Fairness in Routing and Load Balancing *

Jon Kleinberg[†] Yuval Rabani[‡] Éva Tardos[§]

Abstract

We consider the issue of network routing subject to explicit *fairness* conditions. The optimization of fairness criteria interacts in a complex fashion with the optimization of network utilization and throughput; in this work, we undertake an investigation of this relationship through the framework of approximation algorithms.

In a range of settings including both high-speed networks and Internet applications, *max-min fairness* has emerged as a widely accepted formulation of the notion of fairness. Informally, we say that an allocation of bandwidth is *max-min fair* if there is no way to give more bandwidth to any connection without decreasing the allocation to a connection of lesser or equal bandwidth. Given a collection of transmission routes, this criterion imposes a certain equilibrium condition on the bandwidth allocation, and some simple flow control mechanisms converge quickly to this equilibrium state. Indeed, the vast majority of previous work on max-min fairness has focused on this issue of associating rates with connections that are specified by a *fixed* set of paths. Very little work has been devoted to understanding the relationship between the way in which one selects paths for routing, and the amount of throughput one obtains from the resulting max-min fair allocation on these paths.

In this work we consider the problem of selecting paths for routing so as to provide a bandwidth allocation that is as fair as possible (in the max-min sense). We obtain the first approximation algorithms for this basic optimization problem, for single-source unsplittable routings in an arbitrary directed graph. Special cases of our model include several fundamental load balancing problems, endowing them with a natural fairness criterion to which our approach can be applied. Our results form an interesting counterpart to earlier work of Megiddo, who considered max-min fairness for single-source fractional flow. The optimization problems in our setting become NP-complete, and require the development of new techniques for relating fractional relaxations of routing to the equilibrium constraints imposed by the fairness criterion.

*A preliminary version of this work appeared at the 1999 IEEE Symposium on Foundations of Computer Science.

†Dept. of Computer Science, Cornell University, Ithaca NY 14853. Email: kleinber@cs.cornell.edu. Supported in part by an Alfred P. Sloan Research Fellowship, an ONR Young Investigator Award, NSF Faculty Early Career Development Award CCR-9701399, and BSF grant 96-00402.

‡Computer Science Department, Technion — IIT, Haifa 32000, Israel. Email: rabani@cs.technion.ac.il. Work at the Technion supported by BSF grant number 96-00402, Ministry of Science contract #9480198, and by the Fund for the Promotion of Research at the Technion.

§Dept. of Computer Science, Cornell University, Ithaca NY 14853. Email: eva@cs.cornell.edu. Research partially supported by NSF grant CCR-9700163, ONR grant N00014-98-1-058, and BSF grant 96-00402.

1 Introduction

Fairness in routing. A basic problem in network optimization is the efficient routing of traffic between pairs of *terminal nodes* that wish to communicate. One of the fundamental notions that arises in such a setting is that of *fairness*; we want to allocate bandwidth to the connections in a way that does not unnecessarily "starve" any of them. Although it is an intuitively natural concept, finding a concrete definition of fairness that captures the goals of efficient routing is a subtle issue — we wish to prevent starvation of individual connections in a way that allows all connections the opportunity to receive as large a bandwidth allocation as possible.

An elegant framework that has gained wide acceptance in the networking community is the notion of *max-min fairness* [2, 5] — it forms the basis for bandwidth allocation in both high-speed networks and a range of Internet applications. It is defined via a type of equilibrium: An allocation of bandwidths, or *rates*, to a set of connections is said to be *max-min fair* if it is not possible to increase the allotted rate of any connection while decreasing only the rates of connections which have larger rates. In other words, no connection can increase its bandwidth at the expense of connections which are better off than it is. This turns out to be equivalent to another natural definition of fairness — that the list of allotted rates, when sorted in increasing order, is lexicographically as great as possible. This lexicographic definition allows one to directly compare different bandwidth allocations, and speak of the *fairest* allocation.

The vast majority of work on max-min fairness has focused on the setting in which connections are specified by a *fixed* set of paths, and one wants to associate rates with these paths. It is easy to show that the max-min fair allocation for a fixed set of paths is unique, and a number of simple, efficient algorithms have been developed to compute this allocation (e.g. [1, 2, 5]). A wide range of network routing protocols employ such algorithms to enforce max-min fairness (or a close approximation) on the paths used for routing connections. Note that all of this takes place, however, *after* the paths themselves have been chosen; very little work has been devoted to understanding the relationship between the way in which one selects paths for routing, and the amount of throughput one obtains from the resulting fair allocation on these paths. Suppose we want to select paths for routing so as to provide a bandwidth allocation that is as fair as possible (in the max-min sense); how should we go about doing this? Megiddo [12] addressed this problem in the setting of *single-source fractional flow*, in which flow must be sent fractionally to a collection of terminals from a common source, and provided an elegant polynomial-time algorithm.

In this work, we consider the setting in which each connection must be routed on a single path — i.e. we seek an *unsplittable flow*. The single-source case here presents qualitatively new issues from those encountered in Megiddo's setting, for we can show that the fundamental analogue of his problem is now NP-complete. A number of basic load balancing problems arise naturally as special cases of this single-source unsplittable flow model. We obtain the first approximation algorithms for the problem of optimizing over path selection to provide the fairest possible routing. The issues that arise in this framework turn out to involve a number of interesting and very basic trade-offs between the throughput and the type of *equilibrium* constraints imposed by max-min fairness.

We now provide a concrete formulation of these optimization problems, and then summarize our results in more detail.

Formulating the problem: Max-min fairness and approximation guarantees. We seek routings from a common *source* node to a collection of terminals in a network. A routing, in the present framework, consists of two components — the choice of paths that the traffic will use, and the allocation of available bandwidth on these paths to the different connections. Thus, let $G = (V, E)$ be a directed graph with a capacity $c_e \geq 0$ on each edge. We designate a *source* $s \in V$ and a set of *terminals* $t_1, \ldots, t_k \in V$. A *routing* of the terminals consists of a set of paths $\{P_1, \ldots, P_k\}$ — with P_i a path from s to t_i — and an *allocation vector* $r = (r_1, r_2, \ldots, r_k)$. We view the s-t_i connection as being assigned path P_i, with bandwidth allocation, or rate, r_i. We say that this routing is *feasible* if, for all edges e, the total bandwidth allocated for paths using e is at most c_e; that is, the sum of r_i over all P_i containing e is at most c_e.

One can derive max-min fairness from the following intuitive approach to finding the "fairest" allocation: One should first make sure that the minimum bandwidth given to any connection is as large as possible; then, ignoring this "minimum" connection, one should make sure that the minimum bandwidth given to any of the connections that can still get additional bandwidth is as large as possible; and so on. More formally, given two k-tuples of numbers $z = (z_1, \ldots, z_k)$ and $z' = (z'_1, \ldots, z'_k)$, each in non-decreasing order, we say that z *lexicographically dominates* z' if $z = z'$, or there is some index j for which $z_j > z'_j$ and $z_i = z'_i$ for all $i < j$. Given two allocation vectors r and r', we say that r is *as fair as* r' (written $r' \preceq r$) if the sorted order of the coordinates of r lexicographically dominates the sorted order of the coordinates in r'. We will say that r and r' are *equivalent* if both $r' \preceq r$ and $r \preceq r'$. This relation defines a total order on the equivalence classes of allocation vectors; the vectors in the unique maximal equivalence class under $\preceq$ are thus the *fairest* allocations.

One can also use the following equivalent definition of a routing with allocation vector r being "fairest" in the max-min sense: There is no way to increase any entry r_i without decreasing some other entry r_j such that $r_j \leq r_i$.

As we discussed above, max-min fairness in the networking community has been applied primarily to the setting in which one is given not only a set of connections in a network, but also the paths $\{P_i\}$ that they are to use. Thus the only issue is to determine the allocation vector, which is unique in this case; and this can be accomplished by a variety of efficient algorithms (see e.g. [1, 2, 5]). Network protocols that employ max-min fairness thus enforce the following *max-min equilibrium* condition:

> (†) For any routing with paths $\{P_i\}$ and allocation vector r, r must be a fairest allocation given the paths P_i.

The crucial issue raised in the discussion above is then the following. We wish to choose paths for routing a set of connections, with the bandwidth allocation vector then uniquely determined by the equilibrium condition (†). The amount of bandwidth utilization in a fairest allocation depends heavily on the set of paths $\{P_i\}$ that one chooses; some choices of paths allow for much greater fair utilization of the network than others. The fundamental question we seek to address is that of determining the fairest *routing*, optimizing over all possible choices of paths, with the allocation vector determined by (†). For example, does the fairest routing achieve the maximum possible throughput of *any* routing? This was the precise problem considered by Megiddo [12] in the context of the single-source *fractional* flow problem, in which all connections share a common endpoint, but one can divide the flow for a single connection fractionally over many paths. In addition to providing a polynomial-time algorithm for computing a fairest routing, he showed that the fairest flow is a maximum flow — with fractional flow, one does not sacrifice throughput by imposing fairness.

In this work, we focus on the analogous problem, computing a fairest routing, in the setting of single-source unsplittable flow [3, 8, 9]. Once we move to unsplittable flow, the basic problem becomes NP-complete, even in the *unit-capacity* case with all c_e equal to 1. More precisely, we can prove that the following decision problem is NP-complete in the unit-capacity case: given G, s, the terminals $\{t_i\}$, and a vector r^*, is there a routing of the terminals for which the allocation vector r satisfies $r^* \preceq r$? (Additionally, we can show that the fairest flow need not be a maximum unsplittable flow.)

In view of this NP-completeness result, we focus on obtaining both general approximation algorithms and exact algorithms for polynomial-time special cases. The optimization problems here are over the ordering on allocation vectors defined by fairness — hence, since there is no single numerical measure, we must be careful in how we define our notion of approximation to the optimum. We propose the following two natural definitions of approximation. First, we say that r is a *coordinate-wise* c-approximation to r^* if for each j, the j^{th} smallest entry in r is at least $1/c$ times the value of the j^{th} smallest entry in r^*. As a weaker notion, we say that r is a *prefix-sum* c-approximation to r^* if for each j, the sum of the j smallest entries in r is at least $1/c$ times the sum of the j smallest entries in r^*. In other words, a prefix-sum approximation ensures that the subsets of terminals with the smallest allocations receive sufficient bandwidth.

When we move to approximate solutions, it is very important that we can keep in mind that the equilibrium condition (†), or a relaxed version of (†), serves as an additional *feasibility* requirement on the solutions we can produce: in effect, we are able to choose only the paths P_i, for then the network uses (†) to enforce the unique equilibrium allocation vector r. (Of course, in the fairest routing, the allocation r will necessarily be in equilibrium.) This requirement rules out, for example, the following simple approach based on the Dinitz-Garg-Goemans unsplittable flow approximation algorithm [3]: compute the fairest fractional flow using Megiddo's algorithm, scale all resulting allocations down by a factor of 2, and route them as unsplittable demands. The problem is that these scaled demands are generally very far from equilibrium for the paths used. For example, if the fairest fractional flow has allocations of widely varying magnitude, it is easy to find examples in which the Dinitz-Garg-Goemans algorithm produces a routing where flow paths with both small and large allocation share edges, and the allocation vector is arbitrarily far from satisfying the equilibrium condition (†). All previous single-source unsplittable flow algorithms [8, 9] exhibit the same problem.

Summary of results: Routing. For the single-source unsplittable flow problem on an arbitrary directed graph with unit capacities, we indicated above that finding a fairest allocation vector is NP-complete. We develop a general approximation algorithm for this problem by relaxing both the optimality and the equilibrium requirements. First, what do we mean by relaxing the equilibrium requirements? For a constant c, we say that an allocation vector r is in a state of *c-approximate equilibrium* if it is not possible to raise the value of an entry r_i without decreasing some other entry r_j such that $r_j \leq cr_i$. Thus, 1-approximate equilibrium indeed corresponds to max-min equilibrium; we believe that approximate relaxations of these natural equilibrium notions raise a number of interesting issues in their own right.

We give an algorithm that produces a routing whose allocation is in 2-approximate equilibrium, and is a coordinate-wise 2-approximation to the allocation of the fairest fractional routing.

We develop the algorithm by computing a fairest flow for the following "discretized" version of the fairest routing problem. Suppose we only consider routings whose allocation vectors have entries that are all inverse powers of two; we will call such routings and allocation vectors *binary.* Then we can restrict our fairness ordering $\preceq$ to binary allocation vectors, and seek a fairest allocation of this type. We show how to find a fairest binary routing in polynomial time, for the single-source unsplittable flow problem on an arbitrary unit-capacity directed graph G. It is not difficult to show that the fairest binary routing we obtain is both a coordinate-wise 2-approximation to the unrestricted fractional optimum, and in a state of 2-approximate equilibrium.

We find the existence of a polynomial-time algorithm for fairest binary routings somewhat surprising, given that the same problem for unrestricted routings is NP-complete. As a basic building block in the algorithm, we first establish the special case that if all terminals can be routed with at most two paths on any edge, then the fairest unrestricted routing (which will be binary) can be computed in polynomial time. We then apply this result over increasingly large cuts in the graph G to piece together an optimal binary flow.

A natural problem is to provide a good approximation to the fairest unsplittable routing in an arbitrary directed graph without relaxing the equilibrium condition (†); we leave this as an open question.

Summary of results: Load Balancing. The setting of single-source unsplittable flow contains a range of *load balancing* problems. We begin by providing algorithms for two of the most natural of these *without* relaxing the equilibrium condition (†).

- First, the single-source unsplittable flow problem on a *two-level* unit-capacity graph is equivalent to the following load balancing problem: we have a set of jobs $J = \{J_1, \ldots, J_k\}$, and a set of machines $M = \{M_1, \ldots, M_n\}$; for each job J_i, there is a set $S_i \subset M$ on which J_i can be run. Each machine has the same "processing power." We wish to assign each job to a machine, and our fair *allocation vector* $r = (r_1, \ldots, r_k)$ specifies the fraction of processing power each job J_i receives on its assigned machine. We will call this the *uniform load balancing* problem.

- More generally, each job J_i can have an *upper bound* u_i on the amount of processing power it wants. In this setting, we will only consider allocation vectors r for which $r_i \leq u_i$ for each i. We will call this the *non-uniform load balancing* problem; this problem too can be encoded in the single-source unsplittable flow problem, with the upper bounds u_i appearing as capacities.

We first show that a fairest allocation vector for the uniform load balancing problem can be computed in polynomial time. This can be viewed as a natural analogue of Megiddo's result to a setting with unsplittable assignments; the tractability of the problem comes essentially from its connection with bipartite matching, although it is important to note that the allocations in the optimal fractional and integer flows are not the same.

Finding a fairest allocation for the non-uniform load balancing problem is NP-complete; indeed, even determining whether every job can achieve its upper bound u_i is an NP-complete problem considered by Lenstra, Shmoys, and Tardos [10]. We give a polynomial-time algorithm that produces a prefix-sum 2-approximation to the fairest allocation. The approximate allocation we produce is (following our discussion above) in max-min equilibrium. We begin from a fairest *fractional* allocation of jobs to machines — here the allocation of one job J_i can be spread over several machines in its set S_i — computed via Megiddo's algorithm. We then build on the fractional rounding algorithm in [10] to obtain the approximation. Our prefix-sum approximation in fact shows a type of integrality gap in this multi-coordinate setting; it is a prefix-sum 2-approximation to the optimal fractional allocation. We will describe simple examples in which there cannot be a coordinate-wise $O(1)$-approximation to this fractional optimum.

Organization. The remaining three sections of the paper can be read independently. Section 2 develops the algorithms for the load balancing problem without relaxing the equilibrium condition (†). In Section 3 we consider the single-source fair unsplittable flow problem on an arbitrary unit capacity directed graph. We develop a general approximation algorithm for this problem by relaxing both the optimality and the equilibrium requirements. Finally, Section 4 shows that the single-source unsplittable fair flow problem is NP-complete on unit capacity directed graphs.

2 Fair Load Balancing Algorithms

The *fair load balancing problem* is concerned with assigning jobs to machines. Assume that we have a set of jobs $J = \{1, \ldots, k\}$, and a set of machines $M = \{M_1, \ldots, M_n\}$; and for each job j, there is a set $S_j \subset M$ on which job j can be run. An *assignment* is a function $\mathcal{F} : J \to M$ so that $\mathcal{F}$ assigns each job j to a machine in S_j. First we consider the special case of the uniform load balancing problem, and show that an optimum fair solution can be found in this case. Then we consider extensions to problems where the jobs have different needs.

Uniform Load Balancing

The uniform fair load balancing problem can be restated as follows. We want to assign jobs to machines, and choose a load ℓ_j for each job j so that the following two conditions hold. First, if $A(i)$ denotes the set of jobs assigned to machine M_i, we must have that $\sum_{j \in A(i)} \ell_j \leq 1$. Second, the set of allocated loads sorted from smaller to larger should be lexicographically maximal.

If we are *given* an assignment of jobs to machines the corresponding fair loads are very easy to compute:

Lemma 2.1 *Given an assignment of jobs to machines, the fairest allocation load is to assign load $\ell_j = \frac{1}{d_i}$ to job j, where job j has been assigned to a machine M_i with $d_i = |A(i)|$.*

This lemma simply represents the constraint imposed by the equilibrium condition (†). Our goal is now to optimize over all assignments of jobs to machines. Based on Lemma 2.1, our primary objective is to minimize $d_{\max} = \max_i |A(i)|$, the maximum number of jobs that go on the same machine. Our secondary objective function is to have as few jobs as possible assigned to such highly loaded machines, and so on. We obtain the following equivalent formulation of the load balancing problem. In an assignment $\mathcal{F}$ let d_i denote the number of jobs assigned to machine M_i; corresponding to the standard view of assignment problems in terms of bipartite graphs, we will also refer to d_i as the *degree* of M_i.

Lemma 2.2 *The uniform fair load balancing problem is equivalent to finding an assignment $\mathcal{F}$ so that the sequence of degrees d_i for $i = 1, \ldots, m$ when sorted from large to small is lexicographically as small as possible.*

We will use $\mathcal{F}$ more generally to denote a possibly partial assignment of jobs to machines; we write $|\mathcal{F}|$ to denote the number of jobs assigned by $\mathcal{F}$. We say that $\mathcal{F}$ is a (partial) *assignment of maximum degree* d if the maximum number of jobs assigned to a machine is d, and $\mathcal{F}$ is a *maximum assignment* of degree at most d if the number of unassigned jobs is the least possible among all assignments of degree at most d. Given any assignment $\mathcal{F}$ of jobs to machines of degree at most d, we can use augmenting paths to find a maximum assignment $\mathcal{F}'$ of degree at most d. We will refer to this process as $\textsc{Augment}(\mathcal{F}, d)$. Our load balancing algorithm starts with $\mathcal{F}_0 = \emptyset$, and defines $\mathcal{F}_d$=$\textsc{Augment}(\mathcal{F}_{d-1}, d)$ iteratively for $d = 1, 2, \ldots$ until all jobs get assigned. The assignment at termination is then returned; we denote this assignment $\mathcal{F}^*$.

The assignment found by $\mathcal{F}'$=$\textsc{Augment}(\mathcal{F}, d)$ has the following properties. First, all jobs assigned in $\mathcal{F}$ are also assigned in $\mathcal{F}'$. Second, if d_i and d'_i denotes the degree of machine M_i in assignments $\mathcal{F}$ and $\mathcal{F}'$ then $d_i \le d'_i$. Both of these properties follow from the augmenting path algorithm: augmenting paths never use the backwards edges leaving the sink or entering the source.

These two properties imply that the final assignment $\mathcal{F}^*$ in a sense contains an optimum assignment for all degrees d. Let d_i be the number of jobs assigned to machine M_i by the final assignment $\mathcal{F}^*$. Then we have the following.

Lemma 2.3 *For all integers d we have that $|\mathcal{F}_d| = \sum_i \min(d_i, d)$. Further, if the degree of a machine i is less than d in assignment $\mathcal{F}_d$, then the degree will not change throughout the rest of the algorithm.*

Proof. The first statement will follow from the monotonicity of the degrees during the augmentations. Consider assignment $\mathcal{F}_d$. The augmentations done after this assignment will not decrease the degrees due to the monotonicity property. Hence $|\mathcal{F}_d| \le \sum_i \min(d_i, d)$. The right hand side is the size of the assignment of maximum degree d obtained by deleting edges from $\mathcal{F}^*$ entering nodes of degree more than d. The matching $\mathcal{F}_d$ is a maximum such matching, so we also have the opposite inequality $|\mathcal{F}_d| \geq \sum_i \min(d_i, d)$. The second statement follows immediately from the first one and the monotonicity of the degrees. ■

The essence of why this algorithm is optimal is contained in the following lemma. Let $r_d = k - |\mathcal{F}_d|$ denote the number of unassigned jobs in the maximum assignment of degree d.

Lemma 2.4 *Let $\mathcal{F}^*$ be the assignment found the algorithm, and $\mathcal{F}'$ some other assignment. For any degree d let f_d and f'_d denote the number of machines of degree d in $\mathcal{F}^*$ and $\mathcal{F}'$ respectively; let r_d denote the minimum possible number of unassigned jobs in an assignment of maximum degree d. We have that*

$$r_d \leq f'_{d+1} + 2f'_{d+2} + 3f'_{d+3} + \ldots,$$

$$r_d = f_{d+1} + 2f_{d+2} + 3f_{d+3} + \ldots.$$

Proof. To see the first statement, we can delete edges out of $\mathcal{F}'$ to create a matching of maximum degree d. We need to delete i edges from each machine with degree $d+i$, so the right hand side is the number of unassigned jobs at the end of this process. This is at least r_d by the definition of r_d.

To see the second statement we use the lemma above. Deleting i edges from each machine of degree $d+i$, we recreate the degree sequence of $\mathcal{F}_d$, hence the number of jobs unmatched at the end of this process is exactly $r_d = k - |\mathcal{F}_d|$. ■

The lemma immediately implies that the assignment M is optimum.

Theorem 2.5 *The algorithm above finds the optimum assignment of jobs to machines for the load balancing problem.*

Non-Uniform Load Balancing

Next we consider a more general version of the fair load balancing problem on machines. We will still assume that machines are uniform, in that the maximum possible load of each machine is the same. However, jobs will no longer be uniform.

Assume that we have an *upper bound* u_j for the amount of processing power a job j can use. Now an assignment of jobs to machines, and loads ℓ_j for each job j, must satisfy the following.

(i) $\ell_j \leq u_j$.

(ii) If $A(i)$ denotes the set of jobs assigned to machine i, we must have that $\sum_{j\in A(i)} \ell_j \leq 1$.

(iii) The allocation of loads to jobs satisfies the max-min equilibrium condition (†): we cannot increase the load of one job j with $\ell_j < u_j$ without decreasing the load of some other job j' that has $\ell_{j'} \leq \ell_j$.

We can think of this assignment problem as a flow problem in the following three-layer graph. We have a source s connected to nodes representing each of the machines M_i with an edge of capacity 1. There is an edge of infinite capacity from machine node i to job node j if

M_i belongs to S_j. Finally there is an edge from each job node j to a corresponding terminal t_j with capacity u_j.

A fairest fractional flow [12] in this network corresponds to a fairest fractional assignment of job loads to machines. Let f denote this fairest fractional assignment, and let ℓ_j denote the load of job j in f. We say that a job j is *integrally assigned* to machine M_i is the entire allocation of job j is to machine i; otherwise, we say that it is *partially assigned* to those machines on which it receives a strictly positive allocation. An assignment $\mathcal{F}$ of jobs to machines is *integral* if all jobs are integrally assigned by $\mathcal{F}$. At various points, we will use the notation $A(i)$ to refer to the set of jobs assigned to a machine M_i.

For an integral assignment of jobs to machines, the fairest allocation of loads can be computed on each machine independently, and it has a very simple form that follows from the definition of max-min equilibrium. For a real number $x \in [0,1]$, let $B(x)$ be the allocation of loads to jobs in a set $A(i)$ defined by allocating $\min(u_j, x)$ to each job $j \in A(i)$; the *total load* allocated by $B(x)$ is the sum $\sum_{j \in A(i)} \min(u_j, x)$. Then we have

Lemma 2.6 *The fairest allocation of load to $A(i)$ is $B(x^*)$, where x^* is the maximum $x \in [0,1]$ for which the total load of $B(x)$ is ≤ 1.*

The parameter x^* can thus be viewed as a *critical load* that caps the allocation to jobs whose upper bound u_j is too large. We can formulate this notion in a way that also extends to fractional assignments of jobs to machines, as follows.

Lemma 2.7 *In a (possibly fractional) assignment of jobs to machines, the fairest allocation of loads to the set $A(i)$ has the following property. The machine M_i has an associated* critical load *m_i; and the load of any job $j \in A(i)$ is $\ell_j = \min(m_i, u_j)$. For each machine, either the machine is fully loaded, or $m_i = \max(u_j)$ over all $j \in A(i)$. Further, if some job $j \in A(i)$ does not achieve its upper bound u_j, then*

$$m_i = \min_{j \in A(i): \ell_j < u_j} \ell_j.$$

That is, m_i is the minimum load among jobs on M_i that are not able to achieve their upper bounds.

In our algorithm, we will construct an integral assignment of jobs to machines, and then use the fairest allocation of load given by Lemma 2.6. Notice that finding the fairest assignment is NP-hard, as deciding if there is an assignment in which all jobs can be assigned their maximum load $\ell_j = u_j$ is a standard NP-hard scheduling problem [10].

Our goal will be to make the resulting loads closely approximate the load sequence of the fractional assignment. Instead of a coordinate-wise approximation of loads, we will give a prefix-sum 2-approximation. We observe that it might not be possible to approximate the vector of fairest fractional loads $\ell_1, \ell_2, \ldots$ with a fair integer assignment. Consider, for example, a problem in which we have n machines $M_1, \ldots, M_n$. Assume that the first job can be assigned to any machine, and has $u_1 = 1$. In addition to this job, for each machine M_i, there are $n-1$ jobs with $u_j = 1/n$ that can only be assigned to M_i. Now fractionally, we can assign each job its maximum load $\ell_j = u_j$ by spreading the load of job 1 across all machines. However, in any integer assignment, the fair load for that assignment will have

$\ell_1 = 1/n$; i.e., there is no way to approximate the optimal fractional fair load of job 1 in an integer assignment.

We use Megiddo's algorithm [12] to obtain a fairest fractional flow f. We then use the rounding algorithm of Lenstra, Shmoys and Tardos [10] to create an integral assignment $\mathcal{F}^*$ from f. Let $A(i)$ be the set of jobs assigned to machine M_i by $\mathcal{F}^*$, and let ℓ_j^A denote the fair load of this assignment.

Theorem 2.8 ([10]) *Assume f is a fractional assignment of jobs to machines assigning load ℓ_j to job node j. The approximation algorithm of [10] constructs an integral assignment $\mathcal{F}^*$ so that for each machine M_i, $A(i)$ consists of all the jobs that were integrally assigned to M_i by f, plus at most one additional job that was partially assigned to M_i. Since f did not assign more than 1 unit of load to any machine, we consequently have $\sum_{j\in A(i)} \ell_j \leq 1 + \max_{j\in A(i)} \ell_j$.*

First we analyze the fair loads of the set of jobs $A(i)$ for a single machine M_i. Let j_i denote the job with maximum load ℓ_j in $A(i)$,

Lemma 2.9 *The fair load of all jobs $j \in A(i)$ with the possible exception of job j_i have $\ell_j \leq 2\ell_j^A$.*

Proof. We say that M_i is *saturated* by f if some job j that was integrally assigned to M_i by f has a load $\ell_j < u_j$. Let m_i denote the critical load (in the sense of Lemma 2.7) of machine M_i in the fairest fractional assignment f. We define m_i' to be m_i if M_i is saturated by f; otherwise, we define m_i' to be the maximum of u_j over all jobs j that were integrally assigned to M_i by f.

In the analysis, we consider the allocation $B(m_i'/2)$ and show that it has total load ≤ 1. It then follows that any job that was integrally assigned to M_i has integral fair load at worst a factor of two smaller than its allocation in the fractional solution. Moreover, if $j' \in A(i)$ is the unique job that was partially assigned to M_i by f, and its load decreases by more than a factor of two, then we must have $\ell_{j'} > m_i'$, whence j' is the job of maximum load, and constitutes the exceptional job in the statement of the lemma.

Thus it remains only to prove that $B(m_i'/2)$ has total load at most 1. To see this, we consider two cases, depending on whether or not M_i is saturated by f. If M_i is saturated, then some job that was integrally assigned to M_i is given load m_i by f; in $B(m_i'/2)$ this job's load decreases by $m_i'/2$ and creates enough room for one extra job with load at most $m_i'/2$. If M_i is not saturated, then $m_i' = u_j$ for some integrally assigned job j; again, in $B(m_i'/2)$ this job's load decreases by $m_i'/2$, creating room for an extra job with load at most $m_i'/2$. ■

Theorem 2.10 *Let ℓ_j^A denote the load of job j in the assignment created by our algorithm. For each h the sum of the h smallest loads of ℓ_j^A is at least a half of the sum of the smallest h loads in the fairest fractional assignment f.*

Proof. Sort the jobs by increasing order of their load ℓ_j^A. On each machine M_i the job j_i has the maximum fractional load. This job will have maximum fair load ℓ^A among those assigned to M_i by $\mathcal{F}^*$, and hence we can assume that j_i is the last among all jobs in $A(i)$. Assume for this proof that the jobs are indexed in this order, i.e. $\ell_1^A \leq \ell_2^A \leq \cdots \leq \ell_{j_i}^A$.

Consider the prefix sum $t_h^A = \sum_{j \le h} \ell_j^A$ for each h. We will show that these values 2-approximate the corresponding optimum values. In particular we will show that for each h we have $\sum_{j \le h} \ell_j \le 2t_h^A$. This implies the theorem: The value t_h^A, the sum of the smallest h loads in the algorithm's assignment, needs to be compared to the smallest h values in loads ℓ_j, whereas here we compare it to a set of h values that may not be the smallest.

Consider the subset of the jobs j that are assigned to machine M_i. We now show that

$$\sum_{j \le h, j \in A(i)} \ell_j \le 2 \sum_{j \le h, j \in A(i)} \ell_j^A.$$

To see this consider two cases. Let j_i be the index of the job in $A(i)$ with maximum load ℓ_{j_i}. If $j_i > h$ then the inequality is true term by term due to Lemma 2.9. If $j_i \le h$ then our assumption that j_i is ordered last in $A(i)$ implies that all jobs in $A(i)$ participate in the sum. Now the statement follows from Lemma 2.8 as the sum of the loads in the fractional assignment is bounded by

$$\sum_{j \le h, j \in A(i)} \ell_j \le 1 + \max_{j \in A(i)} \ell_j \le 2.$$

Summing over all machines M_i we get the desired bound. ■

3 Single-Source Fair Routing in Graphs

In this section we give a 2-approximation to the fairest unsplittable routing for the single-source fair flow problem in arbitrary unit-capacity directed graphs. The problem is specified by a directed graph $G = (V, E)$, a source $s \in V$, and terminals $t_1, t_2, \ldots, t_k \in V$ (all edge capacities are 1). We first show that the *fairest binary unsplittable flow*, i.e., the most fair unsplittable flow whose allocation vector consists only of inverse powers of 2, can be found in polynomial time. Then we show that this binary fair flow is in 2-approximate equilibrium, and is also a coordinate-wise 2-approximation to the fairest fractional flow.

Fair Routings of Congestion Two

As a basic building block in the algorithm, we first establish the special case that if all terminals can be routed with at most two paths on any edge, then the fairest unrestricted allocation (which will be binary) can be computed in polynomial time. We will say that a set of paths has *congestion two* if at most two paths use any edge.

The assumption that all terminals can be routed with paths of congestion two implies that there is an unsplittable flow sending .5 from s to each of the terminals.

Lemma 3.1 *If .5 units of flow can be routed to all terminals then the fairest unsplittable flow is a flow that routes either .5 or 1 to each terminal, sending 1 to as many terminals as possible.*

Our goal in this special case can be rephrased as follows. We wish to create paths from the source to each of the terminals so that the following conditions hold.

(i) The set of paths has congestion two.

(ii) The number of paths that are involved in shared edges is as small as possible.

Given such a routing we can send a flow of value 1 on the paths that do not go through shared edges, and a flow of value .5 on all other paths. The main theorem of this subsection gives a way to find such paths in polynomial time, and also shows that there is such a routing in which the corresponding unsplittable flow is a maximum flow from s to the terminals.

Theorem 3.2 *Assume there are paths from s to the terminals with congestion two. Then there is a set of paths to the terminals with congestion at most two, where the number of paths that do not share edges with other paths is maximum subject to this condition, and the corresponding flow is a maximum flow from s to the terminals. This set of paths can be found in polynomial time.*

Proof. Let m denote the maximum number of disjoint paths from s to the terminals. Let f denote a maximum integer flow that sends at most 1 unit of flow to any terminal. For notational simplicity assume that the flow is sent to terminals $t_1, \ldots, t_m$, and let $P_1, \ldots, P_m$ denote the paths used by this flow, so that P_i is a path from s to t_i.

Let f' be the flow that corresponds to the paths to the terminals with congestion two. The flow f' sends .5 units of flow to each terminal. We plan to combine the flows f' and f to obtain the desired paths. Consider the flow $f' - f$ in the residual graph of G with respect to f. A path decomposition of this flow contains half-integral flow paths $Q_{m+1}, \ldots, Q_k$ where for $j = m+1, \ldots, k$, Q_j ends at terminal t_j, and starts at one of the first m terminals (a different one for each path; notice that by our assumptions, $m \geq k/2$).

The paths $P_1, \ldots, P_m$ and $Q_{m+1}, \ldots, Q_k$ satisfy the following.

(i) The paths P_i are disjoint.

(ii) The paths Q_j do not use edges of the P_i paths forwards, but may use them backwards.

(iii) The paths Q_j have congestion at most two.

Any set of paths $Q_{m+1}, \ldots, Q_k$ that satisfies the last two properties can be used to augment flow along the P_i paths. By sending .5 units of flow along each path Q_i we get a maximum flow (of value m) that sends at least .5 units of flow to each terminal. However, this is not an unsplittable flow as the augmentation might cause one unit of flow from s to t_i, for some $i \leq m$, use two paths.

If there is a set S of $2m - k$ of the paths P_i with the property that the Q_j don't use backward edges from any $P_i \in S$, and don't start at the terminal associated with any $P_i \in S$, then we can use each of the Q_j paths to augment the flow f by .5 without affecting the one unit of flow sent along those P_i that belong to S. This means that we get the desired maximum unsplittable flow f'', and a path decomposition of this flow gives the paths claimed by the theorem.

Our goal is thus to modify the paths Q_j for $j = m+1, \ldots, k$ so as to satisfy the assumption above:

- If a path Q_j uses one of the edges in the paths P_i backwards (i.e., uses the residual edge), then the corresponding terminal t_i is an endpoint of a (possibly different) path $Q_{j'}$.

Once we have such paths we can use the argument above to obtain the theorem.

We will modify the paths using a process that is similar to the Gale-Shapley stable marriage algorithm [4]. (Indeed, we can carry out the remainder of the proof through a reduction to the stable marriage problem; however, we feel it is simpler here to provide a direct argument.) We say that paths P_i and Q_j *meet* if Q_j has the terminal t_i as an endpoint, or if a contiguous segment of Q_j consists of backward edges from P_i. Note that there may be many such meetings, in this sense, between the same pair of paths P_i and Q_j. Suppose a path P_i meets a path Q_j, but t_i is not an endpoint of any path $Q_{j'}$. If there are many paths that meet P_i, then let Q_j be the path that meets P_i at an edge e closest to its terminal t_i. We change Q_j so that it begins from this terminal t_i, and continues along the backward edges of P_i until the meeting point e; it then continues as before. This re-routing of a path Q_j leads to an alternate set of paths that also satisfies the above properties, and hence can also be used for augmentation.

We repeat this process until there are no pairs of paths P_i and Q_j that satisfy the condition above. We now want to argue that this process terminates; for when it does, we have the set of augmenting paths needed to find the flow f''. To show termination, note that each re-routing decreases the number of distinct meetings between a path in $\{P_1, \ldots, P_m\}$ and a path in $\{Q_{m+1}, \ldots, Q_k\}$: before the re-routing, path Q_j met some other path $P_{i'}$ before meeting P_i, and this meeting is now eliminated. ■

For the next subsection we will need a version of this theorem that routes flows in smaller units. For some integer $\gamma > 0$ let v_γ denote the maximum value of a flow that sends at most $1/\gamma$ flow to each terminal. By considering each edge as a set of γ parallel edges we get the following.

Corollary 3.3 *Assume there is a flow that routes $1/(2\gamma)$ units of flow from s to each of the terminals, then there is an unsplittable flow that routes $1/(2\gamma)$ or $1/\gamma$ units of flow to each terminal and has value v_γ.*

Constructing a Binary Allocation

In this subsection we show how to construct the fairest binary flow in polynomial time, using the algorithm of Corollary 3.3. Let 2^{-c} be the maximum power of two such that there is a flow of value 2^{-c} from s to all of the terminals t_i. The lexicographic definition of the fairest binary flow implies that we must send at least 2^{-c} units of flow to each terminal. We use the Corollary 3.3 with $\gamma = 2^{c-1}$ to find flow paths from the source to each of the terminals. Let S_c denote the s-side of a minimum cut of value v_γ. If there are many such min-cuts, let S_c be the inclusion-wise minimal. (We will frequently identify cuts with their s-sides, hence referring to S_c as a cut.) From the paths obtained above we keep only the parts after leaving the cut S_c, and will recursively find beginning parts that match up with these paths.

There are two facts that we need about the inclusion-wise minimal min-cut S_c. First, any maximum flow saturates the edges leaving S_c. Second, no terminal that received only 2^{-c} flow in the routing above is contained in S_c. This latter statement follows as the minimum cut S_c consists of nodes reachable in the residual graph from s, and if a terminal with only 2^{-c} flow were reachable, then its flow could be increased.

The first observation allows us to define the following smaller problem that we solve recursively. Let the graph G' be obtained from G by considering the subgraph on S_c and adding to this graph all the edges leaving S_c. We keep all terminals in S_c, and replace the terminals outside of S_c by 2^{c-1} new terminals at the end of each of the edges entering S_c. The second property of S_c implies that in the new problem there is a flow that sends 2^{-c+1} units of flow from s to each of the terminals in G'. Each edge leaving S_c has 2^{c-1} new terminals, and so if each of these terminals receive 2^{-c+1} flow then the cut S_c has to be saturated.

Recursively we obtain a fairest binary flow f' on the subproblem on G'. We obtain the solution to the original problem by taking the flow paths of the flow f' to the terminals in S_c. The flow paths to the new terminals at the end of the edges leaving S_c are combined with the segments of the paths obtained in the first iteration to obtain the desired paths and flow.

It is not hard to show by induction on c that the flow created this way is the fairest binary flow.

Theorem 3.4 *The algorithm given above constructs a fairest binary flow.*

A simple corollary of the construction is the following.

Corollary 3.5 *There are nested cuts S_c for $c = 1, 2, \ldots$, such that $S_c \subseteq S_{c+1}$ for all values of c, $S_c = V$ for a sufficiently large value of c, and the following property holds. In a fairest binary flow, all terminals in S_1 receive 1 unit of flow, and all terminals in $S_{c+1} - S_c$ receive either 2^{-c} or 2^{-c+1} units of flow.*

The Overall Approximation Guarantee

Now consider the problem of finding an approximate fair flow. Our algorithm finds the fairest binary flow. We claim here that this flow satisfies our approximation guarantee.

Theorem 3.6 *The fairest binary flow is in 2-approximate equilibrium.*

Proof. We use Corollary 3.5 for the proof. Suppose a terminal t_i receives 2^{-c} units of flow. We need to prove that we cannot increase the flow to t_i without decreasing the flow to some other terminal t_j that receives at most 2^{-c+1} units of flow. Consider the cut S_c in the Corollary. The terminal t_i is on the sink side of S_c. The cut S_c is saturated, so we cannot increase the flow to t_i without decreasing some other flow across the cut S_c. However, all terminals on the sink side of S_c receive at most 2^{-c+1} units of flow. ■

It is easy to see that the fairest binary flow is a prefix-sum 2-approximation of the fractional fair flow. This fact follows essentially as the fairest binary flow saturates the cuts S_c of Corollary 3.5. We need to use more about Megiddo's optimal fractional flow algorithm to see that the binary flow is in fact a coordinate-wise 2-approximation of the fairest fractional flow.

Theorem 3.7 *The fairest binary flow is a coordinate-wise 2-approximation to the fairest fractional flow.*

Proof. Consider the fairest fractional flow f'. For any value α let T_α denote the set of terminals that receive at least α flow. Megiddo proved the fairest fractional flow is a maximum flow from s to each of the sets T_α simultaneously. This implies the following analogue of Lemma 2.3. For a value α let v_α denote the maximum value of a flow from s to the terminals, where each terminal receives at most α units of flow. If for each terminal t_i that receives some $d_i > \alpha$ flow we delete from f' $d_i - \alpha$ units of flow from s to t_i, then we obtain a flow f'_α of value v_α.

Let $\{S_c\}$ denote the cuts of Corollary 3.5. We get the claimed coordinate-wise 2-approximation if we show that the fairest fractional flow f' must send at most 2^{-c+1} units of flow to every terminal outside of S_c. We prove this by contradiction. Let c be such that some terminal t_i outside of S_c received some $d_i > 2^{-c+1}$ units of flow in f'. Let $\alpha = 2^{-c+1}$, and consider the flow f'_α. By our assumption f'_α does not saturate the cut S_c. The fairest binary flow shows that the maximum flow value v_α is equal to the capacity of the cut S_c plus α times the number of terminals in S_c. However, f'_α does not saturate the cut S_c, and hence has smaller value than v_α. This contradiction proves that f' must have sent at most 2^{-c+1} units of flow to each terminal outside of S_c. ■

4 The NP-Completeness of Fairest Allocation

We formulate here a decision problem associated with computing a fairest routing, and show that it is NP-complete. The reduction is somewhat complicated for the following reason: We are dealing with a single-source flow problem with unit capacities, and to obtain an NP-complete problem here, one typically needs to introduce terminals with different (unsplittable) demand values. Lacking a notion of demand in our problem, we must simulate such demands using the constraints imposed by the equlibrium condition (†).

Theorem 4.1 *The following problem is NP-complete: given a single-source routing problem with a unit-capacity directed graph G, source s, terminals $\{t_i\}$, and an allocation vector r^*, is there a routing of the terminals whose equilibrium allocation vector r satisfies $r^* \preceq r$? (I.e. r is at least as fair as r^*.)*

Proof. The problem is in NP since we can exhibit the paths in such a routing, and in polynomial time compute its equilibrium allocation vector in order to compare it to r^*.

To show NP-hardness, we reduce from a special case of the non-uniform load balancing problem considered by Lenstra, Shmoys, and Tardos [10]. We have a set of jobs $J = \{J_1, \ldots, J_k\}$, and a set of machines $M = \{M_1, \ldots, M_n\}$; for each job J_i, there is a set $S_i \subset M$ on which J_i can be run. Each job J_i has a *requirement* r_i with the property that each r_i is equal to either $\frac{1}{2}$ or 1; we wish to assign each job J_i to a machine in S_i so that the sum of the requirements assigned to each machine is at most 1. Moreover, our instance has the property that $\sum_i r_i = n$ — that is, the total of the requirement values is equal to the number of machines. So the feasibility condition indeed requires that each machine receives either a single job of requirement 1 or two jobs of requirement $\frac{1}{2}$. Let $J' \subseteq J$ denote the jobs of requirement 1, and let $J'' \subseteq J$ denote the jobs of requirement $\frac{1}{2}$; we write $k' = |J'|$ and $k'' = |J''|$, and observe that our condition $\sum_i r_i = n$ can be expressed as $k' + k''/2 = n$.

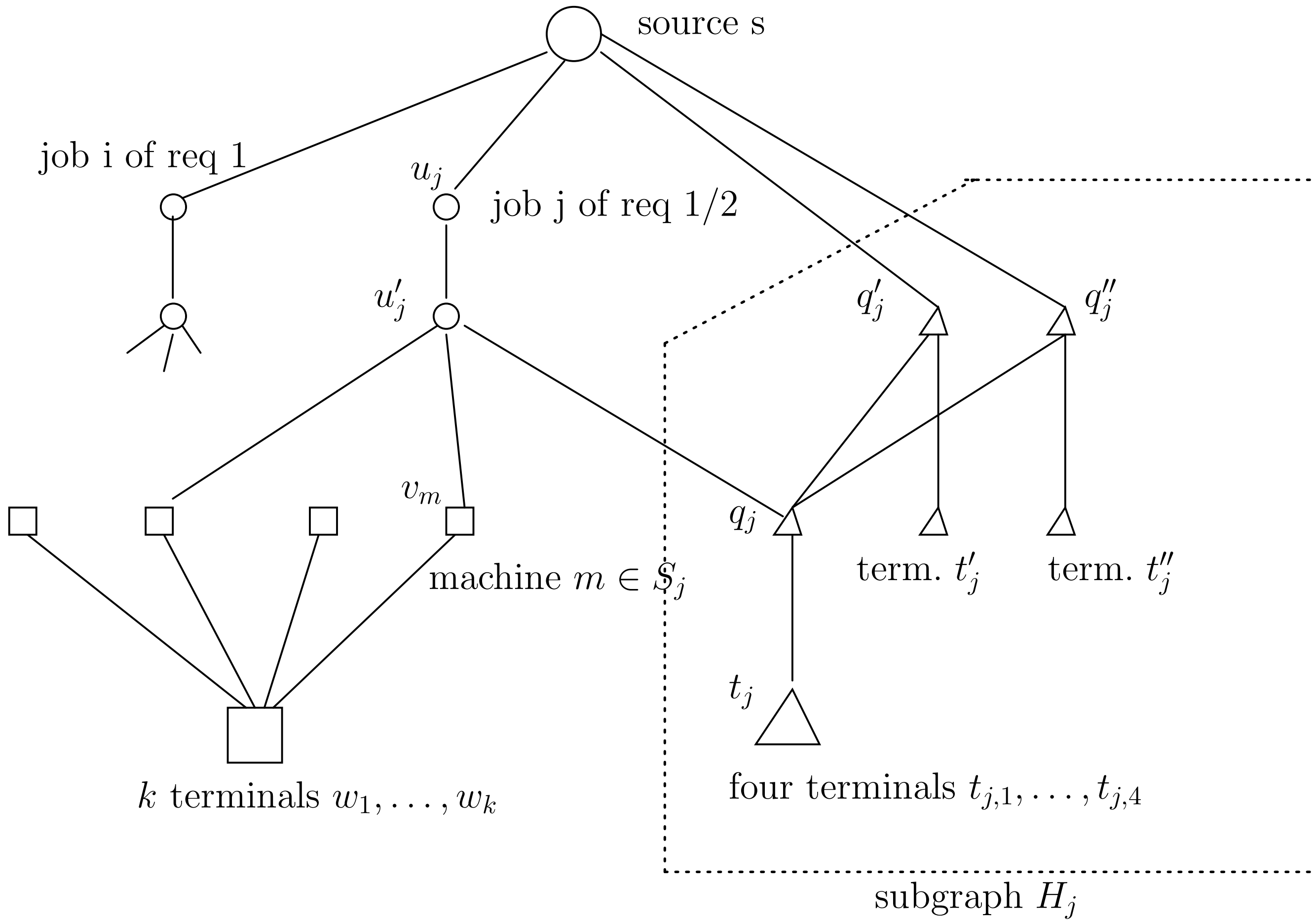

Figure 1: The NP-completeness reduction

We construct the following single-source fair routing problem to encode this decision problem. We refer the reader to Figure 1 for the overall layout of the construction. For simplicity of presentation, we will describe certain nodes as *containing* several terminals; if we wish each terminal to be identified with a distinct node, we can attach each of them via a new degree-one node.

For each job J_j, we create nodes u_j and u'_j, with edges (s, u_j) and (u_j, u'_j). For each machine M_m, we create a node v_m, and edges (u'_j, v_m) for each pair (j, m) such that machine M_m belongs to the set S_j. We also create a single node w that will hold k terminals $w_1, \ldots, w_k$, and add edges (v_m, w) for each k.

This defines the "core" of the construction, through which we encode the condition that job J_j can only be assigned to a machine in S_j. Moreover, if we view this portion of the graph in isolation, we can observe the following: Since there are only n edges entering w, and k terminals at w, we know that the fairest allocation for these terminals would have k' entries equal to 1 and k'' entries equal to $\frac{1}{2}$, as we want. However, we have not yet controlled which "job nodes" get the value $\frac{1}{2}$, and which get 1. This is what we accomplish in the remainder of the construction.

For each job J_j with a requirement of $\frac{1}{2}$, we attach a subgraph H_j containing a total of six terminals $t'_j, t''_j, t_{j,1} \ldots, t_{j,4}$ as shown in Figure 1. We do not create anything additional for the jobs with requirement 1. Thus the complete set of terminals is the set $\{w_i\}$, together with the six terminals from each of the subgraphs H_j.

The subgraph H_j is designed to achieve the following effect. The four terminals at t_j will each get an allocation of 1/4; if one path bound for t_j passes through each of q'_j and q''_j, then each of t'_j and t''_j will get an allocation of 3/4. Finally, two paths bound for t_j can pass through the edge (u_j, u'_j), leaving room for a path bound for a terminal at w to receive an allocation of 1/2. Thus, overall, the fairest allocation will allow a single path to w to get a value of 1/2. Note that for edges (u_j, u'_j) with no subgraph H_j attached, on the other hand, we can have a single path to w with an allocation of 1.

We now make this precise. Define r^* to be a vector consisting of $4k''$ entries equal to 1/4, k'' entries equal to 1/2, $2k''$ entries equal to 3/4, and k' entries equal to 1. If there is a feasible allocation of jobs to machines in the original load balancing problem, then it is easy to construct a routing whose equilibrium allocation r satisfies $r^* \preceq r$. We have a path $\langle s, u_j, u'_j, v_m, w\rangle$ for each job J_j assigned to machine M_m, with an amount of flow equal to the requirement of job J_j. We have two paths $\langle s, u_j, u'_j, q_j, t_j\rangle$, carrying flow 1/4 each, for each job J_j with requirement 1/2; these serve to saturate the edge (u_j, u'_j). The other two terminals on t_j will have paths through q'_j and q''_j respectively, carrying 1/4 units of flow; and the terminals t'_j and t''_j can then receive 3/4 units of flow each on their unique paths from s.

Conversely, suppose there is a routing whose equilibrium allocation r satisfies $r^* \preceq r$. The vector r can only contain at most $4k''$ entries equal to 1/4, so these must be associated with all the terminals of the form $t_{j,i}$. Also, r can only contain at most k'' entries equal to 1/2, so these must be associated with paths that pass through $k''/2$ of the edges into w. All the remaining terminals must get a flow value of at least 3/4 — this therefore consists of of terminals of the form t'_j, t''_j, as well as k' of the terminals at w.

Each terminal of the form t'_j, t''_j must get at least a flow of 3/4, so at most one path bound for t_j can pass through each of q'_j and q''_j. Hence at least two of these paths must pass through the edge (u_j, u'_j). We claim that in fact exactly two of these paths pass through each such edge (u_j, u'_j) (and hence exactly one passes through each of q'_j and q''_j). For suppose that at least three passed through (u_j, u'_j). Then no terminal bound for w could use (u_j, u'_j), and so some edge (u_ℓ, u'_ℓ) would carry two paths bound for w, each with a flow of 1/2; therefore, it would follow that no terminal from t_ℓ could make use of the edge (u_ℓ, u'_ℓ), and this would force more than one path bound for t_ℓ to pass through one of q'_ℓ or q''_ℓ, a contradiction.

Thus, k' of the terminals at w get a flow value equal to 1, so we observe that at least k' of the edges of the form (u_j, u'_j) must carry a single path only. We will call these *pure* edges, and the other edges of the form (u_j, u'_j) *mixed*.

We have therefore established the following two properties of our routing with allocation at least as fair as r^*:

• There are k'' mixed edges of the form (u_j, u'_j); these are associated with indices j for which J_j has requirement 1/2, and on each one, there is a single path bound for w with a flow of 1/2.

• There are $k - k'' = k'$ pure edges of the form (u_j, u'_j); these are associated with indices j for which J_j has requirement 1, and on each one, there is a single path bound for w with a flow of 1.

Hence for each edge (u'_j, v_m) that carries positive flow, we can assign job J_j to machine M_m; this will be a feasible assignment of jobs to machines in the original load balancing problem. ■

References

[1] Y. Afek, Y. Mansour, Z. Ostfeld, "Convergence complexity of optimistic rate based flow control algorithms," *Proc. 28th ACM STOC*, 1996.

[2] D. Bertsekas, R. Gallager, *Data Networks*, Prentice-Hall, 1987.

[3] Y. Dinitz, N. Garg, M. Goemans, "On the single-source unsplittable flow problem," *Proc. 39th IEEE FOCS*, 1998.

[4] D. Gale, L. Shapley, "College Admissions and the Stability of Marriage," *Amer. Math. Monthly* 69(1962).

[5] J. Jaffe, "Bottleneck flow control," *IEEE Trans. Communication*, 29(1981), p.p 954–962.

[6] R.M. Karp, "On the computational complexity of combinatorial problems," *Networks* 5(1975), pp. 45–68.

[7] F. Kelly, "Charging and rate control for elastic traffic," *European Trans. Telecommunications* 8(1997).

[8] J. Kleinberg, "Single-source unsplittable flow," *Proc. 37th IEEE FOCS*, 1996.

[9] S. Kolliopoulos, C. Stein, "Improved approximation algorithms for unsplittable flow problems," *Proc. 38th IEEE FOCS*, 1997.

[10] J.K. Lenstra, D. Shmoys, É. Tardos, "Approximation algorithms for scheduling unrelated parallel machines," *Proc. 28th IEEE FOCS*, 1987.

[11] A. Mayer, Y. Ofek, M. Yung, "Approximating max-min fair rates via distributed local scheduling with partial information," *Proc. IEEE INFOCOM*, 1996.

[12] N. Megiddo, "Optimal flows in networks with sources and sinks," *Math Programming* 7(1974).

[13] S. Plotkin, "Competitive Routing in ATM networks," *IEEE J. Selected Areas in Communications*, 1995.