Simultaneous Optimization via Approximate Majorization for Concave Profits or Convex Costs

Ashish Goel [*]
Stanford University

Adam Meyerson [†]
University of California, Los Angeles

September 3, 2004

Abstract

For multi-criteria problems and problems with poorly characterized objective, 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 first class 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 second class 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 has 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 perform simultaneous optimization for a wide variety of problems.

We first relate simultaneous α-approximation for both classes to α-approximate majorization. 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 and discuss several non-trivial applications. These applications include finding a $(\log n)$-majorized solution for multicommodity flow, and finding approximately best α for various forms of load balancing problems. Our techniques can also be applied to produce approximately fair versions of the facility location and bi-criteria network design problems. In addition, we demonstrate interesting connections between distributional load balancing (where the sizes of jobs are drawn from known probability distributions but the actual size is not known at the time of placement) and approximate majorization.

[*]Department of Management Science and Engineering and (by courtesy) Computer Science, Stanford University. Research supported in part by an NSF CAREER award 0133968, a Terman Engineering Fellowship, and an Alfred P. Sloan Faculty Fellowship. Email: ashishg@stanford.edu

[†]Department of Computer Science, University of California, Los Angeles. Research supported by NSF grant CCR-0122581 and ARO grant DAAG55-98-1-0170. Email: awm@cs.ucla.edu

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 in polynomial time (for example, using the ellipsoid algorithm [12]). 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 1.1 *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 1.2 *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 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 concurrent

flow problem, an α-approximation to the 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 [19, 11], bandwidth allocation [11, 1], clustering [21], and load balancing [10] 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 [17, 6, 7, 1, 20] and theoreticians (see chapter 13 of [25]). Some of the more widely discussed measures are max-min fairness [1, 4], variance related measures [17], 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 notation of approximate majorization as defined by Goel, Meyerson, and Plotkin [10]. 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, 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.

   For the special case of the multicommodity flow problem, we prove the existence of a solution with $\alpha = O(\log n)$ where n is the number of nodes in the network.

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 (in n) sized linear program. Algorithms for computing least majorized elements in different settings were previously known [34, 26, 27, 33]. However, unlike previous work, we use recent fractional packing techniques [30] to obtain particularly

efficient algorithms when the set of constraints form a packing or a covering problem.

We extend our framework to several integer programming problems. For the problem of allocating jobs to unrelated $(1-\infty)$ machines, we combine a rounding technique of Shmoys and Tardos [32] with our linear programming techniques to prove $\alpha^* \leq 2$ and to find a solution with $\alpha \leq 2$. We then give a PTAS for finding the best α for scheduling on identical machines (section 3.4.2). This algorithm uses dynamic programming techniques, unlike all the above algorithms which used linear programming related techniques. Our techniques are also applicable to the facility location and cost-distance problems; details are in sections 3.4.3 and 3.4.4.

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 [18, 9] 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 [19]. 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 [26] as a subroutine. Goel, Meyerson, and Plotkin [11] 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 majorization. Kleinberg and Kumar [21] 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 [10] made explicit the connection between the prefix-sum approximations used earlier [11, 21] and majorization. They also proved that the greedy online algorithm for allocating identical jobs to unrelated $(1-\infty)$ machines is globally logarithmic-fair. Recent work by Azar *et al.* [3] gave a 2-fair solution for load balancing non-identical jobs on unrelated machines via the ellipsoid algorithm for linear programming; we independently arrived at an equivalent result using more general methods.

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 *suffix* of vector $\vec{x}$, the sum of the j largest coordinates. We now define majorization.

Definition 2.1 *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 [13]. 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 convex functions C.*

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 2.2 *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 2.3 *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 2.4 *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 2.5 *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 term "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: First, let's assume that x is a simultaneous α-approximation. It follows that for any canonical utility function U, we have $\alpha U(x) \geq U(y)$ for any feasible vector y. In particular, we observe that the jth prefix $U = P_j$ is a canonical utility function. Thus for any feasible y, we have for all j, $\alpha P_j(x) \geq P_j(y)$ and thus x is globally α-fair.

Now suppose x is globally α-fair. This means that for any feasible allocation y, we have $x \prec^{\alpha} y$, i.e., $\alpha P_j(x) \geq P_j(y)$. Observing that $P_j(x)$ preserves scalar multipliers, we obtain $P_j(\alpha x) \geq P_j(y)$. We now imagine decreasing the largest coordinate(s) of αx simultaneously until some prefix becomes equal to the corresponding prefix in y. The first such prefix will be $P_n(x)$. Call this new vector z. Once this inequality reaches equality, we can always guarantee that since $S_j(z) = P_n(z) - P_{n-j}(z)$, we will have both required properties and we conclude that $z \prec y$. Applying theorem 2.1 and observing that U is increasing and all coordinates of z are only smaller than those of αx, we have $U(\alpha x) \geq U(z) \geq U(y)$ for any canonical utility function U. Since U is concave, we have $U(x) \geq (1/\alpha)U(\alpha x) + (1 - 1/\alpha)U(0)$ and since $U(0) = 0$ we can conclude that $\alpha U(x) \geq U(y)$ for any utility function U and feasible y. Thus x is a simultaneous α-approximation.

■

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

Proof: First, let's assume that x is a simultaneous α-approximation. In terms for load balancing, this means that for any canonical congestion function C and feasible solution y, we have $C(x/\alpha) \leq C(y)$. In particular, take $C = S_j$. We observe that the suffixes are convex as required. It follows that for each j, we have $S_j(x/\alpha) \leq S_j(y)$ and thus $S_j(x) \leq \alpha S_j(y)$ and $x \prec_{\alpha} y$. So x is globally α-balanced.

Now suppose x is globally α-balanced. It follows that for any y, we have $x \prec_{\alpha} y$. This is equivalent to saying $S_j(x) \leq \alpha S_j(y)$ for all j. But then $S_j(x/\alpha) \leq S_j(y)$ for all j. We now consider increasing the smallest coordinate(s) of x/α until the final suffix becomes equal to that of y. This process preserves the suffix inequalities, and once the last suffix is equal we can guarantee the prefix properties as well and thus $z \prec y$. This enables us to conclude that $C(x/\alpha) \leq C(z) \leq C(y)$ and thus x is a simultaneous α-approximation. ■

Definition 2.6 *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 $\vec{x}$.*

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^*:

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

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.

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

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.

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

Proof: Any feasible $\vec{z}$ must be feasible for the original linear program. In order to maximize γ, we will set x_i^j to be as large as possible. It follows that $x_i^j = \min(x_i, U^j)$. The last inequality can be regrouped to sum the j smallest x_i^j plus the difference of the larger x_i^j and U^j. Since we always have $x_i^j \leq U^j$, the

way to maximize this sum will be to set U^j to be the $(j+1)$-th smallest term, so in order to maximize γ we will pick the appropriate U^j and guarantee $P_j \geq \gamma P_j^*$ as desired. So the added constraints guarantee the optimum value of α^*. ■

If the constraints are not linear but instead form a convex set, we can still solve the resulting programs efficiently if we are given an efficient separation oracle [12].

3.2 Fast fairness approximations for packing problems

Packing constraints are a set of linear constraints $A\vec{z} \leq \vec{b}$ 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 [30]. We would like 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:

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

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 $\vec{c} \geq 0$, 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 [30] 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 $O(n \log n)$.*

Proof: Consider the optimum $\vec{x}$. We immediately observe that $x_{(i)} = x_{(i+1)}$ for all $i \geq j$. Suppose for some $i < j$ we have $x_{(i)} > 0$ and $x_{(i)} < x_{(i+1)}$. If we increase $x_{(i)}$ by a small ϵ and decrease $x_{(i')}$ by $\epsilon/(j-i)$ for all $i' > i$ then we will remain in the polytope (since the order of the x_i and the sum of the jth prefix are unchanged). On the other hand, if we decrease $x_{(i)}$ by the same ϵ and increase $x_{(i')}$ by $\epsilon/(j-i)$ for all $i' > i$ then we again remain in the polytope. Since one of these two directions does not make the solution worse, we can conclude that for every i, either $x_{(i)} = 0$ or $x_{(i)} = x_{(i+1)}$. In order to be in the polytope, if the smallest i coordinates of $\vec{x}$ are zero, the remaining coordinates will all have to be at least $1/(j-i)$. We can sort the x_i in decreasing order of their cost c_i; since swapping the values of the coordinates of $\vec{x}$ cannot cause it to leave the polytope, it is clear that the largest c_i will correspond to the smallest x_i and so forth. Thus there are only j possible vectors $\vec{x} \in P$ which could minimize $\vec{c} \bullet \vec{x}$ and we can produce them all (and check their costs) in polynomial time. ■

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:

Minimize α subject to:

$$A\vec{z} \leq \alpha b$$

$$\sum_{i \in S} x_i \geq P^*_{|S|} \text{ for all } S \subseteq \{1, \cdots, n\}$$

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.*

Proof: 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 $c_i > c_j \Rightarrow x_i \leq x_j$. ■

Thus if the original linear program constraints are packing constraints, we can produce fast globally α-fair vector solutions while guaranteeing $\alpha \leq \alpha^*(1+\epsilon)$. 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 convex programs

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 $0 \leq \beta \leq 1$. 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: Let $\vec{z}$ denote the vector $\frac{1}{k}\sum_{i=1}^k \vec{x_i}$. Let $q_1, q_2, \ldots q_j$ denote the indices of the j smallest coordinates of $\vec{z}$, i.e. $\vec{z}_{(p)} = \vec{z}_{q_p}$ for $p \leq j$. Since $P_j(\vec{x_i})$ is the sum of the j smallest coordinates of $\vec{x_i}$, it follows that

$$\sum_{i=1}^{k} \sum_{p=1}^{j} (\vec{x_i})_{q_p} \geq \sum_{i=1}^{k} P_j(\vec{x_i}).$$

The LHS of the above expression is exactly $kP_j(\vec{z})$, concluding the proof of the lemma for the prefixes P_j. The same approach proves the lemma for the suffixes S_j. ■

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

Proof: We consider the n vectors which optimize 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; an example illustrating this is given in section 3.4.3.

3.3.1 Multicommodity Flow

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 [31, 22, 8].

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 [30] 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, $O(\log \frac{P_n^*}{nP_1^*})$. 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 will now formalize this intuition to prove a stronger bound of $O(\log n)$ on α^*.

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

Proof: 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 down to get an $O(\log n)$-fair solution. ■

3.4 Applications to Integer Programming Problems

3.4.1 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 [14]. 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; in fact a result of Lenstra *et al.* [23] shows that finding a $\frac{3}{2}$ approximation to the smallest α is 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 [19].

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 [32]. 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. In general a 1-balanced allocation need not exist for the integer version of the problem [2]. A comparable algorithm for non-identical jobs and unrelated machines was discovered independently by Azar *et al.* [3]; however their techniques depends upon using the ellipsoid method to solve an exponentially large linear program (we can use our packing/covering result to produce a faster solution).

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.

3.4.2 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. We will now present a PTAS for this problem. The idea of a polynomial-time approximation scheme was introduced by Hochbaum and Shmoys [15]; we will use techniques similar to theirs but will also take advantage of several interesting combinatorial properties of global α-balanced solutions. As in the case of unrelated machines, there might not exist a globally 1-balanced solution; Alon *et al.*[2] give such an example.

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 i 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 a candidate optimum solution.

Lemma 3.10 *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 i be a machine which has two or more jobs and satisfies $L_i > 2L_{min}$. We move the smallest job on machine i to the least loaded machine. Let x be the size of this job and let L'_i and L'_{min} be the new loads on these two machines. Since machine i had two or more jobs, $x \leq L_i/2$ and therefore $L'_i \geq L_i/2 > L_{min}$. Also, $L'_{min} = L_{min} + x \leq L_{min} + L_i/2 < L_i$. We now have $L_{min} < L'_{min} < L_i$ and $L_{min} < L'_i < L_i$. Further, the sum of the load on the two machines did not change. We can now write $\mathcal{L}(\mathcal{A}') = D\mathcal{L}(\mathcal{A})$ for some doubly stochastic matrix D. But $x \prec y$ iff $x = Dy$ for some doubly stochastic matrix D [25]. Hence, $\mathcal{L}(\mathcal{A}') \prec \mathcal{L}(\mathcal{A})$, completing the first half of our proof.

The number of machines with load more than L_i is the same in both $\mathcal{A}$ and $\mathcal{A}'$. But $\mathcal{A}'$ has one fewer machine with load L_i. Hence, at least one suffix in $\mathcal{L}(\mathcal{A})'$ is strictly smaller than the corresponding suffix in $\mathcal{L}(\mathcal{A})$, contradicting $\mathcal{L}(\mathcal{A}) \prec \mathcal{L}(\mathcal{A}')$. ■

Assume that x_j refers to the size of the j-th smallest job.

Definition 3.1 *For $1 \leq p \leq m$, let $\overline{y}_p$ denote the quantity $\frac{\sum_{j \leq n-p} x_j}{m-p}$. A p-balanced allocation is an assignment of jobs to machines such that*

1. *the p largest jobs are allocated to machines $m-p+1, m-p+2, \ldots, m$, one job to each machine, and*
2. *for $1 \leq i \leq m-p$, the load on the i-th machine is in the range $(\overline{y}_p/2, 2\overline{y}_p)$, and*
3. *the loads on the first $m-p$ machines are non-decreasing.*

Lemma 3.10 implies that for any $1 \leq k \leq m$, there exists a p-balanced allocation which minimizes the suffix S_k. Further, there exists a p-balanced allocation which minimizes α. Of course the allocations (and the corresponding p values) which minimize different suffixes and/or α need not be the same. We will exploit this structure to derive a dynamic programming based PTAS. We first define a discrete load vector.

Definition 3.2 *A (δ, p)-discrete load vector is an $(m-p)$-dimensional vector where the components of the vector are non-decreasing, and each component is*

1. *of the form $(\overline{y}_p/2)(1+\delta)^k$ for some integer $k > 0$, and*
2. *less than $2\overline{y}_p(1+\delta)^2$.*

Definition 3.3 *A (δ, p)-discrete load vector Y is said to be feasible if there exists a p-balanced allocation with load vector L such that $Y_i \geq L_i$ for $1 \leq i \leq m-p$.*

Definition 3.4 *A (δ, p)-discrete load vector Y is said to be slack if it is feasible and there exists a p-balanced allocation with load vector L such that $Y_i \geq L_i(1+\delta)$ for $1 \leq i \leq m-p$.*

Definition 3.5 *Given a fixed constant $\delta \in (0,1)$, a collection $\mathcal{Z} = \langle Z_1, Z_2, Z_3, \ldots, Z_m \rangle$ of sets of vectors is said to be δ-comprehensive if for each p, the set Z_p satisfies the following properties:*

1. *All vectors in Z_p are (δ, p)-discrete load vectors, and*
2. *All vectors in Z_p are feasible, and*
3. *All slack (δ, p)-discrete load vectors are in Z_p.*

If we are given a δ-comprehensive collection $\mathcal{Z}$, we can consider each set Z_p in $\mathcal{Z}$, then consider each (δ, p)-discrete load vector $Y \in Z_p$, augment Y to obtain an m-dimensional vector Y' by introducing $m-p$ new components corresponding to the p largest jobs, and then use the vector Y' to estimate S_k. Let S'_k denote the minimum value of the suffix S_k obtained using this method. The definition of a δ-comprehensive collection guarantees that $S^*_k \leq S'_k \leq S^*_k(1+\delta)^2$. A similar guarantee holds for α^*. Hence, in order to obtain a PTAS for S^*_k or α^*, it suffices to obtain an algorithm that generates a δ-comprehensive collection for any fixed $\delta \in (0,1)$ in time that is polynomial in n and m. That is the subject of the next theorem.

Theorem 3.11 *Given a fixed constant $\delta \in (0,1)$, it is possible to generate a δ-comprehensive collection in time $n^{O(\frac{1}{\delta}\log\frac{1}{\delta})}$.*

Proof: Focus on any one value of p and assume without loss of generality that $1 \leq p < m < n$. In any (δ, p)-discrete load vector Y, let m_k denote the number of components that are equal to $(\overline{y}_p/2)(1+\delta)^k$. Clearly, $0 \leq m_k \leq m-p < m$. Further, specifying the sequence $m_1, m_2, m_3 \ldots$ completely specifies Y. The largest value of k for which m_k is non-zero can be at most $\log_{1+\delta}(4(1+\delta)^2) = O(1/\delta)$. Hence, all (δ, p)-discrete load vectors can be enumerated in time $m^{O(1/\delta)}$.

We will now outline a dynamic program that takes a (δ, p)-discrete load vector Y as input, and in time $(mn)^{O(\frac{1}{\delta}\log\frac{1}{\delta})}$, answers YES or NO. Further, this program will always answer NO if Y is not feasible, and always answer YES if Y is slack. Using this dynamic program as a test coupled with the enumeration process for (δ, p)-discrete load vectors yields an effective proof of this theorem.

Let $D(x)$ represent the function $\left(\overline{y}_p/2\right)(1+\delta)^{\left\lceil \log_{1+\delta}\frac{x}{\overline{y}_p/2}\right\rceil}$. Construct a new set of jobs with sizes x'_j, $1 \leq j \leq n-p$, where

$$x'_j = \begin{cases} 0 \text{ if } x_j \leq \delta\overline{y}_p, \text{ and} \\ D(x_j) \text{ otherwise.} \end{cases}$$

Jobs with $x'_i = 0$ will be called "small". Let T represent the multi-set of the new job sizes, excluding small jobs. No one job can be larger than $2\overline{y}_p$ and the smallest value in T is at least $\delta\overline{y}_p$. Hence, the number of job sizes is at most $\lceil \log_{1+\delta} 2/\delta \rceil = O((1/\delta)\log(1/\delta))$. Hence the number of distinct sub-multisets of T, is $n^{O((1/\delta)\log(1/\delta))}$; further these multisets can be enumerated in time $n^{O((1/\delta)\log(1/\delta))}$. For any sub-multiset U of T, let $s'(U)$ denote the sum of the elements in U. For any $i, 1 \leq i \leq m-p$, define the indicator

$$I_{U,i} = \begin{cases} 1, \text{ if it is possible to assign jobs from } U \text{ to machines } i, i+1, \ldots m-p \text{ such} \\ \qquad \text{that the load on machine } k, i \leq k \leq m-p \text{ is at most } Y_k, \text{ and} \\ 0 \text{ otherwise.} \end{cases}$$

We extend the definition of I to $i = m-p+1$. In this case, define $I_{U,m-p+1} = 1$ if U is empty and $I_{U,m-p+1} = 0$ otherwise. The following self-explanatory recurrence applies to $I_{U,i}$ for $1 \leq i \leq m-p$:

$$I_{U,i} = \max\{I_{U-V',i+1} : V' \subseteq U, s'(V') \leq Y_i\}.$$

If $I_{U,i} = 1$ then let $V'_{U,i}$ refer to a maximizer of the expression in the RHS.

The straightforward dynamic program to compute the quantities $I_{U,i}$ and $V'_{U,i}$ for all $U \subseteq T$ and all $i, 1 \leq i \leq m-p$, has table size $mn^{O((1/\delta)\log(1/\delta))}$. Computing each entry takes time $n^{O((1/\delta)\log(1/\delta))}$.

If the value $I_{T,1}$ is 0, then the dynamic program answers NO. If the value $I_{T,1}$ is 1, then consider the vector $\langle V'_1, V_2, \ldots, V'_{m-p}\rangle$ defined as follows: $V'_1 = V'_{T,1}$ and for $1 < i \leq m-p$, $V'_i = V'_{U-V'_1-V'_2-\ldots V'_{i-1},i}$. The vector $\langle V'_1, \ldots V'_{m-p}\rangle$ represents the allocation used by the dynamic program. The dynamic program allocates jobs with sizes x'_i and excludes small jobs. Before the dynamic program can answer YES, these assumptions must be removed and a feasible schedule constructed with the original job sizes. For each job-size $x' > 0$ used by the dynamic program, consider the set of jobs $J(x') = \{j : D(x_j) = x'\}$. If V'_i has k jobs of size x', then assign k arbitrarily chosen jobs from $J(x')$ to machine i. Let V_i represent the new allocation of jobs to machines. To allocate the small jobs, the algorithm keeps adding small jobs arbitrarily to any machine from 1 to $m-p$, while ensuring that the load on machine i does not exceed Y_i. If the allocation of small jobs succeeds, the dynamic program answers YES. If it fails, the dynamic program answers NO.

If the dynamic programming test described above answers YES, then the input (δ, p)-discrete load vector Y is clearly feasible, since a feasible allocation has been found. If Y is slack, then there exists a p-balanced allocation such that the the load L_i on machine i, $1 \leq i \leq m-p$, satisfies $L_i(1+\delta) \leq Y_i$. Since $D(x) < x(1+\delta)$, there exists a feasible allocation of jobs with sizes x'_j to machines 1 to $m-p$ without exceeding the loads Y_i. Hence the dynamic program will find such an allocation, and the value $I_{T,1}$ would be 1. While allocating the small jobs, the dynamic program will fail (i.e. answer NO) only if a small job can not be allocated to any of the machines $1, \ldots, m-p$ without exceeding the loads Y_i. This implies that the load on machine i, $1 \leq i \leq m-p$, must already be greater than $Y_i - \delta\overline{y}_p$. Hence the total allocated load on the first $m-p$ machines is already greater than $\sum_i Y_i - \delta(m-p)\overline{y}_p = \sum_i Y_i - \delta\sum_i L_i \geq \sum_i L_i(1+\delta) - \delta\sum_i L_i \geq \sum_i L_i$. But $\sum_i L_i$ is the total size of all the jobs that need to be allocated, and hence, all small jobs must have been accommodated.

Since $m < n$, the dynamic programming test described above, combined with the enumeration process for all $\delta, p)$-discrete load vectors, generates a comprehensive collection in time $n^{O((1/\delta)\log(1/\delta))}$. ■

The above method is clearly effective, since for each load vector in the comprehensive collection, we

have a feasible allocation. Hence, the above PTAS for S_k^* and α^* also produces a corresponding $(1+O(\delta))$-approximate allocation.

3.4.3 Facility Location and K-Median

We consider the fair version of the facility location problem, first discussed by Kumar and Kleinberg [21]. We are given some cost bound C and asked to produce a set of facilities with total purchase cost at most C such that the distances to these facilities are balanced among users. Notice that in this case we are considering balancing; each customer wants a *small* distance to the nearest facility.

Since this is a balancing (minimization) problem, theorem 3.7 does not apply. It could be the case that no solution with a small value of α exists. For example, consider a metric space with three points a, b, c where $d(a,b) = 1$ and $d(a,c) = d(b,c) = \sqrt{n}$. We place $\frac{n-1}{2}$ demand points at each of a and b, and one point at c. Suppose we are permitted two facilities. The solution which minimizes the maximum distance (the first suffix) places the facilities at a and c for a maximum distance of 1. The solution which minimizes the total distance (nth suffix) places centers at a and b for an total distance of $\sqrt{n}$. Since there are only three possible integral solutions, it is straightforward to see that any solution has $\alpha \geq \sqrt{n}$. In fact, even if we permit fractional facilities, this bound still holds. This sort of example was previously observed by [21].

It follows that in order to obtain a good solution in terms of α, we will have to exceed our given bound on the number of facilities. The goal will be to minimize the amount by which we exceed the cost bound C while obtaining a small α.

We consider writing the appropriate integer programs for each suffix. We can solve the linear relaxation of these programs to obtain fractional assignments of points to facilities. Observing that the gap between the k-center and k-median solutions (these are the extremal suffixes) is at most a factor of n, we can classify the suffixes by the value of S_i^*/i and select only $\log n$ of them (this strategy closely resembles that used in the proof of theorem 3.7). Averaging these solutions together will not work, but we can instead take the union of the facilities selected. This gives a solution with $\alpha = O(1)$ but exceeds the facility cost by a factor of $O(\log n)$. We can now perform rounding as in the results of [16] to obtain an integer solution with the same properties. Note that exceeding the cost constraint is necessary (this is a result of [21]).

The above allows us to extend previous work to the capacitated version of the problem and to facilities with nonuniform costs, while also demonstrating that the result of [21] can be reached as a direct result of our general framework.

3.4.4 Cost-Distance Network Design

Consider the problem of designing a network. We are given a set of clients which we need to connect to a single server (or any of several identical servers). We purchase various types of connective cable. A cable between two locations has some installation cost and some latency; many different types of cables may exist and the cost and latency of the different cable types may depend upon the pair of nodes being connected. We would like to purchase a network with cost at most some given constant C while providing the best possible quality-of-service (in terms of latency) to the clients. This type of network design problem was introduced by Marathe *et al.* [24], and a bi-criterion approximation of $O(\log n, \log n)$ was given on the cost

constraint and the maximum (worst-case) latency of any user. Meyerson, Munagala, and Plotkin [28] gave a bi-criterion $O(\log n, \log n)$ approximation for the version in which we wish to optimize for *average* latency of the clients. We will consider designing a network which must *balance* the latency of the clients. Chekuri, Khanna, and Naor [5] derandomized the algorithm of [28] through linear program rounding. We can solve their linear program with additional constraints to produce optimum fractional solutions for each prefix. We observe that this problem is more general than the balanced facility location problem described earlier (the reduction proceeds by building a graph with zero latency edges connecting "facility" nodes to the server and zero cost edges connecting demand points to facility nodes), so once again we will be forced to exceed the cost bound C.

Our algorithm once again proceeds by solving the linear programs for each suffix, then applying the rounding algorithm of Chekuri *et al.*. We note that this procedure can be performed on *any* linear program solution, to obtain a result within $O(\log n, \log n)$ of this fractional solution. In addition, their routine guarantees that the latency from any client to the server will not exceed $O(\log n)$ times its fractional latency (this follows from the assignment of the lower-fractional-latency client as the "center" in each stage of the rounding). It follows that this yields a solution for each suffix which is within $O(\log n)$ of the optimum prefix values while exceeding the cost C by at most $O(\log n)$. We now select $\log n$ of these suffixes by using the same scaling technique from the facility location problem (choose a new suffix each time the ratio of value to number of terms decreases by half). By taking the union of the edges from these solutions, we produce an $O(\log n)$-balanced solution with cost at most $O(\log^2 n)$ times C.

4 Approximate Majorization and Distributional Load Balancing

In the Distributional Load Balancing[1] problem [18, 9], 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 [18] gave an $O(1)$-approximation for arbitrary distributions. Goel and Indyk [9] 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)

[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 [29].

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 [25] *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 [25] *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 [25]. In particular, we would minimize $\mathbf{E}[\max\{L_j\}]$. Combined with the result of Kleinberg, Rabani, and Tardos [19] 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.*

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 $\mu > 0$*

Proof: 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. ■

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. The proof of theorem 4.5 is complex and may well be of independent mathematical interest[2]. Unfortunately, the proof is quite technical and is deferred to appendix A.

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 $f(\alpha) = \alpha^2$.

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.*

Proof: 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[3] 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 $(B(N_1,q), B(N_2,q), ldots, B(N_m,q))$.

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

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

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 3.4.1).

[2] In fact, theorem 4.4 can be obtained as a corollary of theorem 4.5 by treating a Poisson variable as the limit of a Bernoulli variable.

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

2. Graham's rule gives a 2-approximation if the machines are identical (extending the result in [9] to include the binomial case).
3. PTAS for the case when each job can go to each machine and the job sizes are independent but not identically distributed, using the dynamic program in section 3.4.2.

The excellent textbook by Marshall and Olkin [25] presents results such as theorem 4.1 and 4.2 for several other distributions. It is an interesting open problem to determine whether scaling theorems along the lines of theorems 4.4 and 4.5 also exist for some of these other distributions, making them amenable to our framework.

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 out the shorter (and more elegant) proof of theorem 4.4 which we have presented.

References

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

[2] N. Alon, Y. Azar, G. Woeginger, and T. Yadid. Approximation schemes for scheduling. *Proceedings of the Eighth ACM-SIAM Symposium on Discrete Algorithms*, pages 493–500, 1997.

[3] Y. Azar, L. Epstein, Y. Richter, and G. Woeginger. All-norm approximation algorithms. *Journal of Algorithms*, 52(2):120–133, 2004.

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

[5] C. Chekuri, S. Khanna, and S. Naor. A deterministic algorithm for the cost-distance problem. *Proc. 12th Annual Symposium on Discrete Algorithms*, 2001.

[6] 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.

[7] 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.

[8] 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.

[9] A. Goel and P. Indyk. Stochastic load balancing with exponential jobs. *IEEE Foundations of Computer Science*, pages 769–778, Oct 1999.

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

[11] A. Goel, A. Meyerson, and S. Plotkin. Combining fairness with throughput: Online routing with multiple objectives. *Journal of Computer and Systems Sciences*, 63(1):62–79, 2001. A preliminary version appeared in ACM Symposium on Theory of Computing, 2000.

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

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

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

[15] D. Hochbaum and D. Shmoys. Using dual approximation algorithms for scheduling problems: Theoretical and practical results. *Journal of the ACM*, pages 144–162, 1987.

[16] K. Jain and V. Vazirani. Primal-dual approximation algorithms for metric facility location and k-median problems. *Proceedings of the Twenty-Ninth Annual IEEE Symposium on Foundations of Computer Science*, 1999.

[17] 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.

[18] J. Kleinberg, Y. Rabani, and E. Tardos. Allocating bandwidth for bursty connections. *SIAM J. Comput.*, 30(1):191–217, 2000.

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

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

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

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

[23] J. Lenstra, D. Shmoys, and E. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Math Programming*, pages 259–271, 1990.

[24] 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.

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

[26] N. Megiddo. Optimal flows in networks with sources and sinks. *Math Programming*, 7(3):97–107, 1974.

[27] N. Megiddo. A good algorithm for lexicographically optimal flow in multiterminal networks. *Bull Amer. Math. Soc.*, pages 407–409, 1977.

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

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

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

[31] F. Shahrokhi and D.W. Matula. The maximum concurrent flow problem. *Journal of the ACM*, 37(2):318–334, 1990.

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

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

[34] A. Veinott. Least d-majorized network flows with inventory and statistical applications. *Management Sci.*, 17(9):547–567, 1971.

A Proof of theorem 4.5

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 [25] for $X \prec Y$ is

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

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 $0 \le t < 1$, let $I_n(t)$ be the smallest number k such that $P_k^n \ge t$, 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 $E_{N+1}(t) \ge E_N(t)\cdot(N+1)/N$. It is sufficient to prove this result for values of t in the set $\{P_k^N, P_k^{N+1}\}$ where $0 \le k \le N+1$ 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 $P_{k-1}^N \le P_k^{N+1} \le P_k^N$.

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

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

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. ■