Simultaneous Optimization via Approximate Majorization for Concave Profits or Convex Costs

Ashish Goel [1] Adam Meyerson [2]

December 2002

CMU-CS-02-203

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

[1]Department of Computer Science, University of Southern California, Los Angeles CA . Research supported in part by an NSF CAREER award 0133968, and the DARPA NMS program contract co. N66001-00-C-8066 (SAMAN). Email: agoel@pollux.usc.edu

[2]Aladdin Project, Carnegie-Mellon University. Research supported by NSF grant CCR-0122581 and ARO grant DAAG55-98-1-0170. Email: adam@cs.cmu.edu

Abstract

For multi-criteria problems and problems with poorly characterized objective functions, it is often desirable to simultaneously approximate the optimum solution for a large class of objective functions. We consider two such classes:

1. Maximizing all symmetric concave functions, and
2. Minimizing all symmetric convex functions.

The former corresponds to maximizing profit for a resource allocation problem (such as allocation of bandwidths in a computer network). The concavity requirement corresponds to the law of diminishing returns in economics. The latter corresponds to minimizing cost or congestion in a load balancing problem, where the congestion-cost is some convex function of the loads.

Informally, a simultaneous α-approximation for either class is a feasible solution that is within a factor α of the optimum for all functions in that class. Clearly, the structure of the feasible set and the constraints have a significant impact on the best possible α and the computational complexity of finding a solution that achieves (or nearly achieves) this α. We develop a framework and a set of techniques to do simultaneous optimization for a wide variety of problems.

We first relate simultaneous α-approximation for both classes to α-approximate majorization in an "if and only if" fashion. Then we prove that α-approximately majorized solutions exist for logarithmic values of α for the concave profits case. For both classes, we present a polynomial time algorithm to find the best α if the set of constraints is a polynomial-sized linear program. We illustrate how standard rounding techniques can be leveraged to extend our framework to several integer programming problems. We also present a dynamic programming algorithm to obtain a PTAS for the optimum α for the convex cost problem, where the constraints correspond to allocating jobs to identical machines. Finally, we demonstrate some interesting connections between distributional optimization and approximate majorization.

Keywords: algorithms, fairness, majorization, load balancing, routing, approximation, linear programming, integer programming

1 Introduction

Simultaneous Optimization: Consider the problem of maximizing $U(x)$ subject to $x \geq 0$ and some additional constraints (such as $Ax \leq C$), where $x = \langle x_1, x_2, \ldots, x_n \rangle$ is an n-dimensional vector and U is an n-variate symmetric concave function. This general scenario captures problems where we need to allocate resources to users in order to maximize the overall profit/utility; here, x is the vector of resource allocations to different users. Concavity of the objective function corresponds to the "law of diminishing returns" from economics. Symmetry corresponds to saying that all users are equally important. 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 will further assume that $U(0) = 0$ and that U is non-decreasing in each argument. These are both natural restrictions for profit/utility functions. We will call such functions *canonical utility functions*. The two simplest examples are $\sum_i x_i$ and $\min\{x_1, x_2, \ldots, x_n\}$.

Now consider the problem of minimizing $C(x)$ subject to $x \geq 0$ and some additional constraints where C is a symmetric convex function such that $C(0) = 0$ and C is non-decreasing in each argument. We will call these functions *canonical congestion functions*. This general scenario captures several load balancing problems; here x is the vector of loads on different machines. Convexity of C is a natural assumption since most natural measures of congestion are convex. Symmetry indicates that all machines are equally important; as before, the constraints, and hence the optimum solution need not be symmetric. Some common canonical congestion functions are $\sum_i x_i$, $\max\{x_1, x_2, \ldots, x_n\}$, $\sum_i x_i^2$ (variance), and $\sum_i \frac{x_i}{1-x_i}$ (the M/M/1 queueing delay function).

These problems can often be solved by using piece-wise linear approximations of the concave or convex function, and then applying techniques for linear objective functions. For example, if the set of constraints are linear, then these problems can often be solved using interior programming techniques [9]. However, we are interested in finding a feasible solution which is a good approximation *simultaneously* for all canonical utility functions or all canonical congestion functions – hence the term "simultaneous optimization". Clearly, the exact quality of the solution achievable depends on the set of constraints. In this paper, we develop a general framework and a set of techniques that are applicable to a wide variety of constraints.

Before proceeding further, we define simultaneous optimization more precisely.

Definition 1 *A feasible vector x is a simultaneous α-approximation for a resource allocation problem if $U(x) \geq U(y)/\alpha$ for all other feasible solutions y and all canonical utility functions U.*

Since convex functions can be arbitrarily steep, an analogous definition would be too strict for resource allocation problems. Instead we define:

Definition 2 *A feasible vector x is a simultaneous α-approximation for a load balancing problem if $C(x/\alpha) \leq C(y)$ for all other feasible solutions y and all canonical congestion functions C.*

In order to understand the motivation behind simultaneous optimization, it helps to think of specific instances. For example, consider the integer multicommodity flow problem, where x_i is the total amount shipped of commodity i. Then a simultaneous α-approximation would be an α-approximation to the integer concurrent flow problem, an α-approximation to the integer maximum flow problem, and an α-approximation to a whole class of similarly interesting utility functions. Rather than list a host of such problems, we now point out the connections between simultaneous optimization and fairness.

Simultaneous optimization and fairness: Consider the problem of designing an algorithm to *fairly* allocate a fixed resource among n individuals given some constraints. This general scenario can be specialized to several routing [14, 7], bandwidth allocation [7, 1], clustering [16], and load balancing [8] problems. To proceed further, we must first decide on a good definition of fairness. This is quite non-trivial, as can be seen by the large number of (often disparate) fairness measures proposed by both system builders [12, 3, 4, 1, 15] and theoreticians (see chapter 13 of [19]). Some of the more widely discussed measures are max-min fairness [1, 2], variance related measures [12], and average utility. 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 maximizing fairness should be equivalent to maximizing some symmetric concave function (for resource allocation problems) or minimizing some symmetric convex function (for load balancing problems). Hence there is a concrete connection between simultaneous optimization and fairness; this connection will influence much of the vocabulary we use in the rest of this paper.

Our results: We use the notion of approximate majorization as defined by Goel, Meyerson, and Plotkin [8]. For a resource allocation problem, the relevant notion of approximate majorization is global α-fairness: a vector x is said to be globally α-fair if for all $1 \leq k \leq n$, the sum of the k smallest components of x is at least $1/\alpha$ times the sum of the k smallest components for any other feasible solution y. For a load balancing problem, the relevant notion is global α-balance: a vector x is said to be globally α-balanced if for all $1 \leq k \leq n$, the sum of the k largest components of x is at most α times the sum of the k largest components for any other feasible solution y. Majorization, approximate majorization, and global α-fairness and global α-balance are defined formally in section 2.

We first establish (section 2) that global α-fairness and global α-balance are exactly equivalent to simultaneous α-approximation for resource allocation and load balancing problems, respectively. This motivates the following questions:

1. **Existence:** Can simultaneous α-approximation be achieved for small values of α? We prove (section 3) that for any resource allocation problem where the set of feasible solutions is a convex set, there exists a globally α-fair solution with α being logarithmic in some natural problem parameters. No such guarantee can exist for resource allocation problems in general; we omit the details.

2. **Algorithms:** Do efficient algorithms exist for finding or approximating the smallest possible α (denoted α^*) for a given resource allocation or load balancing problem such that a simultaneous α-approximation exists? In section 3, we present polynomial time algorithms for finding α^*, as well as the corresponding simultaneous α^*-approximate solution for both resource allocation and load balancing problems, if the set of constraints is a polynomial sized (in n) linear program. Our algorithm can be made particularly efficient (using techniques from [23]) if the set of constraints form a packing or a covering problem. We illustrate how our results can be extended to integer programming problems by considering two important examples. For the integer multicommodity flow (or integer routing) problem, we argue that $\alpha^* = O(logn)$ where n is the number of nodes in the network, and present a polynomial time algorithm to achieve $\alpha = O(\log n)$ using randomized rounding. For the problem of allocating jobs to unrelated $(1-\infty)$ machines, we combine a rounding technique of Shmoys and Tardos [25] with our linear programming techniques to prove $\alpha^* \leq 2$ and to find a solution with $\alpha \leq 2$. Our techniques are also applicable to the facility location and cost-distance problems, but we omit these results for lack of space.

   We also give a PTAS for finding the best α for scheduling on identical machines (section 3.4.3). This algorithm uses dynamic programming techniques, unlike all the above algorithms which used linear programming related techniques.

3. **Connections to other problems:** We then demonstrate a surprising relationship between majorization and the problem of distributional load balancing. In this problem, we must assign jobs to machines in order to minimize the expected total size of jobs on the most-loaded machine. Rather than having fixed, known sizes, the size of each job will be determined independently by some probability distribution *after* the allocation of jobs to machines is made. Several results are known for this problem [13, 6] under various assumptions about the nature of the probability distribution. We show that if the job size distributions are all Poisson or all Binomial, finding a globally α-balanced allocation of the *mean* size leads to an α-approximation to the original problem. We can then apply our linear programming technique to find such α-balanced solutions for the best possible α and apply rounding techniques where necessary. As an intermediate step, we prove that a Poisson random variable with mean $\mu(1+\epsilon)$ is more majorized (intuitively, more sharply concentrated) than $(1+\epsilon)$ times a Poisson random variable of mean μ. We also prove a similar theorem for binomial variables. These theorems may well be of independent interest. Details are in section 4.

Related Work: A systematic study of approximate fairness for combinatorial optimization problems was initiated by Kleinberg, Rabani, and Tardos [14]. They considered a fair allocation to be one which is max-min fair, and an approximately fair allocation as one which is "close" to the max-min fair allocation. They presented an algorithm to find the max-min fair allocation for the problem of allocating identical jobs to unrelated $(1-\infty)$

machines and also gave a 2-approximation to the max-min fair allocation for unsplittable routing using Megiddo's fractional solution [20] as a subroutine. Goel, Meyerson, and Plotkin [7] observed that max-min fairness can result in very bad throughput for network routing and bandwidth allocation. They presented an online algorithm that is globally polylogarithmic-fair for the problem of routing and bandwidth allocation without realizing the connection between their definition of fairness and approximate majorization. Kleinberg and Kumar [16] studied fair allocations for the problems of bandwidth allocation, completion-time scheduling, and facility location; we allude to their results for specific problems in section 3. Goel, Meyerson, and Plotkin [8] were the first to define the notion of approximate majorization. They proved that the greedy online algorithm for allocating identical jobs to unrelated $(1-\infty)$ machines is globally logarithmic-fair.

2 Approximate Majorization and Simultaneous Optimization

For any vector $\vec{x} = < x_1, x_2, \ldots, x_n >$, denote the i-th smallest component of x by $x_{(i)}$. We define $P_j(\vec{x}) = \sum_{i=1}^{j} x_{(i)}$. This is the jth *prefix* of vector $\vec{x}$; the sum of the j smallest coordinates. We define $S_j(\vec{x}) = \sum_{i=1}^{j} x_{(n+1-i)}$. This is the jth *postfix* of vector $\vec{x}$, the sum of the j largest coordinates. We now define majorization.

Definition 3 *Given two n-dimensional vectors x and y, x is said to be majorized by y (y majorizes x) if each of the following is true: (1) $\forall j \leq n$, $P_j(\vec{x}) \geq P_j(\vec{y})$, and (2) $\forall j \leq n$, $S_j(\vec{x}) \leq S_j(\vec{y})$. We use the notation $x \prec y$ to denote the above relation.*

In this paper we will only concern ourselves with vectors where each component is non-negative. Majorization was first studied by Hardy, Littlewood, and Polya [10]. They also proved:

Theorem 2.1 *$x \prec y$ iff $U(x) \geq U(y)$ for all symmetric concave functions U. Equivalently, $x \prec y$ iff $C(x) \leq C(y)$ for all symmetric concave functions C*

that Another interpretation of this relation is that $x \prec y$ if and only if x is more profitable than y for all canonical utility functions (for a resource allocation problem) or that x is less congested than y for all canonical congestion functions (for a load balancing problem).

Definition 4 *A* globally fair solution *is an assignment of resources which produces a vector of allocations to users which is majorized by any other feasible resource allocation.*

If a globally fair solution were to exist for a given problem, this would be the optimum for all canonical utility or congestion functions, as the case may be. This, of course, is too good to be true for most problems. There exist simple problems where a globally fair solution does not exist. For example, the constraints $\{x_1 + x_2 + x_3 = 6; 2x_1 + x_2 \leq 3; x_1, x_2, x_3 \geq 0\}$ do not admit a globally fair solution. We now introduce α-supermajorization which is a slight variation of the notion of supermajorization defined by Hardy, Littlewood, and Polya.

Definition 5 *Given two n-dimensional vectors x and y, x is said to be α-supermajorized by y if $\alpha P_j(\vec{x}) \geq P_j(\vec{y})$ for all $j \leq n$. This is denoted by $x \prec^{\alpha} y$.*

Definition 6 *Given two n-dimensional vectors x and y, x is said to be α-submajorized by y if $S_j(\vec{x}) \leq \alpha S_j(\vec{y})$ for all $j \leq n$. This is denoted by $x \prec_{\alpha} y$.*

Definition 7 *A vector is said to be globally* α-fair *if it is α-supermajorized by any other feasible vector. A vector is said to be globally* α-balanced *if it is α-submajorized by any other feasible vector.*

We will frequently omit the "global" for the sake of brevity. We now establish a concrete, "if and only if" connection between approximate majorization and simultaneous optimization.

Theorem 2.2 *A feasible solution x is a simultaneous α-approximation for a resource allocation problem iff x is globally α-fair.*

Proof: If x is not globally α-fair, then there exists an integer j between 1 and n and a feasible solution y such that $P_j(x) < P_j(y)/\alpha$. But the function P_j satisfies all the requirements for being a canonical utility function, and hence x can not be a simultaneous α-approximation.

Now assume that x is globally α-fair. Let y be any feasible vector and U be any canonical utility function. $P_j(\alpha x) \geq P_j(y)$ for all $1 \leq j \leq n$. Notice that the vector αx may not be feasible, but we are not going to need it to be feasible in our proof. There must exist another vector z such that $z_j \leq \alpha x_j$ for all $1 \leq j \leq n$ and $z \prec y$. By theorem 2.1, $U(z) \geq U(y)$. Since U is non-decreasing, $U(\alpha x) \geq U(z)$. Since U is concave, $U(x) \geq (1/\alpha)U(\alpha x) + (1 - 1/\alpha)U(0)$. Finally, since $U(0) = 0$, we obtain $U(x) \geq (1/\alpha)U(\alpha x)$ which implies that $U(x) \geq (1/\alpha)U(y)$ and completes the proof of the theorem.

■

Theorem 2.3 *A feasible solution x is a simultaneous α-approximation for a load balancing problem iff x is globally α-balanced.*

The proof of theorem 2.3 is similar to that of 2.2 and is omitted from this version. We define $P_j^* = \max_{\vec{x}} P_j(\vec{x})$ and $S_j^* = \min_{\vec{x}} S_j(\vec{x})$ for feasible values of $\vec{x}$ throughout.

3 Majorized Linear Programs

3.1 Algorithm

Many optimization problems can be reduced to linear or integer programs. We present a technique for finding an α-fair or α-balanced solution to any linear program, for the minimum possible value of α. The results of this section are presented for finding α-fair vectors (thus maximizing the resources allocated to each user). Analogous results hold for the case of α-balanced vectors (minimizing the assignment to each user) with appropriate changes in the direction of inequalities.

We need to find a vector $\vec{x}$ such that $\alpha P_j(\vec{x}) \geq P_j^*$ for all j and the minimum possible α. This leads to a simple two step process. We first determine the value of P_j^* for each j, then impose the global fairness constraints and minimize α. Each of these steps can be performed by solving modified linear programs.

Feasibility requires that $A\vec{z} \leq \vec{b}$ for some matrix A, constant vector $\vec{b}$, and variable vector $\vec{z}$ (which includes the x_i but may also include other additional variables). Consider the following linear program for finding P_j^*:

$$\begin{gathered}\text{Maximize } \left(\sum_{i=1}^{n} x_i'\right) - (n-j)U \text{ subject to:}\\ A\vec{z} \leq b\\ x_i' \leq x_i \text{ for all } i \in \{1, \cdots, n\}\\ x_i' \leq U \text{ for all } i \in \{1, \cdots, n\}\end{gathered}$$

The above linear program is polynomial-sized; in fact it adds only $2n$ constraints and $n+1$ variables, so it's essentially the same size as the original linear program and can be solved efficiently.

Lemma 3.1 *The above linear program produces P_j^*.*

Proof: Consider any $\vec{z}$. The vector $\vec{z}$ is feasible (with some choice of x_i' and U) if and only if $\vec{z}$ was feasible for the original linear program. In order to maximize the objective function, it is clear that we will have $x_i' = \min(x_i, U)$. Since we subtract $n - j$ copies of U, it follows that the objective function is at most $P_j(\vec{z})$. On the other hand, if we set $U = x_{(j)}$ then the objective function will be exactly $P_j(\vec{z})$. It follows that the optimum solution to the linear program is in fact P_j^* as desired. ■

We still need to produce the optimum α and the vector $\vec{x}^*$. For that, we write the following linear program.

$$\begin{gathered}\text{Maximize } \gamma \text{ subject to:}\\ A\vec{z} \leq b\\ x_i^j \leq x_i \text{ for all } i \text{ and } j\\ x_i^j \leq U^j \text{ for all } i \text{ and } j\\ \left(\sum_{i=1}^{n} x_i^j\right) - (n-j)U^j \geq \gamma P_j^* \text{ for all } j\end{gathered}$$

This linear program adds n^2+n new variables and $2n^2+n$ constraints. This may make it larger than the original LP, but it is still polynomial in size. The proof of the following lemma is similar to the proof of lemma 3.1 and we omit it.

Lemma 3.2 *The above linear program produces a globally α-fair solution for the minimum value of $\alpha = \frac{1}{\gamma}$.*

3.2 Fast fairness approximations for packing problems

Packing constraints are a set of linear constraints $A\vec{z} \le \vec{b}$ with the additional properties that matrix A has nonnegative entries and vector $\vec{z}$ is constrained to have nonnegative coordinates. Optimization problems on packing constraints occur in a wide variety of settings and can be approximated very efficiently [23]. We would like to be able to apply fast approximation techniques for packing problems to the problem of finding a globally α-fair feasible solution on a set of packing constraints. Unfortunately, the linear programs described in the previous sections are *not* packing programs. Even if the original matrix A is nonnegative, the constraints added to compute fairness do not have the packing property. We can rewrite the linear program we need to solve in order to find prefix P_j^* as follows:

$$\begin{gathered}\text{Minimize } \lambda_j \text{ subject to:}\\ A\vec{z} \le \lambda_j b\\ \textstyle\sum_{i\in S} x_i \ge 1 \text{ for all } S \subseteq \{1,\cdots,n\} \text{ with } |S| = j\end{gathered}$$

Here $P_j^* = 1/\lambda_j$. We will represent the set of equations which insures each subset of j variables sums to at least one by the polytope $\mathcal{P}_j$. Provided we can, for any $\vec{c} \ge 0$, produce the $\vec{x} \in \mathcal{P}_j$ which minimizes $\vec{c} \bullet \vec{x}$, we can run the algorithm of Plotkin, Shmoys, and Tardos [23] to produce a fast approximation to λ_j. The existence of such a polynomial time *minimization oracle* is the subject of the following theorem.

Theorem 3.3 *We can produce $\vec{x} \in \mathcal{P}_j$ to minimize $\vec{c} \bullet \vec{x}$ in time $O(n \log n)$.*

We omit a detailed proof of the theorem. The crucial step is to prove that there exists an optimum solution such that, for some threshold value c_T, all x_i with $c_i > c_T$ are zero and all other x_i are identical. Then we can sort the x_i and check each of the n possible values of c_T to find the optimum solution. This gives a fast algorithm to produce approximately optimum prefixes for each j. We still need to explain how to produce an approximately-fair vector. We can rewrite the linear program to find α^* as follows:

$$\begin{gathered}\text{Minimize } \alpha \text{ subject to:}\\ A\vec{z} \le \alpha b\\ \textstyle\sum_{i\in S} x_i \ge P_{|S|}^* \text{ for all } S \subseteq \{1,\cdots,n\}\end{gathered}$$

We will need a minimization oracle which can, given a vector $\vec{c}$, produce the vector $\vec{x} \in P$ which minimizes $\vec{c} \bullet \vec{x}$. In this case our polytope is the set of vectors $\vec{x}$ which have prefix j at least equal to P_j^* for every j.

Theorem 3.4 *We can produce a polynomial time minimization oracle for the above linear program.*

The minimizing vector $\vec{x}$ has $x_{(1)} = P_1^*$ and $x_{(j+1)} = P_{j+1}^* - P_j^*$. This can be computed in $O(n \log n)$ time by observing that $c_i > c_j \Rightarrow x_i \le x_j$. Thus if the original linear program constraints were packing constraints, we can produce fast globally α-fair vector solutions while guaranteeing $\alpha \le \alpha^*(1+\epsilon)$. Similarly, if the original linear program constraints are covering constraints, we can produce fast globally α-balanced vector solutions.

3.3 Existential upper bounds on α^* for LP solutions

The previous section described methods for finding a globally α-fair vector for a linear program. Similar methods (reversing the signs on certain inequalities) can be used to find α-balanced vectors. There is no guarantee that a solution with $\alpha = 1$ exists, so we will present existential bounds indicating that the minimum α is not too large for resource allocation problems. We assume that all feasible vectors $\vec{x}$ have nonnegative coordinates, and that for any feasible $\vec{x_1}, \vec{x_2}$ the weighted average $\beta\vec{x_1} + (1-\beta)\vec{x_2}$ is feasible for $0 \le \beta \le 1$. We will use the term "nonnegative convex program" to apply to such a set of feasible points. Any linear program with non-negativity constraints on $\vec{x}$ satisfies the above properties. Intuitively, the average of two vectors is only closer to being fair than either of the original vectors. We formalize this idea in the next two lemmas. The convexity assumption implies that the averaged vectors are in fact feasible.

Lemma 3.5 *Given k solutions $\vec{x_1}$ through $\vec{x_k}$, for any j we have $P_j(\frac{1}{k}\sum_{i=1}^{k}\vec{x_i}) \geq \frac{1}{k}\sum_{i=1}^{k}P_j(\vec{x_i})$ and also $S_j(\frac{1}{k}\sum_{i=1}^{k}\vec{x_i}) \leq \frac{1}{k}\sum_{i=1}^{k}S_j(\vec{x_i})$.*

Proof: In determining prefix j of the averaged vector, we will sum j coordinates. Consider these same coordinates in vector $\vec{x_i}$. Since $P_j(\vec{x_i})$ is the sum of the j smallest coordinates, it follows that the sum of these j specific coordinates must be at least $P_j(\vec{x_i})$. This holds for every i, and the actual value of the jth prefix in the average is the average of the sum of these j coordinates in each of the $\vec{x_i}$. The same approach proves the lemma for the S_j. ■

Theorem 3.6 *For any nonnegative convex program, there exists an n-fair solution.*

Proof: We consider the vector which optimizes for each prefix. Take the average of those n vectors. By convexity, this average is feasible. By lemma 3.5 and non-negativity it must have each prefix within $\frac{1}{n}$ of the best possible. Thus it is n-fair. ■

Theorem 3.7 *For any nonnegative convex program, there exists an $O(\log\frac{P_n^*}{nP_1^*})$-fair solution.*

Proof: We will produce a set of vectors to average, such that there are at most $\log(P_n^*/(nP_1^*))$ vectors in the set, and for every prefix P_j one of the vectors we have selected is within a factor of 2 of the optimum. Given such a set, it will follow from lemma 3.5 that the theorem holds.

We construct our set of vectors as follows. We add the vector producing P_1^*. We discard the vectors producing optimum prefix 2, 3, and so forth until we find some i with $P_i^*/i > 2P_1^*$. We then add this vector to our set, and continue until we find some j with $P_j^*/j > 2P_i^*/i$ and so forth. We observe that as j increases, P_j^*/j is nondecreasing. This enables us to bound the total number of vectors as required. ■

The quantity $P_n^*/(nP_1^*)$ may be thought of as the *inherent imbalance* of the problem since it measures the ratio of the maximum utility that we can provide on the average to the maximum utility we can provide to the "poorest" individual. Thus the above result indicates that α^* is at most logarithmic in the inherent imbalance of any non-negative convex program. It is possible to show that no analogue of the above theorem exists for load balancing problems; we omit the details.

3.4 Applications to Integer Programming Problems

3.4.1 Multicommodity Flow and Offline Routing

We consider the problem of multicommodity flow. We are given a set of source-sink pairs $\{s_i, t_i\}$ and asked to route flows from source to sink through a capacitated network. Each pair represents a communication request, and our goal is to provide fair service to all our communicating customers. This problem has been studied in detail and fast polynomial-time algorithms are known for the *optimization version* where we wish to maximize the total communication or maximize the communication for the pair receiving least. The first is known as maximum flow and the second is maximum concurrent flow [24, 17, 5].

We wish to produce the most majorized solution. This is a straightforward application of our techniques for linear programs; we can write a linear program for each prefix and solve it. Since multicommodity flow is a packing problem, we could alternately use the algorithm of [23] and produce a fast $1+\epsilon$ approximation to each prefix.

It is therefore possible to produce an α-fair solution for the minimum α, in polynomial time. But what is this α? We need to show that it cannot be too large. Theorem 3.7 immediately gives us a bound, $O(\log\frac{P_n^*}{nP_1^*})$. However, in some cases the maximum flow might be much more than n times the best flow which can be allocated to the customer receiving least.

We observe that reducing the flow for some pair by one unit cannot enable us to increase the flow of other pairs by more than n units (since each path has at most n edges). Intuitively, this means that flows of very different values cannot have the same bottleneck edges. We formalize this intuition to prove the following theorem, indicating that $\alpha^* \leq O(\log n)$. The proof of the theorem is deferred to appendix A

Theorem 3.8 *For multicommodity flow, there exists an α-fair solution with $\alpha = O(\log n)$.*

We consider the offline version of the routing problem discussed by [7]. We are again given pairs $\{s_i, t_i\}$ and we are asked to connect these pairs. We are constrained in that each pair must be connected by *exactly one path.* We would like to determine the paths and bandwidth allocations such that our solution is as fair as possible. We can simply write a linear program for this problem and apply randomized rounding. We may exceed capacities by $O(\log n)$ so we must scale back the flows and end up with an $O(\log^2 n)$-fair solution.

We can improve this result by *explicitly* summing a subset of the fractional solutions as in the proof of theorem 3.7. This gives a new solution which is $O(1)$-fair when compared to solutions of the original, but exceeds capacity constraints by an $O(\log n)$ factor. We perform randomized rounding on this solution, choosing a random path for each commodity. This selection exceeds the enhanced capacities by only $O(1)$, with high probability (by application of Chernoff bounds). We now scale back the capacities to produce an $O(\log n)$-fair solution for the original problem. This result improves the bandwidth allocation result of [16] and the offline version of [7].

3.4.2 Balancing Non-Identical Jobs on Unrelated Machines

We consider the problem of allocating jobs to unrelated machines in the 1-∞ model. Each job can execute on a subset of the machines and requires the same amount of resources regardless of the machine where it is allocated. We would like to minimize the load on each machine. Of course the total load is always fixed, and algorithms for approximately minimizing the maximum load are known [11]. We approach this problem from the viewpoint of fairness, trying to create an approximately-submajorizing vector of allocations. By reduction from the knapsack problem, finding the smallest α^* is NP-hard.

We can write the fractional relaxation for the integer program version of this problem by allowing a job to be fractionally assigned to different machine. We apply the techniques presented in this section to produce the most-balanced fractional solution. The proof of the following lemma is immediate from the work of [14] and we will omit it.

Lemma 3.9 *The fractional solution for this problem is* 1*-balanced.*

It follows that $P_k^{(F)} \leq P_k^*$ for every prefix k. We employ the approach used by Shmoys and Tardos for the Generalized Assignment Problem [25]. They show how to convert the fractional solution F to a feasible integer allocation $\mathcal{A}$ which satisfies $L_j^{(\mathcal{A})} \leq L_j^{(F)} + r_j$, where r_j is the size of the largest job allocated to machine j by the allocation $\mathcal{A}$. For $1 \leq k \leq m$, let R_k be the sum of the k largest job-sizes in the problem instance. Now, $P_k^{(\mathcal{A})} \leq P_k^{(F)} + R_k$. But $R_k \leq P_k^*$ since in any feasible solution, the sum of the loads of the (at most k) machines that contain the k largest jobs is at least R_k. It follows that $P_k^{(\mathcal{A})} \leq 2P_k^*$ i.e. the algorithm outlined above yields a globally 2-balanced allocation. Further, in appendix C, we show that a 1-balanced allocation need not exist for the integer version of the problem.

Besides being interesting in its own right, this result is also useful as a subroutine for the distributional load balancing problem discussed later in this paper.

Our techniques are also applicable to the fair version of the facility location problem [16] and the cost-distance problem [18, 21]. However, the results are quite technical, and we omit them from this version. Instead, we move on to the problem of assigning non-identical jobs to identical machines, which requires a new and interesting set of techniques.

3.4.3 Non-identical jobs on identical machines

We now consider the problem of allocating non-identical jobs to identical machines. It is easy to see that Graham's rule results in a globally 2-balanced solution for this problem. Our goal is to find a globally α-balanced solution for the smallest possible α, given an instance of this problem. In this paper, we obtain a PTAS for this problem. The PTAS uses several interesting combinatorial properties of global α-balanced solutions, but for reasons of space, we defer the description of the PTAS to appendix D. We have not been able to find an instance of this problem for which no globally 1-balanced solution exists. Finding such a counter-example or proving that a globally 1-balanced solution always exists remains an interesting open problem.

4 Approximate Majorization and Distributional Load Balancing

In the Distributional Load Balancing[1] problem [13, 6], we are given n jobs such that the size of the i-th job is a random variable X_i. The $\{X_i\}$s are independent but not necessarily identical. We are also given m machines. Each job needs to be allocated to exactly one machine. Let the total load on machine j be L_j. Now, L_j is a random variable and so is the maximum load $L = \max_j L_j$. Our goal is to find the allocation that minimizes $\mathbf{E}[L]$. Kleinberg, Rabani, and Tardos [13] gave an $O(1)$-approximation for arbitrary distributions. Goel and Indyk [6] gave a 2-approximation for the case when all job sizes are Poisson and a PTAS when the jobs sizes are exponential. The central theme of this section is the following: *Finding globally α-balanced allocations leads to an α-approximation for several interesting special cases of the distributional load balancing problem.* We will illustrate this theme by concentrating on two special cases:

1. Each X_i is a Poisson random variable (mean μ_i).
2. Each X_i is a binomial random variable with parameters (n_i, q) i.e. X_i is the sum of n_i independent Bernoulli variables, where each Bernoulli variable is 1 with probability q and 0 with probability $1-q$.

First observe that the sum of two independent Poisson variables with means μ_1 and μ_2 is a Poisson variable with mean $\mu_1 + \mu_2$, and the sum of two independent binomial variables with parameters (n_1, q) and (n_2, q) is a binomial variable with parameter $(n_1 + n_2, q)$. The load L_j on any machine is now a Poisson/binomial variable parameterized by the sum of the parameters for the jobs allocated to that machine. Let λ_j represent the mean of the Poisson variable corresponding to the load on machine j (for the Poisson case) and let N_j represent the sum of the parameters n_i for all jobs allocated to machine j (for the binomial case). We are going to find globally α-balanced vectors using the parameters μ_i for the Poisson case and the parameters n_i for the binomial case.

Let $\phi(L_1, L_2, \ldots L_m)$ be any function which is symmetric and convex in its arguments. The function max is symmetric and convex; so is the function $\sum_j f(L_j)$ where f is any convex function. Before we discuss approximate majorization and its relation to distributional load balancing, we state the key theorems that relate distributional load balancing to *exact* majorization:

Theorem 4.1 [19] *If $L_1, L_2, \ldots, L_m$ are Poisson variables with means $\lambda_1, \lambda_2, \ldots, \lambda_m$, and $\phi(L_1, L_2, \ldots, L_m)$ is symmetric and convex, then $\mathbf{E}[\phi(L_1, L_2, \ldots, L_m)]$ is a symmetric and convex function of $\lambda_1, \lambda_2, \ldots, \lambda_m$.*

Theorem 4.2 [19] *If $L_1, L_2, \ldots, L_m$ are binomial variables with parameters $(N_1, q), (N_2, q), \ldots, (N_m, q)$, and $\phi(L_1, L_2, \ldots, L_m)$ is symmetric and convex, then $\mathbf{E}[\phi(L_1, L_2, \ldots, L_m)]$ is a symmetric and convex function of $N_1, N_2, \ldots, N_m$.*

Recall (from section 2) that a most majorized solution $\vec{x}$ minimizes $\phi(\vec{x})$ for any symmetric and convex function ϕ. Hence, if we could find a globally 1-balanced vector for the mean loads $\lambda_1, \ldots, \lambda_m$ (for the Poisson case) or $N_1, \ldots, N_m$ (for the binomial case), we would minimize $\mathbf{E}[\phi(L_1, L_2, \ldots, L_m)]$ for all convex, symmetric functions ϕ *simultaneously.* This is important because any natural measure of fairness must correspond to minimizing a convex function [19]. In particular, we would minimize $\mathbf{E}[\max\{L_j\}]$. Combined with the result of Kleinberg, Rabani, and Tardos [14] that a globally 1-balanced vector exists and can be found for the problem of allocating identical sized jobs to unrelated (1-∞) machines, we have the following corollary:

Corollary 4.3 *Given the problem of allocating i.i.d. Poisson or binomial jobs to unrelated (1-∞) machines, an allocation which simultaneously minimizes $\mathbf{E}[\phi(\vec{L})]$ for all symmetric convex function ϕ can be found in polynomial time.*

[1]This problem was originally called the stochastic load balancing problem. We have decided to rename the problem to make it explicit that we are performing allocation decisions on distributions and not on jobs drawn from given distributions as in most of the textbook examples [22].

As observed earlier, there are several interesting problems where the most majorized vector need not exist. We now relate global α-balance to distributional load balancing. First let us assume that ϕ is monotonic i.e. $\phi(\vec{L})$ does not decrease if we increase any component of the vector $\vec{L}$. Now, let $\mathcal{P}(\mu)$ denote a Poisson variable of mean μ and let $B(n, q)$ denote a binomial variable with parameters (n, q). Further let us say that a real-valued random variable X is majorized by another real-valued random variable Y iff

(1) $\mathbf{E}[X] = \mathbf{E}[Y]$, and (2) For any convex function g, $\mathbf{E}[g(X)] \leq \mathbf{E}[g[Y]]$.

Intuitively, if X is majorized by Y then X is less spread out or more sharply concentrated. We now establish the following results:

Theorem 4.4 [Scaling theorem for Poisson variables] *The random variable $\mathcal{P}(\mu(1+\epsilon))/(1+\epsilon)$ is majorized by $\mathcal{P}(\mu)$ for all $\epsilon > 0$ and $\mu > 0$*

Theorem 4.5 [Scaling theorem for binomial variables] *The random variable $B(M, q) \cdot (N/M)$ is majorized by $B(N, q)$ for all $M > N$.*

The scaling theorems say that if we multiply μ (for Poisson) or N (for binomial) by a factor α, the resulting random variable has a distribution that is "better" or more concentrated than the original random variable multiplied by α. Theorem 4.4 can be derived as a corollary from theorem 4.5 for all rational values of ϵ by viewing a Poisson random variable as a limiting case of a binomial variable; continuity then implies theorem 4.4 for all $\epsilon > 0$. The proof of theorem 4.5 is complex and may well be of independent mathematical interest. Unfortunately, the proof is also quite long and is deferred to appendix B. It is also possible to prove theorem 4.4 directly without invoking theorem 4.5; this alternate proof is also presented in appendix B. Now assume that $\phi(\alpha\vec{L}) \leq f(\alpha)(\vec{L})$ for some monotonically increasing function f and all $\alpha > 1$. If ϕ is chosen to be the max function then $f(\alpha) = \alpha$; if ϕ is the sum of the squares of the components of $\vec{L}$ then $f(\alpha) = \alpha^2$. The proof of the following theorem is deferred to appendix B due to lack of space.

Theorem 4.6 *A globally α-balanced allocation of the means of the given jobs (for the Poisson case) or the parameters n_i (for the binomial case) yields an $f(\alpha)$-approximation for the distributional load balancing problem.*

The following results are now immediate for Poisson or binomial jobs:

1. A 2-approximation if the job sizes are not identical, and if each job may be executed only on a subset of the machines (combined with section 3.4.2).
2. Graham's rule gives a 2-approximation if the machines are identical (extending the result in [6] to include the binomial case).
3. PTAS for the case when each job can go to each machine, but the job sizes are not i.i.d. using ideas from appendix D (but not the result directly since the result guarantees a $1+\epsilon$ approximation to α^*, not $\alpha^* = 1+\epsilon$).

The excellent textbook by Marshall and Olkin [19] presents results such as theorem 4.1 and 4.2 for several other distributions. It may be possible to capture those distributions as well, using the framework described here along with appropriate parameterizations and scaling theorems.

Acknowledgement

Ashish Goel would like to thank David Tse for introducing him to the concept of majorization. We would also like to thank Krishnan Kumaran for helpful discussions, and Bruce Hajek for pointing the shorter (and more elegant) proof of theorem 4.4 which we have sketched in appendix B.

References

[1] Y. Afek, Y. Mansour, and Z. Ostfeld. Convergence complexity of optimistic rate based flow control algorithms. *Proceedings of the 28th ACM Symposium on Theory of Computing*, pages 89–98, 1996.

[2] D. Bertsekas and R. Gallager. *Data Networks.* Prentice-Hall, 1992.

[3] D. Chiu and R. Jain. Analysis of the increase/decrease algorithms for congestion avoidance in computer networks. *Journal of Computer Networks and ISDN*, 17(1):1–14, June 1989.

[4] S. Floyd. Connections with multiple congested gateways in packet-switched networks part 1: One-way traffic. *Computer Communications Review*, 21(5):30–47, October 1991.

[5] 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.

[6] A. Goel and P. Indyk. Stochastic load balancing with exponential jobs. *IEEE Foundations of Computer Science*, 1999.

[7] A. Goel, A. Meyerson, and S. Plotkin. Combining fairness with throughput: Online routing with multiple objectives. *32nd ACM Symposium on Theory of Computing*, 2000.

[8] A. Goel, A. Meyerson, and S. Plotkin. Approximate majorization and fair online load balancing. *To appear in ACM-SIAM Symposium on Discrete Algorithms*, 2001.

[9] M. Grotschel, L. Lovász, and A. Schrijver. *Geometric algorithms and combinatorial optimization.* Springer-Verlag, 1987.

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

[11] D. Hochbaum. *Approximation algorithms for NP-hard problems.* PWS Publishing Company, 1997.

[12] R. Jain, D. Chiu, and W. Hawe. A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. *DEC Research Report TR-301*, September 1984.

[13] J. Kleinberg, Y. Rabani, and E. Tardos. Allocating bandwidth for bursty connections. *Proc. 29th ACM Symposium on Theory of Computing*, 1997.

[14] J. Kleinberg, Y. Rabani, and E. Tardos. Fairness in routing and load balancing. *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, 1999.

[15] C. E. Koksal, H. I. Kassab, and H. Balakrishnan. An analysis of short-term fairness in wireless media access protocols. *ACM SIGMETRIC*, June 2000.

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

[17] T. Leighton, F. Makedon, S. Plotkin, C. Stein, E. Tardos, and S. Tragoudas. Fast approximation algorithms for multicommodity flow problems. *J. Comp. and Syst. Sci.*, 50:228–43, 1995.

[18] M.V. Marathe, R. Ravi, R. Sundaram, S.S. Ravi, D.J. Rosenkrantz, and H.B. Hunt III. Bicriteria network design problems. *Proc. 22nd International Colloquium on Automata, Languages and Programming (ICALP), Szeged, Hungary, LNCS 944*, pages 142–171, July 1995.

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

[20] N. Megiddo. Optimal flows in networks with sources and sinks. *Math Programming*, 1974.

[21] A. Meyerson, K. Munagala, and S. Plotkin. Cost-distance: Two metric network design. *Proc of 41st IEEE Symposium on Foundations of Computer Science*, 2000.

[22] M. Pinedo. *Scheduling: Theory, Algorithms, and Systems.* Prentice Hall, 1995.

[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] F. Shahrokhi and D.W. Matula. The maximum concurrent flow problem. *Journal of the ACM*, 1990.

[25] D. Shmoys and E. Tardos. An approximation algorithm for the generalized assignement problem. *Mathematical Programming*, A 62:461–474, 1993.

A Proof of theorem 3.8

Proof of theorem 3.8: We now prove the following theorem:

For multicommodity flow, there exists an α-fair solution with $\alpha = O(\log n)$.

We can assume that the number of edges and the number of clients are both bounded by the square of the number of nodes. Let n represent the number of nodes. We create many different graphs, with G_i containing those edges with capacity at least n^i. The capacity of an edge in graph G_i is the minimum of its actual capacity and n^{i+8}. Consider prefix P_j with $n^\beta \leq P_j^* \leq n^{\beta+1}$. We will solve for this prefix in the graph $G_{\beta-5}$ (or G_0 if $\beta \leq 5$). The value of this prefix in graph $G_{\beta-5}$ is very close to P_j^*. The removed edges cannot account for more than $n^{\beta-1}$ total units of flow, so they cannot significantly reduce the prefix. On the other hand, the edges whose capacity was decreased have enough capacity to give *every* commodity flow $n^{\beta+1}$ so reducing their capacity cannot reduce the value of the prefix. Since each G_i considers only prefixes in a polynomial range, they each produce $O(\log n)$-fair solutions. We will sum these solutions, but we may exceed capacity. However we observe that an edge has its full capacity in only 9 graphs, and the capacity decreases geometrically elsewhere. Thus the capacities are exceeded by only $O(1)$ and we can scale back to get an $O(\log n)$-fair solution. ■

B Proofs of theorems 4.5, 4.4, and 4.6

Proof of the scaling theorem for Binomial variables (theorem 4.5): We will now prove the following theorem:

The random variable $B(M,q) \cdot (N/M)$ is majorized by $B(N,q)$ for all $M > N$.

It is sufficient to prove the theorem for $M = N+1$; applying this proof inductively will immediately result in a proof for any $M > N$. Let F_X, F_Y be the cumulative distribution functions of two real-valued, non-negative random variables X and Y. Then, the necessary and sufficient condition [19] for $X \prec Y$ is

$$(\forall t,\ 0 \leq t \leq 1) \int_0^t F_X^{-1}(U)dU \geq \int_0^t F_Y^{-1}(U)dU \tag{1}$$

Intuitively, the above equation says that the integral over the space of mass t which contains the smallest values is larger for X than for Y; this is analogous to saying that the sum of the smallest t components of a vector A is larger than the same sum for B.

Let $p_k^n = \binom{n}{k} q^k (1-q)^{n-k}$ represent the probability of $B(n,q)$ being exactly k. Let $P_k^n = \sum_{i=0}^k p_i^n$ represent the probability of $B(n,q)$ being at most k. For any $0 \leq t < 1$, let $I_n(t)$ be the smallest number k such that $P_k^n \geq t$, and let $E_n(t) = \int_0^t I_n(U)dU$. Adapting equation 1 to our scenario, our goal is now to prove that $E_{N+1}(t) \geq E_N(t) \cdot (N+1)/N$. It is sufficient to prove this result for values of t in the set $\{P_k^N, P_k^{N+1}\}$ where $0 \leq k \leq N+1$ since the functions E_N and E_{N+1} are both linear in the intervals defined by these sets of points. The proof will proceed in several steps.

Relating P_k^{N+1} and P_k^N: $\binom{N+1}{i} = \binom{N}{i} + \binom{N}{i-1}$. Therefore $p_i^{N+1} = \binom{N+1}{i} q^i (1-q)^{N+1-i} = (1-q)\binom{N}{i} q^i (1-q)^{N-i} + q\binom{N}{i-1} q^{i-1}(1-q)^{N+1-i} = (1-q)p_i^N + qp_{i-1}^N$. Summing up, we get

$$P_k^{N+1} = P_{k-1}^N + (1-q)p_k^N, \tag{2}$$

which also implies $P_{k-1}^N \leq P_k^{N+1} \leq P_k^N$.

Evaluating E_{N+1} and E_N at P_k^{N+1}:

$$E_{N+1}(P_k^{N+1}) = \sum_{i=0}^k i \binom{N+1}{i} (1-q)^{N+1-i} q^i$$

$$\begin{aligned} &= (1-q)\sum_{i=0}^{k} i\binom{N}{i}(1-q)^{N-i}q^i + q\sum_{i=0}^{k} i\binom{N}{i-1}q^{i-1}(1-q)^{N+1-i} \\ &= \sum_{i=0}^{k-1} i\binom{N}{i}q^i(1-q)^{N-i} + k(1-q)p_k^N + qP_{k-1}^N \end{aligned}$$

But $\sum_{i=0}^{k-1} i\binom{N}{i}q^i(1-q)^{N-i} + k(1-q)p_k^N$ is exactly $E_N(P_k^{N+1})$ (using equation 2). Hence,

$$E_{N+1}(P_k^{N+1}) = E_N(P_k^{N+1}) + qP_{k-1}^N \tag{3}$$

We now inductively prove that $E_N(P_k^{N+1}) = NqP_{k-1}^N$. The base cases $k = 0, 1$ are easy to verify. Assume the above is true for all $k < K$. Now $E_N(P_K^{N+1}) = E_N(P_{K-1}^{N+1}) + (K-1)(P_{K-1}^N - P_{K-1}^{N+1}) + K(P_K^{N+1} - P_{K-1}^N)$. Using the inductive hypothesis and simplifying, $E_N(P_K^{N+1}) = qP_{K-2}^N + (K-1)qp_{K-1}^N + K(1-q)p_K^N$. For our inductive hypothesis to continue to hold, we need to prove that $(K-1)qp_{K-1}^N + K(1-q)p_K^N = Nqp_{K-1}^N$. Now, $(K-1)qp_{K-1}^N + K(1-q)p_K^N = N(\binom{N-1}{K-2}q^K(1-q)^{N+1-K} + \binom{N-1}{K-1}q^K(1-q)^{N+1-K}) = Nq\binom{N}{K-1}q^{K-1}(1-q)^{N+1-K} = Nqp_{K-1}^N$ which completes the proof of $E_N(P_k^{N+1}) = nqP_{k-1}^N$. Plugging this back into equation 3 gives $E_{N+1}(P_k^{N+1}) = E_N(P_k^{N+1}) \cdot (N+1)/N$.

Evaluating E_{N+1} and E_N at P_k^N:

$$\begin{aligned} E_{N+1}(P_k^N) &= E_{N+1}(P_k^{N+1}) + (k+1)qp_k^N \\ &= E_N(P_k^{N+1}) + qP_{k-1}^N + (k+1)qp_k^N \qquad \text{[Using equation 3]} \\ &= E_N(P_{k-1}^N) + k(1-q)p_k^N + qP_{k-1}^N + (k+1)qp_k^N \\ &= E_N(P_{k-1}^N) + kp_k^N + qP_k^N, \end{aligned}$$

But $E_N(P_{k-1}^N) + kp_k^N = E_N(P_k^N)$ which gives

$$E_{N+1}(P_k^N) = E_N(P_k^N) + qP_k^N. \tag{4}$$

The quantity $E_N(P_k^N)/P_k^N$ must increase with k. Since this quantity is exactly nq when $k = n$, we can conclude that $E_N(P_k^N) \leq nqP_k^N$. Plugging this inequality into equation 4 gives $E_{N+1}(P_k^N) \geq E_N(P_k^N) \cdot (N+1)/N$.

This completes the proof of theorem 4.5. ■

Proof (alternate) of the scaling theorem for Poisson variables (theorem 4.4): We now present a shorter, simpler proof of the following theorem, without invoking theorem 4.5 which has a more involved proof:

> The random variable $\mathcal{P}(\mu(1+\epsilon))/(1+\epsilon)$ is majorized by $\mathcal{P}(\mu)$ for all $\epsilon > 0$ and $\mu > 0$.

Let $X_1, X_2, \ldots, X_N$ be i.i.d. variables, with each X_i being 1 with probability p and 0 with probability $1-p$. Further, let $Y_1, Y_2, \ldots, Y_N$ be i.i.d. random variables such that each Y_i is $1+\epsilon$ with probability $\frac{p}{1+\epsilon}$ and 0 otherwise. It is easy to establish that $X_i \prec Y_i$. Thus, for any symmetric, convex, N-variate function g, $\mathbf{E}[g(X_1, \ldots, X_N)] \leq \mathbf{E}[g(Y_1, \ldots, Y_N)]$. For any convex univariate function f, $f(\sum_i X_i)$ is a symmetric, convex, N-variate function of $X_1, \ldots, X_N$. Hence, for any convex function f, $\mathbf{E}[f(\sum_i X_i)] \leq \mathbf{E}[f(\sum_i Y_i)]$. Equivalently, $\sum_{i=1}^N X_i \prec \sum_{i=1}^N Y_i$ for all $N \geq 1$ and all $p > 0$.

A Poisson random variable with mean $\mu(1+\epsilon)$, denoted $\mathcal{P}(\mu(1+\epsilon))$, can be interpreted as $\sum_{i=1}^N X_i$ with $p = (1+\epsilon)\mu/N$ in the limit as N tends to infinity. Similarly, the scaled Poisson random variable $(1+\epsilon)\mathcal{P}(\mu)$ can be interpreted as $\sum_{i=1}^N Y_i$ under the same conditions. The theorem is now immediate. ■

Proof of theorem 4.6:

> A globally α-balanced allocation of the means of the given jobs (for the Poisson case) or the parameters n_i (for the binomial case) yields an $f(\alpha)$-approximation for the distributional load balancing problem.

We will present the proof for binomial variables. Let $\vec{N} = (N_1, N_2, \ldots, N_m)$ be a globally α-balanced allocation of the parameter n_i to machines and let $\vec{N}^*$ be the allocation that minimizes $\mathbf{E}[\phi(\vec{L})]$. Let $\vec{N}/\alpha$ represent the vector obtained by dividing each component[2] in $\vec{N}$ by α. Since $\vec{N}$ is α-balanced, we can find another vector $\vec{N}'$ such that $\vec{N}/\alpha$ is coordinate-wise less than $\vec{N}'$ and $\vec{N}' \prec \vec{N}^*$. We will use $B(\vec{N})$ to denote a vector of independent random Bernoulli variables $(B(N_1, q), B(N_2, q), ldots, B(N_m, q))$.

$$\begin{aligned}
\mathbf{E}[\phi(B(\vec{N}))] &\leq f(\alpha)\mathbf{E}[\phi(B(\vec{N}/\alpha))] && \text{[By theorem 4.5 and convexity of } \phi] \\
&\leq f(\alpha)\mathbf{E}[\phi(B(\vec{N}'))] && \text{[By monotonicity of } \phi] \\
&\leq f(\alpha)\mathbf{E}[\phi(B(\vec{N}^*))] && \text{[Using } \vec{N}' \prec \vec{N}^* \text{ and theorem 4.2]}
\end{aligned}$$

which completes the proof. The proof for the Poisson case is similar and is omitted. ■

C $\alpha^* > 1$ for unrelated machine scheduling problem

For the problem of scheduling non-identical jobs on unrelated (1-∞) machines, a globally 1-balanced solution may not exist. Consider the problem where there are three machines and five jobs. Jobs 1, 2, and 3 have sizes 7, 6, and 5 respectively, and can only go to machines 1, 2, and 3, respectively. Job 4 has size 5 and can go to either of machines 2 and 3. Job 5 has size 3 and can go to either of machines 1 and 3. In this case, $P_1^* = 10$ but the allocation that minimizes P_1 must result in $P_2 \geq 20$ (figure 1. Also, $P_2^* = 19$, but the allocation that minimizes P_2 must have $P_1 \geq 11$ (figure 1). Hence, there does not exist an allocation that can simultaneously minimize P_1 and P_2. Also, by reduction from $0-1$ knapsack, finding the smallest possible α is NP-hard.

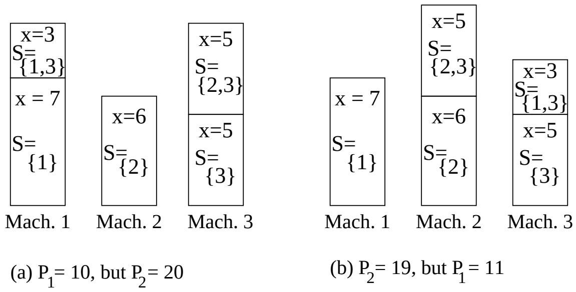

Figure 1: There does not exist an allocation that simultaneously optimizes P_1 and P_2 for this problem instance.

D A PTAS for α^* for identical machine scheduling

We will study the problem of obtaining fair allocations when all machines are identical. We have not yet been able to resolve the question of whether globally 1-balanced allocations exist for this problem. In this section, we sketch a PTAS for approximating α^* for this problem. We will use the term $\mathcal{L}(\mathcal{A})$ to represent the vector of loads on the m machines, given allocation $\mathcal{A}$. Further, define an allocation $\mathcal{A}$ to be *locally unbalanced* if there exists a machine j which has more than one job and the load on this machine is more than twice the load on the least loaded machine. The following lemma will help identify some structure in our candidate optimum solution.

Lemma D.1 *For any locally unbalanced allocation $\mathcal{A}$, there exists another allocation $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}') \prec \mathcal{L}(\mathcal{A})$ but it is not true that $\mathcal{L}(\mathcal{A}) \prec \mathcal{L}(\mathcal{A}')$.*

Proof: Let L_{min} refer to the load on the least loaded machine. Let j be a machine which has two or more jobs and satisfies $L_j > 2L_{min}$. Such a machine is guaranteed to exist since the allocation is locally unbalanced.

[2]The components may be fractional after this division; it is easy to take care of that by rounding components carefully.

We move the smallest job on machine j to the least loaded machine. Let x be the size of this job and let L'_j and L'_{min} be the new loads on these two machines. Since j had two or more jobs, $x \leq L_j/2$ and therefore $L'_j \geq L_j/2 > L_{min}$. Also, $L'_{min} = L_{min} + x \leq L_{min} + L_j/2 < L_j$. We now have $L_{min} < L'_{min} < L_j$ and $L_{min} < L'_j < L_j$. Further, the sum of the load on the two machines did not change. This can only improve all the prefixes (see [8] for a detailed explanation), and hence $\mathcal{L}(\mathcal{A}') \prec \mathcal{L}(\mathcal{A})$. It is easy to verify that at least one prefix gets strictly reduced, and therefore, it is not true that $\mathcal{L}(\mathcal{A}) \prec \mathcal{L}(\mathcal{A}')$. ■

The proof works by showing that a local exchange can result in a more majorized allocation; we omit the details. We can now conclude that any solution which minimizes the prefix P_k or minimizes α has a nice structure: there exists a number $p, 1 \leq p \leq m$ such that

1. The p most loaded machines have exactly one job each, and these p jobs are the largest jobs.
2. All the other machines have at most twice as much load as the least loaded machine.

Whether we are computing P_k^* or α^*, we can guess each of the m possible values of p in turn and pick the one that gave the best result. Given the number p we can immediately assign the p most loaded jobs to the first p machines. We now concentrate on solving the remaining subproblem, for which we have the following guarantee: the most loaded machine has at most twice the load of the least loaded machine. Thus, if $\overline{L}$ is the average load on all the machines, we are guaranteed that $2L_j \geq \overline{L} \geq L_j/2$. An algorithm to compute P_k^* and α up to a $1+\epsilon$ factor accuracy in time $n^{O(1/\epsilon)}$ for this problem is sketched below.

Given any $\delta > 0$ define a job to be small if its size is less than $\delta\overline{L}$. Delete all the small jobs from the problem instance. Then discretize the remaining "big" jobs by increasing the size x_i of a job to the smallest number $x'_i \geq x_i$ such that x'_i is of the form $\delta(1+\delta)^q\overline{L}$ where $q \geq 0$ is an integer. This discretization can affect the optimum prefix by at most a factor of $1+\delta$. Now assume that the load on each machine is also of the form $\delta(1+\delta)^q\overline{L}$. Since all loads lie in the range $[\overline{L}/2, 2\overline{L}]$, the number of possible load values is $O(\log 4/\log(1+\delta)) = O(1/\delta)$. Thus the number of possible load vectors is at most $m^{O(1/\delta)}$. For each load vector, we say that the load vector is achievable if all the big jobs can be packed with the resultant load vector being dominated by the desired load vector. Whether a load vector is achievable or not can be computed using dynamic programming in time $n^{O(1/\delta)}$ (using the fact that due to discretization, the number of possible subsets of jobs is at most $n^{O(1/\delta)}$). Then, for each achievable load vector for the big jobs, pack the small jobs starting from the least loaded machine to make the loads as balanced as possible. Among all the resulting allocations, find the one with the best value of P_k and then, when all the P_k^* are known, the one with the best α.

The resulting values are a $(1+\delta)^c$ approximation for some small constant $c > 1$. Setting $(1+\delta)^c = 1+\epsilon$ gives an algorithm that computes the values P_k^* and α^* to within an accuracy of a $1+\epsilon$ factor and takes time $n^{O(1/\epsilon)}$. Further this algorithm is effective in that it also finds the corresponding allocations which achieve the optimum P_k^* and α^*.