocking flow algorithm for acrylic networks. *J. Algorithms 13*, 489–501.

WALLACHER, C. 1991. A generalization of the minimum-mean cycle selection rule in cycle canceling algorithms. Tech. Rep. Preprints in Optimization, Institute für Angewandte Mathemtik, Technische Universitat Braunschweig, Germany.

WALLACHER, C., AND ZIMMERMAN, U. 1991. A combinatorial interior point method for network flow problems. Tech Rep. Preprints in Optimization, Institute für Angewandte Mathemtik, Technische Universitat Braunschweig, Germany.

RECEIVED OCTOBER 1997; REVISED OCTOBER 1997; ACCEPTED JUNE 1998