

FAST, DISTRIBUTED APPROXIMATION ALGORITHMS FOR POSITIVE LINEAR PROGRAMMING WITH APPLICATIONS TO FLOW CONTROL*

YAIR BARTAL[†], JOHN W. BYERS[‡], AND DANNY RAZ[§]

Abstract. We study combinatorial optimization problems in which a set of distributed agents must achieve a global objective using only local information. Papadimitriou and Yannakakis [*Proceedings of the 25th ACM Symposium on Theory of Computing*, 1993, pp. 121–129] initiated the study of such problems in a framework where distributed decision-makers must generate feasible solutions to *positive linear programs* with information only about local constraints. We extend their model by allowing these distributed decision-makers to perform local communication to acquire information over time and then explore the tradeoff between the amount of communication and the quality of the solution to the linear program that the decision-makers can obtain.

Our main result is a distributed algorithm that obtains a $(1 + \epsilon)$ approximation to the optimal linear programming solution while using only a polylogarithmic number of rounds of local communication. This algorithm offers a significant improvement over the logarithmic approximation ratio previously obtained by Awerbuch and Azar [*Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science*, 1994, pp. 240–249] for this problem while providing a comparable running time. Our results apply directly to the application of network flow control, an application in which distributed routers must quickly choose how to allocate bandwidth to connections using only local information to achieve global objectives. The sequential version of our algorithm is faster and considerably simpler than the best known approximation algorithms capable of achieving a $(1 + \epsilon)$ approximation ratio for positive linear programming.

Key words. linear programming, approximation algorithm, primal-dual, flow control

AMS subject classifications. 68W15, 68W25

DOI. 10.1137/S0097539700379383

1. Introduction. Processors in a distributed environment must make decisions based only on local data; thus fast distributed algorithms must often do without global information about the system as a whole. This is exactly why computing many target functions in distributed models quickly is provably hard [15]. However, quite surprisingly, some of the most interesting global optimization problems can be very closely approximated based only on local information and a modest amount of local communication.

Our work is motivated by the application of developing flow control policies which must achieve global objective functions. Flow control is the mechanism by which routers of a network distribute the available network bandwidth across connections. In our work, routing policies determine the routes in the network that connections must use to transmit packets. Regulating the rates at which the connections may inject data along these fixed routes is the problem of flow control. This connection-oriented, or rate-based, approach to flow control is a standard for routing available bit-rate

*Received by the editors October 11, 2000; accepted for publication (in revised form) December 30, 2003; published electronically August 6, 2004. Much of this work was completed while the authors were affiliated with the International Computer Science Institute in Berkeley, CA, and the University of California, Berkeley. This research was supported in part by NSF operating grants CCR-9304722 and NCR-9416101 and by a grant from the U.S.-Israel Binational Science Foundation. The work presented in this paper appeared in preliminary form in [6] and [10].

<http://www.siam.org/journals/sicomp/33-6/37938.html>

[†]Department of Computer Science, Hebrew University, Jerusalem, Israel (yairb@cs.huji.ac.il).

[‡]Department of Computer Science, Boston University, Boston, MA (byers@cs.bu.edu).

[§]Department of Computer Science, Technion, Haifa, Israel (danny@cs.technion.ac.il).

traffic in ATM networks [9] and is expected to become widely used in packet-switched networks. In this approach, each router in the network must make regulatory decisions based only on local information, which typically consists of the current transmission rates of connections using the router. Most existing flow control policies try to satisfy local objective functions such as *max-min fairness* [13, 7, 1, 11]. However, there are many other practical scenarios in which global objective functions are the appropriate choice. For example, in a commercial intranetwork in which users are paying for use of the network bandwidth (possibly at different rates), the administrator would want to use a flow control policy which maximizes total revenue. We express such a flow control policy objective as a *positive linear program*, a linear program (LP) in which all entries of the constraint matrix are nonnegative. Complicating the issue is the problem that routers must generate feasible solutions to this LP quickly, and based only on available information.

Motivated by this application and other related applications, Papadimitriou and Yannakakis considered the problem of having distributed decision-makers assign values to a set of variables in an LP, where the agents have limited information [18]. In one scenario that they describe, each agent, acting in isolation, must set the value of a single primal variable, knowing only the constraints affecting that variable in the LP. In the context of flow control, where the objective is to maximize the total flow through the network, this corresponds to a setting in which connections only know how many other connections share each of the routers they intend to use. When all edge capacities are 1, their “safe” algorithm sets each connection’s flow to the reciprocal of the maximum number of connections which share an edge with that connection. It is not hard to see that the worst-case approximation ratio achieved by the “safe” algorithm is $\Theta(\Delta)$, where Δ is the maximum number of connections that share an edge. They also prove that the “safe” algorithm achieves the best possible worst-case ratio when agents may not communicate, leaving open the possibility that much better ratios can be obtained when agents can interact.

We extend their model to allow computation to proceed in a sequence of *rounds*, in each of which agents can communicate a fixed-size message to their immediate neighbors, where agents are neighbors if and only if they share one or more constraints in the LP. Our goal is to determine the number of rounds necessary to achieve a $(1 + \epsilon)$ approximation ratio to the optimum LP solution. Although we focus on the application of flow control, this study could also be performed on a range of resource allocation problems, including those described in [18]. We note that similar models for describing the interaction between connections and routers in both theoretical and practical evaluations of flow control policies have been suggested in [3, 1, 5, 16].

Of course, a centralized administrator with complete information could solve the problem exactly using one of the well-known polynomial-time algorithms for linear programming (see, for example, [14]). Recently, however, much faster algorithms that produce approximate solutions to positive LPs to within a $(1 + \epsilon)$ factor of optimal have been designed. The sequential algorithm of Plotkin, Shmoys, and Tardos [19] repeatedly identifies a globally minimum weight path and pushes more flow along that path. More recently, faster approximation algorithms have been considered for several related flow and bin-packing problems [8, 12] using this same principle of repeatedly choosing a good flow, and incrementally increasing the rate of that flow. The technical difficulty is to balance the amount of flow increase, such that the required approximation is achieved, with the number of needed steps (i.e., running time). In all of these algorithms a global operator choosing the appropriate unsatisfied constraints is used. Moreover, the more general multicommodity flow problem cannot be formulated as

a positive LP, unless the number of dual variables or the number of constraints is exponential. In these cases one must use a separation oracle, since only polynomially many flows can be handled. For positive LPs, however, the number of constraints is polynomially bounded and thus one can increase all dual variables simultaneously. This is the approach taken by the algorithm of Luby and Nisan [17]. This algorithm, which has both a fast sequential and parallel implementation, repeatedly performs a global median selection algorithm on the values of the dual variables to choose a threshold, then increases values of dual variables above this threshold. Although these algorithms have efficient implementations, they both perform global operations, which makes them unsuitable for fast distributed implementation, since emulating those global operations requires a polynomial number of distributed rounds, and we are interested in much more time-efficient solutions.

The only previously known result for a distributed flow control algorithm with a global objective function is an algorithm of Awerbuch and Azar [3] which gives a logarithmic approximation ratio and also runs in a polylogarithmic number of rounds. Their algorithm is based on fundamental results from competitive analysis [2, 4]. The deterministic algorithm we present produces $(1 + \epsilon)$ approximate solutions to positive LPs, both in general and for the flow control problem, and builds on ideas used in these other algorithms [3, 17, 19]. Our algorithm is most closely related to the algorithm of Luby and Nisan but eliminates the need for the complex global selection operations and a global normalization step upon termination, enabling fast implementation in a distributed setting. Those simplifications carry over to serial and parallel settings as well, where we have a dramatically simpler implementation which saves a $\frac{1}{\epsilon}$ factor in the running time over the Luby–Nisan algorithm. Finally, we can parameterize the algorithm to quantify a tradeoff between the number of rounds and the quality of the approximation. In practice, we can run the algorithm for any number of *phases*, with the guarantee that after a constant number of phases, we have a logarithmic factor approximation, and after a logarithmic number of phases, we have a $(1 + \epsilon)$ approximation.

The rest of the paper is organized as follows. We begin with a presentation of our algorithm first as an easily understandable and implementable serial algorithm for approximately solving positive linear programming in section 2. In section 2.3, we prove that the algorithm achieves a $(1 + \epsilon)$ approximation ratio, and we analyze its running time. We formulate our distributed model and give an explanation of the correspondence between flow control policies and positive LPs in section 3. Then in section 3.2, we present the distributed implementation applicable to the flow control problem and explain the modifications to the analysis for this case.

2. Sequential approximation algorithms. We consider positive LPs represented in the following standard form, which is well known to be as general as arbitrary positive linear programming.

<u>PRIMAL</u>	<u>DUAL</u>
$\max Y = \sum_{j=1}^n y_j$	$\min X = \sum_{i=1}^m x_i$
$\forall i, \sum_j a_{ij} y_j \leq 1$	$\forall j, \sum_i a_{ij} x_i \geq 1$
$\forall j, y_j \geq 0$	$\forall i, x_i \geq 0$
$\forall i, j, a_{ij} \geq 0$	$\forall i, j, a_{ij} \geq 0$

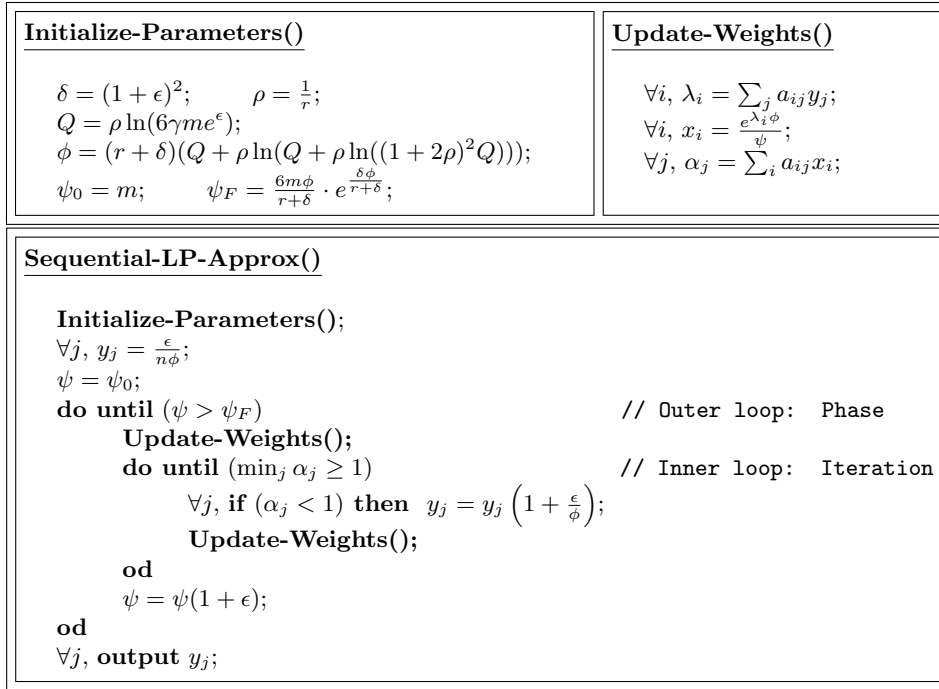


FIG. 1. The sequential positive LP approximation algorithm.

We will further assume that the LP is presented to the algorithm in a normalized form in which the a_{ij} are either 0 or satisfy $\frac{1}{\gamma} \leq a_{ij} \leq 1$. In the sequential case, one can convert an arbitrary LP in standard form into an LP in normalized form in linear time simply by dividing all constraints by $a_{max} = \max a_{ij}$ and setting $\gamma = \frac{a_{max}}{a_{min}}$ (where $a_{min} = \min\{a_{ij} | a_{ij} > 0\}$). We will describe how to perform this transformation efficiently in a distributed setting in section 3.

2.1. Overview of results. The parameterized algorithm for approximately solving a positive LP in normalized form is given in Figure 1. The main theorem that we will prove about the performance of this algorithm relates the quality of the approximation to the running time as follows.

THEOREM 1. *For any $0 < \epsilon \leq 1$ and $0 < r \leq \ln(\gamma m)$, the algorithm produces a feasible $(r + (1 + \epsilon)^2)$ approximation to the optimum primal linear programming solution and runs in $O\left(\frac{nm \ln(\frac{\gamma m}{r})}{r\epsilon}\right)$ time.*

The following corollary clarifies the tradeoff between the running time and the quality of the approximation and follows directly from Theorem 1.

COROLLARY 2. *For any $\epsilon \leq 1$, the algorithm produces a $(1 + \epsilon)$ approximation to the optimum in $O\left(\frac{nm \ln(\frac{\gamma m}{\epsilon})}{\epsilon^2}\right)$ time. For any $1 \leq r' \leq \ln(\gamma m)$, the algorithm produces a $(1 + r')$ approximation to the optimum in $O\left(\frac{nm \ln^2(\gamma m)}{r'}\right)$ time.*

Proof of Corollary 2. To prove the first claim of the corollary, set $r = \epsilon$ in Theorem 1, and to prove the second, set $r = 1 + r' - (1 + \epsilon)^2$ and choose ϵ so that $0 < \epsilon < \sqrt{2} - 1$, which implies $r > 0$ in Theorem 1. □

2.2. Description of the algorithm. In the sequential implementation of our algorithm presented in Figure 1, the main body of the program in the bottom panel

runs in a sequence of *phases*, the number of which ultimately depends only on the desired approximation ratio. Within a phase, values of appropriate primal variables y_j are increased monotonically until a threshold is reached. We refer to a set of increase operations across all primal variables constituting the inner loop of our main program as an *iteration* of our algorithm, noting that the number of iterations per phase may vary. We will demonstrate that at the time slot t ending each phase, the nonnegative settings for the primal variables $y_j(t)$ and the dual variables $x_i(t)$ are primal and dual feasible, respectively, satisfying the constraints below.

$$\text{Primal Feasibility: } \forall i, \sum_{j=1}^n a_{ij}y_j(t) \leq 1.$$

$$\text{Dual Feasibility: } \forall j, \sum_{i=1}^m a_{ij}x_i(t) \geq 1.$$

2.2.1. Moving between pairs of feasible solutions. Since the values of primal variables increase monotonically in our algorithm, it is crucial to carefully select the variables to increase in each phase. To this end, our algorithm uses exponential weight functions which have been employed in a variety of related contexts, including [2, 4, 17, 19, 3]. To do so, we associate each dual constraint with a measure $\alpha_j(t) = \sum_i a_{ij}x_i(t)$, and we associate each primal constraint with a measure $\lambda_i(t) = \sum_j a_{ij}y_j(t)$. Throughout the algorithm, the values of the dual variables x_i are tied to the values of “neighboring” primal variables y_j by the exponential weighting function defined in **Update-Weights()**:

$$\forall t : x_i(t) = \frac{e^{\lambda_i(t)\phi}}{\psi} = \frac{e^{\sum_j a_{ij}y_j(t)\phi}}{\psi},$$

where ϕ is a constant which again depends only on the desired approximation ratio, and ψ is a scaling factor which is initialized to m , then grows geometrically with the phase number until it reaches ψ_F , which is the termination condition.

By establishing this connection between primal and dual variables, when the value $\alpha_j(t)$ is less than 1, the dual constraint $\sum_i a_{ij}x_i(t) \geq 1$ is violated, but can be satisfied by a sufficient increase in y_j . This relationship suggests the following idea for an algorithm, which we employ. Start with a pair of dual and primal feasible solutions. Scale down the dual feasible solution by a multiplicative factor, making the solution dual infeasible, and causing some dual constraints to be violated. Then move back to a dual feasible solution by increasing those primal variables j for which $\alpha_j(t) < 1$, and repeat the process until a satisfactory approximation is achieved.

A hypothetical depiction of the intermediate primal and dual feasible solutions $Y(t) = \sum_j y_j(t)$ and $X = \sum_i x_i(t)$ at the end of each phase relative to the value of the optimal solution is shown in Figure 2. We maintain the invariant that the values of the intermediate primal feasible solutions are monotonically nondecreasing over time, but no such guarantee is provided for the intermediate dual feasible solutions. Linear programming feasibility ensures that values of intermediate primal feasible solutions are necessarily smaller than or equal to the value of the program, denoted by OPT, and similarly, values of intermediate dual feasible solutions are necessarily larger than or equal to the value of the program. Upon termination, we prove that the final (and maximal) primal feasible solution is within a desired (in this case, $1 + \epsilon$) factor of the minimal intermediate dual feasible solution. By linear programming duality, this

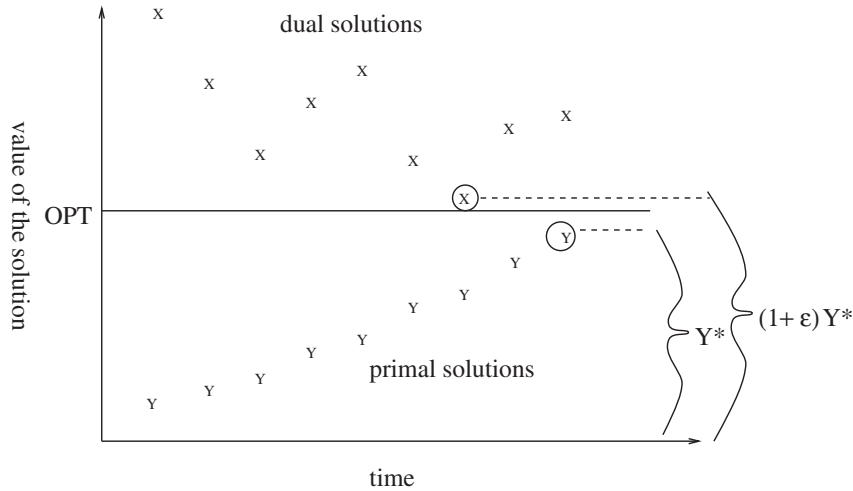


FIG. 2. Intermediate primal and dual feasible solutions.

implies that the final primal feasible solution gives the desired approximation to the value of the program.

In the sequential implementation presented in Figure 1, the bottleneck operation is to recompute the α_j 's after each iteration, which takes $O(nm)$ time. In fact, the running time of this bottleneck operation can be more precisely written as $O(E)$, where E is the total number of nonzero entries of the constraint matrix of the LP. When multiplied by the total number of iterations, which we demonstrate to be at most polylogarithmic in section 2.3, we have the bound on the total sequential running time.

2.3. Analysis of the algorithm. In this section, we bound the approximation ratio of our approximation algorithm and present an analysis of its sequential running time. We will later extend both of these results in a straightforward way to the distributed case.

First we prove a claim we made earlier, that at the end of each phase both the primal and the dual solutions are feasible. From the definition of the algorithm, the values of primal variables increase monotonically, and therefore the values of intermediate primal solutions also increase monotonically. Thus to carry out the analysis of the approximation ratio, it will remain only to prove that the value of the final primal feasible solution and the value of the minimal dual feasible solution are at most a $(1 + \epsilon)$ factor apart. In the proof we use the following three facts, which all follow from the initialization of the parameters given in **Initialize-Parameters()** in Figure 1.

FACT 3. $\phi \geq \epsilon$.

Proof. Fact 3 immediately follows from the definition of ϕ and from the fact that $\epsilon \leq 1$. \square

FACT 4. $\phi = O\left(\frac{(r+\delta)\ln\left(\frac{\gamma m}{r}\right)}{r}\right)$.

Proof. By definition, $\phi = (r + \delta)(Q + \rho \ln(Q + \rho \ln((1 + 2\rho)^2 Q)))$ and $Q = \rho \ln(6\gamma m e^\epsilon)$. Thus $Q \geq \rho$ and we have $\phi = O((r + \delta)(Q + \rho \ln Q))$. Since by definition

we also have $Q = O(\rho \ln(\gamma m))$ (recall that ρ is a shorthand for $1/r$), it suffices to show

$$\rho \ln(\gamma m) + \rho \ln(\rho \ln(\gamma m)) = O(\rho \ln(\rho \gamma m)).$$

Now since $r \leq \ln(\gamma m)$, we have that $\ln(\gamma m) = O(\ln(\rho \gamma m))$, and $\ln(\rho \ln(\gamma m)) = O(\ln \rho + \ln \ln(\gamma m)) = O(\ln \rho + \ln(\gamma m)) = O(\ln(\rho \gamma m))$, and the result follows directly. \square

FACT 5. $e^{\phi - \epsilon} \geq \gamma \psi$.

Proof. Substituting ψ_F for ψ and multiplying by e^ϵ , we must show

$$e^\phi \geq (\gamma e^\epsilon) \frac{6m\phi}{r + \delta} e^{\frac{\delta\phi}{r+\delta}}$$

or

$$e^{\frac{r\phi}{r+\delta}} \geq (6\gamma m e^\epsilon) \left(\frac{\phi}{r + \delta} \right)$$

or

$$e^{\frac{\phi}{r+\delta}} \geq \left(6\gamma m e^\epsilon \frac{\phi}{r + \delta} \right)^{\frac{1}{r}}.$$

Now by taking the natural logarithm of both sides above, we need to show

$$\frac{\phi}{r + \delta} \geq \frac{1}{r} \left[\ln(6\gamma m e^\epsilon) + \ln \left(\frac{\phi}{r + \delta} \right) \right].$$

Recall from the definitions that $\rho = \frac{1}{r}$, $Q = \rho \ln(6\gamma m e^\epsilon)$, and $\frac{\phi}{r+\delta} = Q + \rho \ln(Q + P)$, where $P = \rho \ln((1 + 2\rho)^2 Q)$. Therefore, it is enough to prove the following:

$$\begin{aligned} & Q + \rho \ln(Q + P) - Q - \rho \ln(Q + \rho \ln(Q + P)) \\ &= \rho \ln \left(\frac{Q + P}{Q + \rho \ln(Q + P)} \right) \geq 0. \end{aligned}$$

Since P and ρ are clearly nonnegative, and $Q \geq 1$, to show that $\rho \ln\left(\frac{Q+P}{Q+\rho \ln(Q+P)}\right)$ is nonnegative we need $P \geq \rho \ln(Q + P)$ or, substituting for P ,

$$\begin{aligned} \rho \ln((1 + 2\rho)^2 Q) &\geq \rho \ln(Q + \rho \ln((1 + 2\rho)^2 Q)), \\ (1 + 2\rho)^2 Q &\geq Q + \rho \ln((1 + 2\rho)^2 Q), \\ 2Q(2\rho + 2) &\geq \ln((1 + 2\rho)^2 Q). \end{aligned}$$

Using $2 \ln((1 + 2\rho)Q) \geq \ln((1 + 2\rho)^2 Q)$, and $2Q(2\rho + 2) \geq 2Q(2\rho + 1)$, we have to show

$$2Q(2\rho + 1) \geq 2 \ln((1 + 2\rho)Q).$$

The final inequality follows from the fact that $x \geq \ln x$, which completes the proof. \square

2.4. Feasibility. Recall that the algorithm maintains the invariant that dual feasibility is achieved prior to each increase in ψ .

FACT 6 (dual feasibility). *At the end of each phase, for all j , $\alpha_j \geq 1$.*

We next prove that the y_j 's are primal feasible throughout the execution of the algorithm, using Claim 7 to help perform the induction. In the proof it will be convenient to treat the initial increase of y_j 's from 0 to their initialized value as iteration 0.

CLAIM 7. *For all i and for every iteration, $\Delta\lambda_i \leq \frac{\epsilon}{\phi}$.*

CLAIM 8 (primal feasibility). *For all i , $\lambda_i \leq 1$ throughout the execution of the algorithm.*

We prove these two claims simultaneously by induction over iterations of the algorithm.

Proof. Let $J_i = \{j | a_{ij} > 0\}$. The first step is to prove that the claims hold for iteration 0:

$$\lambda_i = \sum_{j \in J_i} a_{ij} y_j \leq \sum_{j \in J_i} a_{ij} \frac{\epsilon}{n\phi} \leq \frac{\epsilon}{\phi}.$$

Since $\phi \geq \epsilon$ this also implies that Claim 8 holds at iteration 0.

Consider a subsequent iteration, and let Δv denote the change in variable v during that iteration. We have that for all j , $\Delta y_j \leq y_j \frac{\epsilon}{\phi}$ by the rate of increase in an iteration, so for all i ,

$$\Delta\lambda_i = \sum_j a_{ij} \Delta y_j \leq \sum_j a_{ij} y_j \frac{\epsilon}{\phi} = \lambda_i \frac{\epsilon}{\phi} \leq \frac{\epsilon}{\phi},$$

where the final inequality holds by the inductive hypothesis of Claim 8. This completes the proof of Claim 7.

To complete the proof of Claim 8, we consider two cases for λ_i separately. We first consider values i for which $\lambda_i < 1 - \frac{\epsilon}{\phi}$ prior to an iteration. From the proof of Claim 7 we have that after such an iteration, $\lambda'_i \leq \lambda_i + \frac{\epsilon}{\phi} < 1$, giving the desired result.

Next we consider those i for which $\lambda_i \geq 1 - \frac{\epsilon}{\phi}$ prior to an iteration. Fix such an i and fix any $j \in J_i$. We have that

$$\alpha_j = \sum_k a_{kj} x_k \geq a_{ij} x_i = a_{ij} \frac{e^{\lambda_i \phi}}{\psi} \geq a_{ij} \frac{e^{\phi - \epsilon}}{\psi}.$$

By Fact 5, we have that by our choice of ϕ , $e^{\phi - \epsilon} \geq \gamma\psi$, and hence $\alpha_j \geq a_{ij}\gamma \geq 1$, by the definition of γ . By the increase rule in the algorithm, we never increase the value of primal variable y_j if $\alpha_j \geq 1$, so in fact no primal variable in J_i increases in this iteration. Therefore, λ_i does not increase during this iteration and remains smaller than 1 by the induction hypothesis, completing the proof. \square

2.5. Proof of a $(1 + \epsilon)$ approximation ratio. We now turn to bound the approximation ratio obtained by the algorithm stated as the first half of Theorem 1.

CLAIM 9. *For any $0 < \epsilon \leq 1$ and $0 < r \leq \ln(\gamma m)$, the algorithm produces a feasible $(r + (1 + \epsilon)^2)$ approximation to the optimum primal linear programming solution.*

We use the notation $\Delta Y = \sum_j \Delta y_j$ to denote the aggregate change in the y values over the course of an iteration and similar notation for other variables. We begin with the following lemma.

LEMMA 10. *For every iteration,*

$$\frac{\Delta X}{\Delta Y} \leq \phi(1 + \epsilon).$$

Proof. As $\epsilon \leq 1$, we have from Claim 7 that $\Delta\lambda_i\phi \leq \epsilon \leq 1$. It follows that

$$\begin{aligned} \Delta x_i &= x_i (e^{\Delta\lambda_i\phi} - 1) \\ &\leq x_i \Delta\lambda_i\phi(1 + \Delta\lambda_i\phi) \\ &\leq x_i \Delta\lambda_i\phi(1 + \epsilon), \end{aligned}$$

using the inequality $e^z - 1 \leq z(1 + z)$ for $z \leq 1$.

Prior to a given iteration, let $S = \{j | \alpha_j < 1\}$. S is the set of indices of primal variables which will be active in the upcoming iteration, i.e., those variables y_j whose values will increase in the iteration. (Recall that a variable y_j may be increased several times in a phase.) The lemma follows from the following sequence of inequalities:

$$\begin{aligned} \Delta X &= \sum_i \Delta x_i \leq \sum_i x_i \Delta\lambda_i\phi(1 + \epsilon) \\ &= \sum_i x_i \sum_{j \in S} a_{ij} \Delta y_j \phi(1 + \epsilon) \\ &= \sum_{j \in S} \Delta y_j \sum_i a_{ij} x_i \phi(1 + \epsilon) \\ &= \sum_{j \in S} \Delta y_j \alpha_j \phi(1 + \epsilon) \\ &< \Delta Y \phi(1 + \epsilon). \end{aligned}$$

The final inequality holds from the definition of S . \square

In stating and proving the next lemma, we require a more precise description of our notation. We now consider the change in the values of the dual variables X over the course of a *phase*. In the proof, we let X' denote the sum of the x_i at the end of the current phase, and we let X denote the sum of the x_i at the end of the previous phase, i.e., before they are scaled down. We let ΔX denote the change in the sum of the x_i over the course of the current phase. We further define X^* to be the minimum over all dual feasible solutions obtained at the end of each phase and let Y_L be the primal feasible solution obtained at the end of the final phase. Fact 6 and Claim 8, respectively, imply that X^* is dual feasible and Y_L is primal feasible. In conjunction with Lemma 11 below, this implies the approximation result stated in Claim 9, by linear programming duality.

LEMMA 11.

$$Y_L \geq \frac{X^*}{r + (1 + \epsilon)^2}.$$

Proof. We prove the lemma for two separate cases: the first is an easy case in which the initial primal feasible solution is a close approximation to the optimum, and in the second we repeatedly apply Lemma 9 to bound the ratio between X^* and Y_L . Let X_0 denote the value for X before the initialization of the y_j 's, that is, $X_0 = \sum_i \frac{e^0}{m} = 1$, and let X_1 denote the value for X after the first phase. Similarly, let X_L denote the value of the dual solution at the end of the final phase.

Case I. $X_L \leq \frac{r+\delta}{\phi(1+\epsilon)}(X_1 - X_0)$. Letting Y_1 denote the value of the primal solution at the end of the first phase, we apply Lemma 9 to the iterations comprising the first phase, giving us

$$X_1 - X_0 \leq Y_1\phi(1 + \epsilon).$$

Since the values of the primal feasible solutions increase monotonically throughout the course of the algorithm, the lemma holds by the following sequence of inequalities:

$$X^* \leq X_L \leq (r + \delta)Y_1 \leq (r + (1 + \epsilon)^2)Y_L.$$

Case II. $X_L > \frac{r+\delta}{\phi(1+\epsilon)}(X_1 - X_0)$. Since the values X are scaled by a $\frac{\psi}{\psi'} = \frac{1}{1+\epsilon}$ factor just following the end of each phase, the earlier definitions imply that

$$X' = X \frac{\psi}{\psi'} + \Delta X.$$

By rewriting this expression and applying the inequality $e^z \geq 1 + z$, we have

$$X' = X \frac{\psi}{\psi'} \left(1 + \frac{\Delta X(1 + \epsilon)}{X} \right) \leq X \frac{\psi}{\psi'} \left(e^{\frac{\Delta X(1+\epsilon)}{X}} \right).$$

Now, using $X^* \leq X$ and applying Lemma 10 yields

$$X' \leq X \frac{\psi}{\psi'} \left(e^{\frac{\Delta Y \phi(1+\epsilon)^2}{X^*}} \right).$$

Let ψ_1 and ψ_L to be the initial value of ψ and the value of ψ in the last phase of the algorithm, i.e., $\psi_L < \psi_F$. Using the bound above repeatedly to compare X_L with X_1 gives us

$$(1) \quad X_L \leq X_1 \frac{\psi_1}{\psi_L} \left(e^{\frac{Y_L \phi(1+\epsilon)^2}{X^*}} \right).$$

Since X_L is dual feasible and the optimal solution is bounded below by 1 (by the normalized form of the program) we have that $X_L \geq 1 = X_0$. Also note that by the assumption $r \leq \ln(\gamma m)$, we have $\phi \geq (r + \delta)$. We can now use these facts in conjunction with the fact that $X_L > \frac{r+\delta}{\phi(1+\epsilon)}(X_1 - X_0)$ to obtain

$$X_1 < \frac{X_L \phi(1 + \epsilon)}{r + \delta} \leq X_L \left(\frac{\phi(1 + \epsilon)}{r + \delta} + 1 \right) \leq X_L \frac{\phi(2 + \epsilon)}{r + \delta}.$$

Using the bound above in (1) and observing that $\psi_1 = m$ and $\psi_L \geq \psi_F/(1 + \epsilon)$ we get

$$e^{\frac{Y_L \phi(1+\epsilon)^2}{X^*}} \geq \frac{\psi_F}{m \frac{\phi(2+\epsilon)}{r+\delta} (1 + \epsilon)} = \frac{6m \frac{\phi}{r+\delta} e^{\frac{\delta \phi}{r+\delta}}}{m(2 + \epsilon)(1 + \epsilon) \frac{\phi}{r+\delta}} = \frac{6e^{\frac{\delta \phi}{r+\delta}}}{(2 + \epsilon)(1 + \epsilon)} \geq e^{\frac{(1+\epsilon)^2 \phi}{r+(1+\epsilon)^2}},$$

where the final inequality holds by substituting $\delta = (1 + \epsilon)^2$ and using $\epsilon \leq 1$. We therefore get

$$\frac{Y_L}{X^*} \geq \frac{1}{r + (1 + \epsilon)^2},$$

concluding the proof of the lemma for Case II. \square

2.6. Running time. For the algorithm provided so far, we have the following running time bounds, which are slightly weaker than those stated in Theorem 1. These weaker bounds are presented since a natural translation of these time bounds and the preceding analysis extends directly to the distributed algorithm which we present in section 3. After proving these bounds, we provide a simple improvement which applies only in the sequential case and which is used to give the time bounds stated in Theorem 1.

CLAIM 12. *The sequential algorithm runs in $O\left(\frac{(r+1)nm \ln^2\left(\frac{\gamma m}{r}\right) \ln\left(\frac{(r+1)\gamma n \ln m}{r\epsilon}\right)}{r^2\epsilon^2}\right)$ time.*

Proof. We bound the number of phases by measuring the change in ψ , which increases by a $(1 + \epsilon)$ factor per phase. From the definitions in Figure 1, and using Fact 4 for the final equality, we have that

$$\left\lceil \log_{1+\epsilon} \left(\frac{\psi_F}{\psi_0} \right) \right\rceil = O \left(\frac{\frac{\delta\phi}{r+\delta} + \ln\left(\frac{\phi}{r+\delta}\right)}{\epsilon} \right) = O \left(\frac{\phi}{(r+\delta)\epsilon} \right) = O \left(\frac{\ln\left(\frac{\gamma m}{r}\right)}{r\epsilon} \right).$$

We now bound the number of iterations in a phase by computing the maximum number of iterations needed to increase all α_j values above 1. For a given j , we say that y_j is *large* once $y_j \geq \frac{\gamma}{\phi} \ln(\gamma\psi)$. We show that once y_j is large, $\alpha_j \geq 1$, and therefore j no longer participates in the phase. Let the set $I_j = \{i | a_{ij} > 0\}$. Therefore for all $i \in I_j$,

$$\lambda_i = \sum_k a_{ik} y_k \geq \frac{1}{\gamma} y_j \geq \frac{1}{\phi} \ln(\gamma\psi),$$

$$\alpha_j = \sum_i a_{ij} x_i = \sum_i a_{ij} \cdot \frac{e^{\lambda_i \phi}}{\psi} \geq 1.$$

Initially, $y_j = \frac{\epsilon}{n\phi}$ and at every iteration it increases by a factor of $1 + \frac{\epsilon}{\phi}$. Therefore the number of iterations y_j can participate in before y_j becomes large is at most

$$\left\lceil \log_{1+\frac{\epsilon}{\phi}} \left(\frac{n\phi}{\epsilon} \cdot \frac{\gamma}{\phi} \ln(\gamma\psi) \right) \right\rceil = O \left(\frac{\phi}{\epsilon} \cdot \ln \left(\frac{\gamma n}{\epsilon} \ln(\gamma\psi) \right) \right).$$

From Fact 5 we have $\ln(\gamma\psi) = O(\phi)$. By using this fact, and by another application of Fact 4, we bound the number of iterations during a phase by

$$\begin{aligned} O \left(\frac{\phi}{\epsilon} \cdot \ln \left(\frac{\gamma n \phi}{\epsilon} \right) \right) &= O \left(\frac{r+\delta}{r\epsilon} \ln \left(\frac{\gamma m}{r} \right) \ln \left(\frac{\gamma n}{\epsilon} \frac{r+\delta}{r} \ln \left(\frac{\gamma m}{r} \right) \right) \right) \\ &= O \left(\frac{(r+1) \ln \left(\frac{\gamma m}{r} \right) \ln \left(\frac{(r+1)\gamma n \ln m}{r\epsilon} \right)}{r\epsilon} \right). \end{aligned}$$

Note that the term $\frac{r+1}{r}$ cannot be removed since r can be any value satisfying $0 < r \leq \ln(\gamma m)$. Multiplying this by the earlier bound on the number of phases and using the fact that each iteration can be computed in $O(nm)$ time, this completes the proof of Claim 12. \square

To prove the time bound stated in the second half of Theorem 1, we need to give a sequential algorithm which completes in $O\left(\frac{nm \ln\left(\frac{\gamma m}{r}\right)}{r\epsilon}\right)$ time. This running time can be achieved by an algorithm which performs exactly one iteration per phase. In

the sequential case, we accomplish this by increasing a candidate y_j not merely by a multiplicative factor of $1 + \frac{\epsilon}{\phi}$ per iteration, but by increasing it by the amount which causes α_j to reach a value of 1 directly. (Note that this procedure is straightforward to implement in the sequential case, but not in the distributed case.) This improvement causes each phase to terminate in a single iteration; Claims 7 and 8 still hold (the other claims are unaffected); and we achieve the time bound stated in Theorem 1.

3. The distributed model. We now consider the following model in the spirit of Papadimitriou and Yannakakis [18], in which distributed agents generate approximate solutions to positive LPs in the standard form presented in section 2.

We associate a *primal agent* with each of the n primal variables y_j and a *dual agent* with each of the m dual variables x_i . Each agent is responsible for setting the value of their associated variable. For any i, j such that $a_{ij} > 0$, we say that dual agent i and primal agent j are *neighbors*. In each distributed round of computation, each agent may broadcast a fixed-size message to all of its neighbors; i.e., in one round each primal agent j may broadcast one message to its set of dual neighbors $I_j = \{i | a_{ij} > 0\}$, and each dual agent i may broadcast one message to its set of primal neighbors $J_i = \{j | a_{ij} > 0\}$.

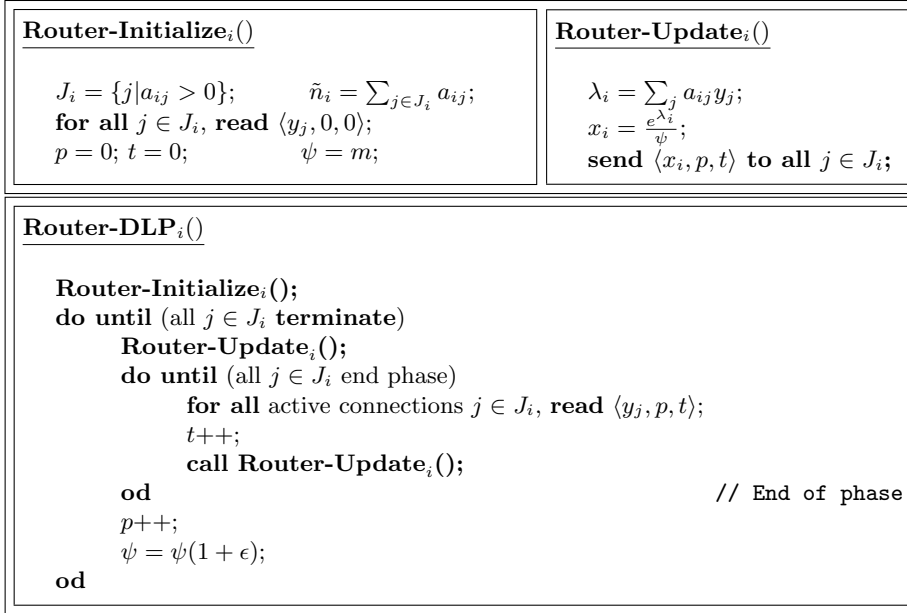
After a fixed number of rounds, the agents must choose feasible values for their variables to minimize (in the case of the primal) the approximation ratio, $\frac{\text{OPT}}{\sum y_j}$, where OPT is the value of the optimal solution to the LP. We then study the tradeoff between the number of rounds and the quality of the approximation ratio obtained.

The application of flow control in a network with per-flow queuing motivates the following mapping to our model of primal agents and dual agents. Each of n connections transmits data along a fixed path in the network, and a connection corresponds to a primal agent. Each of the paths traverses an ordered subset of the m routers which comprise the network, and these routers correspond to dual agents. At a given time step t , each connection j transmits at a given rate into the network, thereby establishing the value $y_j(t)$ of its primal variable. Once these new flow values stabilize, each router i uses its local load to set a value for the primal variable $x_i(t)$. Based on a simple function (the sum) of the values of dual variables along its path, the source uses this control information to compute a new flow value $y_j(t+1)$. To compute this sum, each connection transmits a fixed-length control message which loops through the routers along its path and back to the source. As mentioned earlier, this simple and natural model of communication between connections and routers corresponds to models previously suggested in both practical and theoretical studies of flow control [3, 1, 5, 16].

Each router i has capacity C_i , which it may share among the connections which utilize it, while each connection accrues benefit B_j for every unit of end-to-end capacity which it receives. Therefore, the connections act as the primal agents and the routers act as the dual agents in the following positive LP:

$$\begin{aligned} & \max \sum_{j=1}^n B_j y_j, \\ & \forall i, \sum_j \tilde{a}_{ij} y_j \leq C_i, \\ & \forall i, j, \tilde{a}_{ij} = 1 \text{ or } 0. \end{aligned}$$

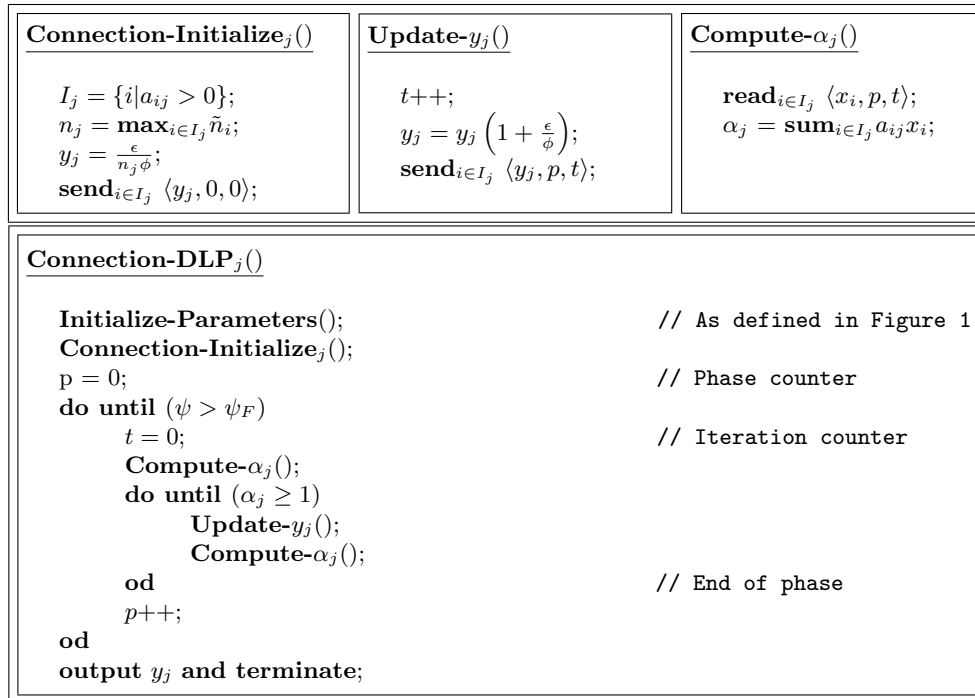
Clearly, this positive LP can be converted to standard form by the local operation $a_{ij} = \frac{\tilde{a}_{ij}}{B_j C_i}$. In a synchronous model, each round takes time equal to the maximum

FIG. 3. The distributed algorithm at router i .

round-trip time experienced by a connection in the network. However, this synchronization assumption can and will subsequently be relaxed with no changes to the algorithms we propose. A final note is that the message size we use in our implementation can be bounded by a number of bits polynomial in $\log m$, $\log \gamma$, and $1/\epsilon$.

3.1. The distributed approximation algorithm. Several additional complications must be addressed in the definition and description of the distributed algorithm provided in Figures 3 and 4 for routers and connections, respectively. Since global operations cannot be performed efficiently, each connection and router must be able to independently compute the values of all of the parameters described in the serial implementation. In the case of parameters which are fixed, such as the value of m (the number of nodes in the network), and for the parameters which affect the approximation ratio, r and ϵ , we assume that these values are known in advance to all connections and routers. We do not assume that n , the number of connections, is globally known. In the sequential case, knowledge of this parameter was required to initialize the variables y_j so as to satisfy Claims 7 and 8. In the distributed setting, each connection j instead computes a local estimate of n , n_j , which it can compute in two distributed rounds, and which then used in initialization satisfies Claims 7 and 8. Finally, the parameter γ used to convert the program into normalized form may not be globally known, in which case the LP cannot be normalized efficiently. Approximately solving such programs in the distributed setting adds considerable complexity, and we defer providing techniques for doing so until section 3.2.

Connections and routers communicate using the message-passing model described in section 3. As in the serial algorithm, agents track time in terms of phases and iterations. When transmitting the value of a variable using the **send** primitive, agents timestamp the transmission with their current phase number p and iteration t . Likewise, in receiving the value of a variable using the **read** primitive, agents specify

FIG. 4. The distributed algorithm at connection j .

their phase number p and iteration t , and they wait until they receive the appropriate value. For simplicity, we assume that these control messages reliably flow through the path, although, in practice, retransmissions would likely be necessary. Also, strict alternation between primal and dual rounds eliminates the possibility of deadlock.

In our implementation, message-passing primitives enable control to alternate between connections and routers at a local level. This is not to say that control is globally synchronized—in fact, at any instant in time, connections in separate areas of the network might not even be working on the same phase. However, it is the case that any given router is working on only a single phase at an instant in time. Therefore, all the connections through a router which is currently working on phase i are either actively working on phase i themselves or are idle and awaiting permission to proceed to phase $i + 1$. Aside from message-passing, the other technical obstacle in converting the centralized algorithm to a distributed algorithm is the condition for ending a phase. In the centralized algorithm, a phase terminates when $\min_k \alpha_k \geq 1$. Since we cannot hope to track the value of this global expression in our distributed model, we instead let each connection j check whether $\alpha_j \geq 1$ locally and independently. When the condition is satisfied, connection j terminates its phase by incrementing its phase number and informing its neighboring routers.

The analysis of feasibility and the bounds on the quality of the approximation are identical to those for the centralized algorithm. This is the case because the value of any primal variable at the time that the corresponding connection completes phase i satisfies the conditions placed on primal variables after phase i in the centralized implementation. A similar statement holds with respect to the values of dual variables at the time their corresponding routers complete phase i . These statements hold for

each primal and dual variable independently, and irrespective of the fact that phase completion times may not occur uniformly across the network. As for the distributed running time, the following corollary to Claim 12 holds.

COROLLARY 13. *The distributed algorithm runs in $O\left(\frac{(r+1) \ln^2\left(\frac{\gamma m}{r}\right) \ln\left(\frac{(r+1)\gamma n \ln m}{r\epsilon}\right)}{r^2 \epsilon^2}\right)$ rounds.*

3.2. Distributed techniques to convert to special form. Recall that we can convert a program in standard form to the normalized form by dividing all constraints by a_{max} , thereby setting $\gamma = \frac{a_{max}}{a_{min}}$. If bounds on the values of a_{max} and a_{min} are known in advance, for example, if connections and routers can all bound the min and max values of the edge capacities and benefit coefficients, then γ can be estimated. But these bounds may not be globally known; moreover, the value of γ , which impacts the running time of our algorithm, depends on the values of the entries of the matrix. We now show that solving a problem in standard form can be reduced to solving problems in a *special form* (similar to a form used by Luby and Nisan in [17]), where the value of γ depends only on m and ϵ and does not affect the approximation ratio obtained, nor does it significantly affect the running time of our algorithm. Moreover, this transformation can be done distributively in a constant number of rounds, without global knowledge of a_{max} and a_{min} .

A precondition for transforming an LP Z in standard form to an LP Z' in special form is that we can generate a feasible solution for Z with value c which approximates the value of the optimal solution for Z to within a factor of τ : $c \leq \text{OPT} \leq c\tau$. If this precondition is satisfied, we can perform the following transformation, which bounds the value of the a'_{ij} in Z' by $\frac{\epsilon^2}{m\tau} \leq a'_{ij} \leq 1$ for all i and j , giving $\gamma' = \frac{m\tau}{\epsilon^2}$. Note that the value of γ now depends on the extent to which we can bound the value of the LP, but not on the relative values of the constraints (which could be very small).

Define $\nu = \frac{m}{c\epsilon}$, and perform the following transformation operation on the constraints:

$$a'_{ij} = \begin{cases} \frac{\epsilon}{\nu\tau c} & \text{if } a_{ij} < \frac{\epsilon}{c\tau}, \\ 1 & \text{if } a_{ij} > \nu, \\ \frac{a_{ij}}{\nu} & \text{otherwise.} \end{cases}$$

This transformed LP has the following properties:

1. If $\{y'_j\}$ is a primal feasible solution for Z' , then

$$\left\{ y_j = \begin{cases} 0 & \text{if } \exists i \text{ such that } a'_{ij} = 1, \\ \frac{y'_j}{\nu} & \text{otherwise} \end{cases} \right\}$$

is primal feasible for Z , and $\sum_j y_j \geq \frac{\sum_j y'_j}{\nu} - \epsilon \cdot \text{OPT}$.

2. If $\{y_j\}$ is primal feasible for Z , then $\{y'_j = \frac{y_j\nu}{1+\epsilon}\}$ is primal feasible for Z' , and $\sum_j y'_j = \sum_j \frac{y_j\nu}{1+\epsilon}$.
3. $\frac{\epsilon}{\nu\tau c} \leq a'_{ij} \leq 1$ for all i and j .

Property 3 is clearly true, but the other two properties require the short proofs below.

Proof of property 1. Take a feasible solution $\{y'_j\}$ for Z' and let $\{y_j\}$ be as

specified. Then for any fixed value of i ,

$$\begin{aligned} \sum_j a_{ij} y_j &= \sum_{j|a'_{ij}<1} a_{ij} y_j \leq \sum_{j|a'_{ij}<1} \nu a'_{ij} y_j \\ &\leq \sum_{j|a'_{ij}<1} \nu a'_{ij} \frac{y'_j}{\nu} \leq 1. \end{aligned}$$

The final inequality holds by the feasibility of the solution $\{y'_j\}$, so the solution $\{y_j\}$ is feasible for Z . The value of this solution satisfies

$$\begin{aligned} \sum_j y_j &= \sum_j \frac{y'_j}{\nu} - \sum_{j|\exists i \text{ s.t. } a'_{ij}=1} \frac{y'_j}{\nu} \\ &\geq \sum_j \frac{y'_j}{\nu} - \sum_{j|\exists a'_{ij}=1} \frac{1}{\nu} \\ &\geq \sum_j \frac{y'_j}{\nu} - \frac{m}{\nu} \\ &\geq \sum_j \frac{y'_j}{\nu} - \epsilon \cdot \text{OPT}. \end{aligned}$$

The first inequality holds by the fact that for any feasible $\{y'_j\}$ and for any i and j , $a'_{ij} y'_j \leq 1$. The second inequality holds by the bound on the number of routers in the network, and the final inequality holds by the definition of ν . \square

Proof of property 2. Take a feasible solution $\{y_j\}$ for Z and let $\{y'_j\}$ be as specified. Then for any fixed value of i ,

$$\begin{aligned} \sum_j a'_{ij} y'_j &= \sum_j a'_{ij} \left(\frac{y_j \nu}{1 + \epsilon} \right) \\ &= \sum_{j|a_{ij} > \frac{\epsilon}{c\tau}} a'_{ij} \left(\frac{y_j \nu}{1 + \epsilon} \right) + \sum_{j|a_{ij} \leq \frac{\epsilon}{c\tau}} a'_{ij} \left(\frac{y_j \nu}{1 + \epsilon} \right) \\ &\leq \sum_{j|a_{ij} > \frac{\epsilon}{c\tau}} \frac{a_{ij} y_j}{1 + \epsilon} + \sum_{j|a_{ij} \leq \frac{\epsilon}{c\tau}} \frac{\epsilon y_j}{(1 + \epsilon) \tau c} \\ &\leq \frac{1}{1 + \epsilon} + \frac{\epsilon}{(1 + \epsilon) \tau c} \sum_j y_j \\ &\leq \frac{1}{1 + \epsilon} + \frac{\epsilon}{(1 + \epsilon)} = 1. \end{aligned}$$

The final line holds from the bound on the optimal solution for Z : $\sum_j y_j \leq \text{OPT} \leq c\tau$, so the solution y'_j is feasible. \square

Now we generate an approximate solution to Z by performing the transformation to special form and then computing a $(1 + \epsilon)$ approximation $\{y'_j\}$ for Z' using our algorithm. We transform this solution to $\{y_j\}$ using the transformation described in

property 1 and get a primal feasible solution Y for our LP such that

$$\begin{aligned} Y = \sum_j y_j &\geq \frac{\sum_j y'_j}{\nu} - \epsilon \cdot \text{OPT} \geq \frac{\text{OPT}'}{\nu(1+\epsilon)} - \epsilon \cdot \text{OPT} \\ &\geq \text{OPT} \left(\frac{1}{(1+\epsilon)^2} - \epsilon \right). \end{aligned}$$

The first inequality is from property 1, the second is based on the fact that $\{y'_j\}$ is a $(1+\epsilon)$ approximation to the value of Z' (denoted by OPT'), and the final inequality is from property 2.

Next we need to explain how to choose the parameters c and τ so as to guarantee the precondition $c \leq \text{OPT} \leq c\tau$. Recall that I_j denotes the set of edges incident to connection j , $I_j = \{i | a_{ij} > 0\}$, and J_i denotes the set of connections incident to edge i , $J_i = \{j | a_{ij} > 0\}$. Now define

$$\beta_i = \min_{l \in J_i} \max_{k \in I_l} a_{kl},$$

a quantity which can be locally computed in one round for each router i . Also, let $\beta = \min_i \beta_i$, and for each connection j , define $\hat{\beta}_j = \min_{i \in I_j} \beta_i$. It is relatively easy to show that $\frac{1}{\beta} \leq \text{OPT} \leq \frac{m}{\beta}$. The first inequality holds from the primal feasibility of the solution in which the connection j used in the evaluation of the minimum β_i is assigned flow $y_j = \frac{1}{\beta}$. The second inequality holds from the dual feasibility of the solution in which each router i is assigned weight $x_i = \frac{1}{\beta_i}$.

Therefore, we can set $c = \frac{1}{\beta}$, and $\tau = m$ in the sequential implementation, giving $\gamma' = \frac{m^2}{\epsilon^2}$ and an $O\left(\frac{nm \ln(m/r\epsilon)}{r\epsilon}\right)$ running time.

3.3. Approximating β in the distributed setting. In the sequential case, knowledge of β is enough to perform the transformation to special form, but connections and routers may not know this value. We now describe a technique in which we distributively subdivide the LP into subprograms based on local estimates of β . The value of each subprogram is bounded, so we can work in special form. Then we recombine solutions in such a way as to only assign nonzero rates to connections with good estimates of β , but we prove that this only reduces the sum of the rates by a small factor.

Set $p = \lceil \frac{1}{\epsilon} \rceil$, and for $q = 0, \dots, p-1$, define the sets

$$G_t^q = \left\{ j \mid \left(\frac{m}{\epsilon} \right)^{p(t-1)+q} \leq \hat{\beta}_j < \left(\frac{m}{\epsilon} \right)^{pt+q} \right\}$$

for integer t . It is clear that each connection belongs to exactly p of these sets. Independently for each value of q , each router i assigns flow only to connections which are members of $G_{T_{iq}}^q$, where T_{iq} is the minimal value of t for which $G_t^q \cap J_i$ is nonempty. In effect, this means that the algorithm is run on the network p successive times. From the connection's point of view, it runs p successive algorithms, using β_j as an approximation for β . In each of these p trials, it can be rejected (i.e., given no flow) by some of the routers. The final flow assigned to connection j is the average of the flows given in the p independent trials. We will prove that this procedure does not decrease the sum of the rates by more than an additional $(1-\epsilon)^2$ factor.

Now define $\text{OPT}(X)$ to be the value of the modified LP when flow can *only* be assigned to connections in the set X . It is not difficult to show that $\text{OPT}(G_t^q)$ is

bounded between $(\frac{\epsilon}{m})^{p(t-1)+q}$ and $(\frac{\epsilon}{m})^{p(t-1)+q} \cdot (\frac{m}{\epsilon})^{1/\epsilon} \cdot m$. Thus, we have that for each set G_t^q , the special form of the modified LP for connections in G_t^q satisfies the precondition with $c = (\frac{\epsilon}{m})^{p(t-1)+q}$ and $\tau = (\frac{m}{\epsilon})^{1/\epsilon} \cdot m$. Therefore, for each of these LPs, we can use $\gamma = \frac{m\tau}{\epsilon^2} = (\frac{m}{\epsilon})^{2+1/\epsilon}$.

We now turn to bound the approximation ratio. Consider a particular $q \in \{0, \dots, p-1\}$, and let T and Q be the unique integers such that β is in the interval defined by G_T^q and $\beta > (\frac{m}{\epsilon})^{pT+Q-1}$. For $q \neq Q$ and for the dual feasible setting $\{x_i = \frac{1}{\beta_i}\}$,

$$\begin{aligned} \text{OPT} \left(\bigcup_{t>T} G_t^q \right) &\leq \sum_{i|i \in I_t, t \in G_t^q, t>T} x_i \\ &\leq \frac{m}{(\frac{m}{\epsilon})^{pT+Q}} \leq \frac{\epsilon}{\beta} \leq \epsilon \cdot \text{OPT}. \end{aligned}$$

This implies that $\text{OPT}(G_T^q) \geq (1 - \epsilon) \text{OPT}$ for all $q \neq Q$. The quality of the solution we obtain is therefore bounded below by

$$\frac{1}{p} \sum_{q \neq Q} \text{OPT}(G_T^q) \geq \frac{p-1}{p} (1 - \epsilon) \text{OPT} \geq (1 - \epsilon)^2 \text{OPT}.$$

Putting everything together, we have a distributed algorithm that assumes global knowledge only of m and the approximation parameters r and ϵ . This algorithm finds a primal feasible $(r + (1 + \epsilon)^5)$ approximation of the optimal solution, and terminates in $O\left(\frac{(r+1) \ln^2(m/r\epsilon) \ln(\frac{(r+1)mn}{r\epsilon})}{r^2\epsilon^5}\right)$ distributed rounds.

4. Discussion. We studied the problem of having distributed decision-makers with local information generate feasible solutions to positive LPs. Our results explore the tradeoff between the amount of local communication that these agents may perform and the quality of the solution they obtain, measured by the approximation ratio. While we have provided an algorithm which obtains a $(1 + \epsilon)$ approximation ratio in a polylogarithmic number of distributed communication rounds, proving nontrivial lower bounds on the running time needed to obtain a $(1 + \epsilon)$ approximation remains an open question, as does the challenging problem of providing fast approximation algorithms for general LPs.

Acknowledgments. We would like to thank Christos Papadimitriou for stimulating discussions and useful insights in the early stages of this work. We also thank Dick Karp, Mike Luby, Dorit Hochbaum, and the anonymous SICOMP referees for their helpful comments on earlier versions of results presented in this paper.

REFERENCES

- [1] Y. AFEK, Y. MANSOUR, AND Z. OSTFELD, *Convergence complexity of optimistic rate based flow control algorithms*, in Proceedings of the 28th ACM Symposium on Theory of Computing, 1996, pp. 89–98.
- [2] J. ASPNES, Y. AZAR, A. FIAT, S. PLOTKIN, AND O. WAARTS, *On-line load balancing with applications to machine scheduling and virtual circuit routing*, in Proceedings of the 33rd Annual IEEE Symposium on Foundations of Computer Science, 1992, pp. 164–173.
- [3] B. AWERBUCH AND Y. AZAR, *Local optimization of global objectives: Competitive distributed deadlock resolution and resource allocation*, in Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science, 1994, pp. 240–249.

- [4] B. AWERBUCH, Y. AZAR, AND S. PLOTKIN, *Throughput competitive on-line routing*, in Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science, 1993, pp. 32–40.
- [5] B. AWERBUCH AND Y. SHAVITT, *Converging to approximated max-min flow fairness in logarithmic time*, in Proceedings of the 17th IEEE INFOCOM Conference, 1998, pp. 1350–1357.
- [6] Y. BARTAL, J. BYERS, AND D. RAZ, *Achieving global objectives using local information with applications to flow control*, in Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science, Miami Beach, FL, 1997, pp. 303–312.
- [7] D. BERTSEKAS AND R. GALLAGHER, *Data Networks*, Prentice-Hall, Englewood Cliffs, NJ, 1987.
- [8] D. BIENSTOCK, *Potential function methods for approximately solving linear programming problems: Theory and practice*, in CORE Lecture Series; available from CORE, University Catholique de Louvain, Belgium.
- [9] F. BONOMI AND K. FENDICK, *The rate-based flow control framework for the available bit rate ATM service*, IEEE Network Magazine, 9 (2) (1995), pp. 24–39.
- [10] J. BYERS, *Maximizing Throughput of Reliable Bulk Network Transmissions*, Ph.D. Thesis, University of California at Berkeley, Berkeley, CA, 1997.
- [11] A. CHARNY, *An Algorithm for Rate Allocation in a Packet-Switching Network with Feedback*, Technical Report MIT/LCS/TR-601, MIT Laboratory for Computer Science, Cambridge, MA, 1994.
- [12] N. GARG AND J. KOENEMANN, *Faster and simpler algorithms for multicommodity flow and other fractional packing problems*, in Proceedings of the 39th Annual Symposium on Foundations of Computer Science, 1998, pp. 300–309.
- [13] J. JAFFE, *Bottleneck flow control*, IEEE Trans. Comm., 29 (1981), pp. 954–962.
- [14] H. KARLOFF, *Linear Programming*, Birkhäuser, Boston, MA, 1996.
- [15] N. LINIAL, *Distributive graph algorithms—global solutions from local data*, in Proceedings of the 28th Annual IEEE Symposium on Foundations of Computer Science, 1987, pp. 331–335.
- [16] S. LOW AND D. LAPSLEY, *Optimization flow control I: Basic algorithm and convergence*, IEEE/ACM Trans. Networking, 7 (1999), pp. 861–874.
- [17] M. LUBY AND N. NISAN, *A parallel approximation algorithm for positive linear programming*, in Proceedings of the 25th ACM Symposium on Theory of Computing, 1993, pp. 448–457.
- [18] C. PAPADIMITRIOU AND M. YANNAKAKIS, *Linear programming without the matrix*, in Proceedings of the 25th ACM Symposium on Theory of Computing, 1993, pp. 121–129.
- [19] S. PLOTKIN, D. SHMOYS, AND E. TARDOS, *Fast approximation algorithms for fractional packing and covering problems*, in Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science, 1991, pp. 495–504. Full version appears as Technical Report ORIE-999, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY, 1995.