Pricing for Fairness: Distributed Resource Allocation for Multiple Objectives

Sung-woo Cho *
University of Southern California

Ashish Goel †
Stanford University

November 16, 2005

Abstract

In this paper, we present a simple distributed algorithm for resource allocation which simultaneously approximates the optimum value for a large class of objective functions. In particular, we consider the class of canonical utility functions U that are symmetric, non-decreasing, concave, and satisfy $U(0) = 0$. Our distributed algorithm is based on primal-dual updates. We prove that this algorithm is an $O(\log \rho)$-approximation for all canonical utility functions simultaneously, i.e. without any knowledge of U. The algorithm needs at most $O(\log^2 \rho)$ iterations. Here n is the number of flows, m is the number of edges, R is the ratio between the maximum capacity and the minimum capacity of the edges in the network, and ρ is $\max\{n, m, R\}$.

We extend this result to multi-path routing, and also to a natural pricing mechanism that results in a simple and practical protocol for bandwidth allocation in a network. When the protocol reaches equilibrium, the allocated bandwidths are the same as when the distributed algorithm converges; hence the protocol is also an $O(\log \rho)$ approximation for all canonical utility functions.

*Department of Computer Science, University of Southern California. Email: sungwooc@cs.usc.edu . Research supported by NSF Award CCR-0133968.

†Departments of Management Science and Engineering and (by courtesy) Computer Science, Stanford University. Email: ashishg@stanford.edu. Research supported by NSF CAREER Award 0133968 and an Alfred P. Sloan Faculty Fellowship.

1 Introduction

In this paper, we study the classic problem of distributed allocation of bandwidths to flows (i.e. source-destination pairs) in a network (see [17, 2, 4, 20, 1, 5, 3, 11, 7] for some of the recent research on the problem). Apart from being important in its own right, this problem also models a wide variety of other resource allocation problems. Specifically, we will be interested in obtaining distributed algorithms that are approximately optimum for a large class of objective functions, *simultaneously*. Our main result is a distributed algorithm for the case when each flow must use a single pre-specified route; our algorithm requires only polylogarithmic number of iterations and guarantees that the vector of allocated bandwidths is a logarithmic approximation, simultaneously, for all non-decreasing, symmetric, concave objective functions. The algorithm naturally extends to the multi-path routing case where each flow can use multiple routes which are not specified in advance, with weaker guarantees on running time.

Our algorithm is surprisingly simple and natural; in fact, it is simple enough to be converted into a TCP-like protocol. The class of objective functions we consider encompasses all "reasonable" social objective functions (i.e. fairness functions) that we know of. Our results are useful whenever the objective function is poorly understood (eg. customer satisfaction, fairness) or when there are multiple objectives (eg. when a social planner wants to simultaneously satisfy both socialists and capitalists). Before describing our results in greater detail, we will first describe and then motivate the problem we study.

1.1 Problem description

The bandwidth allocation problem with fixed routes consists of a (directed or undirected) graph $G = (V, E)$. Edge e has capacity c_e. There are n source-destination (s_i, t_i) pairs, and for each pair we are given a unique route p_i from s_i to t_i. Each such pair is called a *flow*. The goal is to allocate bandwidths to flows such that the capacity constraints are not violated. We assume that R is the ratio of the maximum capacity of any edge to the minimum capacity of any edge. Let m be the number of edges, and define $\rho = \max\{n, m, R\}$. Let x_i be the bandwidth allocated to flow i. Then the constraints are $x \geq 0$ and $\forall e, \sum_{i:e \in p_i} x_i \leq c_e$. We will refer to the set of all feasible allocations as S.

In this paper, we are interested in optimizing a large class of objective functions simultaneously. Let U be an n-variate real-valued function. We will say that U is a *canonical utility function* if U is symmetric in its arguments, concave, non-decreasing, and $U(0) = 0$. This captures a large class of social objective functions; more details about the importance of this class are presented in section 1.2. Let U^* denote the maximum value of $U(y)$ subject to $y \in S$. We will define the simultaneous-approximation-ratio $r(x)$ of a feasible solution x as follows:

$$r(x) = \max_{U:U \text{is canonical}} \frac{U^*}{U(x)}.$$

Thus $r(x)$ can be thought of the worst possible approximation ratio that x provides for "reasonable" social objective functions. There always exists a feasible solution x with $r(x) = O(\log \rho)$ [19, 12], and this is essentially the best achievable bound [19]. Our goal is to obtain a simple and efficient distributed algorithm which guarantees $r(x) = O(\log \rho)$. We define a distributed algorithm for this problem to be one where there is an agent corresponding to each flow and an agent corresponding to each edge. Along the lines of Bartal, Byers, and Raz [4] and Kelly, Maulloo, and Tan [17], the flow agents control the allocated bandwidth x_i and the edge agents maintain dual costs (or shadow prices) l_e. In one iteration, first each flow agent is told the sum of the dual costs of all the edges in its route. Then, each flow agent decides how to update x_i based only on this aggregate dual cost. And finally, each edge agent updates its dual cost based only on the change in the total allocated bandwidth on the edge in this iteration. This has a natural interpretation as a simple protocol where the total edge costs are conveyed to the flow agent by piggybacking as a header-field in the data packets transmitted between s_i and t_i.

While the notion of a network and flows is convenient to describe this problem and related work, this problem also models fairly general resource allocation problems. Our algorithm (as well as many earlier algorithms for the bandwidth allocation problem) does not crucially use the graph structure. Hence we can think of edges as resources, flows as tasks, and route p_i as the set of resources required for task i. All our algorithms and analyses continue to work in this general setting.

We also study two generalizations of the above problem. The first generalization is to relax the requirement that only a single route can be used for a given flow, and that this route must be specified upfront. Thus, along with the total bandwidth x_i, each flow agent must also determine a set of routes and a distribution of this bandwidth among different routes. Here, a distributed algorithm is one where each flow agent is allowed to use (as a primitive) the computation of a shortest path (under the dual costs) from s_i to t_i in each iteration. This is in line with the single objective versions of the multiple-route bandwidth allocation problem [23, 10]. Our algorithm naturally generalizes to this model, at the expense of an increased number of iterations. As before, there is a natural interpretation of this problem as a resource allocation problem with an appropriate shortest cost oracle instead of the shortest path computation.

The second generalization is to allow utility functions U which satisfy concavity, symmetry, and the non-decreasing property, but do not necessarily satisfy $U(0) = 0$. This includes functions such as $\sum_i \log x_i$. For this (even larger) class, we show that our distributed algorithm achieves the optimum for all functions U in this class, given $O(\log \rho)$ times more capacity (i.e. the approximation ratio for canonical utility functions translates into the amount of resource augmentation needed when $U(0) \neq 0$).

1.2 Motivation and related work

The Transport Control Protocol (TCP) is by far the most widely used solution to the bandwidth allocation problem in practice. In more abstract settings, Kelly, Maulloo, and Tan [17] proposed a distributed algorithm for this problem for the case where $U(x) = \sum_i U_i(x_i)$, and each U_i is a concave function. Their algorithm uses a primal-dual framework where the dual prices (which they call shadow prices) are maintained by edge agents and primal flows are maintained by the flow agents. All communication is local, i.e., takes place between a flow agent and an edge agent on the path of the flow. The resulting solution is proportional with respect to the dual prices, and hence, their framework is widely referred to as the "proportional-fairness" framework[1]. Subsequent work by Low, Peterson, and Wang [20] and others has shown that several variants of TCP essentially perform the above computation for different choices of utility functions. Since the behavior of different variants of TCP is quite different (different variants work better in different settings), the above work raises the following natural question: *Is it possible to obtain solutions which are simultaneously good for a wide range of utility functions?*.

Bartal, Byers, and Raz [4] presented a distributed algorithm for the above problem when $U(x) = \sum_i x_i$. Unlike the work of Kelly *et al.*, this work presents a running time analysis. They prove that a simple local computation along with local communication can lead to almost optimum solutions in polylogarithmic number of iterations. Their work builds on the positive linear programming framework of Luby and Nisan [21]; for their problem, the positive linear programming framework is essentially identical to the fractional packing framework developed by Plotkin, Shmoys, and Tardos [23], and later simplified by Garg and Konemann [10]. Each edge-agent maintains dual costs, and each flow-agent uses these dual costs to update its own flow. Recently, Garg and Young [11] have shown how a simple MIMD (multiplicative increase multiplicative decrease) protocol can approximate the above objective. These results lead to the following natural question: *Can we obtain distributed algorithms with similar rigorous running time analyses for more involved utility functions?*.

[1]The symmetry requirement in our work implies that our class does not contain all utility functions to which the proportional-fairness framework applies. However, since our class does not require the utility function to be a sum of uni-variate functions, it contains important functions such as min which can not be addressed using the proportional-fairness framework. Hence the two classes of utility functions are incomparable.

Building on a series of papers about multi-objective fairness [18, 14], Kleinberg and Kumar [19] studied the problem of bandwidth allocation in a centralized setting with multiple fairness objectives. Goel and Meyerson [12] and Bhargava, Goel, and Meyerson [6] later expanded this work to a large class of linear programs and related it to simultaneous optimization [13]. In particular, the above sequence of papers resulted in a centralized bandwidth allocation algorithm for computing a single allocation which is simultaneously an $O(\log \rho)$ approximation for all canonical utility functions. Goel and Meyerson [12] build on the notion of majorization due to Hardy, Littlewood, and Polya [15, 16, 22]. This leaves open the following question: *Can there be efficient distributed algorithms which achieve the same results?* Cho and Goel [7] made some partial progress towards this problem by giving a centralized algorithm which maintains only one set of dual costs. However their single-dual algorithm was still inherently centralized and their techniques do not offer much insight towards obtaining the results in this paper.

All three questions raised above are very similar, even though they arise in different contexts. They point towards a need for distributed bandwidth allocation algorithms which are good across multiple objectives and have provably good running times. We address precisely this problem.

The class of utility functions we consider is not arbitrarily chosen. This is a large class, and contains the important subclass $\sum_i f(x_i)$ where f is a uni-variate concave function (f must also be non-decreasing and $f(0)$ must be 0). Concavity is a natural restriction, since it corresponds to the "law of diminishing returns" from economics. Symmetry corresponds to saying that all users are equally important[2]. The requirement for U being non-decreasing is natural for a resource allocation problem. The requirement that $U(0) = 0$ is also natural in many (but not all) settings. The class of canonical utility functions includes important special functions such as min, $\sum_i x_i$, $\sum_i \log(1 + x_i)$, and $P_j(x) =$ the sum of the j smallest x_i's. The class of canonical utility functions also contains a series of functions which together capture max-min fairness. Most interestingly, there is a concrete connection between this class and our intuitive notion of fairness. Suppose there exists some function which measures the fairness of an allocation. It seems natural that the allocation (x_1, x_2) should be deemed as fair as (x_2, x_1) and less fair than $(\frac{x_1+x_2}{2}, \frac{x_1+x_2}{2})$. This assumption implies that for any natural definition of fairness, maximizing fairness should be equivalent to maximizing some symmetric concave function; certainly, all the definitions of fairness that we found in literature are of this form.

Some important utility functions satisfy symmetry, concavity, and the non-decreasing property but are not 0 at 0. One example is the function $\sum_i \log x_i$, which is particularly important for two reasons: it is the objective maximized by TCP Vegas [20] which is an important version of TCP, and this also corresponds to the objective function required to solve Leontief economies [8].

In order to implement these ideas in the setting of a centralized social planner, it would also be important to derive a natural pricing mechanism which implements simultaneous optimization.

1.3 Overview of results and techniques

We use one of the simplest possible distributed algorithms: we start with the variables x_i and dual costs l_e both being very small. During each iteration, the i-th flow agent increments x_i by a small amount if the total dual cost along route p_i is less than 1. And each edge agent updates its dual cost multiplicatively depending on the amount of new bandwidth utilized on this edge. Since edge costs increase monotonically, the algorithm would terminate when every flow agent sees a total dual cost of 1 or more on its route. We think of the bandwidth allocations x_i as the primal variables, and accordingly, will refer to flow agents as primal agents where that notation is more appropriate.

Our main result is that with appropriate parameters, the above algorithm terminates in $O(\log^2 \rho)$ iterations with a feasible allocation x for the bandwidth allocation problem with fixed routes, and $r(x) = O(\log \rho)$ i.e. for all canonical utility functions U and all feasible allocations y, $U(x) = U(y)/O(\log \rho)$.

[2]Notice that the constraints are not required to be symmetric, and hence, the optimum solution need not be symmetric even though the objective function is symmetric.

We extend this result to the multi-path routing case. The simultaneous-approximation-ratio $r(x)$ remains the same but the number of iterations becomes polynomial in ρ. This is in line with what we observe for the single-objective case; the fixed-route case corresponds to positive linear programming [21] and requires only polylogarithmic iterations [2, 4] whereas the multi-path route case requires polynomially many iterations [23, 10, 9].

For both results, if we drop the requirement that $U(0) = 0$, the same algorithm leads to a resource-augmented approximation. The final solution x violates the capacity constraints by a factor of $O(\log \rho)$ but for all feasible (with respect to the original capacities) solutions y and all symmetric, concave, non-decreasing functions U, $U(x) \geq U(y)$.

We also interpret our algorithm as a natural pricing mechanism (hence the title of our paper). We further build on this connection to give a TCP-like protocol that has the same equilibrium point as our algorithm; detailed control theoretic and convergence analysis of this protocol is a promising direction.

The standard analysis technique for the kind of primal-dual update algorithm we use is to look at the final dual costs and prove that they provide a certificate of optimality (or approximation ratio) for the primal variables. Such an approach is not going to work for us. We provide a quick explanation for the reader who is conversant with the standard single-objective primal-dual analysis for this problem, since that provides a very good insight into the hardness of this problem. Consider a line network with $2n$ edges. There are n "long" flows $L_1, L_2, \ldots L_n$ each of which uses each of the first n edges, and there are n more "short" flows $S_1, S_2, ..S_n$, each of which uses a distinct edge from the remaining n edges. The optimum value for the canonical utility function min for this problem is clearly $1/n$. Very early in the execution of the algorithm, the dual cost of each of the first n edges will become $1/n$, and the primal variables for the long flows will stop updating. Much later, the cost of each of the last n edges will become 1 as well, and the primal variables for the short flows will stop updating. Thus, the final dual prices are $1/n$ for the first n edges, and 1 for the last n edges, for a total dual cost of $n + 1$. The dual cost for each flow is 1, and hence the dual cost of an (infeasible) allocation y which satisfies $\min_i y_i = 1$ is $2n$. If we were to try to use the final dual prices to obtain a bound on the maximum value of $\min_i x_i$, we would get a bound of $(n + 1)/(2n) > 1/2$. This bound is not very useful, since the optimum value is $1/n$. Thus, we have to analyze the algorithm at many different intermediate points to obtain an approximation guarantee for all canonical utility functions. This is a novel feature of our analysis.

In section 2 we provide the background needed for our analysis; in particular, we describe the connection between majorization and simultaneous optimization. In section 3 we present our algorithm for the fixed route case, and prove a weaker result with polynomial number of iterations and $r(x) = O(\log \rho)$. This is done for ease of exposition – this weaker algorithm conveys all the intuition behind our algorithm and also provides a convenient starting point for two different extensions. In section 4.1 we describe the (slight) changes we need to make to the basic algorithm to reduce the number of iterations to $O(\log^2 \rho)$. In section 4.2 we describe the (slight) changes we need to make to the basic algorithm to handle the multiple-routes case. In section 4.3 we describe how our algorithm corresponds to a natural pricing mechanism and how our algorithm can be thought of as a TCP-like protocol.

2 Background – Approximate Majorization

In this section we describe the framework for simultaneous optimization of linear programs developed by Goel and Meyerson [12]. This framework works in a centralized setting. Define the k-th prefix, $P_k(x)$ to be the sum of the k smallest components of x (not $x_1 + x_2 + \ldots + x_k$ but $x_{\sigma(1)} + x_{\sigma(1)} + \ldots x_{\sigma(k)}$ where σ is the permutation that sorts x in increasing order). Let P_k^* denote the maximum possible value of $P_k(x)$ subject to the given constraints

Definition 2.1 Approximate Majorization: *A feasible solution x is said to be α-majorized if $P_k(x) \geq P_k(y)/\alpha$ for all $1 \leq k \leq n$ and all feasible solutions y.*

Informally, the k poorest users in an α-majorized solution get at least $1/\alpha$ times as much resource as the k poorest individuals in any other feasible allocation. Intuitively, this seems to have some ties to fairness. The following theorem [12] makes this intuition concrete:

Theorem 2.1 *A feasible solution x is α-majorized if and only if $U(x) \geq U(y)/\alpha$ for all feasible solutions y and all canonical utility functions U.*

The following theorem is also immediate from [12]:

Theorem 2.2 *A feasible solution x is α-majorized if and only if $U(x) \geq U(y/\alpha)$ for all feasible solutions y and all symmetric, non-decreasing, concave functions U.*

In fact, the above theorems hold not just for resource allocation, but for an arbitrary set of constraints (integer, convex etc.) as long as the constraints imply $x \geq 0$. Thus the notion of α-majorization captures simultaneous optimization; theorem 2.1 corresponds to simultaneous approximation of canonical utility functions, whereas theorem 2.2 guarantees a weaker resource-augmented approximation but for a larger class of functions (i.e. $U(0)$ need not be 0).

We have reduced the problem of approximating uncountably many canonical utility functions to the problem of approximating n simple prefix-sum functions. However, for this framework to be useful, we need to demonstrate that α-majorized solutions exist with small values of α; the following theorem does exactly that [12].

Theorem 2.3 *If the set of feasible solutions is convex and non-negative, then there exists an $O(\log \frac{P_n^*}{nP_1^*})$-majorized solution.*

For many problems of interest, the above theorem translates into $\alpha = O(\log n)$. For example, for the bandwidth allocation problem with unit capacities, $P_n^* \leq n$ whereas $P_1^* \geq 1/n$, implying the existence of an $O(\log n)$-majorized solution. For non-uniform capacities, the guarantee becomes $O(\log n + \log R)$ where R is the ratio of the maximum to the minimum capacity. However, even if there exists an α-majorized solution, it is not clear a priori that finding such a solution is computationally tractable. The next theorem [12] resolves this issue assuming linear constraints. Here, α^* is the smallest possible value of α for which an α-majorized solution exists.

Theorem 2.4 *Given the constraints $\{Ax \leq c, x \geq 0\}$, both α^* and an α^*-majorized solution can be found in time polynomial in n and the number of constraints.*

Similar techniques were developed by Tamir [24] to compute majorized elements. Let us focus on a proof of theorem 2.4 that highlights the difficulties involved in making the above framework carry over in a distributed setting. We will restrict ourself to the bandwidth allocation problem for simplicity, where A and b are both required to be non-negative. In order to compute α^*, we first need to compute P_k^*. Computing P_k^* is equivalent to solving the following linear program:

$$\begin{gathered}\text{Minimize } \lambda_k \text{ subject to:} \\ Ax \leq \lambda_k c \\ \textstyle\sum_{i \in S} x_i \geq 1 \text{ for all } S \subseteq \{1, \cdots, n\} \text{ with } |S| = k\end{gathered}$$

P_k^* would be $1/\lambda_k^*$ where λ_k^* is the solution to the above linear program. The linear program described above is a fractional packing problem, and can be solved efficiently [23] if we are given a *dual optimization subroutine* to solve the following program:

$$\alpha_k(l) = \text{ Minimize } w \cdot x \quad \text{ subject to: } \quad P_k(x) = 1; x \geq 0 \tag{1}$$

where $l_e \geq 0$ is the "dual cost" of edge e and $w_i \geq 0$ represents the dual cost for flow i. The dual cost w_i for flow i is computed by simply adding the dual costs l_e of each edge e used by the flow. It is important to note

that the dual costs are artifacts of the solution methodology and do not correspond to any real entity in the original problem. The dual optimization subroutine can be implemented with time complexity $O(n \log n)$ in the centralized setting. In order to find P_k^* for all $k, 1 \leq k \leq n$, we need to solve n fractional packing problems. The quantity α^* and the corresponding α^*-majorized solution can then be computed using similar techniques. To make this algorithm distributed, it appears that we need to address two issues:

1. Efficient distributed implementation of the dual subroutine, and
2. Efficient solution to the n different problems corresponding to different prefixes P_j.

This was the approach taken in an earlier paper by Cho and Goel [7], but unfortunately, this approach did not lead to a distributed algorithm. In this paper, we present an algorithm that does not implement the dual subroutine at all; in fact the algorithm that we present in the next section is much simpler than using the above approach for even one prefix.

3 The Basic Algorithm and Analysis

In this section we present a distributed algorithm for simultaneous optimization in the fixed-route case, and the proof of its feasibility and approximation guarantee. The algorithm in this section requires polynomially many iterations; we will reduce the number of iterations to polylogarithmic in section 4.1 and point out the extension to the multiple routes case in section 4.2.

Recall that c_e is the capacity and l_e is the dual cost of an edge e. Also recall that x_i is the amount of bandwidth and the w_i is the dual cost for a flow i. The dual cost w_i is given by $\sum_{e \in p_i} l_e$ for flow i where p_i is the route of flow i. We define Λ_e as the load of an edge e i.e. $\Lambda_e = (\sum_{i:e \in p_i} x_i)/c_e$. We will use parameter t for denoting these values at time t (i.e. during the t-th iteration). For example, $l_e(t)$ implies the dual cost of an edge e at time t. For brevity, we define $\Delta x_i(t) = x_i(t+1) - x_i(t)$ and $\Delta \Lambda_e(t) = \Lambda_e(t+1) - \Lambda_e(t)$. Let m be the number of edges, and define ρ as $\max\{n, m, R\}$. We now state the algorithm:

```
INITIALIZE()
1  δ ← 12 ln ρ + 2
2  for each edge e
3  do l_e(0) ← δ/(2ρ^3)
4  for each flow i
5  do x_i(0) ← min_e c_e / 2n
6  t ← 0
```

```
DISTRIBUTED-MAJORIZATION()
1  while there is any flow i s.t. w_i(t) < 1
2  do t ← t + 1
3     /* bandwidth allocation */
4     for each flow i
5     do if w_i(t − 1) < 1
6           then x_i(t) ← x_i(t − 1) + min_e c_e / nδ
7           else x_i(t) ← x_i(t − 1)
8     /* edge-length update */
9     for each edge e
10    do l_e(t) ← l_e(t − 1)(1 + δΔΛ_e(t − 1))
```

Informally, the algorithm does the following. Initially, all the dual costs are very small, and all flow agents can increment their primal variables. At some time t, one or more flow agents find that their routes are now too expensive, and they become inactive (i.e. stop incrementing their primal variables). At this point, the prefix P_1 gets frozen but the dual costs and the other prefix functions may keep increasing. Hence we need to use intermediate dual costs at different times as certificates of approximate optimality for different P_j's. We combine this idea with several combinatorial properties of the solution to the dual subroutine (1) to analyze our algorithm.

Feasibility: The proof of feasibility is straightforward. At time 0, $\Lambda_e(0) = \frac{\sum_{i:e\in p_i} x_i(0)}{c_e} \leq \frac{n\frac{\min_e c_e}{2n}}{c_e} \leq \frac{1}{2}$. Let t be the smallest time such that $\Lambda_e(t) \geq 1$. Then,

$$1 > l_e(t-1) = l_e(t-2)(1+\delta\Delta\Lambda_e(t-2)) = l_e(0)\prod_{\tau=0}^{t-2}(1+\delta\Delta\Lambda_e(\tau)).$$

Since $\delta\Delta\Lambda_e(\tau) \leq \delta\frac{\frac{\min_e c_e}{n\delta}\cdot n}{c_e} \leq 1$, we can use the relation $e^{\frac{a}{2}} < 1+a$ for $0 \leq a \leq 1$ to obtain

$$1 > l_e(0)\prod_{\tau=0}^{t-2} e^{\frac{\delta\Delta\Lambda_e(\tau)}{2}} = l_e(0)\left(e^{\frac{\sum_{\tau=0}^{t-2}\delta\Delta\Lambda_e(\tau)}{2}}\right) > l_e(0)\left(e^{\frac{\delta\{\Lambda_e(t-1)-\Lambda_e(0)\}}{2}}\right).$$

Taking the logarithm on both sides,

$$\Lambda_e(t) \leq \Lambda_e(0) + \Delta\Lambda_e(t-1) + \frac{2\ln\frac{1}{l_e(0)}}{\delta} \leq \frac{1}{2} + \frac{1+6\ln\rho-2\ln\frac{\delta}{2}}{\delta} \leq 1$$

The above equation implies that the load of an edge e is less than 1 at time t and any time before t. After time t, the load of the edge e does not increase, because every flow through e has reached the maximum weight possible[3].

Approximation guarantee: Using basic LP duality, the problem of maximizing P_k has an optimal primal solution x and a corresponding (i.e. equal in value) dual solution $(\sum_e l_e \cdot c_e)/\alpha_k(l)$ where

$$\alpha_k(l) = \min_y \frac{\sum_i \left(\left(\sum_{e\in p_i} l_e\right)\cdot y_i\right)}{P_k(y)} = \min_y \frac{\sum_i (w_i \cdot y_i)}{P_k(y)}.$$

Suppose that we are given a feasible primal solution x, dual costs l_e for the edges and dual costs w_i for the flows (derived from l). We will refer to the w_i's as weights. Since flow agents with $w_i > 1$ do not increment their flows, it is useful to define the following two functions which essentially truncate w_i at 1:

$$v_i(w) = \begin{cases} 1 & \text{if } w_i \geq 1 \\ w_i & \text{otherwise} \end{cases}, \qquad \text{and } \beta_i(w) = \min_x \frac{\sum_j v_j(w)\cdot x_j}{P_i(x)}.$$

The v_i are analogous to w_i and the β_i are analogous to α_i. Without loss of generality, it is assumed that w is sorted in decreasing order. We now prove two important combinatorial properties of the solution to the dual subroutine (1):

Lemma 3.1 *$\beta_i(w)$ has the following properties.*

1. *$\beta_i(w) = \frac{\sum_{j\geq\gamma} v_j(w)}{i-\gamma+1}$ for some γ such that $1 \leq \gamma \leq i$.*
2. *Define a symbol '$\leq$' on vectors a and b as follows.*

$$a \leq b \Leftrightarrow [a_{j'} < b_{j'}\ \exists\ j'] \text{ and } [a_j \leq b_j\ \forall\ j] \text{ for all } a_j, b_j \geq 0$$

Then, $a \leq b$ implies that $\beta_i(a) \leq \beta_i(b)$.

Proof:

1. Goel and Meyerson [12] showed that the solution y of $\alpha_k(l)$ satisfies the following conditions:

[3]Note that we did not directly use the fact that updates to x_i are small in each iteration; we just used the fact that $\delta\Delta\Lambda_e(t) < 1$. We are going to exploit this in section 4.1.

(1) $w_{j'} \leq w_j \Rightarrow y_{j'} \geq y_j$

(2) The solution x is two valued. In particular, there exists a value λ such that for all j, $y_j = 0$ or λ.

Since $\beta_i(w)$ is a special form of $\alpha_k(l)$, $\beta_i(w)$ also has the above properties. Thus, there should be an index γ, a value λ and an optimal solution y for $\beta_i(w)$ such that $y_j = 0$ if $j < \gamma$ and $y_j = \lambda$ otherwise. Hence,

$$\beta_i(w) = \frac{\sum_j v_j(w) \cdot y_j}{P_i(y)} = \frac{\sum_{j \geq \gamma} v_j(w) \cdot y_j}{P_i(y)} = \frac{\sum_{j \geq \gamma} v_j(w) \cdot \lambda}{(i - \gamma + 1) \cdot \lambda} = \frac{\sum_{j \geq \gamma} v_j(w)}{(i - \gamma + 1)}.$$

2. Inequality $a \leq b$ implies that $v_j(a) \leq v_j(b)$. This implies that for any x, $\frac{\sum_j v_j(a) \cdot x_j}{P_i(x)} \leq \frac{\sum_j v_j(b) \cdot x_j}{P_i(x)}$. This equation holds even if we take a function $\min_x$ for both sides.

■

Lemma 3.2 *Given a weight vector w, let μ denote the cardinality of the set $\{w_j : w_j \geq 1\}$. If $\beta_i(w) < 1$ for some i, then $\sum_{j:w_j<1} w_j \ < \ i - \mu$.*

Proof: Suppose that $\beta_i(w) = (\sum_{j \geq \gamma} v_j(w))/(i - \gamma + 1)$, as guaranteed by lemma 3.1. Then,

$$\begin{aligned}
\sum_{j:w_j<1} w_j &= \sum_{j \geq 1} v_j(w) - \mu \\
&= \sum_{1 \leq j < \gamma} v_j(w) + \sum_{j \geq \gamma} v_j(w) - \mu \\
&\leq \gamma - 1 + \sum_{j \geq \gamma} v_j(w) - \mu \\
&= \gamma - 1 + (i - \gamma + 1)\beta_i(w) - \mu \\
&< \gamma - 1 + (i - \gamma + 1) - \mu = i - \mu
\end{aligned}$$

■

Since $\beta_i(w) < 1$ implies that $i > \mu$, we must have $i - \mu > 0$. We now prove our main result. The basic idea is to look at the first time $\beta_i(w)$ becomes greater than 1 and use the dual costs at that time to obtain a certificate of approximate optimality for P_i, using lemma 3.2 to relate the increase in dual costs till that time to the increase in the primal variables.

Theorem 3.3 *The algorithm* Distributed-Majorization *gives us an $O(\log \rho)$-majorized solution.*

Proof: For simplicity, $\sum_e l_e(t)c_e$ and $\beta_i\Big(w(t)\Big)$ are denoted by $D(t)$ and $\beta_i(t)$ respectively. Suppose that $\beta_i(T_i - 1) < 1$ and $\beta_i(T_i) \geq 1$.

$$\begin{aligned}
D(T_i) &= \sum_e \Big(l_e(T_i - 1)\Big(1 + \delta \Delta\Lambda_e(T_i - 1)\Big)c_e \Big) \\
&= D(T_i - 1) + \delta \sum_e \Big(l_e(T_i - 1)\Delta\Lambda_e(T_i - 1)c_e \Big) \\
&= D(T_i - 1) + \delta \sum_e \left(l_e(T_i - 1) \sum_{j:e \in p_j} \Delta x_j(T_i - 1) \right) \\
&= D(T_i - 1) + \delta \sum_j \left(\sum_{e \in p_j} l_e(T_i - 1)\Delta x_j(T_i - 1) \right) \\
&= D(T_i - 1) + \delta \sum_j \Big(w_j(T_i - 1)\Delta x_j(T_i - 1) \Big)
\end{aligned}$$

For one iteration, we increase x_j only when $w_j < 1$. Thus,

$$D(T_i) \;=\; D(T_i-1) + \delta \sum_{j:w_j(T_i-1)<1} w_j(T_i-1)\Delta x_j(T_i-1).$$

By lemma 3.2,

$$D(T_i) < D(T_i-1) + \delta\Big(i-\mu(T_i-1)\Big)\frac{\min_e c_e}{n\delta}$$

where $\mu(t) = \Big|\Big\{w_j(t) : w_j(t) \geq 1\Big\}\Big|$. Note that $\Big(i-\mu(T_i-1)\Big)\frac{\min_e c_e}{n\delta} = P_i\Big(\Delta x(T_i-1)\Big)$. Since the above argument also holds at all times $t \leq T_{i-1}$, we have

$$\begin{aligned}
D(T_i) &< D(0) + \delta \cdot \sum_{0\leq t\leq T_i-1} P_i\Big(\Delta x(t)\Big) \\
&\leq \frac{\delta}{2\rho^3}\cdot \max_e c_e \cdot m + \delta \cdot \sum_{0\leq t\leq T_i-1} P_i\Big(\Delta x(t)\Big) \\
&= \delta\frac{1}{2\rho}\cdot\frac{\max_e c_e}{\rho}\cdot\frac{m}{\rho} + \delta \cdot \sum_{0\leq t\leq T_i-1} P_i\Big(\Delta x(t)\Big) \\
&\leq \delta\cdot\frac{1}{2n}\cdot\min_e c_e + \delta \cdot \sum_{0\leq t\leq T_i-1} P_i\Big(\Delta x(t)\Big) \\
&= \delta\left(P_1\Big(x(0)\Big) + \sum_{0\leq t\leq T_i-1} P_i\Big(\Delta x(t)\Big)\right) \\
&\leq \delta P_i\Big(x(T_i)\Big).
\end{aligned}$$

Since $1 \leq \beta_i(T_i) \leq \alpha_i\Big(l_e(T_i)\Big)$ for all T_i by the lemma 3.1 part 2,

$$\frac{D(T_i)}{\alpha_i\Big(l_e(T_i)\Big)P_i\Big(x(T_i)\Big)} < \delta \;\text{ for all } i.$$

Using duality,

$$P_i\Big(x(T_n)\Big) \geq P_i\Big(x(T_i)\Big) \geq \frac{P_i^*}{\delta} = \frac{P_i^*}{O(\log\rho)} \;\text{ for all } i.$$

■

Running time: Let I be the number of iterations of our distributed algorithm, and let f be an edge which sees increase in flow during the last iteration. This edge must have also seen an increase in its flow during each of the previous iterations. Since the final solution is feasible,

$$c_f \geq \sum_{i:f\in p_i} x_i(0) + I\cdot\frac{\min_e c_e}{n\delta} \geq \frac{\min_e c_e}{2n} + \frac{I\cdot\min_e c_e}{n\delta} = \frac{\min_e c_e}{n}\left(\frac{1}{2}+\frac{I}{\delta}\right).$$

Hence, $I = O\left(n\delta\cdot\frac{\max_e c_e}{\min_e c_e}\right) = O(nR\log\rho)$.

4 Extensions and Improvements

4.1 Polylogarithmic number of iterations for the fixed route case

Replace line 5 in algorithm DISTRIBUTED-MAJORIZATION with $x_i(t) = x_i(t-1)(1+1/\delta)$. Since the routes are fixed, and any flow which is active at time t must have also been active at time $t-1$, the new load

introduced on any edge in the t-th iteration can be at most a $(1/\delta)$-fraction of the load already introduced on the edge in the first $t-1$ iterations. If the loads after time $t-1$ are feasible, then the new load introduced at time t is at most a $(1/\delta)$-fraction of the edge's capacity; by the footnote in section 3, this is sufficient to guarantee feasibility of the final solution. The proof of approximation ratio does not depend on the size of the update. The running time analysis is now simple. The initial bandwidth for any flow is $\min_e c_e/(2n)$. Hence, the number of iterations is at most

$$\log_{1+1/\delta}\left(\frac{\max_e c_e}{\min_e c_e/(2n)}\right) = O(\delta \log \rho) = O(\log^2 \rho).$$

4.2 Multi-path routing

We modify the behavior of the flow agents in the basic algorithm as follows. Initially, each flow agent chooses an arbitrary path and uses this path to allocate to itself a bandwidth of $\min_e c_e/(2n)$. In each iteration, each flow agent computes the shortest path from s_i to t_i using the dual costs. Let p_i be this path, and let w_i be the cost. If $w_i < 1$, the flow agent allocates additional $\min_e c_e/(2n)$ bandwidth to itself along p_i. The edge agents perform exactly the same initializations and updates as in the basic algorithm described in section 3, and the same analysis as in section 3 applies. Hence, we get a simultaneous optimization of $O(\log \rho)$ in polynomial number of iterations. Unfortunately, the techniques in section 4.1 can not be applied here; we can either reduce the number of iterations to polylogarithmic, or handle multi-path routing, but not both. The number of iterations increases by another factor of m; we omit the details and make no attempt to optimize the polynomial in this abstract.

4.3 Pricing for fairness for fixed routes

The algorithm, as described, must start from the precise initial conditions we defined. In order to convert this distributed algorithm into a TCP-like protocol, we must allow it to start with arbitrary primal flows, i.e. we need an algorithm which settles into an equilibrium. Also, if a social planner decides to implement our scheme for resource allocation, it would be desirable to have the scheme supported by a pricing mechanism that has a natural interpretation and settles into an equilibrium. We solve both problems using the same idea – the dual cost of an edge is computed differently for different flow agents. Intuitively, if one flow agent introduces 0.2 units of flow and the other introduces 0.3 units of flow on the same edge, then it is unfair to penalize the first agent for the extra 0.1 units of flow allocated to the second agent. Hence, the dual cost $l_e(i)$ of edge e for flow agent i should be computed by first reducing every flow x_j to $\min\{x_i, x_j\}$. Let the total load on edge e after this reduction be $\Lambda_e(i)$. Then define $l_e(i)$ as $\theta_e e^{\delta \Lambda_e(i)}$, where θ_e is chosen such that the cost functions used by this protocol match with the initial dual costs of the basic distributed algorithm in section 3. i.e. $\theta_e = \tilde{l}_e/e^{\delta \tilde{\Lambda}_e}$, where $\tilde{l}_e$ and $\tilde{\Lambda}_e$ are the costs and allocations on edge e at the beginning of the basic algorithm. Each edge agent can compute this quantity locally. The dual cost of flow agent i is $w_i = \sum_{e \in p_i} l_e(i)$.

Theorem 4.1 *If each flow agent updates its flow according to the differential equation*

$$dx_i/dt = -\log w_i$$

where w_i is as computed above, then there is a unique equilibrium for the resulting dynamic system, and at this equilibrium, $r(x) = O(\log \rho)$.

We omit the proof of the above theorem; the basic idea is to show that any equilibrium must correspond to the limiting behavior of the basic algorithm in section 3 as the amount of incremental bandwidth allocated in each iteration goes to 0. It is quite intriguing that such a simple and natural protocol achieves a very strong simultaneous approximation for all canonical utility functions. The convergence properties of this protocol deserve further study, specially for delayed feedback.

References

[1] Y. Afek, Y. Mansour, and Z. Ostfeld. Convergence complexity of optimistic rate based flow control algorithms. *Journal of Algorithms*, 30(1):106–143, 1999.

[2] A. Awerbuch and Y. Azar. Local optimization of global objectives: competitive distributed deadlock resolution and resource allocation. *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 240–49, 1994.

[3] B. Awerbuch and Y. Shavitt. Converging to approximated max-min flow fairness in logarithmic time. *Proceedings of the 17th IEEE Infocom conference*, pages 1350–57, 1998.

[4] Y. Bartal, J. Byers, and D. Raz. Global optimization using local information with applications to flow control. *38th Annual Symposium on Foundations of Computer Science*, pages 303–312, 1997.

[5] Y. Bartal, M. Farach-Colton, M. Andrews, and L. Zhang. Fast fair and frugal bandwidth allocation in atm networks. *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 92–101, 1999.

[6] R. Bhargava, A. Goel, and A. Meyerson. Using approximate majorization to characterize protocol fairness. *Proceedings of ACM Sigmetrics*, pages 330–331, June 2001.

[7] S. Cho and A. Goel. Bandwidth allocation in networks: a single dual-update subroutine for multiple objectives. *Lecture Notes in Computer Science (proceedings of the first Workshop on Combinatorial and Algorithmic Aspects of Networks (CAAN), Aug 2004)*, 3405:28–41.

[8] B. Codenotti, A. Saberi, K. Varadarajan, and Y. Ye. Leontief economies encode nonzero sum two-player games. *SODA 2006 (to appear)*, 2006.

[9] L.K. Fleischer. Approximating fractional multicommodity flows independent of the number of commodities. *SIAM J. Discrete Math.*, 13(4):505–520, 2000.

[10] N. Garg and J. Konemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *39th Annual Symposium on Foundations of Computer Science*, pages 300–309, 1998.

[11] N. Garg and N. Young. On-line, end-to-end congestion control. *IEEE Foundations of Computer Science*, pages 303–312, 2002.

[12] A. Goel and A. Meyerson. Simultaneous optimization via approximate majorization for concave profits or convex costs. *Technical report CMU-CS-02-203, Computer Science Department, Carnegie Mellon University*, December 2002.

[13] A. Goel, A. Meyerson, and S. Plotkin. Approximate majorization and fair online load balancing. *Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms*, pages 384–390, Jan 2001.

[14] A. Goel, A. Meyerson, and S. Plotkin. Combining fairness with throughput: Online routing with multiple objectives. *Journal of Computer and Systems Sciences*, 63(1):62–79, 2001.

[15] G.H. Hardy, J.E. Littlewood, and G. Polya. Some simple inequalities satisfied by convex functions. *Messenger Math.*, 58:145–152, 1929.

[16] G.H. Hardy, J.E. Littlewood, and G. Polya. *Inequalities*. 1st ed., 2nd ed. Cambridge University Press, London and New York., 1934, 1952.

[17] F.P. Kelly, A.K. Maulloo, and D.K.H. Tan. Rate control in communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research Society*, 49:237–252, 1998.

[18] J. Kleinberg, Y. Rabani, and E. Tardos. Fairness in routing and load balancing. *J. Comput. Syst. Sci.*, 63(1):2–20, 2001.

[19] A. Kumar and J. Kleinberg. Fairness measures for resource allocation. *Proceedings of 41st IEEE Symposium on Foundations of Computer Science*, 2000.

[20] S. Low, L. Peterson, and L. Wang. Understanding TCP Vegas: a duality model. *Proceedings of ACM Sigmetrics*, 2001.

[21] M. Luby and N. Nisan. A parallel approximation algorithm for positive linear programming. *Proceedings of 25th Annual Symposium on the Theory of Computing*, pages 448–57, 1993.

[22] A.W. Marshall and I. Olkin. *Inequalities: theory of majorization and its applications.* Academic Press (Volume 143 of Mathematics in Science and Engineering), 1979.

[23] S. Plotkin, D. Shmoys, and E. Tardos. Fast approximation algorithms for fractional packing and covering problems. *Math of Oper. Research*, pages 257–301, 1994.

[24] A. Tamir. Least majorized elements and generalized polymatroids. *Mathematics of Operations Research*, 20(3):583–589, 1995.