Chapter 20
A Faster Deterministic Maximum Flow Algorithm

V. King* S. Rao† R. Tarjan§

Abstract

We describe a deterministic version of a 1990 Cheriyan, Hagerup, and Mehlhorn randomized algorithm for computing the maximum flow on a directed graph with n nodes and m edges which runs in time $O(mn + n^{2+\epsilon})$, for any constant ϵ. This improves upon Alon's 1989 bound of $O(mn + n^{8/3}\log n)$ [A] and gives an $O(mn)$ deterministic algorithm for all $m > n^{1+\epsilon}$. Thus it extends the range of m/n for which an $O(mn)$ algorithm is known, and matches the 1988 algorithm of Goldberg and Tarjan [GT] for smaller values of m/n.

1 Introduction

The maximum flow problem is as follows: Given a directed graph with a non-negative capacity on each edge, and with two distinguished nodes, the source and the sink, compute the maximum value of flow from the source to the sink, where flow is a function which assigns each edge a number between 0 and the capacity so that for all nodes except the source and the sink, the total flow of edges directed into a node must equal the flow of edges leaving the node. The value of a flow is the total flow over all edges leaving the source. Two recent surveys of work in this area are [AMO, GTT].

In 1988, Goldberg and Tarjan [GT] devised a generic max flow algorithm in which flow was pushed from one node to the next along the current "eligible" edge incident to each node. The specific choice of which eligible edge to choose next, after a current edge became ineligible, was left open. They made use of dynamic trees to keep track of paths of current edges so that flow could be pushed down a long sequence of current edges efficiently. In 1989, Cheriyan, Hagerup and Mehlhorn introduced randomization as a means of deciding from among the eligible edges incident to a node which to choose next, and they gave a clever framework to enable the analysis of this strategy. The randomized strategy was simple: each node randomly numbers its edges and the lowest numbered edge which is eligible goes first. The [GT] result pushed the running time down to $O(mn\log(n^2/m))$ where n is the number of nodes and m is the number of edges. The [CHM] result is an algorithm which with high probability has cost $O(mn + (n\log n)^2)$. This meant an $O(mn)$ algorithm for all but sparse graphs.

We describe a deterministic algorithm to compute the maximum flow on a directed graph with n nodes and m edges which runs in time $O(mn + n^{2+\epsilon})$, for any constant ϵ. This improves upon Alon's 1989 bound of $O(mn+n^{8/3}\log n)$ [A] and gives an $O(mn)$ deterministic algorithm for all $m > n^{1+\epsilon}$. Thus it extends the range of m/n for which an $O(mn)$ algorithm is known, and matches the 1988 algorithm of Goldberg and Tarjan [GT] for smaller values of m/n.

Our algorithm is essentially a deterministic version of the 1990 Cheriyan, Hagerup, Mehlhorn [CHM] randomized algorithm. They reduce the problem of solving a maximum flow to executing a strategy in a certain two person combinatorial game where the payoff reflects the cost of the computation. Randomization is used to choose a successful strategy. In 1989, Alon found the fastest deterministic algorithm for maximum flow, by derandomizing their strategy, but at a cost of a factor of $n^{2/3}/\log n$ in the payoff. By slightly modifying the rules of the game, we are able to find a deterministic strategy which improves the bound on the payoff to within an n^{ϵ} factor of the randomized version.

2 The Game

Cheriyan, Hagerup, and Mehlhorn [CHM] reduced the cost of finding a maximum flow to $O(mn + n^{3/2}m^{1/2} + P(n^2, nm)\log n + C(n^2, mn))$ where $P(n,m)$ is the maximum number of points that can be collected by the adversary in the following game and $C(n,m)$ is the cost of implementing the algorithm's strategy: Let G=(U,V,E) be any undirected bipartite graph, with $|U| = |V| = n$ and $|E| = m$.

The first player is the algorithm, who for each node $u \in U$, orders the edges adjacent to u. As the game proceeds, the highest remaining edge adjacent to each node $u \in U$ which is still in the graph is called u's "designated" edge. The remaining moves are by the adversary. She may either (i) remove a node $v \in V$ and

*NEC Research Institute
†NEC Research Institute
‡Princeton University

all its adjacent edges from the graph, scoring a point for every designated edge adjacent to v; or (ii) remove only an edge, but scoring no points for this move.

Cheriyan, Hagerup, and Mehlhorn analyzed the strategy in which the algorithm chooses a random ordering for each node u. We vary the rules of the game, by permitting the algorithm to act adaptively, and permitting the algorithm to redesignate edges.

We state this game more precisely as follows. Initially, the algorithm designates an edge for each node in U. Then the game proceeds in rounds until all the nodes in V are removed.

- The adversary makes one of the following moves.
  - It can remove any edge from the graph. When the adversary removes a designated edge we call it an *adversary edge kill.* It scores zero points for this move.
  - It can remove any node in V and its incident edges. It scores a point for each designated incident edge.
- The algorithm can make any sequence of the following moves.
  - It must designate an edge for each node $u \in U$ that is not currently designated to a node in V.
  - It may redesignate an edge, i.e., it may shift a designation from one edge incident to a node $u \in U$ to another. The adversary scores a point for each redesignation.

Here, we redefine $P(n,m)$ to be the number of points scored in this version of the game, and $C(n,m)$ to be the overhead incurred by the algorithm. With small modifications of [CHM] we can prove the following relationship between the game above and the running time of a maximum flow algorithm based on [CHM].

THEOREM 2.1. *Maximum flow in a n-node m-edge graph can computed in time*

$$C(n^2, nm) + O(mlogn + n^{3/2}m^{1/2} + P(n^2, nm)\log n).$$

Our main result is to bound the $P(n,m)$ where the algorithm is deterministic. We prove the following theorem.

LEMMA 2.1. *There is a deterministic algorithm where*

$$P(n,m) = O\ (\ n^{1/2+\epsilon}m^{1/2} + \#\textit{adversary-edge-kills}/(n^{2\epsilon})),$$

and

$$C(n,m) = O\ (\ m + m^{1/2}n^{1/2-\epsilon} + \frac{m^{1/2}}{n^{1/2+\epsilon}}(P(m,n)) + \#\textit{adversary-edge-kills}).$$

By appropriately bounding #adversary-edge-kills in section 7, we can show that $P(n,m) < O(n^{1/2+\epsilon}m^{1/2})$, and $C(n,m) = O(n+m)$. This gives the desired running time for a deterministic maximum flow computation.

In the next section, we sketch the intuition behind our game strategy. In Section 4 we describe it more technically. For those readers unfamiliar with the [CHM] maximum flow algorithm, we include a description of it in Section 5. Section 6 explains how our version of the game fits into the maximum flow algorithm, and Section 7 contains the cost analysis of the game, following [CHM].

3 The Idea

To illustrate the main idea, we make two simplifying assumptions: that the adversary may only make moves of the second kind, that is, edges may be removed only when their endpoint is removed, and that each node $u \in U$ has degree at least l.

We prove the following, which given our assumptions, yields a bound of mn^{ϵ}/l on the number of points scored by the adversary. If we set $l = \sqrt{m/n}$, we can see the relation to lemma 2.1.

For each $v \in V$, let $r(v)$ be the ratio of the number of designated edges incident to v to the initial degree of v. Let $U_r = \{$ edges which are designated to nodes with ratios at least r$\}$.

Claim: The strategy of designating the edges to the node with the smallest ratio keeps all ratios below n^{ϵ}/l.

We observe that since no designated edges to v are removed before v is removed, $r(v)$ never decreases. Thus, half the edges designated to v were designated when the ratio of v was greater than $r(v)/2$. Since a node u only makes such a designation when all its other edges are incident to nodes with ratios at least as high, and each has, by assumption, degree l, we know that at least $l|U_r|/2$ edges are incident to nodes of ratio $r/2$, and therefore:

$$|U_{r/2}| \geq (|U_r|/2)(l)(r/2).$$

Suppose that for some $v, r(v)$ exceeds n^{ϵ}/l. By induction, this implies

$$|U_{n^{\epsilon}/l2^j}| \geq n^{\epsilon j}/2^{j^2}.$$

For $\epsilon > 2/\sqrt{\log n}$, there exists a j such that the number of nodes is greater than n, giving a contradiction. □

4 The Edge Designation Strategy

In this section, we describe the full details of the edge designation strategy and then we prove that it is correct.

Recall that the game is played on a graph $G = (U, V, E)$ with $|U| = |V| = n$ and $|E| = m$.

The algorithm plays the game using the following concepts. We maintain a subset U' of U containing all nodes with degree greater than l, in the remaining graph. Each node v in V keeps track of the ratio $r(v)$ of the number of designated incident edges, (u, v), where $u \in U'$, to v's initial degree. Each node in U' keeps track of the ratios of its neighbors. However, to keep the algorithm's overhead down, the possible range of ratios is divided into discrete levels. A node in V is considered to be in the bottom level (or level 0) if its ratio lies below r_0. Level i contains nodes whose ratio lies in the interval $[r_i, r_{i+1})$, where $r_i = 2^i r_0$.. Thus, a node in U' has only an estimate, *erl*, of the ratio level of each of its neighbors to within one level. When an edge must be designated for a node $u \in U'$, the algorithm chooses the edge incident to the v with the minimal $erl(v)$.

The idea of this strategy (as above) is that no node v can ever be removed with a ratio as high as the ratio of a certain top ratio level, t. Unlike the simplified case in Section 3, $r(v)$ may drop, from the one-time removal of a node from U', and from the adversary edge kills. The latter factor complicates matters; we cannot prove that a node in V does not enter the top ratio level. Thus, to prevent a node, $v \in V$, from being removed when it is in the top ratio level, the algorithm redesignates edges so that v's ratio drops.

We can show that the number of redesignations needed to drop v's ratio level is small compared to the number of edge kills that have recently occurred. Thus, the total number of redesignations is small when compared to the total number of edge kills. We can prove the following theorem.

THEOREM 4.1. *There is an algorithm strategy where*

$$P(n,m) = O\ (\ n^{1/2+\epsilon}m^{1/2}) + 8\#\textit{adversary-edge-kills}/(n^{2\epsilon}),$$

and

$$C(n,m) = O\ (\ m + m^{1/2}n^{1/2-\epsilon} + \frac{m^{1/2}}{n^{1/2+\epsilon}}(P(m,n)) + \#\textit{adversary-edge-kills}).$$

We proceed by stating the algorithm's strategy in section 4.1, bounding the number of points scored by the adversary in section 4.2, and bounding $C(n,m)$ in section 4.3.

4.1 The Strategy

Let $U' = \{u \in U |\ degree\ of\ u > l\}$; and

$$r(v) = |\{designated\ edges\{u,v\} | u \in U'\}|/(d(v),$$

where $d(v)$ is the initial degree of v.

We will use the following subroutine:

UPDATE_ERL(v)
if v is in level i and $erl(v) \neq i$ then do
  $erl(v) := i$;
  inform all $u \in U'$ incident to v.

1. {Initialize}
   For all v, $erl(v) := 0$.
   For each node $u \in U'$:
     designate an edge {u,v}
     such that $erl(v)$ is minimal
     over all v incident to u;
     UPDATE_ERL(v);
     if $erl(v) = t$, RESET.
2. If a node $v \in V$ is removed by the adversary,
   then for each $u \in U'$ such that
   $\{u, v\}$ was a designated edge do
     designate an edge $\{u, v'\}$
     such that $erl(v')$ is minimal
     over all nodes incident to u
     in the remaining graph;
     UPDATE_ERL(v');
     if $erl(v') = t$, RESET.
   If, as a consequence, the degree
   of u falls below l, then
     the edge designated from u
     is "killed" by the algorithm.
   (See step 3 and 4.)
3. If the degree of u is below l,
   u may designate any edge
   incident to it, as needed.
4. If $\{u, v\}$ is the victim of an
   adversary or algorithm edge kill and
   the level of v falls to $erl(v) - 2$ then
     UPDATE_ERL(v).

The RESET proceeds as follows. Denote by V_i the nodes in V that lie in or above ratio level i. Denote by U_i the nodes in U' whose designated edges are incident to nodes in ratio level i or above.

1. $k := t$.
2. While the $|U_{k-2}| \geq (r_{k-1}l)\,|U_k|/2$ do

$k := k - 2$.

3. a. For each v in level $k-2$,
   UPDATE_ERL(v).
   b. For each $u \in U_k$,
   undesignate its designated edge.
   c. For each $v \in V_k$, UPDATE_ERL(v);
   {Since $r(v) = 0$, set $erl(v) := 0$ }.
   d. For each $u \in U_k$, do
   designate the edge (u, v)
   such that $erl(v)$ is minimal
   over all v incident to u
   in the remaining graph;
   UPDATE_ERL(v).

(It can be shown using a similar argument as in the previous section that this procedure must terminate above level $t - \log_{r_0 l/2} 2n + 1$.)

In the following analysis, we show that the level k at which the RESET procedure stops must have suffered many edge kills.

4.2 Point analysis

In this section, we analyze the number of points scored by the adversary. The main point of the analysis is that the RESET must work as required. Then, the argument proceeds as in the case presented in section 3.

The following crucial lemma for the point analysis ensures that the reset will find a level that suffered many edge kills.

LEMMA 4.1. *At any point in the algorithm, at every level $k \geq 3$, either*

1. $|U_{k-2}| > (r_{k-1})\,|U_k|/4$, *or*
2. *there were at least $(r_{k-1}l)\,|U_k|/8$ edge kills at level $k-2$ or higher since the previous time step 5 of the RESET procedure occurred at or below level k. (We consider a RESET to happen at level h if that is the value of k when step 3 of the RESET procedure is performed.)*

Proof: We will show that the above lemma is true by using the following fact.

FACT 4.1. *At any point in the algorithm, for all $k > 0$ such that $U_k \neq \emptyset$, $\exists \hat{U}_k \subseteq U_k$ such that $|\hat{U}_k| \geq |U_k|/2$ and every v incident to a node $u \in \hat{U}_k$ there was a time after the previous RESET (or the start of the algorithm, if there was none) during which v was in V_{k-1}.*

To prove the lemma, we assume that condition (1) is false for a level k and show that condition (2) must hold.

Since each node $u \in U_k$ has degree at least l, Fact 1 implies that $l|U_k|/2$ edges are incident to nodes in V_{k-1} at points of time since the previous RESET. Hence, there must have been at least $r_{k-1}l|U_k|/2$ designated edges over this period to nodes in V_{k-1}. But the negation of condition (1) implies that $|U_{k-2}| < r_{k-1}l|U_k|/4$. Thus at least $r_{k-1}l|U_k|/4$ designated edges were removed by the adversary from nodes which are or had been in V_{k-1}. A node that drops from level $k-1$ or higher to below $k-2$ must lose at least half its designated edges at level $k-2$. This implies that at least $r_{k-1}l|U_{k-1}|/8$ designated edges were removed when they were incident to v's whose ratios were at level $k-2$ or higher. That is, there were at least $r_{k-1}\,|U_{k-1}|/8$ edge kills since that last RESET, satisfying condition (2) of the lemma.

Proof of fact:

We let $\hat{U}_k$ be the union over all $u \in U_k$ such that (u, v) is a designated edge, and the edge was designated after at least half the currently designated edges incident to v were designated.

Consider a node u such that (u, v) is designated and $v \in V_k$. The algorithm designated that edge when:

1. the node that u was previously designated for was killed by the adversary or
2. when u is redesignated by the RESET procedure of the algorithm.

In either case, v must have been the node with the minimal erl of any of u's neighbors.

For $u \in \hat{U}_k$, and u's designated edge is (u, v), then $erl(v)$, and, consequently, the erl's of each of u's neighbors were at least $k-1$ right before the time of the designation.

We must show that u's neighbors were in ratio level $k-1$ *since the previous RESET on level k.* We accomplish this by induction on the number of RESETS. This is clearly true before there were any RESETS.

Consider that the previous RESET was on level $k(R)$.

For levels $k \geq k(R)+1$, any node in U_k that entered level $k-1$ entered after the RESET, thus the claim is trivially true.

For levels $k < k(R)$, the claim is inductively true.

For level $k = k(R)$, the claim holds since the RESET updates the erl for all v who might have $erl(v) = k-1$, but have a lower $r(v)$, i.e., those v in level $k-2$, and no edge kills occur during the RESET. Thus, if a node, v, keeps its $erl(v) = k-1$ throughout the RESET, it is at level $k-1$ afterwards. Any node, v, that attains an $erl(v) = k-1$ after the RESET must have been at level $k-1$ since an $erl(v)$ is never raised to a level until the v actually attains that level.

□

End of proof of lemma 4.1

Given the lemma above, we can bound the number

of points scored by the adversary as follows.

LEMMA 4.2. *The number of points scored by or given to the adversary is bounded by*

$$r_{t-1}m + nl + 8\#\textit{edge-kills}/(r_0 l),$$

where #edge-kills denotes the number of algorithm edge kills (which is at most n since each node's degree only decreases below l once) plus the number of adversary edge kills.

Proof: We consider designated edges incident only to nodes in U', i.e., of degree greater than l, in the ratio computation, and assume that the adversary gets l points for each node after it leaves U'. This is bounded by nl.

Now recall that every node the adversary kills has ratio at most r_{t-1}. This implies that when the adversary removes a node, she can score no more than $1/r_{t-1}$ points per edge removed, or no more than mr_{t-1} points total, from edges incident to nodes in U'.

Finally, when the algorithm redesignates edges a point must be given to the adversary. All the redesignations are done in the RESET procedure of the algorithm. In step 3 of the RESET procedure we know that $|U_k|$ redesignations are performed. At this level we know condition 1 of lemma 4.1 doesn't hold by step 2 of the algorithm. Thus, there must have been at least $r_{k-1}l\,|U_k|/8$ edge kills at level $k-1$ since the previous RESET at level k or below. We assign the cost of redesignation to these edge kills. By the lemma, we know that these edge kills will never be assigned the cost of any more redesignations.

There are at least $(r_{k-1}l)/8$ edge kills per redesignation so that the total number of redesignations is bounded above by

$$8\#\text{edge-kills}/r_0 l.$$

We have shown that the total number of points the adversary gains is bounded above by the expression in the lemma. □

Now if we plug some numbers in, we get the following corollary:

COROLLARY 4.1. *The game can be played so that the number of points scored by or given to the adversary is bounded by*

$$O(n^{1/2+\epsilon}m^{1/2} + \frac{\#\textit{adversary-edge-kills}}{n^{2\epsilon}}),$$

where #adversary-edge-kills denotes the number of adversary edge kills.

Proof: Set $r_0 = n^\epsilon/\sqrt{m/n}$, and $l = n^\epsilon\sqrt{m/n}$ and $t = O(1/\epsilon)$ and plug in to the lemma. □

4.3 Cost of Implementing the Strategy

Now we will show that $C(n,m) = O(m + n + (n + P(n,m))/r_0\#\textit{adversary} - \textit{edge} - \textit{kills}/r_0)$.

Consider that the algorithm needs to take the following actions.

- $P(n,m)+n$ times some node in u needs to designate an incident edge, (u,v), where v has minimal $erl(v)$ over u's neighbors.
- The algorithm must keep track of edge removals.
- UPDATE-ERL(v) informs the neighbors of v of a new $erl(v)$ when the ratio level of v rises (step 1 or 2 or RESET step 3(d)) or drops two levels (step 4),
- UPDATE-ERL(v) also informs the neighbors of v when the ratio level of v drops and UPDATE-ERL(v) is called by the RESET (RESET step 3(a) and 3(c)).

To minimize the overhead for these actions, we maintain the following data structure. For each $u \in U$ and each level $i = 0, 1, ..., t-1$ we maintain a linked list containing all edges between u and the v's with $erl = i$ which are still in the graph.

With this data structure, each designation requires $O(t)$ time. Thus, item 1 above requires a total of $O(tP(n,m) + tn)$ time.

The removal of an edge, $e = (u,v)$ takes constant time to maintain the linked lists of u, and if e is designated to recompute the ratio level of v. Thus, item 2 above requires a total of $O(m)$ time.

Now we consider item 3. This occurs when the ratio of a node v rises to a new level, and the $erl(v)$ reflects the old level, or v falls two levels below the $erl(v)$. Thus, these revisions can only occur after at least $r_0(\textit{initial degree of } v)$ edge removals or designations involving v have occurred. The cost per revision is no greater than the current degree of v, so that total cost of these revisions per node is no greater than (degree of v)(#edge-kills involving v + #designations involving v)/r_0(initial degree of v). When summed over all nodes v, this is $\leq (n + P(n,m) +$ #edge-kills)/r_0.

Finally, we analyze item 4. Steps 3a and 3c involve the revision of the erl of nodes at or above level $k-2$. The total number of edges to nodes at or above level $k-2$ is $\leq |U_{k-2}|/r_{k-2}$. At the time of the RESET, $|U_{k-2}| \leq r_{k-1}l|U_k|/2$, so the total number of edges to nodes at level $k-2$ is less than $l|U_k|/2$. But each such RESET occurs after $r_{k-1}l|U_k|/8$ edge kills at level $k-1$ since the previous RESET. So the cost of updating all these edges per edge-kill is $O(\#\text{ edge-kills}/(r_0))$.

Thus, $C(n,m)$ is $O(m + n + P(n,m)/r_0 +$ #edge-kills/r_0). Since the number of algorithm edge

kills is at most n, we obtain $C(n,m) = O(m+n+(n+P(n,m))/r_0\#adversary-edge-kills/r_0)$.

5 The Maximum Flow Algorithm

We describe the [CHM] algorithm here. Our only modification to it is the implementation of the currentedge procedure, which is described in following sections. We assume that the reader is familiar with the generic maximum flow algorithm and the use of dynamic trees [ST] to implement it, as described by Goldberg and Tarjan [GT]. The terms *cap, link, cut, d(v), saturate, residual capacity* r_f, *excess flow* are essentially the same in [GT] and [CHM] and are used here without definitions.

The incrementally strong polynomial algorithm in [CHM] may be viewed as an extension of the generic algorithm, with a few modifications. Unlike the [GT] algorithm, the maximal flow is computed in stages with an incomplete graph (V, E^*) into which each pair of directed edges is added one by one (using an Addedge operation). In order to allow one of the edges of the pair to be saturated upon its addition, *"visible" excess flow* into a node, $e*(v)$, rather than excess flow $e(v)$ is used to determine the maximum flow that may be pushed from v, where

$$e*(v) = max\{e(v) - \sum_{(v,u)\in E\setminus E*} cap(v,u), 0\}.$$

As in [GT], dynamic tree structures are used to maintain information about a forest F of edges, so that a sequence of pushes along a path in the forest may be done quickly. The edges in the forest must be *eligible* for a push, that is, they must have residual capacity and the labels of their tails must be one higher than the labels of their heads. Each node in the forest has at most one such edge emanating from it. Upon saturation of an edge, the edge is cut from the forest. Upon relabeling of a node, the edges directed into that node must be cut from the forest. When a new eligible edge is picked for a node, that edge is added to the forest with a link operation.

When there is more than one eligible edge emanating from a node in the graph, the [GT] algorithm does not specify which edge should be added to F. A major contribution of the [CHM] paper is show how these choices can be turned into a two-person game between the algorithm and an adversary, where the number of cuts induced by node relabelings is the payoff to the adversary. This number is used to bound not only the number of these cut operations but also the number of saturating pushes.

The [CHM] algorithm randomly chooses among the eligible edges. The contribution of this paper is to make this choice deterministically, with only an n^ϵ gain in cost. We allow the algorithm to make its choice adaptively and to change its choice while the edge is still eligible and before it is saturated.

Both the [GT] and [CHM] algorithms use the operations treepush, cut, and link. The treepush in [CHM] is different in that the trees may grow as high as needed so that each tree operation has amortized cost bounded by O(log n). Also, because of the Addedge operation, visible excess flow may accumulate in a non-root node, in the [CHM] version and therefore, treepushes may start from a non-root node.

Below, we state the main routines of the [CHM] algorithm.

```
   procedure currentedge (v)
   { This will be described
     in the next section}

procedure treepush (v)
   let (u,w) = currentedge(u).
   If (v,w) is not in F then link (v,w)
   Find the first edge (x,y) on the path
        to the root with
        residual capacity <= e*(v),
if there is one.
   Cut (x,y) and saturate it.
   Send e*(v) units of flow
   along the maximal path from v in F.


procedure relabel(v)
   For all u with currentedge(u) = (u,v),
        (u,v) in F, do cut(u,v)
   d(v) =d(v)+1.


main algorithm
   initialize as in [GT]
   For all edge pairs (u,v) and (v,u) of G,
    in order of decreasing size
    of cap(u,v) + cap (v,u),
   do
        add (u,v) and (v,u) to F and
        saturate (u,v) if d(v) < d(u).
        while some node v has e*(v) >0,
         if v has no eligible edge
                then relabel(v)
                else treepush(v).
```

This algorithm may be viewed, for the most part, as a version of the generic maximum flow algorithm, and thus, the proof of its correctness follows from [GT].

6 Determining current edges–how the game fits in

The game is played on a bipartite graph $G' = (U', V', E')$ such that for each node w in the max flow problem and each possible label k between 1 and $2n-1$, there is a node $u_{w,k}$ in U', and a node $v_{w,k-1}$ in V', for a total of $O(n^2)$ nodes. For each edge (w, w') in the max flow problem, there are $2n-1$ copies of edges $(u_{w,k}, v_{w',k-1})$ for $k = 1, ..., 2n-1$.

The designated edge from each $u_{w,k}$ determines the max flow algorithm's choice of current edge for w when w has label k. Thus we have:

```
procedure currentedge(w)
 Let k be the current label of w.
 Let v_{w', k-1} be the endpoint
 of the designated edge from u_{w,k}.
 Then the currentedge(w) = (w,w').
```

We relate the adversary's strategy to the max flow computation as follows.

When the max flow algorithm sets the label of a node w to k, all game edges incident to $u_{w,k}$ except those which are currently eligible are removed by the adversary from G'.

When the max flow algorithm relabels a node w from k to $k+1$, it must cut its incoming edges from the forest F. These edges are the current edges of nodes incident to w in F. In the game, the adversary removes the node $v_{w,k}$ and collects a point for each designated edge incident to $v_{w,k}$. *That is, each edge cut from F corresponds to a point gained by the adversary.*

When an edge is saturated, the corresponding edge (with the current endpoint labels) is removed from G', by the adversary. That is, *each adversary edge-kill corresponds to a saturating push.*

Finally, when the game algorithm performs a redesignation, the max flow algorithm changes the current edge and therefore must cut the old current edge from F if it is there. *Again, each edge cut corresponds to a point gained by the adversary.*

The number of points scored by the adversary $P(n^2, mn)$ is an upper bound on the number of edges that were cut from the forest *before* the edge was saturated. The number of saturating pushes is the number of adversary edge-kills in the game.

7 Analysis of the cost

In this section, we prove the following bound on the running time of the algorithm using the analysis in [CHM].

THEOREM 7.1. *The running time of the maximum flow algorithm is $O(mn + n^{3/2+\epsilon}m^{1/2})$ for any constant ϵ.*

The proof follows from the lemmas below. They are similar to those proved in more detail in [CHM]. Here, we highlight the modifications required for our analysis.

LEMMA 7.1. *The running time of the maximum flow algorithm is*

$$C(n^2, mn) + O(m \log n + n^2 + (\#treepush + \#cuts)\log n),$$

where C is the total cost of computing the current edges, $m \log n$ is the cost of sorting the capacities of the new edges, and n^2 is the cost of relabeling.

LEMMA 7.2.

a

$$\#treepushes \leq \#links + \#cuts + \#saturating\_pushes + 2m$$

b $\#links \leq \#cuts + n$

c $\#cuts \leq P(n^2, mn) + \#saturating\_pushes$

Proof: a) Each treepush results in either a link, a cut, or it reduces the visible excess of a non-root node to 0. But a non-root node gets positive visible excess only from a saturating push which affects one node or from the addition of a new edge which results in at most two nodes gaining excess.

c) Each cut is caused by a saturation or results in a point being scored.

□

LEMMA 7.3.

$\#saturating\_pushes \leq O(n^{3/2}m^{1/2}) + m + P(n^2, mn)$

Proof: We say that an edge is saturated by a *regular push bundle* if at some point the flow equals the capacity in one direction and all subsequent pushes are in the opposite direction, until the edge is saturated.

CLAIM 7.1. *The number of saturations by non-regular push bundles is $\leq m + P(n^2, mn)$.*

The direction of pushes may be interrupted before an edge is saturated when an edge is cut from a tree in a point-scoring event (i.e., a relabeling or redesignation). Also, when new edges are added, the sequence begins with a flow of 0.

CLAIM 7.2. *The number of regular push bundles is $O(n^{3/2}m^{1/2})$.*

The analysis is in [CHM] and is independent of changes to that algorithm made here. □

From Section 4 and the fact that the *#adversary-edge-kills* is *#saturating_pushes*, we have:

$$P(n^2, mn) \leq O(\quad n^{3/2+\epsilon}m^{1/2}) + 8/n^{2\epsilon}\#saturating\_pushes.$$

Therefore, we have the following corollary of lemma 7.3.

COROLLARY 7.1.
$\#saturating\_\ pushes \leq O(n^{3/2+\epsilon}m^{1/2})$

Theorem 7.1 follows from this corollary, lemmas 7.1 and 7.2, and our bound on $C(n^2, nm)$ from section 4.3.

References

[AMO] .K. Ahuja, T. L. Magnanti, and J.B. Orlin, "Network flows," in *Handbook in Operations Research and Management, Volume 1: Optimization,* G. L. Nemhauser, A.H. G. Rinnooy Kan, and M.J. Todd, eds., North-Holland, Amsterdam, 1990, 211-360.

[A] N. Alon, "Generating Pseudo-Random Permutations and Maximum Flow Algorithms," *manuscript.*

[CH] J. Cheriyan and T. Hagerup, "A Randomized Maximum-Flow Algorithm," *Proc.IEEE* FOCS (1989), 118-123.

[CHM] J. Cheriyan, T. Hagerup, K. Mehlhorn, "Can a Maximum Flow be Computed in o(nm) Time?" ICALP (1990).

[GT] A.V. Goldberg and R. E. Tarjan, "A New Approach to the Maximum Flow Problem," *Journal of the ACM 35* (1988), 921-940.

[GTT] A.V. Goldberg, E. Tardos, and R. E. Tarjan, "Network flow algorithms," in *Paths, Flows, and VLSI-Layout,* B. Korte, L. Lovász, H. J. Prömel, and A. Schriver, eds., Springer-Verlag, Berlin, (1990), 101-164.

[ST] D. Sleator and R. E. Tarjan, "A Data Structure for Dynamic Trees," *Journal of Computer and System Sciences* (26) (1983), 362-391.