JOURNAL OF ALGORITHMS **17**, 447–474 (1994)

A Faster Deterministic Maximum Flow Algorithm

V. KING

University of Victoria, Victoria, British Columbia V8W 2Y2, Canada

S. RAO

NEC Research Institute, 4 Independence Way, Princeton, NJ 08540

AND

R. TARJAN

NEC Research Institute, 4 Independence Way, Princeton, NJ 08540, and Princeton University, Princeton, NJ 08544

Received March 20, 1992; revised January 27, 1994

Cheriyan and Hagerup developed a randomized algorithm to compute the maximum flow in a graph with n nodes and m edges in $O(mn + n^2 \log^2 n)$ expected time. The randomization is used to efficiently play a certain combinatorial game that arises during the computation. We give a version of their algorithm where a general version of their game arises. Then we give a strategy for the game that yields a deterministic algorithm for computing the maximum flow in a directed graph with n nodes and m edges that runs in time $O(mn(\log_{m/n \log n} n))$. Our algorithm gives an $O(mn)$ deterministic algorithm for all $m/n = \Omega(n^\epsilon)$ for any positive constant ϵ, and is currently the fastest deterministic algorithm for computing maximum flow as long as $m/n = \omega(\log n)$. © 1994 Academic Press, Inc.

1. INTRODUCTION

The maximum flow problem[1] is defined on a directed graph, $F = (\mathcal{N}, A)$, with two distinguished vertices, a *source* s and a *sink* t, and a positive capacity $cap(i, j)$ for every directed edge $(i, j) \in A$. For conve-

[1]This description is taken from [12].

nience we define $cap(i, j) = 0$ if (i, j) is not in A. A *flow* on F is a real-valued function f on vertex pairs having the following three properties:

1. *Skew symmetry.* $f(j, i) = -f(i, j)$. If $f(i, j) > 0$ we say there is flow from i to j.
2. *Capacity constraint.* $f(i, j) \leq cap(i, j)$.
3. *Flow conservation.* $\sum_{(j,i) \in A} f(j, i) = 0$, for all $i \in \mathcal{N} \setminus \{s, t\}$.

The *value* of the flow is the net flow out of the source. The *maximum flow problem* is that of finding a flow of maximum value, called a *maximum flow*.

Algorithms for this problem are classified as either *pseudopolynomial time*, where the running time on an n-node m-edge network with integer capacities not exceeding B is bounded by a polynomial in n, m, and $\log B$, or *strongly polynomial time*, where the running time on an n-node m-edge graph is bounded by a polynomial in n and m. There is a long history of work on finding efficient pseudo- and strongly polynomial time algorithms for the maximum flow problem. A recent survey of work in this area is [7]. In this paper, we primarily discuss the strongly polynomial algorithms of Cheriyan *et al.* in [4, 5].

Using a variety of techniques including randomization, Cheriyan and Hagerup improved the strongly polynomial algorithm of [8] to obtain an algorithm that runs in $O(nm + (n \log n)^2)$ expected time. They use randomization to play a combinatorial game that arises during their maximum flow algorithm. Alon [3] gave a deterministic strategy for playing the game that yielded a maximum flow algorithm that runs deterministically in time $O(mn + n^{8/3} \log n)$. This improved upon the best previous running time for deterministic maximum flow given by the $O(mn \log(n^2/m))$ algorithm of Goldberg and Tarjan [8] when m/n is sufficiently large. Alon's approach yields an $O(mn)$ algorithm when m/n is $\Omega(n^{2/3} \log n)$.

In 1992 [9], we gave a version of the algorithm of [4] where a slightly different combinatorial game arose. We also showed how to play this game deterministically to yield a strongly polynomial maximum flow algorithm that ran in time $O(nm + n^{2+\epsilon})$. Since that time, Phillips and Westbrook [10] gave an $O(mn \log_{m/n} n + \log^{2+\epsilon} n)$ time algorithm for any constant ϵ. They also related the combinatorial game to a load balancing problem. In this paper, we generalize our 1992 result to yield a maximum flow algorithm that runs in time $O(mn \log_{(m/n \log n)} n)$.[2] This is a slight improvement over the algorithm of Phillips and Westbrook. For example, if

[2]We note that this expression only makes sense for $m > n \log n$. When $m < n \log n$, the $O(mn \log(n^2/m))$ result of Goldberg and Tarjan dominates ours.

$m = n \log n \log\log n$ our algorithm runs a factor of $\Omega(\log^{\epsilon} n)$ faster than that of Phillips and Westbrook.[3]

(We should remark, however, that the deterministic pseudo-polynomial time algorithm of Ahuja *et al.* [2] that runs on a network with integer capacities not exceeding B in $O(nm \log(n\sqrt{\log B}/m + 2))$ time is faster than ours unless B is rather large, i.e., $B > 2^{(mn^{1/\log(m/n \log n)}/n)^2}$.)

In [6], Cheriyan *et al.* analyze the total number of real number operations required by their algorithms. They refer to these operations as *flow operations*. They give the first bounds for any maximum flow algorithm where the number of flow operations is $o(nm)$. In particular, their randomized algorithm uses

$$O\left(n^{3/2}m^{1/2} \log n + n^2(\log n)^2/\log\left(2 + n(\log n)^2/m\right)\right)$$

flow operations. We present a deterministic algorithm that for any positive constant $\delta < 1/2$ uses

$$O(n^{3/2-\delta}m^{1/2+\delta} \log n)$$

flow operations and runs in time $O(nm \log_{m/n} n)$ on a graph where $(m/n)^{2\delta} > c \log n$ for a large enough constant c.

We proceed by describing a general version of the combinatorial game of [4, 5] in the following section. In Section 3, we describe the connection between the flow algorithms of [4, 5] and our version of the game. In Section 4, we describe a strategy for a simplified version of our game to illustrate the central idea of our solution. In the remainder of the sections, we present various strategies for playing our version of the combinatorial game.

2. The Game

In this section, we describe a game which is a variation of the game described in [4, 6]. The game is played between a player and an adversary on any undirected bipartite graph $G = (U, V, E)$ with $|U| = |V| = N$ and $|E| = M$, where $M \geq N$. There is a parameter $K(N, M) = O(N^{1/2}M^{1/2})$.

The rules of the game are as follows: Initially, the player "designates" one edge incident to each node in U. The adversary and the player then alternate moves as follows.

[3]We note that the difference between our result and that of Phillips and Westbrook is so minor that one suspects that they could also obtain our result.

An adversary move is one of the following:

1. The adversary can remove any node in V and its incident edges. He scores a point for each designated edge which is incident to the node.

2. The adversary may remove any number of undesignated edges from the graph. The adversary may also at the same time remove any number of designated edges, provided that the total number of designated edges which he removed during the course of the game does not exceed the sum of $K(N, M)$ and twice the number of points scored by the adversary.

The removal of a designated edge is termed an *adversary edge kill.* The adversary scores no points for this move.

The player's move consists of any sequence of the following:

1. She is required to designate an edge for any node in U of positive degree in the remaining graph that does not have an edge currently designated to a node in V.

2. She may *redesignate* an edge; i.e., she may shift a designation from one edge incident to a node $u \in U$ to another edge incident to u. The adversary scores a point for each redesignation.

The game ends when there are no nodes remaining in V.

Given a strategy, $\mathscr{A}$, for the player we define $P_{\mathscr{A}}(N, M)$ to be the maximum number of points any adversary can win over all bipartite graphs with $2N$ nodes and M edges. We define $C_{\mathscr{A}}(N, M)$ to be the worst case cost of implementing the player's strategy (again over graphs with $2N$ nodes and M edges).

Our maximum flow algorithm "plays" this game in the role of the player, as a coroutine, on a graph where $N = 2n^2 - n$ and $M = m(2n - 1)$. The maximum flow algorithm needs $O(\log n)$ time for each point scored by the adversary, and needs to implement the player's strategy. That is, the running time of the maximum flow algorithm depends upon (among other things) $P_{\mathscr{A}}(N, M)$ and $C_{\mathscr{A}}(N, M)$ for some game strategy $\mathscr{A}$.

In this paper, we describe a strategy for playing this game. In particular, we describe

- a player's deterministic strategy, $\mathscr{A}$, where

$$P_{\mathscr{A}}(N, M) = O\left(N^{1/2}M^{1/2}\sqrt{D \log N/\log D}\right)$$

and $C_{\mathscr{A}}(N, M) = O(M \log_D N)$, for any D where $176 < D < (M/N)/\log N$.

We proceed by describing the connection of this game with our maximum flow algorithm, and then by describing strategies for the player. The content of the next section is largely a review of previous work and the algorithms of Cheriyan *et al.* Our contribution are the new strategies in the successive sections.

3. The Game and a Maximum Flow Algorithm

Informally, a preflow is a flow where the flow conservation constraint that flow into a node equals the flow out of the node is replaced by the less restrictive nonnegativity constraint that flow into a node is at least as much as the flow out of the node. That is, there may be nodes in the network with *excess flow* coming into them. The generic push/relabel maximum flow algorithm of [8] pushes excess flow around a flow network using an integer labeling as a guide. Each node, i, maintains a current edge, (i, j), along which some amount of flow can be pushed without violating its capacity constraint and where the label of j is less than the label of i. The algorithm proceeds in a series of push/relabel steps where a push moves around excess flow and a relabel increases the value of the label of a node. A current edge, (i, j), for a node, i, must be changed either when the current edge becomes saturated by a push, that is, the flow across (i, j) equals the capacity of (i, j), or when j is relabeled. A node may be relabeled when it is not incident to any edges which are eligible to be the current edge.

The maximum flow algorithm of Cheriyan *et al.* [6] generalizes the generic push/relabel algorithm so that it manipulates a preflow in subnetworks to eventually compute a maximum flow in the network. They do so by using a generic operation called add-edge that augments the subnetwork by a single edge in the network. They use a number of specializations of this push/relabel/add-edge algorithm to produce a fast algorithm.

One factor in analyzing the running time of their algorithm is the number of current edge changes. These fall into two classes, those caused by saturating pushes and those caused by relabelings. The latter are referred to as *premature target relabelings*, and the number of such changes is denoted by #ptr. They limit #ptr by using a randomized current edge selection strategy. They derive a bound on #ptr by describing a two-person game and bounding the points that can be obtained by one of the players in the game.

In our version of the push/relabel/add-edge algorithm, the current edge of a node can be changed by the algorithm even when this is not made necessary by a relabeling or a saturating push. We define a *premature edge change* to be any current edge change that is not due to a saturating push. That is, a premature edge change is a current edge change due to a relabeling (a premature target relabeling) or any current edge change made by the algorithm while the current edge is still eligible. The number of premature edge changes in the execution of a push/relabel/add-edge algorithm is denoted by #pec. (In the analysis of [6], this quantity is denoted by #ptr.)

We show in this section that #pec in an execution of a push/relabel/add-edge algorithm with our current edge selection strategy can be limited to the number of points scored by the adversary in a play of the game defined in Section 2.

We proceed by briefly describing the generic push/relabel/add-edge algorithm of [6], and then by showing the relationship of the current edge changes in a push/relabel/add-edge algorithm to our game. We remark that this analysis is essentially the analysis in [4].

3.1. *The Generic Push/Relabel/Add-Edge Algorithm*

For any flow network consisting of a directed graph, $F = (\mathcal{N}, A)$, with a capacity function *cap*: $A \to \Re^{+} \cup \{0\}$ and distinguished vertices $\{s, t\}$, a preflow is defined as a real valued function on vertex pairs with the following properties.

1. *Skew symmetry*. $f(j,i) = -f(i,j)$. If $f(i,j) > 0$ we say there is flow from i to j.
2. *Capacity constraint*. $f(i,j) \leq cap(i,j)$.
3. *Flow nonnegativity*. $\sum_{(j,i) \in A} f(j,i) \geq 0$, for all $i \in \mathcal{N} \setminus \{s,t\}$.

A preflow is a generalization of a flow on a graph, the only difference being that a flow satisfies the inequality in the third property above with equality.

A vertex pair (i,j) is a *residual edge* with respect to f if $f(i,j) < cap(i,j)$. The *residual capacity of* (i,j), $rescap(i,j)$, is defined as $cap(i,j) - f(i,j)$ for a residual edge and 0 otherwise. A labeling of $F = (\mathcal{N}, A)$ is a function d: $\mathcal{N} \to \aleph \cup \{0\}$. A labeling d is *valid* for F and a preflow on F if $d(i) \leq d(j) + 1$ for each residual edge (i,j). A vertex pair (i,j) is an *eligible edge* if (i,j) is a residual edge and $d(i) = d(j) + 1$.

The generic push/relabel/add-edge algorithm of [6] maintains a preflow and a labeling on a sequence of subgraphs $F^* = (\mathcal{N}, A^*)$ of $F = (\mathcal{N}, A)$ where $A^* \subseteq A$. During the algorithm the label of t, $d(t)$, is always 0 and the label of $s, d(s)$, is always n where $n = |\mathcal{N}|$. With these conditions and the definition of a valid labeling the maximum label for any node is $2n - 1$. See [8, 6]. We proceed with a brief description of the algorithm.

Given a preflow on a subgraph $F^* = (\mathcal{N}, A^*)$ of a graph $F = (\mathcal{N}, A)$, we define the *excess*, $e(i)$, and the *visible excess*, $e^*(i)$, *with respect to F* as follows:

$$e(i) = \sum_{(j,i)} f(j,i)$$

$$e^*(i) = \max\left(0, e(i) - \sum_{(i,j)\in A\setminus A^*} cap(i,j)\right).$$

We define $\mathcal{N}^+$ to be the set of nodes $\mathcal{N}\setminus\{s,t\}$. We give the definitions of push, relabel, and add-edge operations in [6] below. (We leave out many details involving subroutines and implementation.)

```
**procedure** push((i, j): vertexpair, c: real);
Precondition: $i \in \mathcal{N}^+$, $(i,j)$ is eligible, and
$0 < c < \min(e^*(i), rescap(i,j))$.
    $setflow((i,j), f(i,j) + c)$;

**procedure** relabel(i: vertex);
Precondition: $i \in \mathcal{N}^+$, $e^*(v) > 0$ and no eligible edge with tail $i$.
    $d(i) = d(i) + 1$;

**procedure** add-edge((i, j): vertexpair);
Precondition: $(i,j) \in A \setminus \bar{A}^*$.
    $A^* := A^* \cup \{(i,j)\}$
    **if** $d(i) > d(j)$ **then** $saturate(i,j)$;

**subroutine** set-flow((i, j): vertexpair, c: real)
Precondition: $(i,j) \in A^*$ or $(j,i) \in A^*$.
    $f(i,j) = c$;
    $f(j,i) = -c$;
```

The algorithm of Cheriyan *et al.* [6] proceeds by applying the operations above until none apply. They show that this computes a maximum flow. They implicitly also prove the following fact, which we re-prove here.

FACT 1. *The only edges that will be eligible for a node i while it is labeled k are the edges that were eligible when i's label was set to k.*

Proof. If an edge $(i,j) \in A^*$ is ineligible at the time when i's label was set to k then either (i,j) is saturated or $d(j) > d(i) - 1$. To make a saturated edge eligible for i, a push must occur over (j,i) which requires that $d(j) > d(i)$. Thus, in either case, $d(i)$'s label needs to be increased before (i, j) can be eligible. Thus, an edge, $(i, j) \in A^*$, will never be eligible while $d(i) = k$ if it was not eligible when i was first labeled k.

Now consider an edge, (i, j), that is added to A^* after i is labeled k. If $d(i) > d(j)$, the edge is saturated as soon as it is added. Thus, by the argument above, i must be relabeled before (i, j) can become an eligible edge.

One way that Cheriyan *et al.* specialize the generic algorithm above is through the concept of a *current edge*. That is, the algorithm maintains a current edge for each node that is eligible and will be the only edge that is used to push excess flow out of this node. In the algorithm of [6], a current edge for a node is only changed when the current edge becomes ineligible. This condition does not hold in our implementation of the algorithm of [6].

The next section relates the selection of current edges in a push/relabel/add-edge algorithm to the game described in Section 2.

3.2. *The Game and Current Edge Changes*

We can specialize the current edge selection method of any push/relabel or push/relabel/add-edge algorithm on a flow network, $F = (\mathcal{N}, A)$, using our game strategy on the graph, $G = (U, V, E)$, where

$$U = \{\langle i_u, k\rangle : i \in \mathcal{N},\ k \in \{1, \dots, 2n-1\}\},$$

$$V = \{\langle i_v, k\rangle : i \in \mathcal{N},\ k \in \{0, \dots, 2n-2\}\},$$

and

$$E = \{(\langle i_u, k\rangle, \langle j_v, k-1\rangle) : (i, j) \in \bar{A},\ k \in \{1, \dots, 2n-1\}\}.$$

An execution of a push/relabel/add-edge algorithm and a play of the game on the graph are related as follows.

The current edge in the push/relabel/add-edge algorithm for a vertex $i \in \mathcal{N}$ with $d(i) = k$ is the edge that corresponds to the designated edge of $\langle i_u, k\rangle$ in G[4]. If the degree of node $\langle i_u, k\rangle$ in G falls to 0 and $d(i) = k$ then the current edge of i is set to nil. A node, i, with $d(i) = k$ may be relabeled in the push/relabel/add-edge algorithm only when its current edge has been set to nil.

[4]We remark, that the push/relabel/add-edge algorithm is not affected at all by the choice of designated edges in the game for $\langle i, k'\rangle$ when k' is not the current label of i.

Three types of events in the push/relabel/add-edge algorithm correspond to adversary actions in the game as follows.

1. A saturating push on edge (i, j) when $d(i) = k$ in the push/relabel/add-edge algorithm causes the adversary to remove the edge $(\langle i_u, k\rangle, \langle j_v, k-1\rangle)$ in the game. (Note that this is an adversary edge kill since the edge, $(\langle i_u, k\rangle, \langle j_v, k-1\rangle)$, must have been the designated edge for $\langle i_u, k\rangle$ since (i, j) must be the current edge for i.)

2. A relabeling of a node i from k to $k+1$ in the push/relabel/add-edge algorithm causes the adversary to remove the node $\langle i_v, k\rangle$ from the game graph, and to remove the edges incident to $\langle i_u, k+1\rangle$ from the game graph that correspond to ineligible edges incident to i in the push/relabel/add-edge algorithm. (Note that this may cause an adversary edge kill if the designated edge for the node $\langle i_u, k+1\rangle$ corresponds to an ineligible edge in the flow network.)

3. At termination of the push/relabel/add-edge algorithm every node in V is removed. (By the rules of the game all vertices in V must eventually be removed.)

The following lemma ensures that the procedure above properly implements a push/relabel/add-edge algorithm.

LEMMA 2. *During the execution of the push/relabel/add-edge algorithm, the current edge of any node is always an incident eligible edge if there is one.*

Proof. For a node i, with $d(i) = k$, the designated edge for $\langle i_u, k\rangle$ must correspond to an eligible edge since all the incident edges that correspond to ineligible edges are removed by the adversary as they become ineligible. That is, they are removed when i is first labeled k or as the edges become saturated.

Furthermore, since ineligible edges do not become eligible for i while $d(i) = k$ by fact 1, once the degree of $\langle i_u, k\rangle$ falls to zero there are no eligible edges adjacent to i. ■

The relationships among the number of points scored in the game, #pec, and the running time of a push/relabel/add-edge algorithm are stated in the theorem below.

THEOREM 3. *In any push/relabel/add-edge algorithm on an n-node m-edge graph that uses the current edge selection method above where* *the game is played using strategy $\mathscr{A}$ and where $N = n(2n - 1)$, and $M = m(2n - 1)$,*

(i) *the number of saturating pushes is bounded by $cN^{1/2}M^{1/2} + 2\#\text{pec}$ for some constant c,*

(ii) *the running time of the algorithm is $O(N^{1/2}M^{1/2} \log n + \#\text{pec} \log n) + C_{\mathscr{A}}(N, M)$ and the number of flow operations is $O((N^{1/2}M^{1/2} + \#\text{pec})\log n)$,*

(iii)

$$\#\text{pac} \leq P_{\mathscr{A}}(N, M),$$

and

(iv) *the number of adversary edge kills is less than $K(N, M) + 2P_{\mathscr{A}}(N, M)$ for some $K(N, M) = O(N^{1/2}M^{1/2})$. (That is, the adversary only makes as many adversary edge kills as are allowed in the game.)*

Proof of Theorem 3. Item (i) can be proved by examining the proof of lemma 7.4 in [5]. Item (ii) is proven as Lemma 7.5 in [6]. We remark that we use #pec where [6] uses #ptr, since they denote the number of analogous events for the purposes of the proof in [6].

We define *#adversary edge kills*, *#edge designations*, to be respectively the number of adversary edge kills and the total number of edge designations in the game. And we define *#saturating pushes*, *#edge selections*, and *#relabelings* to be respectively the number of saturating pushes, the number of current edge selections, and the number of node relabelings in the push/relabel/add-edge algorithm. As shorthand, we define $P(\cdot, \cdot)$ to be $P_{\mathscr{A}}(\cdot, \cdot)$, and N to be $n(2n - 1)$ and M to be $m(2n - 1)$.

We prove (iii) by deriving some relationships among various events in the game and in the algorithm as follows.

We relate #edge selections to #edge designations by noting that there is at most one edge selection in the push/relabel/add-edge algorithm for each edge designation in the game except the N initial designations, since initially no node has a current edge in the push/relabel/add-edge algorithm. That is,

$$\#\text{edge selections} \leq \#\text{edge designations} - N. \tag{1}$$

From the correspondence between the push/relabel/add-edge algorithm and the game, every designated edge is eventually either removed by an adversary edge kill or by a node removal or by a redesignation. Since the latter two actions cause a point to be scored in the game, the total number of edge designations is equal to sum of the number of adversary edge kills and the number of points scored in the game. That is,

$$\#\text{edge designations} \leq \#\text{adversary edge kills} + P(N, M). \tag{2}$$

By definition, the total number of premature edge changes (#pec) is the number of current edge changes minus the number of saturating edge pushes. Since for each change there is an edge selection, the following inequality holds:

$$\#\text{pec} \le \#\text{edge selections} - \#\text{saturating pushes}. \qquad (3)$$

We notice that each adversary edge kill is caused either by a saturating push in the push/relabel/add-edge algorithm or by a node relabeling. Thus,

$$\#\text{adversary edge kills} \le \#\text{saturating pushes} + \#\text{relabelings}$$

Since the maximum label of any node is $2n - 1$ and each relabel increases the value of a node's label, we have

$$\#\text{relabelings} \le n(2n-1) = N.$$

Thus,

$$\#\text{adversary edge kills} \le \#\text{saturating pushes} + N. \qquad (4)$$

Combining Eq. (3) with the equation above gives

$$\#pec \le \#edge\ selections - \#adversary\ edge\ kills + \mathrm{N}.$$

Finally, using Eqs. (1) and (2) with the equation above we get

$$\#\text{pec} \le P(N, M).$$

Thus, (iii) holds.
From (i), we have

$$\#\text{saturating pushes} \le cN^{1/2}M^{1/2} + 2P(N, M).$$

To prove (iv), we combine the equation above with Eq. 4 as follows:

$$\begin{aligned}\#\text{adversary edge kills} &\le \#\text{saturating pushes} + N &(5)\\ &\le (cN^{1/2}M^{1/2} + N) + 2P(N, M). &(6)\end{aligned}$$

Thus, if we set $K(N, M) = cN^{1/2}M^{1/2} + N = O(N^{1/2}M^{1/2})$, (iv) holds. ∎

The following theorem relate the time of a version of the Cheriyan *et al.* algorithm to a play on the game that we define in this paper and follows directly from the theorem above.

THEOREM 4. *Given a deterministic strategy, $\mathscr{A}$, for playing our game, there is a deterministic maximum flow algorithm that runs in time*

$$O\big(n^{3/2}m^{1/2}\log n + P_{\mathscr{A}}(n(2n-1), m(2n-1))\log n\big) + C_{\mathscr{A}}(n(2n-1), m(2n-1)),$$

and uses at most $O(n^{3/2}m^{1/2} + P_{\mathscr{A}}(n(2n-1), m(2n-1))\log n)$ flow operations.

We proceed by describing various strategies for playing the game of Sect. 2.

4. The Idea

To illustrate the main idea, we make two simplifying assumptions: that there are no adversary edge kills and that each node has degree at least l. We prove the following:

THEOREM 5. *While the degree of every node in U is at least l, the player has a strategy which keeps the adversary's score below $O(N^{\epsilon}M/l)$ for any parameter $\epsilon > 3/\sqrt{\log n}$.*

We use the following concept throughout the remainder of the paper.

DEFINITION 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.

The player never redesignates an edge. She uses the following method to decide which edge to designate:

PLAYER'S STRATEGY. To designate an edge for $u \in U$, pick $a(u, v)$ such that $r(v)$ is minimal over all v incident to u.

We observe that since there are no adversary edge kills and the player never redesignates an edge, the ratio of a node never decreases.

Let $V_r = \{$nodes in V with ratios at least $r\}$. Let $U_r = \{$nodes in U whose designated edges have endpoints in $V_r\}$.

CLAIM 6. *For any ratio $r \geq 5/l$:*

$$\sum_{v \in V_{r/2}} \textit{initial degree of } v \geq l|U_r|/3.$$

Proof of Claim. Let v be a node in V_r and let $U(v)$ denote the set of $u \in U$ whose designated edge is incident to v. Let $U'(v) \subset U(v)$ denote those nodes whose designated edges were designated when the ratio of v was at least $r/2$. It is easy to observe that if $rl \geq 5$, then $|U'(v)| \geq \lceil |U(v)|/2 \rceil \geq |U(v)|/3$.

Let $u \in U'(v)$. At the time the player designated edge (u, v), all the neighbors of u had ratios at least as great as the ratio of v, that is, at least $r/2$. Since each u has degree at least l, this implies there were at least

$l|U'(v)| \geq l|U(v)|/3$ edges incident to nodes which at some time had a ratio at least $r/2$. Since ratios never decrease, these nodes are currently in $V_{r/2}$.

We may repeat this argument for all $v \in V_r$. Since there is only one designated edge for each u, the sets $U(v)$ are disjoint. Thus, we have that the total number of edges incident to nodes with a ratio at least$r/2$ is at least

$$\sum_{v \in V_r} l|U'(v)| \geq \sum_{v \in V_r} l|U(v)|/3 = l|U_r|/3. \blacksquare$$

By the definition of ratio and the claim, we have

$$|U_{r/2}| \geq r/2 \sum_{v \in V_{r/2}} \text{initial degree of } v \geq (r/2)l|U_r|/3. \tag{7}$$

This is the central idea of our strategy. The expansion factor implied by this equation allows us to limit the maximum ratio of any node as stated in the claim below. In the next section, we use the same concept to bound the number of points yielded in the general strategy.

CLAIM 7. *While the degree of every node in U is at least l, the player's strategy keeps all ratios below N^ε/l for $\varepsilon \geq 3/\sqrt{\log N}$ and sufficiently large N.*

Proof of Claim. Suppose that for some $v, r(v)$ exceeds N^ε/l. Thus $|U_{N^\varepsilon/l}| \geq 1$.

We can apply Eq. 7 j times where the r used in the jth application is $r(v)/2^j = N^\varepsilon/l2^j$ as long as $N^\varepsilon/l2^j$ is greater than $5/l$. By doing this, we obtain the following equations:

$$\begin{aligned} |U_{N^\varepsilon/l2^j}| &\geq \left(\frac{N^\varepsilon}{3 \cdot 2^j}\right)\left(\frac{N^\varepsilon}{3 \cdot 2^{j-1}}\right) \cdots \left(\frac{N^\varepsilon}{3 \cdot 2}\right)(1) \\ &\geq N^{\varepsilon j}/3^j 2^{j^2}. \end{aligned}$$

For $\varepsilon \geq 3/\sqrt{\log N}$ and $j \leq \lceil 3/\varepsilon \rceil$, we note that $N^\varepsilon/2^j > 5$ for sufficiently large N, thus it is valid to apply Eq. 7 as above. Furthermore, when $j = \lceil 3/\varepsilon \rceil$,

$$|U_{N^\varepsilon/l2^j}| > N^3/(3^{3/\varepsilon+1}2^{(3/\varepsilon+1)^2}),$$

which is strictly larger than N for sufficiently large N when $\varepsilon \geq 3/\sqrt{\log N}$. This is a contradiction. $\blacksquare$

The proof of the theorem proceeds as follows. Since the maximum ratio is N^{ϵ}/l, the maximum number of points scored by the adversary for removing a node v is at most (the initial degree of v) $N^{\epsilon}/l + 1$. If we recall that the initial degree of any node v is at least l and we sum over the nodes in V, we obtain an $O(MN^{\epsilon}/l)$ bound on the total number of points that are scored.

4.1. *Limits of This Approach*

There are two problems with this approach. First, we know of no way for the player to quickly and exactly determine which node or nodes have the minimal ratio among the neighbors of a node $u \in U$, since these ratios are changing throughout the game. Our solution to this, which is presented in the next sections, is to use estimates of ratios which are only occasionally updated.

Second, when the degree of a node in U falls below l the expansion argument fails. We remedy this by applying the player's strategy only to a subset U' of U, which consists of nodes of degree at least l. That is, when a node's degree drops below l, its designated edge no longer contributes to the ratio of a node. This has the effect of causing ratios to drop, which adversely affects the expansion argument. A second source of ratio drops which were not considered here is the adversary edge kill. The player compensates for ratio drops by edge redesignations.

The next section discusses strategies for the general game, with no simplifying assumptions.

5. The Edge Designation Strategy

In this section, we describe the full details of an edge designation strategy.

Recall that the game is played on a graph $G = 1(U, V, E)$ with $|U| = |V| = N$ and $|E| = M$. The strategy is specified by three parameters; a real number r_0, and integer l, and a real number $x \geq 2$, where $r_0 l/x \geq 176$.

We make some definitions for the description of our strategy and its analysis. We define

$$V' = \{v \in V \text{ whose initial degree} \geq l\}$$

and

$$U' = \{u \in U \text{ whose degree} \geq l\}.$$

Note that the set V' is fixed throughout the game, while the set U' may be diminishing in size as the game proceeds since the degree of a node in U' may drop.

Now we describe the data structures that the player uses to implement her strategy. A ratio $r(v)$ is maintained for each node $v \in V'$. The *initial degree of a node* v is its degree in $G = (U, V, E)$. Let

$$r(v) = \frac{\text{number of designated edges in } U' \times \{v\}}{\text{initial degree of } v}.$$

The ratios are partitioned into levels. We define $r_i = (1 + 1/x)^i r_0$, for $i > 0$, where x and r_0 are the parameters defined above. Let $t = 3\lceil(\log N)/\log r_0 l - \log 88 - \log x)\rceil + 4$. For each $v \in V$ the *ratio level of* v, $rl(v)$, is defined to be

$$\begin{array}{llll} 0 & \text{if } v \in V \setminus V' \text{ or} & & r(v) < r_0 \\ i & \text{if } v \in V' & \text{and} & r(v) \in [r_{i-1}, r_i) \text{ and } 0 < i < t \\ t & \text{if } v \in V' & \text{and} & r(v) \geq r_{t-1}. \end{array}$$

We define V_k to be the nodes in V' that have ratio level at least k and U_k to be the nodes in U' whose designated edges are incident to nodes in V_k.

For each $v \in V'$ an *estimated ratio level*, $erl(v)$, is maintained, such that $erl(v) \in \{rl(v), rl(v) + 1\}$.

For each node $u \in U'$ and each level $i = 0, 1, \ldots, t$, a list of u's neighbors which have an estimated ratio level of i is maintained. We refer to the set of lists for a node u as *u's neighbor lists*.

The player maintains $r(v)$ and $erl(v)$ for each node in V, the neighbor list for each node u in U', a single neighbor lists for each node $u \in U \setminus U'$ consisting of all its neighbors regardless of their ratios, and a designated edge for each node in U and that is all. The player updates the data structures when the ratio level of a node in v rises or drops.

The ratio of a node $v \in V$ can only increase when the player designates an edge (u, v). When this happens, she adjusts the ratio level of v. The estimated ratio level of v and the neighbor lists of each neighbor of v in U' are adjusted only when $rl(v) > erl(v)$.

Three events may cause the ratio of a node v to drop: (i) an adversary edge kill (removal of a designated edge) incident to v, or (ii) the removal of node u from U' when (u, v) is a designated edge (which we define as a *u-node shift*), or (iii) redesignation by the player from an edge adjacent to v to some other edge. In these situations, the ratio level is adjusted, and the estimated ratio level and the neighbor lists of each neighbor of v in U' are adjusted only when $rl(v) < erl(v) - 1$.

The player's strategy may be described as follows:

EDGE-DESIGNATION STRATEGY. *When an edge must be designated for a node $u \in U'$, an edge incident to a node with lowest erl is chosen.*

When an edge must be designated for a node $u \in U \setminus U'$, any edge may be chosen.

Each time the player designates a new edge to a node v, except for the initial designations, the player checks to see if she has raised the ratio level of v to ratio level t. If so, she calls the RESET procedure repeatedly until no node has ratio level t.

Informally, the RESET procedure finds a level k' such that $|U_{k'-3}| > r_{k'-3}l|U_{k'}|/88x$. It undesignates enough edges so that the ratio level of every node is no greater than $k' - 2$. The neighbor lists are updated to reflect these changes. Each node in U' which no longer has a designated edge is then reassigned a designated edge, using the player's EDGE-DESIGNATION STRATEGY.

We describe the strategy more precisely in the following pseudocode.

Subroutines

```
UPDATE_RL(v)
Update $rl(v)$ to its correct value.

UPDATE_ERL($v$)
    $erl(v) := rl(v)$;
    For all $u \in U'$ incident to $v$, update $u$'s neighbor lists.

REMOVE_EDGE(($u, v$))
Remove $v$ from $u$'s neighbor lists
If $u \in U'$ and $u$'s degree falls below $l$
    $U' := U' \setminus \{u\}$
    If $(u, v)$ is a designated edge do
        UPDATE_RL($v$);
        If $rl(v) < erl(v) - 1$ then UPDATE_ERL($v$)

DESIGNATE_EDGE($u$)
If $u \in U \setminus U'$ then designate any edge incident to $u$
    Else do
        designate an edge $\{u, v\}$ such that $erl(v)$
            is minimal over all $v$ incident to $u$;
        UPDATE_RL(v);
        If $rl(v) > erl(v)$ then UPDATE_ERL($v$).
        Return ($v$).
```

Player's Initialization

```
For all $v \in V'$, $erl(v)$, $r(v) := 0$
For all $u \in U$, DESIGNATE_EDGE($u$)
```

Responses to Adversary Actions

```
1. When the adversary removes edge $(u, v)$ and does not remove $v$:
REMOVE_EDGE($(u, v)$)
If $(u, v)$ was a designated edge then
    $v'$ = DESIGNATE_EDGE($u$)
    If $rl(v') = t$,
        Then repeat RESET until every node's ratio level is less than $t$.
2. When the adversary removes a node $v \in V'$:
Respond as if the adversary removed
each edge $(u, v)$ in (any) sequence.
```

Reset Procedure

```
1. $k := t$.
2. While $|U_{k-3}| \geq (r_{k-3}l)|U_k|/88x$ do
        $k := k - 3$.
3. (a) $W_{k-1} := U_{k-1}$;
   (b) For each $v \in V_{k-1}$, do
        while $rl(v) \geq k - 1$ do
            undesignate any designated edge incident to $v$
              and some $u \in U'$;
            UPDATE_RL($v$);
      If $rl(v) > erl(v)$ or $rl(v) < erl(v) - 1$ then do UPDATE_ERL($v$);
   (d) For each $u \in W_{k-1}$ which does not have a designated edge,
DESIGNATE_EDGE($u$).
4. Return $(k)$.
```

6. Analysis of the Strategy

The *level of a RESET procedure call* is defined to be the value returned by the RESET procedure (i.e., the value of k after step 2 of RESET is executed.)

LEMMA 8. 1. *When the RESET procedure is called repeatedly, each successive call is at a level at least three lower than the level of the previous call.*

2. *The level of a RESET call is at least* 4.

3. *The total number of redesignations is no greater than* $2/5$ *of the sum of the number of u-node shifts and the number of adversary edge kills.*

The proof of Lemma 8 appears in the next subsections. Lemma 8 implies the following theorem:

THEOREM 9. *The total number of points scored by the adversary when the edge designation strategy of section 2 is used is*

$$O(Mr_{t-1} + lN + K(N,M)).$$

Proof. Statements (1) and (2) imply that the repeat RESET instruction terminates. We note that no node in V' with $rl = t$ is every removed by the adversary since its ratio is reduced by RESETs before the adversary's next move.

Since every node in V' which the adversary removes has ratio less than r_{t-1}, the adversary scores less than r_{t-1} points per edge removed, or less than Mr_{t-1} points total, from edges in $U' \times V'$. Additionally, the adversary may score $2Nl$ points from edges outside of $U' \times V'$. Thus the total number of points scored from node removals is less than $Mr_{t-1} + 2Nl$.

The adversary also scores a point for each redesignation. Since the total number of adversary edge kills is bounded above by $2P(N,M) + K(N,M)$ by the rules of the game and the total number of u-node shifts is bounded by N we can conclude from statement (3) of Lemma 8 that the number of redesignations is no greater than $2(2P(N,M) + N + K(N,M))/5$. Therefore,

$$P(N,M) \le Mr_{t-1} + 2Nl + 2(2P(N,M) + N + K(N,M))/5.$$

Solving for $P(N,M)$ gives

$$P(N,M) \le 5(Mr_{t-1} + 2Nl) + 2(K(N,M) + N). \blacksquare$$

6.1. *Expansion or Ratio Drops*

We first prove the following lemma and then show in the next subsection that Lemma 8 follows from it.

As before, we let V_i denote the nodes in V' that lie at or above ratio level i *at the current timestep* and let U_i denote the nodes in U' whose designated edges are incident to nodes in V_i at the current timestep. We say a node $u \in U$ is designated for a node $v \in V$ if its designated edge is incident to v.

LEMMA 10. *Fix any point of time as the "current timestep" in the execution and any level $k \in \{4, 7, \ldots, t\}$. Let RESET_TIME refer to the timestep of the most recent previous RESET call at a level no greater than k or to the timestep at the start of the player's initialization if there was no such RESET call.*

Then either

1. $|U_{k-3}| \geq r_{k-3} l |U_k| / 88x$, *or*

2. *the number of adversary edge kills plus the number of u-node shifts affecting nodes at level $k - 3$ which occurred between RESET_TIME and the current timestep is at least $r_{k-4} 20 l |U_k| / 44x^2$.*

Proof. We first prove the following claim:

CLAIM 11. *Let $\hat{V}$ denote the set of nodes v such that v was in a ratio level at least $k - 2$ at some point since RESET_TIME. Then*

$$\sum_{v \in \hat{V}} \textit{initial degreee of } v \geq l |U_k| / 2x. \tag{8}$$

Proof of Claim. Let $v \in V_k$. Let $U'(v)$ be the nodes in U' with designated edges incident to v. Let $\hat{U}'(v)$ be the nodes in $U'(v)$ that were designated to v at a time since RESET_TIME when the ratio level of v was at least $k - 1$ immediately preceding the designation. We note that the ratio level of v must have been below $k - 1$ since RESET_TIME since the ratio level of every node is at most $k - 1$ at some point during a RESET call at level no greater than k or during the player's initialization.

The number of edges designated to v must increase by a factor of $(1 + 1/x)$ from the time since RESET_TIME that $rl(v)$ became $k - 1$ to the time that $rl(v)$ became k. Thus, at least $\lfloor (1/x) |U'(v)| / (1 + 1/x) \rfloor = \lfloor |U'(v)| / (x + 1) \rfloor$ nodes are in $\hat{U}'(v)$.[5]

That is,

$$|\hat{U}'(v)| \geq \lfloor |U'(v)| / (x + 1) \rfloor \geq |U'(v)| 2x.^{6}$$

Let $u \in \hat{U}'(v)$. At the time the player designated edge (u, v), all the neighbors v' of u had $erl(v') \geq erl(v) \geq rl(v) \geq k - 1$. Since $rl(v')$ is always at least $erl(v') - 1$, $rl(v') \geq k - 2$ at this time. That is, $v' \in \hat{V}$.

We may repeat this argument for all $v \in V_k$. As there is only one designated edge for each u, the sets $U'(v)$ are disjoint. Since $\bigcup_{v \in V_k} U'(v) = U_k$, there are at least $|U_k| / 2x$ nodes in $\bigcup_{v \in V_k} \hat{U}'(v)$. For each such node all of its neighbors are in $\hat{V}$. Since each node in U_k has at least l neighbors, there are at least $l |U_k| / 2x$ edges incident to nodes in $\hat{V}$, and the claim is proved. ■

[5]This holds even when considering rounding since if $k' = k + \lfloor k/x \rfloor$ then $\lfloor k/x \rfloor \geq \lfloor k'/(x + 1) \rfloor$ and if $k' = k + \lceil k/x \rceil$ then $\lceil k/x \rceil \geq \lfloor k'/(x + 1) \rfloor$ for any k.

[6]The second inequality is derived from the fact that $x \geq 2$ and by observing that the initial degree of each node in V' is at least l and thus $|U'(v)| / x \geq r_{k-1} l / x \geq 176$, and thus $\lfloor |U'(v)| / x \rfloor \geq |U'(v)| / 2x$.

Let $\hat{V} \cap V_{k-3}$ be the set of nodes in $\hat{V}$ which have ratio level at least $k-3$ at the current time. Then there are at least $r_{k-4}\Sigma_{v \in \hat{V} \cap V_{k-3}}$ (initial degree of v) designated edges incident to $\hat{V} \cap V_{k-3}$, each of which corresponds to a node in U_{k-3}.

If $|U_{k-3}| \leq r_{k-3}l|U_k|/88x$, then

$$r_{k-4} \sum_{v \in \hat{V} \cap V_{k-3}} \text{initial degree of } v \leq \frac{r_{k-3}l|U_k|}{88x}.$$

Thus

$$\sum_{v \in \hat{V} \cap V_{k-3}} \text{initial degree of } v \leq \frac{l|U_k|}{8x}(1 + 1/x) \leq \frac{l|U_k|}{44x}. \qquad (9)$$

(The second inequality follows since $x \geq 1$.) Equations (8) and (9) imply that

$$\sum_{v \in (\hat{V} \setminus V_{k-3})} \text{initial degree of } v \geq \frac{l|U_k|}{2x} - \frac{l|U_k|}{44x} = \frac{21l|U_k|}{44x}. \qquad (10)$$

Each node in $\hat{V} \setminus V_{k-3}$ has dropped from level $k-2$ to below $k-3$. Thus the number of designated edges that were removed from these nodes since RESET_TIME and while they were in level $k-3$ is at least

$$\lfloor (r_{k-3} - r_{k-4})l21|U_k|44x \rfloor,$$

since each node in $\hat{V}$ has degree at least l. This quantity is greater than

$$(1/x)\frac{r_{k-4}20l|U_k|}{44x}$$

since $l/x > r_0 l/x > 176$.

Since the only RESETs after the call that defined RESET_TIME occurred at a level greater than k, and these caused designated edge removals only on nodes with ratio levels greater than $k-1$, we can conclude that all $r_{k-4}l20|U_k|/44x^2$ designated edge removals correspond to u-node shifts and adversary edge kills that occurred since RESET_TIME. ■

6.2. *Proof of Lemma* 8

The proof of Lemma 8 follows fairly easily from the previous lemma.

1. *When the RESET procedure is called repeatedly, each successive call is at a level at least three lower than the level of the previous call.*

Proof. Suppose there is an uninterrupted sequence of RESET calls in which a call of level p is followed any time later in the sequence by a call at level p', $p' \geq p$. Let the timestep in which the second call is made be the "current timestep" of Lemma 10. Fix k in Lemma 10 to be p'. Then either the call at level p was the call that defined RESET_TIME of Lemma 10, or there was another RESET call after RESET_TIME. In either case, consequence (2) of Lemma 10 is false since there were no adversary actions intervening between RESET_TIME and the timestep of the second call. Then consequence (1) is true at the time of the second RESET call. Therefore, the second RESET call cannot be at level p', giving a contradiction.

Also, RESET calls can only be made on levels p', where $(t - p') \bmod 3 = 0$; thus p' cannot be one or two levels below p. ■

2. *The level of a RESET call is at least* 4.

Proof. Suppose not. Step 2 in the RESET call must have been executed until $k = 1$, or $(t - 1)/3$ times. By the condition in step 2, for $k = 4, 7, 10, \ldots, t$, $|U_{k-3}| \geq r_{k-3} l |U_k|/88x$. We also have that $|U_t| \geq 1$, since $v' \in V_t$ when the RESET call is made. Then

$$|U_1| \geq (1 + 1/x)^{(t-3)+(t-6)+(t-9)+ \cdots +1} (r_0 l/88x)^{(t-1)/3}.$$

$$> (r_0 l/88x)^{(t-1)/3}.$$

For $t = 3\lceil \log N/(\log r_0 l - \log 88 - \log x)\rceil + 4$, $|U_1| > N$, which is a contradiction. ■

3. *The total number of redesignations is no greater than* 2/5 *of the sum of the number of u-node shifts and the number of adversary edge kills.*

Proof. Let $rdes(i, k)$ be the number of redesignations done in the ith RESET call of level k. Let $rdrop(i, k)$ be the number of ratio drops (downward adjustments to $r(v)$ over all $v \in V'$) caused by u-node shifts and adversary edge kills that affected nodes while they were in ratio level $k - 3$, and that occurred between the $(i - 1)$th and ith k-level RESET call. (The 0th RESET call will denote the start of the game.)

CLAIM 12. *$rdes(i,k) \le 2rdrop(i,k)/5$*

Proof of Claim. The number of redesignations in a RESET call at level k is the number of undesignations needed to ensure that no nodes remain at level $k-1$ or higher.

For any node whose ratio is no greater than $(1+1/x)^i r_0$, we only need to undesignate a $(1/x)/(1+1/x)$ fraction of the node's incident designated edges to ensure that its ratio drops to below $(1+1/x)^{i-1} r_0$. Thus, bringing all the nodes' ratios down to $(1+1/x)^{i-1} r_0$ when all the nodes currently have ratio at most $(1+1/x)^i r_0$ requires that at most $\sum_{v \in V_i} \lceil (1/(x+1)) \mathrm{des}(v) \rceil$ edges be undesignated, where $des(v)$ denotes the number of edges designated to v. This can be upper bounded by $|U_i|/(x+1) + |U_i|/(r_{i-1} l)$, since the degree of each node in V_i is at least l. Furthermore, since $r_0 l/x > 176$ and $x \ge 2$, we can bound this by $|U_i|/(x+1) + |U_i|/(176x) < (8/7)|U_i|/x$. That is, at most $(8/7)|U_i|/x$ undesignations are needed to bring the ratios of all the nodes to at most $(1+1/x)^{i-1} r_0$ if the ratios of all the nodes are no higher than $(1+1/x)^i r_0$.

We can lower the ratio levels to below $k-1$ as follows: lower each node with ratio of at most $(1+1/x)^i r_0$ to have ratio at most $(1+1/x)^{i-1} r_0$, starting with $i = i_t$, where i_t is the minimal integer where all nodes have ratio less than $(1+1/x)^{i_t} r_0$, and decrementing i until $i = k-1$.[7] Since the cost of bringing the ratios of all the nodes down from $(1+1/x)^i r_0$ to at most $(1+1/x)^{i-1} r_0$ is less than $(8/7)|U_i|/x$, the number of undesignations needed to ensure that no node remains in level $k-1$ is at most

$$(1/x) \sum_{k-1 \le i \le i_t} (8/7)|U_i|.$$

Since step 2 of the RESET did not terminate at a level higher than k and there are no adversary edge kills or u-node shifts on nodes with ratio larger than $(1+1/x)^{t-1} r_0$, we have for each $j > 0$

$$|U_{3j+k}| \le \frac{|U_{3(j-1)+k}|}{(r_0 l/88x)} \le \frac{|U_{3(j-1)+k}|}{2}.$$

(The second inequality above follows since $r_0 l/x \ge 176$.) We can use this equation to derive that

$$\sum_{i \ge k} |U_i| \le 3 \sum_{j \ge 0} |U_{3j+k}| \le 6|U_k|$$

[7]We remark that i_t may actually be larger than t since edges may have been designated by an immediately preceding call to the RESET procedure.

and that

$$\sum_{i \geq k-1} |U_i| \leq 7|U_{k-1}|.$$

Thus, only $(8/7x)(7|U_{k-1}) = 8|U_{k-1}|$ undesignations are needed to bring the ratio levels of all the nodes down to $k - 1$. Since $|U_{k-1}| \leq |U_{k-3}| \leq r_{k-3}l|U_k|/88x$, by the fact that step 2 of the RESET terminated at level k, we can conclude that

$$rdes(i, k) \leq 8r_{k-3}l|U_k|/88x^2. \tag{11}$$

Let the timestep in which the ith k-level RESET call is made be the current timestep of Lemma 10. Fix the level of the lemma to be k. Since consequence (1) of Lemma 10 fails in the k-level RESET call, consequence (2) holds. Now, $rdrop(i, k)$ is at least as great as the number of adversary edge kills or u-node shifts affecting nodes at level $k - 3$ which occurred between RESET_TIME and the current timestep since it includes these events in its count. Thus,

$$\begin{aligned} rdrop(i, k) &\geq 20r_{k-4}l|U_k|/44x^2 \\ &= 20\frac{r_{k-3}}{1 + 1/x}l|U_k|44x^2 \geq 20r_{k-3}l|U_k|/88x^2. \end{aligned}$$

(The final inequality follows from $x \geq 1$.) Combining Eq. (11) with the equation above yields

$$rdrop(i, k) \geq 20rdes(i, k)/8 = 5rdes(i, k)/2. \blacksquare$$

We prove item (3) by noting that each node shift or adversary edge kill is counted in only one $rdrop(i, k)$ and using the claim above as follows.

Total number of redesignations

$$\begin{aligned} &= \sum_{i=4,\dots} \sum_{k=7,10,\dots} rdes(i, k) \\ &\leq \sum_{i=4,\dots} \sum_{k=7,10,\dots} 2rdrop(i, k)/5 \leq 2\,(\text{total number of ratio drops})/5. \end{aligned}$$

$\blacksquare$

This completes the proof of Lemma 8.

In the next subsection, we calculate the cost of implementing the player's strategy.

6.3. *Cost of Implementing the Strategy*

THEOREM 13. *The implementation cost, $C(N, M)$, of the strategy of Section 5 is $O((t + x/r_0)(N + P(N, M) + K(N, M)) + M)$, where* $t = 3\lceil(\log N)/(\log r_0 l - \log 88 - \log x)\rceil + 4$.

Proof. The total implementation cost of the player is the cost of selecting designated edges plus the cost of maintaining the data structures that she uses.

The total number of times edges are designated is no greater than $N + 3P(N, M) + K(N, M)$, since by the rules of the game

- the number of initial designations is at most N,
- the number of edge designations caused by node removal plus the number of redesignations done by the player is at most $P(N, M)$,
- the number of adversary edge kills is at most $2P(N, M) + K(N, M)$, and
- there are no other edge designations.

Selecting an edge of minimal estimated ratio level for any node can be performed in $O(t)$ time by scanning its t neighbor lists from low to high estimated ratio level. Thus, the total implementation cost for selecting designated edges is $O(t(N + P(N, M) + K(N, M)))$.

We enumerate the data structures that the player may change as follows. The player maintains:

(a) for each node in V, $rl(v)$,

(b) for each node in V, $erl(v)$,

(c) for each node $u \in U'$, a neighbor-list which, for each level $i = 0, \ldots, t$, has a pointer to a linked list of all nodes with $erl = i$ which are incident to u,

(d) and, for each node in U, the designated edge.

The implementation costs of the data structures in (a), (b), and (d) can be bounded as follows:

- Changes in rl require only constant time and occur only when an edge is designated (using $O(N + P(N, M) + K(N, M))$ time) or by an adversary edge kill (using $O(N + P(N, M) + K(N, M))$ time) or by a u-node shift (using $O(N)$ time).

• Changes in $erl(v)$ require only constant time. Whenever a change occurs the neighbor lists of all of v's neighbors are changed. Thus the cost is dominated by the cost of maintaining the neighbor lists.

• Changes in the designated edge of a node require constant time and only occur when there is an edge designation for that node (using $O(N + P(N, M) + K(N, M))$ time.)

Thus, we proceed by bounding the cost of maintaining the neighbor lists.

The neighbor lists are changed only at initialization and when UPDATE_ERL is called. The initialization cost is bounded by $O(tN + M)$, since for each node $u \in U'$, t lists are made, and all of u's neighbors are put in the 0th list.

An UPDATE_ERL may occur when the ratio level of a node v changes either by rising to a level above the current $erl(v)$, or falling two levels below the current $erl(v)$. Between any two consecutive upward revisions of $erl(v)$ either a downward revision of $erl(v)$ occurred or v's ratio rose a complete ratio level. Furthermore, a downward revision of $erl(v)$ can only occur if v's ratio has dropped a complete level since the last revision of $erl(v)$. That is, each call of UPDATE_ERL for a node v can be assigned at most two to one to a complete ratio level change of the node.

For the ratio to have changed a complete level, at least (r_0/x)(initial degree of v) ratio changes must have occurred. The cost per UPDATE_ERL for a node v is proportional to the degree of v which is bounded above by the initial degree of v. So the total cost of these revisions per node is no greater than

$$2\frac{(\text{number of changes to } r(v))(\text{initial degree of } v)}{(r_0/x)(\text{initial degree of } v)}$$

$$= 2\frac{x(\text{number of changes to } r(v))}{r_0}.$$

When summed over all nodes v, this is

$$2\frac{x(\text{number of ratio changes})}{r_0} = O\left(\frac{x(N + P(N, M) + K(N, M))}{r_0}\right).$$

Thus, $C(N, M)$ is $O(t(N + P(N, M) + K(N, M)) + M + x(N + P(N, M) + K(N, M))/r_0)$. ■

7. Specific Edge Designation Strategies and Maximum Flow Algorithms

In this section, we specify the parameters in the strategy of Section 5 to yield a strategy for playing the game. The strategy combined with Theorem 3 yields the main result of this paper.

Theorem 14. *There is a strategy, $\mathscr{A}$, where*

$$P_{\mathscr{A}}(N, M) = O\left(\sqrt{NM}\sqrt{D}\sqrt{\log N/\log D}\right)$$

and $C_{\mathscr{A}}(N, M) = O(M \log N/\log D)$ *for any parameter* D, *where* $176 < D < (M/N)/\log N$.

Proof. We note that, due to the constraints on D, the theorem only applies when $M > 176N \log N$.

We define a strategy $\mathscr{A}$ to be the edge designation strategy of Section 5, where we set $r_0 = (N/M)^{1/2}\sqrt{D}\sqrt{\log N/\log D}$, $x = \log N/\log D$, and $l = \lceil D \log N/\log D)/r_0 \rceil$. With this setting of parameters, the quantities t and r_t in the strategy can be computed as follows:

$$\begin{aligned} t &= \left\lceil \frac{\log N}{\log r_0 l - \log 88 - \log x} \right\rceil + 4 \\ &= O(\log N/\log D) \\ r_{t-1} &= (1 + 1/x)^{t-1} r_0 \\ &= O(r_0). \end{aligned}$$

We can now bound $P_{\mathscr{A}}(N, M)$ as follows

$$\begin{aligned} P_{\mathscr{A}}(N, M) &= O(Mr_{t-1} + lN + K(N, M)) \\ &= O\left(\sqrt{NM}\sqrt{D}\sqrt{\log N/\log D}\right) + O(K(N, M)) \\ &= O\left(\sqrt{NM}\sqrt{D}\sqrt{\log N/\log D}\right). \end{aligned}$$

The first equation above follows from Theorem 9, the second follows by substituting the appropriated values for r_{t-1} and l, and the third follows from the fact that $K(N, M) = O(\sqrt{NM})$ by the rules of the game.

We proceed by bounding $C_{\mathscr{A}}(N, M)$ as follows:

$$\begin{aligned} C_{\mathscr{A}}(N, M) &= O((t + x/r_0)(N + P(N, M) + K(N, M)) + M) \\ &= O((x/r_0)(N + P(N, M) + K(N, M)) + M) \\ &= O((x/r_0)(N + Nl + r_{t-1}M + 2K(N, M)) + M) \\ &= O((x/r_0)(r_{t-1}M) + M) \\ &= O(xM) \\ &= O(M \log N/\log D). \end{aligned}$$

The first equation follows from Theorem 13. The second follows from the fact that $t = O(x)$ and r_0 is less than 1. The third equation follows from Theorem 9. The fourth follows from the facts that $Nl = O(r_0 M)$ and that $N = O(Nl)$ and that $K(N, M) = O(r_t M)$ for the settings of r_0 and l that we are using. The fifth equation follows from the fact that $r_{t-1} = O(r_0)$, and the final equation follows by substituting $\log N/\log D$ for x. ■

COROLLARY 15. *There is a deterministic maximum flow algorithm that runs on an n-node, m-edge graph in time* $O(nm \log_{m/n \log n} n)$.

Proof. If $m < (176 \log 176) n \log n$, we use the algorithm of Goldberg and Tarjan [8]. Otherwise, we use Theorem 4 to conclude that there is a maximum flow algorithm whose running time is a asymptotically bounded by $P_{\mathscr{A}}(n(2n-1), m(2n-1))$ times $\log n$ plus $C_{\mathscr{A}}(n(2n-1), m(2n-1))$ plus $n^{3/2} m^{1/2} \log n$ for any games strategy $\mathscr{A}$.

Then we use a strategy described in Theorem 14, where we set D such that $D \log D = M/N \log N = m/n \log n$. The number of points for this strategy is bounded by $O(nm/\log D)$ and $C(n(2n-1), m(2n-1))$ is bounded by $O(nm \log n/\log D)$ by Theorem 14. ■

COROLLARY 16. *There is a deterministic maximum flow algorithm that given a positive constant δ and an n-node, m-edge graph with* $(m/n)^{2\delta} > (176 \log 176) \log n$ *runs in time*

$$O(nm \log_{m/n} n),$$

and uses at most

$$O(n^{3/2-\delta} m^{1/2+\delta} \log n)$$

flow operations.

Proof. Again, we use Theorem 4 to conclude that there is a maximum flow algorithm whose running time is asymptotically bounded by $P_{\mathscr{A}}(n(2n-1), m(2n-1))$ times $\log n$ plus $C_{\mathscr{A}}(n(2n-1), m(2n-1))$ plus $n^{3/2} m^{1/2} \log n$ for any game strategy $\mathscr{A}$.

Then we use a strategy described in Theorem 14, where we set D such that $D \log D = (M/N)^{2\delta}/\log N = O((m/n)^{2\delta}/\log n)$. The number of points for this strategy can be bounded by $O(n^{3/2-\delta} m^{1/2+\delta})$ and $C_{\mathscr{A}}(n(2n-1), m(2n-1))$ can be bounded by $O(nm \log n/\log(m/n))$ using Theorem 14 as long as $(m/n)^{2\delta} > (176 \log 176) \log n$. ■

REFERENCES

1. R. K. AHUJA AND J. B. ORLIN, A fast and simple algorithm for the maximum flow problem, *Oper. Res.* **37** (1989), 748–759.
2. R. K. AHUJA, J. B. ORLIN, AND R. E. TARJAN, Improved time bounds for the maximum flow problem, *SIAM J. Comput.* **18** (1989), 939–954.

3. N. Alon, Generating pseudo-random permutations and maximum flow algorithms, *Inform. Process. Lett.* **35** (1990), 201–204.
4. J. Cheriyan and T. Hagerup, A randomized maximum-flow algorithm, *in* "Proceedings of the 30th Annual Symposium on Foundations of Computer Science, 1989," pp. 118–123.
5. J. Cheriyan, T. Hagerup, and K. Mehlhorn, Can a maximum flow be computed in $o(nm)$ time? *in* "Proceedings of the 17th International Colloquium on Automata, Languages and Programming, 1990," pp. 235–248.
6. J. Cheriyan, T. Hagerup, and K. Mehlhorn, "An $o(n^3)$-Time Maximum-Flow Algorithm," Technical Report, Max-Planck-Institutfür Informatik, Saarbrücken, Germany, 1990.
7. A. V. Goldberg, É. Tardos, and R. E. Tarjan, Network flow algorithms, *in* "Paths, Flows, and VLSI-Layout," (B. Korte, L. Lovász, H. J. Prömel, and A. Schrijver, Eds.), pp. 101–164, Springer-Verlag, Berlin, 1990.
8. A. V. Goldberg and R. E. Tarjan, A new approach to the maximum-flow problem, *J. Assoc. Comput. Mach.* **35** (1988), 921–940.
9. V. King, S. Rao, and R. Taran, A faster deterministic maximum flow algorithm, *in* "Proceedings of the Third Annual ACM–SIAM Symposium on Discrete Algorithms, 1992," pp. 157–164.
10. S. Phillips and J. Westbrook, Online load balancing and network flow, *in* "Proceedings of the 25th Annual ACM Symposium on Theory of Computing, 1993," pp. 402–411.
11. D. D. Sleator and R. E. Tarjan, A data structure for dynamic trees, *J. Comput. System Sci.* **26** (1983), 362–391.
12. R. E. Tarjan, "Data Structures and Network Algorithms," SIAM, Philadelphia, 1983.