Global Optimization Using Local Information with Applications to Flow Control

Yair Bartal * John W. Byers* Danny Raz*

Abstract

Flow control in high speed networks requires distributed routers to make fast decisions based only on local information in allocating bandwidth to connections. While most previous work on this problem focuses on achieving local objective functions, in many cases it may be necessary to achieve global *objectives such as maximizing the total flow. This problem illustrates one of the basic aspects of distributed computing: achieving global objectives using local information.*

Papadimitriou and Yannakakis [PY93] initiated the study of such problems in a framework of solving positive linear programs *by distributed agents. We take their model further, by allowing the distributed agents to acquire more information over time. We therefore turn attention to the tradeoff between the running time and the quality of the solution to the linear program.*

We give a distributed algorithm that obtains a $(1+\epsilon)$ *approximation to the global optimum solution and runs in a polylogarithmic number of distributed rounds. While comparable in running time, our results exhibit a significant improvement on the logarithmic ratio previously obtained by [AA94]. Our algorithm, which draws from techniques developed by Luby and Nisan [LN93], is considerably simpler than previous approximation algorithms for positive linear programs, and thus may have practical value in both centralized and distributed settings.*

1 Introduction

Processors in a distributed environment make decisions based only on local data. Therefore, fast distributed algorithms must 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 [L87]. However, quite surprisingly, some of the most interesting global optimization problems can be very closely approximated based only on local information.

We study the problem of developing flow control policies with global objective functions. Flow control is the mechanism by which routers of a network distribute the available network bandwidth across connections. When the connections transmit data along fixed routes in the network, flow control is typically performed by allowing routers to regulate the rate at which connections inject data into the network. This connection-oriented, or rate-based, approach is expected to become widely used in packet-based networks and is a standard for routing available bit rate traffic in ATM 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* [BG87, AMO96, C94]. However, there are many other practical scenarios in which global objective functions are the appropriate choice. For example, in a commercial subnetwork 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 show how to express such a flow control policy objective as a positive linear program. Complicating the issue is the problem that routers must generate feasible solutions to this linear program (LP) quickly, and based only on available information.

Motivated by this and related applications, Papadimitriou and Yannakakis considered the problem of having distributed decision-makers assign values to a set of variables in a linear program, where the agents have limited information [PY93]. In one scenario 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 $O(\Delta)$, 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. Our goal is to determine the number of rounds necessary to achieve a $(1+\epsilon)$ approximation ratio to the optimum LP solution. In the setting of

*The International Computer Science Institute in Berkeley, California, and the University of California, Berkeley. Supported in part by NSF operating grants CCR-9304722 and NCR-9416101.

flow control, in one round, connections can communicate their flow values to all routers along their path, and routers can communicate a value describing their load to all connections which use them. 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 [PY93]. We note that similar models for describing the interaction between connections and routers in theoretical evaluations of flow control policies have been suggested in [AA94, AMO96, AS97].

One observation toward achieving our goal is that a centralized administrator with complete information could certainly solve the problem exactly using one of the well known polynomial-time algorithms for linear programming (see for example, [K96]). Recently, much faster algorithms that produce approximate solutions to positive linear programs to within a $(1+\epsilon)$ factor of optimal have been developed. The sequential algorithm of Plotkin, Shmoys and Tardos [PST94] repeatedly identifies a globally minimum weight path, and pushes more flow along that path. The algorithm of Luby and Nisan [LN93] has both a fast sequential and parallel implementation, and repeatedly performs a global median selection algorithm on the values of the dual variables, then increases values of dual variables above this threshold. Although these algorithms have efficient implementations, they both perform global operations which make them unsuitable for fast distributed implementation. Clearly, each of the global operations in these algorithms can be implemented in a polynomial number of distributed rounds, in which agents broadcast the values of relevant variables to all other agents. But we are interested in 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 [AA94], 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 [AAFPW92, AAP93]. The deterministic algorithm we present produces $(1+\epsilon)$ approximate solutions to positive linear programs, both in general and for the flow control problem, and builds on ideas used in these other algorithms [AA94, LN93, PST94]. Our algorithm is most closely related to the algorithm of Luby and Nisan, and affords the following advantages. It 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 algorithm of Luby-Nisan. 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 formulation of our distributed model and an explanation of the correspondence between flow control policies and positive linear programs in Section 2. In Section 3, we present our algorithm first as an easily understandable and implementable serial algorithm for approximately solving positive linear programming. In Section 4, we prove that the algorithm achieves a $(1+\epsilon)$ approximation ratio, and prove that it runs in a polylogarithmic number of rounds. Then in Section 5, we present the distributed implementation applicable to the flow control problem and explain the modification in the analysis of this case.

2 The Model

We consider the following model in the spirit of Papadimitriou and Yannakakis in which distributed agents generate approximate solutions to positive linear programs in the following standard form, which is well known to be as general as arbitrary positive linear programming.

PRIMAL	DUAL
$\max Y = \sum_{j=1}^{n} y_j$	$\min X = \sum_{i=1}^{m} x_i$
$\forall i,\ \sum_j a_{ij} y_j \le 1$	$\forall j,\ \sum_i a_{ij} x_i \ge 1$
$\forall j,\ y_j \ge 0$	$\forall i,\ x_i \ge 0$
$\forall i, j,\ a_{ij} \ge 0$	$\forall i, j,\ a_{ij} \ge 0$

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*. By this definition, primal agents are neighbors only with dual agents, and vice versa. In a *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 transmit messages to its set of dual neighbors $I_j = \{i | a_{ij} > 0\}$ and each dual agent i may transmit messages to its set of primal neighbors $J_i = \{j | a_{ij} > 0\}$. For simplicity, we consider algorithms which alternate between rounds in which primal agents transmit messages and rounds in which dual agents transmit messages. In the abstract model, we make no limitations on how many messages an agent may *receive* in one round, although practical considerations may introduce such limitations. After a fixed number of rounds, the agents must choose feasible values for their variables to (in the case of the primal) minimize the approximation ratio: 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.

In discussing flow control policies, we study a scenario in which each of n connections transmits data along a fixed path in the network, along an ordered subset of the m routers which comprise the network. Each router i has capacity C_i, which it may share among the connections which utilize it, while each connection is willing to pay 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 linear program.

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

Clearly, this positive linear program can be converted to standard form by the local operation $a_{ij} = \frac{\tilde{a}_{ij}}{B_j C_i}$. In a primal round, each connection may transmit a fixed-length message along its path, which each of its neighboring routers will receive. Since routers do not typically initiate transmissions to connection endpoints in existing networks, we implement a dual round by having each connection transmit a control message which loops through the network and back to the source. Then, each router en route, i.e. exactly those routers which *neighbor* a given connection, may modify the contents of the payload of the control message for that connection as it passes through in the dual round. This implementation introduces a restriction on the general model, however, in that each connection only receives a single message from its neighboring routers in each dual round. As mentioned earlier, this simple and natural model of communication between connections and routers corresponds to models previously suggested in other studies of flow control [AA94, AMO96, AS97]. In a synchronous model, each round takes time equal to the maximum round-trip time experienced by a connection in the network.

Another assumption that we make on the LP is that it is given to the algorithm in a normalized form in which the a_{ij} are either 0, or satisfy $\frac{1}{\gamma} \leq a_{ij} \leq 1$. One can convert a problem in standard form to the normalized form simply by dividing all constraints by $a_{max} = \max a_{ij}$, thereby setting $\gamma = \frac{a_{max}}{a_{min}}$, (where $a_{min} = \min\{a_{ij} | a_{ij} > 0\}$). Performing this transformation is straightforward in both the serial setting and in the distributed setting if bounds on the values of a_{max} and a_{min} are known to all agents in advance. In the context of the flow control problem, all agents would need to know are bounds on the min and max values of the edge capacities and benefit coefficients to compute γ. A disadvantage of this approach is that the value of γ, which affects the running time of our algorithm, depends on the values of the entries of the matrix. So we show in Section 5 that solving a problem in standard form can be reduced to solving problems in normalized form where the value of γ depends only on m and ϵ and does not significantly affect the approximation ratio or 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 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 The Algorithm

Our algorithm for approximately solving linear programming runs in *phases* as shown in Figure 1. At the end of each phase, the flow values y_j assigned to the connections are primal feasible and the "weights" on the routers x_i are dual feasible:

$$\textbf{Primal Feasibility: } \forall i, \lambda_i = \sum_j a_{ij} y_j \leq 1.$$

$$\textbf{Dual Feasibility: } \forall j, \alpha_j = \sum_i a_{ij} x_i \geq 1.$$

In a primal feasible solution, each router i has sufficient capacity to route the aggregate flow λ_i allocated to neighboring connections. In a dual feasible solution, each connection j transports flow through neighboring routers with aggregate weight $\alpha_j \geq 1$.

Throughout the algorithm, the values of the x_i are dependent on the values of neighboring y_j by the exponential weighting function: $x_i = e^{\lambda_i \phi}/\psi$, where ϕ is a constant which depends on the desired approximation ratio, and ψ is a scaling factor which depends upon the phase number. We begin each phase by increasing the value of ψ, scaling down the dual feasible values of the x_i from the previous phase and thereby introducing some slack by lightly loosening the x_i's dependence on the y_j. At this point the dual variables may no longer be feasible. This introduction of slack allows us to perform *iterations* in which we slowly increase, or "pump", the values of certain y_j in order to move to a primal solution with greater benefit. The y_j which are pumped in a given iteration are exactly those for which the value of the variable α_j is dual *infeasible*, i.e. smaller than 1. Therefore, when pumping terminates, ending a phase, dual feasibility is again achieved. Primal feasibility is maintained throughout the execution of the algorithm.

The parameter r in the algorithm determines the number of phases the algorithm executes, which trades off against the quality of the approximation. The approximation ratio which we obtain is a $(r + (1+\epsilon)^2)$ approximation to the optimum solution of the LP. We prove that after running for a constant number of phases, we obtain a logarithmic approximation ratio, and after running for a logarithmic number of phases, we obtain the stated ratio. Furthermore, we show that in each phase, we perform at most a polylogarithmic number of "pump" iterations. In the sequential implementation presented in Figure 1, the bottleneck operation is to recompute the α_js after each iteration, which takes $O(nm)$ time.

procedure Round-Update() {
$\forall i,\ \lambda_i = \sum_j a_{ij} y_j$
$\forall i,\ x_i = \frac{e^{\lambda_i \phi}}{\psi}$
$\forall j,\ \alpha_j = \sum_i a_{ij} x_i$
}

procedure Initialize() {
$\delta = (1+\epsilon)^2; \qquad \rho = \frac{1}{r}; \qquad Q = \rho \ln(6\gamma m e^{\epsilon});$
$\phi = (r+\delta)(Q + \rho \ln(Q + \rho \ln(2\rho Q))); \qquad \psi = m; \qquad \psi_F = 6m \frac{\phi}{r+\delta} e^{\frac{\delta\phi}{r+\delta}};$
$\forall i, \tilde{n}_i = \sum_{j \in J_i} a_{ij}; \qquad \forall j,\ n_j = \max_{i \in I_j} \tilde{n}_i; \qquad y_j = \frac{\epsilon}{n_j \phi};$
}

Algorithm LP() {
call Initialize()
repeat until $(\psi > \psi_F)$ { /* Phase */
call Round-Update()
repeat until $(\min_j \alpha_j \geq 1)$ { /* Iteration */
$\forall j$, **if** $(\alpha_j < 1)$ **then** $y_j = y_j \left(1 + \frac{\epsilon}{\phi}\right)$
call Round-Update()
}
$\psi = \psi(1+\epsilon)$
}
$\forall j$, **output** y_j
}

Figure 1: The Linear Programming Approximation Algorithm

Until now, our discussion of the algorithm has centered on the serial implementation. Additional considerations must be taken into account in the definition and description of the distributed algorithm, which we briefly discuss here, leaving the details for Section 5. Since global operations cannot be performed quickly, each distributed processor must be able to independently compute the values of all of the variables described in the serial implementation. In the case of network parameters which are fixed, such as the value of m, and for the parameters which affect the approximation ratio, r and ϵ, we assume that these values are known to all processors. The parameter γ may not be known by all processors, but we describe how to specify this parameter in the event that it cannot be locally computed in Section 5.3. The one remaining parameter is n, the number of connections. We do not assume that this value is globally known - instead, each primal agent j uses a value n_j which it can compute from its neighbors in two distributed rounds.

4 Analysis

In this section, we prove the following main results about our algorithm:

Theorem 1 *For $0 < \epsilon \leq 1$ 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 $O\left(\frac{\ln^2(\gamma m)\ln(\gamma mn/\epsilon)}{r\epsilon^2}\right)$ rounds.*

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

Corollary 2 *For any $\epsilon \leq 1$ there exists a $(1+\epsilon)$ approximation algorithm for positive linear programming that runs in $O\left(\frac{\ln^2(\gamma m)\ln(\gamma mn/\epsilon)}{\epsilon^3}\right)$ rounds. For any $1 \leq r \leq \ln(\gamma m)$ there exists a $(1+r)$ approximation algorithm for positive linear programming that runs in $O\left(\frac{\ln^2(\gamma m)\ln(\gamma mn)}{r}\right)$ rounds.*

The remainder of this section is divided into three parts: first we prove that at the end of each phase both the primal and the dual solutions are feasible. Then we carry on the analysis of the approximation ratio and finally we prove the claimed running time. In the proof we use the following two facts:

Fact 3 *$\phi \geq \epsilon$.*

Fact 4 *$e^{\phi-\epsilon} \geq \gamma\psi$.*

Fact 3 follows from the definitions, and the involved proof of Fact 4 is left for the Appendix.

4.1 Feasibility

Recall that the algorithm has the property that at the end of each phase (prior to increasing ψ), the x_is are a feasible solution to the dual program, implying the following fact:

Fact 5 (Dual Feasibility) *At the end of each phase, for all j, $\alpha_j \geq 1$.*

We next prove that the y_js are primal feasible throughout the execution of the algorithm, using Claim 6 to help perform the induction. Throughout the proof it will be convenient to think of the y_js as being increased from 0 to their actual initial value set in the algorithm. We will refer this operation as iteration 0.

Claim 6 *For all i and for every iteration, $\Delta\lambda_i \leq \frac{\epsilon}{\phi}$.*

Claim 7 (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 Claim 6 holds for iteration 0:

$$\begin{aligned}\lambda_i &= \sum_{j\in J_i} a_{ij} y_j \leq \sum_{j \in J_i} a_{ij} \frac{\epsilon}{n_j \phi} \\ &\leq \frac{\epsilon}{n_i \phi} \sum_{j\in J_i} a_{ij} = n_i \cdot \frac{\epsilon}{n_i\phi} \leq \frac{\epsilon}{\phi}.\end{aligned}$$

Since $\phi \geq \epsilon$ this also implies that Claim 7 holds at iteration 0.

Consider a subsequent iteration, and let Δv denote the change in variable v in 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 7. This completes the proof of Claim 6.

To complete the proof of Claim 7, we consider two cases for λ_i separately. We first consider edges i for which $\lambda_i < 1 - \frac{\epsilon}{\phi}$ prior to an iteration. From the proof of Claim 6 we have that after an iteration on such an edge, $\lambda_i' \leq \lambda_i + \frac{\epsilon}{\phi} < 1$, giving the desired result.

Next we consider edges i for which $\lambda_i \geq 1 - \frac{\epsilon}{\phi}$ prior to an iteration. Let i be such an edge and fix $j \in J_i$. We have that:

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

In the proof of Fact 4 in the appendix we show 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 the algorithm, we never increase the flow on connection j if $\alpha_j \geq 1$, so in fact, no path in J_i increases its flow in this iteration. Therefore, λ_i does not increase during this iteration and remains smaller than 1 by the induction hypothesis, completing the proof. ∎

4.2 Proof of the Approximation

We now turn to bound the approximation ratio obtained by the algorithm stated as the first half of Theorem 1:

Claim 8 *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 9 *For every iteration*

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

Proof: As $\epsilon \leq 1$, we have from Claim 6 that $\Delta\lambda_i\phi \leq \epsilon \leq 1$. It follows that

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

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

Let $S = \{j | \alpha_j < 1\}$ be the set of active connections in the iteration. 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. ∎

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. We let ΔX denote the change in a 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. The following lemma directly implies Claim 8.

Lemma 10

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

Fact 5 and Claim 7 respectively imply that X^* is dual feasible and Y_L is primal feasible. In conjunction with Lemma 10 this implies the approximation result stated in Claim 8, by linear programming duality.

Proof: Since the values X are scaled down by a $\frac{\psi}{\psi'} = 1+\epsilon$ 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 9 yields

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

Now let X_1 be the value of the dual solution at the end of the first phase and ψ_1 be the initial value of ψ. Similarly let X_L be the value of the dual solution at the end of the last phase and ψ_L be the appropriate value of ψ.

Using the bound above repeatedly to compare X_L with X_1 gives us:

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

We again use Lemma 9 applied to the first phase. Let X_0 stand for the value for X before the first initialization of the y_js, that is $X_0 = \sum_i \frac{e^0}{m} = 1$. Let Y_1 denote the value of the primal solution at the end of the first phase. We have that

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

If $X_L \leq \frac{r+\delta}{\phi(1+\epsilon)}(X_1 - X_0)$ then by the monotonicity of the primal solution the claim of the theorem follows:

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

We are left with the case that $X_L > \frac{r+\delta}{\phi(1+\epsilon)}(X_1 - X_0)$. 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 therefore obtain

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

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

$$\begin{aligned} e^{\frac{Y_L\phi(1+\epsilon)^2}{X^*}} &\geq \frac{\psi_F}{m\frac{\phi(2+\epsilon)}{r+\delta}(1+\epsilon)} \\ &\geq \frac{6m\frac{\phi}{r+\delta}e^{\frac{\delta\phi}{r+\delta}}}{m(2+\epsilon)(1+\epsilon)\frac{\phi}{r+\delta}} \\ &\geq e^{\frac{(1+\epsilon)^2\phi}{r+(1+\epsilon)^2}} \end{aligned}$$

by substituting $\delta = (1+\epsilon)^2$, and using $\epsilon \leq 1$. We finally get

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

■

4.3 Running Time

We now prove the second half of Theorem 1, bounding the number of rounds every connection executes before outputting its flow value:

Claim 11 *Our algorithm runs in* $O\left(\frac{\ln^2(\gamma m)\ln(\gamma mn/\epsilon)}{r\epsilon^2}\right)$ *rounds.*

Proof: We bound the number of phases by measuring the change in ψ:

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

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. In particular, we prove that if a connection j participates in a polylogarithmic number of iterations in a phase, α_j increases above 1.

For a given j, we say that y_j is *large* once $y_j \geq \frac{\gamma}{\epsilon^2}$. Initially, $y_j = \frac{\epsilon}{n_j\phi}$ and at every iteration it increases by a factor of $1+\frac{\epsilon}{\phi}$. Therefore the number of iterations connection j can participate in before y_j becomes large is at most

$$\log_{1+\frac{\epsilon}{\phi}}\left(\frac{\gamma}{\epsilon^2}\cdot\frac{n_j\phi}{\epsilon}\right).$$

Now once y_j is large, we have that

$$\Delta y_j \geq y_j\cdot\frac{\epsilon}{\phi} \geq \frac{\gamma}{\epsilon\phi}.$$

Let the set $I_j = \{i | a_{ij} > 0\}$. Therefore for all $i \in I_j$

$$\begin{aligned}\Delta\lambda_i &\geq \sum_k a_{ik}\Delta y_k \geq \frac{1}{\gamma}\Delta y_j \geq \frac{1}{\epsilon\phi} \qquad \text{and} \\ \alpha_j' &= \sum_i a_{ij}x_i' = \sum_i a_{ij}x_i \cdot e^{\Delta\lambda_i\phi} \geq \alpha_j \cdot e^{1/\epsilon}.\end{aligned}$$

Therefore, after $\ln(\gamma m(1+\epsilon))\epsilon$ additional iterations:

$$\alpha_j' \geq \alpha_j \cdot \gamma m(1+\epsilon)$$

where α_j and α_j' denote values at the start and end of these iterations respectively. At the beginning of the first phase, all $\alpha_j \geq \frac{1}{m\gamma}$, since all x_i are initialized to $\frac{\epsilon^0}{m} = \frac{1}{m}$. At the beginning of subsequent phases, all $\alpha_j \geq \frac{1}{1+\epsilon}$, by Fact 5. It follows from the discussion above that after these iterations $\alpha_j \geq 1$.

The bound on the number of iterations during a phase is therefore:

$$\begin{aligned}&\left\lceil \log_{1+\frac{\epsilon}{\phi}}\left(\frac{\gamma\phi n}{\epsilon^3}\right) + \ln(\gamma m(1+\epsilon))\epsilon \right\rceil \\ &= O\left(\frac{\phi}{\epsilon}\ln(\frac{\gamma m n}{\epsilon}) + \ln(\gamma m)\right) \\ &= O\left(\frac{\ln(\gamma m)\ln(\frac{\gamma m n}{\epsilon})}{\epsilon}\right)\end{aligned}$$

With the bound on the number of phases this completes the proof of Claim 11 and Theorem 1. ■

5 The Distributed Implementation and Special Form

In this section, we first present a distributed implementation of our algorithm. Then we describe modifications of the linear program and the analysis which enable us to implement our algorithm when γ is not known to all distributed agents or when we wish to eliminate the dependence of the running time on values of the matrix, i.e. when $\gamma = \frac{a_{max}}{a_{min}}$. Finally, the last subsection describes a distributed technique for dividing our original program into subprograms conforming to these modifications.

5.1 The Distributed Algorithm

The distributed implementation of our algorithm is shown in Figure 2. The top half of the implementation specifies the code executed at the routers and the bottom half specifies the code executed at the connections. Connections and routers communicate by message-passing, in the model of distributed rounds described in Section 2. Much of the complexity in converting the serial implementation into a distributed implementation involves local synchronization. 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.

The other technical obstacle in converting the serial algorithm to a distributed algorithm is the condition for ending a phase: $(\min_j \alpha_j \geq 1)$. Since we cannot hope to compute this minimum value in our distributed model, we instead let each connection check its end-of-phase condition locally and independently. Upon completion, each connection sends an end-of-phase message to its neighboring routers, and waits for those neighboring routers to terminate the phase before proceeding on to the next phase.

5.2 Special Form

When we wish to avoid setting $\gamma = \frac{a_{max}}{a_{min}}$, we can transform an LP in standard form to a *special form* (similar to one used by Luby and Nisan) in place of the transformation to *normalized form* described in Section 2. A precondition for transforming an LP Z in standard form to an LP Z' in special form is that we can approximate the value of the optimal solution for Z to within a factor of τ: $c \leq \mathrm{OPT} \leq c\tau$. 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 $\gamma' = \frac{m\tau}{\epsilon^2}$.

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, proofs of which are omitted:

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 $\sum_j y_j \geq \frac{\sum_j y'_j}{\nu} - \epsilon \cdot \mathrm{OPT}$.

2. If $\{y_j\}$ is primal feasible for Z then $\left\{y'_j = \frac{y_j\nu}{1+\epsilon}\right\}$ is primal feasible for Z', and $\sum_j y'_j = \sum_j y_j \frac{\nu}{1+\epsilon}$.

3. $\frac{\epsilon}{\nu\tau c} \leq a'_{ij} \leq 1$ for all i and j.

We generate an approximate solution to Z by performing the transformation to special form and computing a $(1+\epsilon)$ approximation $\{y'_j\}$ for Z' using our algorithm. We transform this solution to $\{y_j\}$ as described in (1) and get a primal feasible solution Y such that:

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

```
procedure Router-Update_i() {                                    /* Update λ_i and x_i. */
    λ_i = Σ_j a_ij y_j;      x_i = e^{λ_i}/ψ;      transmit x_i to all connections j ∈ J_i;
}

procedure Router-Initialize_i() {                                /* Initialization for Router i */
    J_i = {j | a_ij > 0};      ñ_i = Σ_{j∈J_i} a_ij;      ψ = m;
    transmit ñ_i to all connections j ∈ J_i;
    wait until (initial y_j values arrive from all j ∈ J_i)
}

Algorithm Router-DLP_i() {                                       /* Algorithm for Router i */
    call Router-Initialize_i()
    repeat until (all j ∈ J_i terminate) {
        call Router-Update_i()                                   /* Phase begins after this update */
        repeat until (all j ∈ J_i end current phase) {
            wait until (new y_j values arrive from all active j ∈ J_i)
            call Router-Update_i()
        }                                                        /* End of phase */
        transmit end-of-phase message to all connections j ∈ J_i;
        ψ = ψ(1 + ε);
    }
}
```

```
procedure Connection-Increase_j() {                              /* Update y_j */
    y_j = y_j (1 + ε/φ);      transmit y_j to all routers i ∈ I_j;
}

procedure Connection-Initialize_j() {                            /* Initialization for Connection j */
    I_j = {i | a_ij > 0};      wait until (ñ_i values arrive from all i ∈ I_j)
    δ = (1 + ε)^2;      ρ = 1/r;      Q = ρ ln(6γ m e^ε);
    φ = (r + δ)(Q + ρ ln(Q + ρ ln(2ρQ)));      ψ = m;      ψ_F = 6m (φ/(r+δ)) e^{δφ/(r+δ)};
    n_j = max_{i∈I_j} ñ_i;      y_j = ε/(n_j φ);      transmit y_j to all routers i ∈ I_j;
}

Algorithm Connection-DLP_j() {                                   /* Algorithm for Connection j */
    call Connection-Initialize_j()
    repeat until (ψ > ψ_F) {
        wait until (x_i values arrive from all i ∈ I_j)
        α_j = Σ_i a_ij x_i;                                      /* Phase begins after this update */
        repeat until (α_j ≥ 1) {
            call Connection-Increase_i()
            wait until (x_i values arrive from all i ∈ I_j)
            α_j = Σ_i a_ij x_i;
        }                                                        /* End of phase – become inactive */
        transmit end-of-phase message to all routers i ∈ I_j;
        wait until (end-of-phase messages arrive from all routers i ∈ I_j)
        ψ = ψ(1 + ε);
    }
    output y_j and terminate;
}
```

Figure 2: The Distributed Algorithm

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 as to guarantee $c \leq$ 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 $\widehat{\beta_j} = \min_{i \in I_j} \beta_i$. It is relatively easy to show that $\frac{1}{\beta} \leq$ 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 sequential implementations, giving $\gamma' = \frac{m^2}{\epsilon^2}$, and bounding the running time by $\mathrm{O}\left(\frac{\ln^2(m/\epsilon)\ln(mn/\epsilon)}{r\epsilon^2}\right)$ rounds.

5.3 Distributed Implementation

In the sequential case, knowledge of β is enough to perform the transformation to special form, but distributed agents 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 give flow to connections with good estimates of β, but we prove that this only reduces the total flow by a small factor.

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

$$G_t^q = \left\{ j \;\middle|\; \left(\frac{m}{\epsilon}\right)^{p(t-1)+q} \leq \widehat{\beta_j} < \left(\frac{m}{\epsilon}\right)^{pt+q} \right\}$$

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 $G^q_{T_{iq}}$, where T_{iq} is the minimal set index of connections in J_i for q. 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 $\widehat{\beta_j}$ as an approximation for β. In each of the algorithms, 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 flow by more than an additional $(1-\epsilon)^2$ factor.

Now define 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 OPT(G_t^q) is bounded between $\left(\frac{\epsilon}{m}\right)^{p(t-1)+q}$ and $\left(\frac{\epsilon}{m}\right)^{p(t-1)+q} \cdot \left(\frac{m}{\epsilon}\right)^{1/\epsilon} \cdot m$. Thus, we have that for each set G_t^q, the special form of the modified LP for connections in G_t^q has $\gamma = \left(\frac{m}{\epsilon}\right)^{2+1/\epsilon}$.

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

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

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

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

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 $(r + (1+\epsilon)^c)$-approximation of the optimal solution, and terminates in $\mathrm{O}\left(\frac{\ln^2(m/\epsilon)\ln(mn/\epsilon)}{r\epsilon^4}\right)$ distributed rounds.

6 Discussion

We studied the problem of generating feasible solutions to positive linear programs in a distributed environment, and in particular, the application of flow control. Our results explore the tradeoff between the amount of communication among the distributed agents and the quality of the solution we obtain, measured by the approximation ratio. We give an algorithm which obtains a $(1+\epsilon)$ approximation ratio in a polylogarithmic number of distributed communication rounds. Yet, many theoretical and practical questions remain open.

One obvious question is can the running time be improved? It is not difficult to show that $\frac{1}{\epsilon}$ distributed rounds are required to achieve a $1+\epsilon$ approximation, but other less trivial lower bounds are yet to be found. Another interesting theoretical question is the scope of these results, i.e. can they be applied to non-positive LPs?. Finding fast sequential approximation algorithms for general linear programs could be a start in this direction.

On the more practical side, we are interested in implementing our algorithm as a flow control policy. Our preliminary implementation indicates that further work on fine-tuning the algorithm to improve performance could enable the use of the algorithm for flow control on real networks.

Acknowledgments

We would like to thank Christos Papadimitriou for stimulating discussions and useful insights in the formulative stages of this work. We also thank Mike Luby for comments on an earlier draft of the paper.

References

[AA94] B. Awerbuch and Y. Azar. Local Optimization of Global Objectives: Competitive Distributed Deadlock Resolution and Resource Allocation. In *Proc. of the 35th Ann. IEEE Symp. on Foundations of Computer Science*, pp. 240-249, 1994.

[AAFPW92] 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 *Proc. of the 33rd Ann. IEEE Symp. on Foundations of Computer Science*, pages 164-173, October 1992.

[AAP93] B. Awerbuch, Y. Azar, and S. Plotkin. Throughput Competitive On-line Routing. In *Proc. of the 34th Ann. IEEE Symp. on Foundations of Computer Science*, November 1993.

[AMO96] Y. Afek, Y. Mansour and Z. Ostfeld. Convergence Complexity of Optimistic Rate Based Flow Control Algorithms. In *Proc. of 28th ACM Symposium on Theory of Computing*, pp. 89-98, 1996.

[AS97] B. Awerbuch and Y. Shavitt. Converging to Approximated Max-Min Flow Fairness in Logarithmic Time. Johns Hopkins Tech Report, 1997.

[BG87] D. Bertsekas and R. Gallagher. *Data Networks.* Prentice Hall, 1987.

[C94] 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, April 1994.

[J81] J. Jaffe. Bottleneck Flow Control. In *IEEE Transactions on Commmunication,* COM-29, 1(7), pp. 954-962, July 1981.

[K96] H. Karloff. *Linear Programming.* Birkhauser, 1996.

[L87] N. Linial. Distributive Graph Algorithms - Global Solutions from Local Data. In *Proc. of the 28th Ann. IEEE Symp. on Foundations of Computer Science*, pp. 331-335, 1987.

[LN93] M. Luby and N. Nisan. A Parallel Approximation Algorithm for Positive Linear Programming. In *Proc. of 25th ACM Symposium on Theory of Computing*, pp. 448-457, 1993.

[PY93] C. Papadimitriou and M. Yannakakis. Linear Programming without the Matrix. In *Proc. of 25th ACM Symposium on Theory of Computing*, pp. 121-129, 1993.

[PST94] S. Plotkin, D. Shmoys and E. Tardos. Fast Approximation Algorithms for Fractional Packing and Covering Problems. *Technical Report ORIE-999 of the School of Operations Research and Industrial Engineering, Cornell University*, 1995.

Proof of Fact 4

We have to prove that $e^{\phi-\epsilon} \geq \gamma\psi$. Substituting ψ_F for ψ and multiplying by e^ϵ, we must show

$$\begin{aligned} e^{\phi} &\geq (6\gamma m e^{\epsilon})\frac{\phi}{r+\delta}e^{\frac{\delta\phi}{r+\delta}} \qquad \text{or} \\ e^{\frac{\phi}{r+\delta}} &\geq (6\gamma m e^{\epsilon})^{\frac{1}{r}}(\frac{\phi}{r+\delta})^{\frac{1}{r}}. \end{aligned}$$

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 $P = \rho\ln(2\rho Q)$. Therefore it is enough to prove the following.

$$\begin{aligned} &\tfrac{\phi}{r+\delta} - \rho\ln(\tfrac{\phi}{r+\delta}) - Q \\ &= Q + \rho\ln(Q+P) - \rho\ln(Q+\rho\ln(Q+P)) - Q \\ &= \rho\ln(\tfrac{Q+P}{Q+\rho\ln(Q+P)}) \geq 0. \end{aligned}$$

To show that $\rho\ln(\frac{Q+P}{Q+\rho\ln(Q+P)})$ is nonnegative we need $P \geq \rho\ln(Q+P)$ or

$$\rho\ln(Q+P) - \rho\ln(2\rho Q) = \rho\ln(\tfrac{Q+P}{2\rho Q}) \leq 0.$$

It is left to show then that $\frac{Q+P}{2\rho Q} < 1$, but

$$\begin{aligned} &\tfrac{Q+P}{2\rho Q} = \tfrac{Q+\rho\ln(2Q)+\rho\ln(\rho)}{2\rho Q} \\ &\leq \tfrac{1}{2\rho} + \tfrac{\ln(2Q)}{2Q} + \tfrac{\ln(\rho)}{2\rho} \leq \tfrac{1}{2\rho} + \tfrac{1}{e} + \tfrac{\ln(\rho)}{2\rho}. \end{aligned}$$

This follows from the fact that $Q \geq \rho$ and from $\ln z/z \leq 1/e$. Since $1 + \ln\rho \leq \rho$ we get

$$\frac{Q+P}{2\rho Q} \leq \frac{1}{2} + \frac{1}{e} < 1$$

which completes the proof. ∎