MATHEMATICS OF OPERATIONS RESEARCH
Vol. 19, No. 2, May 1994
Printed in U.S.A.

LOWER AND UPPER BOUNDS FOR THE ALLOCATION PROBLEM AND OTHER NONLINEAR OPTIMIZATION PROBLEMS

DORIT S. HOCHBAUM

We demonstrate the impossibility of strongly polynomial algorithms for the allocation problem, in the comparison model and in the algebraic tree computation model, that follow from lower bound results. Consequently, there are no strongly polynomial algorithms for nonlinear (concave) separable optimization over a totally unimodular constraint matrix. This is in contrast to the case when the objective is linear.

We present scaling-based algorithms that use a greedy algorithm as a subroutine. The algorithms are polynomial for the allocation problem and its extensions and are also optimal for the simple allocation problem and the generalized upper bounds allocation problem, in that the complexity meets the lower bound derived from the comparison model. For other extensions of the allocation problem the scaling-based algorithms presented here are the fastest known.

These algorithms are also polynomial time algorithms for solving with ϵ accuracy the allocation problem and its extension in continuous variables.

1. Introduction. We consider in this paper the separable concave *simple allocation* problem,

$$\text{Max}\left\{ \sum_{i=1}^{n} f_i(x_i) \,\middle|\, \sum_{i=1}^{n} x_i = B,\ x \geq 0 \right\},$$

and its extensions to problems with $\mathbf{x}$ satisfying in addition general polymatroidal constraints. We call the problem over polymatroidal constraints, the *general allocation problem*. These problems are studied for $\mathbf{x}$ either an integer or continuous vector. Other than the concavity of the f_i's, no additional assumptions are made.

The allocation problem and its extensions have been studied extensively. A recently published book by Ibaraki and Katoh (1988), gives a comprehensive review of the state-of-the-art of the problem, as well as more than 150 references. There are numerous applications of these problems in the literature of capital budgeting, portfolio planning, production planning and more. Several such applications are presented in §2. We study here in detail, in addition to the general allocation problem and the simple allocation problem, also the cases of the *generalized upper bounds* problem, the *nested* problem and the *tree* problem. These problems are defined formally in §2.

The integer version of the general allocation problem is characterized as solvable via a greedy allocation procedure. The value of each variable is incremented by one unit at a time if the corresponding increase in the objective function is largest among

Received December 15, 1989; revised October 9, 1992.
AMS 1980 subject classification. Primary: 90C25, 90C60, 68Q25. Secondary: 90C10, 90C45.
IAOR 1973 subject classification. Main: Programming/Algorithms. Cross references: Allocation/Resources.
OR/MS Index 1978 subject classification. Primary: 656 Programming/Nonlinear/Convex.
Key words. Submodular, lower bounds, strong polynomiality, nonlinear optimization.

0364-765X/94/1902/0390/$01.25

all possible increments that are feasible, until all B units are allocated. This greedy algorithm was first devised by Gross (1956) and later by Fox (1966) for the integer simple allocation problem. The difficulty with this algorithm is that it requires $O(B)$ iterative steps, each including at least one evaluation of the objective function. Since the representation of B in the input takes only $\log_2 B$ bits, the running time of the greedy is exponential (or pseudo-polynomial) in the length of B, and hence exponential in the input length.

In order to characterize the complexity of algorithms for the general allocation problem, one needs to know the form of the representation of the objective function, and the complexity of evaluating those functions' values. Since the functions f_i are assumed to be general, the existence of an oracle computing the functions' values is assumed. An oracle call is counted as a single operation in the complexity model employed here. With this complexity model, the running time of the greedy algorithm for the general allocation problem is $O(B[\log n + F])$, where F is the number of operations required to check the feasibility of a given increment in the solution.

This paper contains a number of results concerning the allocation problems and other nonlinear optimization problems over linear constraints. For the general allocation problem and its cases we devise a general purpose algorithm from which upper bounds on the complexity of these problems are established. We also demonstrate lower bounds for the simple allocation problem which has implications on the concrete complexity of constrained nonlinear optimization.

The key to the general purpose algorithm is a proximity theorem. The essence of this theorem is that the greedy algorithm can be applied to these problems with arbitrary increments, rather than unit integer increments, until no such increments are possible. The proximity theorem shows, that only the last increment of each variable can be potentially erroneous, and if removed we get a valid lower bound to the integer optimal solution. This process of scaled greedy increments, can then be repeated with smaller increments. By careful choice of scaling of increments we can demonstrate polynomial running time for all allocation problems. No proximity of this type has been observed before for optimization problems over polymatroidal constraints. The only result that bears some similarity is by Ibaraki and Katoh (1988, pp. 72–74) who prove for the simple allocation problem with continuously differentiable objective, that the value of an optimal integer solution is bounded from above by the ceiling of an optimal continuous solution. This result, even for nondifferentiable objective functions and for all allocation problems, is a corollary of our proximity theorem.

The results given in this paper establish constructively, that the general integer and continuous allocation problems are solvable in *polynomial* time, requiring $O(\log(B/n))$ iterations. This polynomial algorithm is developed using concepts of scaling and proximity between the optimal solutions of two scaled problems. The running time of our algorithm for the general integer allocation problem is

$$O(n(\log n + F)\log(B/n)),$$

and for the continuous case an ϵ-accurate solution (to be defined subsequently) is produced in $O(n(\log n + F)\log(B/\epsilon n))$ steps.

For the general allocation problem and its special cases, the algorithms derived here are the fastest known. For the general allocation problem Groenevelt (1985) established polynomiality, yet gave no explicit complexity expression. (It relies on the separation algorithm for which explicit complexity has not been established.) For the simple allocation problem the fastest algorithm is Frederickson and Johnson's (1982) algorithm, $O(n \log(B/n))$, which is also optimal with respect to the comparison

complexity model. Our algorithm uses a subroutine of Frederickson and Johnson, yet its overall structure is simpler than Frederickson and Johnson's algorithm, while achieving the same running time. This optimal running time is also established for the generalized upper bounds case. This case, that comes up in financial portfolio planning, could be solved as a special case of the tree problem (it corresponds to a star of diameter 2). Our algorithm consists of substantial improvement over this running time. The nested problem is also a special case of a tree problem where the tree is a path. For the nested problem some notable algorithms include that of Galperin and Waksman's (1981) of complexity $O(B \log n)$, Tamir's (1980) of complexity $O(n^2 \log^2 B)$, and Dyer and Walker's (1987) of complexity $O(n \log n \log^2(B/n))$. For the tree problem there is an algorithm with running time $O(n^2 \log(B/n))$ by Ibaraki and Katoh (1988), and the recent most efficient algorithm by Dyer and Frieze (1990) with running time $O(n \log^2 n \log B)$. This latter algorithm uses as a subroutine the algorithm of Frederickson and Johnson to solve a number of simple allocation problems. Our algorithm, when applied to the nested allocation problem and the tree allocation problem, run in time $O(n \log n \log(B/n))$, which is faster than any known algorithm for these problems and is a factor of $\log n$ off the comparison model lower bound.

Another topic presented here is lower bond proofs. Dyer and Frieze say in (1990), "...we do not address the potentially interesting issue of strong polynomial complexity of algorithms for this problem. ...If we were to make appropriate assumptions about the form of the f_i in the general case, it would seem we could obtain strongly polynomial algorithms, but we do not discuss this further here." This implied conjecture, that strongly polynomial algorithms exist for these problems, is refuted here. We show that the dependence on $\log(B/\epsilon)$ in the running time of the algorithm cannot be removed even for the simple case of the allocation problem and hence for the general problem as well (with the possible exception of quadratic objective function). The lower bound result holds for both the comparison model and the algebraic computation tree model with the four arithmetic operations $+, -, \cdot, :$, and comparisons. It holds even if the floor operation is permitted. Since the simple allocation problem is a special case of separable concave maximization problems over totally unimodular constraint matrices or over polymatroidal constraints, then in particular all allocation problems and nonlinear convex network flow problems cannot be solved in strongly polynomial time.

This issue of strong polynomiality is of special interest, as it clearly delineates the distinction in the complexity of nonlinear versus linear problems. For linear objective function, the integer programming problem over totally unimodular constraint matrix is solvable in strongly polynomial time, (Tardos 1986), whereas the separable concave version of this problem is solvable in time which depends on the logarithm of the right-hand side (Hochbaum and Shanthikumar 1990). As a conclusion from the lower bound result given here, the dependence on the right-hand sides cannot be removed.

Our results not only improve on existing algorithms for the general allocation (integer) problem and its special cases, but also provide polynomial algorithms for the problem in *continuous* variables. The continuous general allocation problem has only been solved to data in special cases when the derivatives of the f_i's exist. For example, the "ranking algorithm" suggested by Zipkin (1980) for the allocation problem, requires the existence of the derivatives as well as a solution to a system of nonlinear equations. Similar difficulties are encountered in the algorithms proposed by Yao and Shanthikumar (1987). In contrast, we do not require here the existence of derivatives.

It is not surprising that these difficulties are encountered in the continuous version of the (general) allocation problem since the solution could require infinite representation. For example, let a simple allocation problem be given with $f_1(x_1) = 6x_1 - x_1^3$,

$f_2(x_2) = 0$, and $B = 2$. The optimal solution to the 2 variable allocation problem is $(\sqrt{2}, 2 - \sqrt{2})$, which is irrational. The solution may even lack an algebraic representation. Hence, solving a system of nonlinear equations is a challenging problem even when the nonlinearities are as simple as polynomials. We therefore use the practical notion of *ϵ-accurate* solution to represent the continuous solution to the problem. A solution, $\mathbf{x}^{(\epsilon)}$ is *ϵ-accurate* if there exists an optimal solution $\mathbf{x}^*$ such that $\|\mathbf{x}^{(\epsilon)} - \mathbf{x}^*\|_\infty \leq \epsilon$. That is, ϵ is the accuracy required in the solution space.

Using the proximity results, we show that ϵ-accurate solutions can be obtained by solving a general integer allocation problem obtained by scaling the continuous general allocation problem by a factor of ϵ/n. Hence, the continuous problem is reduced to the integer case, and similar algorithms to those used in the integer case apply to the continuous case. The lower bound results mentioned earlier, establish that the dependence on B/ϵ cannot be removed.

The plan of the paper is as follows. Section 2 defines the general allocation problem, its special cases and several applications. The lower bounds for the comparison model and the algebraic computation tree model are given in §3. Section 4 includes the proximity theorem between the scaled solution and the optimal solution and its consequence regarding the proximity of optimal integer and continuous solutions. This theorem validates the scaling algorithm. Section 5 gives the general algorithm, and in §6 there are adaptations of the general algorithm for the special cases. Finally, §7 has concluding remarks and some open questions.

2. Preliminaries, notation and applications. Given a *submodular* rank function $r: 2^E \to R$, for $E = \{1, \ldots, n\}$, i.e., $r(\phi) = 0$ and for all $A, B \subset E$,

$$r(A) + r(B) \geq r(A \cup B) + r(A \cap B).$$

(for a detailed description of submodular functions see e.g. Nemhauser and Wolsey (1988). The *polymatroid* defined by the rank function r, is the polytope $\{\mathbf{x} | \sum_{j \in A} x_j \leq r(A),\ A \subseteq E\}$.

We call the system of inequalities $\{\sum_{j \in A} x_j \leq r(A),\ A \subseteq E\}$, the *polymatroidal constraints*.

As for the notation in this paper, bold letters are used to denote vectors; $\mathbf{e}$ is the vector $(1, 1, \ldots, 1)$; $\mathbf{e}^j$ is the unit vector with $e^j_j = 1$ and $e^j_l = 0$ for $i \neq j$; all logarithms are base 2 logarithms. The general allocation problem, GAP, is

$$\text{(GAP)} \qquad \begin{aligned} &\max \sum_{j \in E} f_j(x_j), \\ &\sum_{j \in E} x_j = B, \\ &\sum_{j \in A} x_j \leq r(A), \qquad A \subset E \\ &x_j \geq l_j \quad \text{and integer } j \in E. \end{aligned}$$

For $B \leq r(E)$, the problem (GAP) has a feasible (and optimal) solution. The problem is given here with general lower bounds l_j rather than nonnegativity requirements as is common in the literature. From properties of polymatroid, any solution that satisfies all constraints other than the equality constraint (a member of the polymatroid), is a lower bound vector to a feasible solution that satisfies the equality

constraint as well. It is well known that the greedy algorithm solves the problem (GAP) (e.g., Federgruen and Greenevelt 1986a, Ibaraki and Katoh 1988). An important concept of the greedy algorithm is that of an *increment*.

The jth increment at x_j is defined as $\Delta_j(x_j) = f_j(x_j + 1) - f_j(x_j)$. The greedy for GAP is formally described as follows.

Procedure greedy

Input $(\mathbf{l}, r, E)$

Step 0: $\mathbf{x} = \mathbf{l}$, $B \leftarrow B - \mathbf{l} \cdot \mathbf{e}$.

Step 1: Find i such that $\Delta_i(x_i) = \max_{j \in E}\{\Delta_j(x_j)\}$.

Step 2: (Feasibility check), is $\mathbf{x} + e^i$ infeasible?

If yes, $E \leftarrow E - \{i\}$,

else $x_i \leftarrow x_i + 1$

$B \leftarrow B - 1$.

Step 3: If $B = 0$ stop. Output $\mathbf{x}$. If $E = \phi$, stop, output "no feasible solution." Otherwise go to step 1.

Note that the feasibility check is polynomial for the general allocation problem (using the ellipsoid method, (Grötchel, Lovász and Schrijver 1981), and can be performed trivially in a single step for the simple allocation problem. Note also that the output of "no feasible solution" in Step 3 never materializes if r is indeed a submodular function, $B \leq r(E)$, and the choice of the vector $\mathbf{l}$ is such that it does not violate the polymatroidal constraints (that is, $\mathbf{l}$ is a member of the polymatroid).

The *greedy* algorithm is applicable to special cases of GAP. These cases are generated by restricting the set of the polymatroidal constraints to specially structured collection of subsets of E, rather than the entire power set. Important special cases that have been studied in the literature are:

(1) *The simple resource allocation* problem (SRA):

$$\text{(SRA)} \qquad \max \sum_{j=1}^{n} f_j(x_j),$$

$$\sum_{j=1}^{n} x_j = B,$$

$$x_j, \quad \text{nonegative integers } j = 1, \ldots, n.$$

The earliest explicit investigations of SRA are due to Koopman (1953). One early application is the distribution of search effort problem for which a discrete version is discussed by Charnes and Cooper (1958), and a continuous version by Koopman (1957). In the discrete version, an object may be located in one of n given locations. There is a known probability for the object to be in position j, p_j. The probability of detecting the object in position j depends on the amount of effort allocated to the search in position j, x_j. This probability is $(1 - e^{-\alpha x_j})p_j$, for a positive constant α. For a total amount of effort b, the problem is SRA with the objective function,

$$\max \sum_{j=1}^{n} (1 - e^{-\alpha x_j}) p_j.$$

As this function is concave and separable, the problem is an instance of SRA. Another interesting application is described by Luss and Gupta (1975) for a problem of allocating marketing effort.

(2) The *generalized upper bounds* resource allocation problem (GUB):

$$\text{(GUB)} \qquad \max \sum_{j=1}^{n} f_j(x_j),$$

$$\sum_{j=1}^{n} x_j = B,$$

$$\sum_{j \in S_i} x_j \leq b_i, \quad i = 1, \ldots, m,$$

$$x_j, \quad \text{nonnegative integers } j = 1, \ldots, n$$

where $\{S_1, S_2, \ldots, S_m\}$ is a partition of $E = \{1, \ldots, n\}$, i.e., disjoint sets the union of which is E.

The GUB problem has an application to optimal portfolio selection of securities with consideration of risk factors. Here there are n securities representing potential investment opportunity. The securities are sorted in classes according to risk level associated with the investment. This risk level is the estimated standard deviation on the return. The objective is to minimize the ratio of expected return to variance. It was shown (see Elton, Gruber and Podberg 1976 and Zipkin 1980) that such a problem can be written, when correlation coefficients are constant, as a separable convex minimization problem on variables $y_j = \sigma_j x_j$ for σ_j the standard deviation, and x_j the proportion invested in security j. That problem is SRA in nonnegative continuous variables, except for the existence of one additional variable that is unrestricted in sign. That variable represents the weighted sum of risks of the chosen investment. The problem is then solved as an allocation problem with the weighted sum of risks treated as a fixed parameter. It is further desirable that the proportion of securities of each risk level σ_i, S_i, does not exceed a certain prescribed proportion p_i. The additional constraints are then

$$\sum_{j \in S_i} \sigma_i x_j \leq p_i, \qquad i = 1, \ldots, m.$$

Dividing through the ith constraint by σ_i gives a GUB allocation problem formulation over continuous variables.

(3) The *nested* resource allocation problem:

$$\max \sum_{j=1}^{n} f_j(x_j),$$

$$\sum_{j=1}^{n} x_j = B,$$

$$\sum_{j \in S_i} x_j \leq b_i, \qquad i = 1, \ldots, m,$$

$$x_j, \quad \text{nonnegative integers } j = 1, \ldots, n$$

where $S_1 \subset S_2 \subset \cdots \subset S_m \subset E$. Notice that $b_1 \leq \cdots \leq b_m$, otherwise, if $b_i > b_{i+1}$, then the ith constraint is redundant and can be omitted.

Tamir (1980) considered a production problem application. It is given that the horizon is finite, $[0, T]$, and a restriction that product units can be sold only at m prescribed times $0 < t_1 < t_2 < \cdots < t_m = T$. The utility of selling k items at time t_j is a monotone nondecreasing concave function $f_j(k)$. The production rate is constant, with a units of time required to produce one unit of the product, and there is no initial stock or inventory holding cost. This problem is formulated as a problem concave maximization problem with the constraints:

$$a \sum_{j=1}^{n} x_j \leq t_i, \qquad i = 1, \ldots, n,$$

for x_j nonnegative integers. This problem is the nested problem once we divide through each inequality by a.

Another unrelated application is in determining optimal sample sizes for the purpose of estimation. The *stratified sampling problem* is an instance of the nested problem discussed by Sanathanan (1970).

(4) The *tree* resource allocation problem:

$$\max \sum_{j} f_j(x_j),$$

$$\sum_{j=1}^{n} x_j = B,$$

$$\sum_{j \in S_i} x_j \leq b_i, \qquad i = 1, \ldots, m,$$

$$x_j, \quad \text{nonnegative integers } j = 1, \ldots, n.$$

The sets S_i are derived by some hierarchical decomposition of E into disjoint subsets and the repeated decomposition of each of the subsets. Each set thus generated is among the sets S_i $i = 1, \ldots, m$. Describing each set as a node and the decomposition as edges from the parent set to its subsets, one gets a tree on m nodes which is a branching, i.e., the indegree of each node except the root (corresponding to the set E), is one.

Note that the nested problem is a special case of the tree problem when the tree is a path. Also the generalized upper bounds is a special case of the tree problem with all nodes other then the root being leaves; that is, the tree is a star.

One application of this problem is in the context of product storage and transportation. A company needs to buy B units of a product from up to n different suppliers that are geographically dispersed. The utility function for possessing k units for the product from supplier j is $f_j(k)$, where all f_j's are concave monotone nondecreasing functions. The product from the suppliers is routed to local warehouses, where transportation capacity is limited and warehouse capacity is limited. From local warehouses the product is transported to regional warehouses and from there to distribution centers. Each product may be transported to the nearest warehouse facility, but the routes are predetermined. The capacity of each warehouse and center is bounded by the minimum of its own capacity and the capacity of the routes leading to the warehouse. So for a warehouse i, the set of products assigned to be shipped via that warehouse is S_i. For a total number of m warehouses and centers the constraint

is,

$$\sum_{j \in S_i} x_j \leq C_i, \qquad i = 1, \ldots, m,$$

where C_i is the capacity of warehouse i.

A variation of this problem with n different items and objective function representing the lot ordering cost and holding cost, an *optimal lot sizing problem*, is presented by Ziegler (1982) and Bitran and Hax (1981). There the objective function is a minimization of a separable convex function. In the constraints each variable x_j appears with a coefficient v_j that represents the amount of storage required by a unit of that product. In order to derive the allocation problem form of constraints the variables are scaled by the factor v_j.

Many researchers have discussed other cases of the general allocation problem. One such case is the *network* allocation problem that is not defined here. The tree problem is a special instance of the network problem when the network forms a tree. Federgruen and Groenevelt (1986b)) showed that the *network* allocation problem is a special case of GAP, and presented an application to an oil and gas bidding problem. In this application, the network is a bipartite graph. An adaptation of the proximity-scaling algorithm gives faster running time (than that of the greedy described in Federgruen and Groenevelt (1986b)) to the oil and gas bidding problem, where the factor F for checking the feasibility of an increment is equivalent to testing whether in a given residual network an increment of a unit of flow from one of the sources is feasible. For a network on m arcs this can be trivially implemented in $O(m)$. More efficient implementations are potentially possible.

The greedy is valid for all these problems, also with **1** replacing **0** as a lower bound (for proof see Federgruen and Groenevelt (1986a) or Ibaraki and Katoh (1988)).

3. Lower bounds. In contrast to combinatorial linear programming (with zero-one coefficients in the constraint matrix) where there is a strongly polynomial algorithm to solve the problem (see Tardos (1986)), this is not the case when we introduce a concave objective function, even if it is separable and consists only of two (or one) variables, and even if the maximum subdeterminant is 1 (i.e., the constraint matrix is totally unimodular), and even if there is only one constraint in addition to nonnegativity. We first present a comparison model lower bound followed by an algebraic tree model lower bound.

3.1. *A comparison model lower bound.* Here, we rely on a result of information theory according to which there is no algorithm that finds a value in a monotonic decreasing n-array that is the first to be smaller than some specified constant in less than $\log_2 n$ comparisons.

Consider the allocation problem defined for $c > 0$,

$$\max \sum_{j=1}^{n} f_j(x_j) + c \cdot x_{n+1},$$

$$\sum_{j=1}^{n+1} x_j = B,$$

$$x_j \geq 0 \quad \text{and integer } j = 1, \ldots, n+1.$$

Let the functions f_j be concave and monotonic increasing in the interval $[0, \lfloor \frac{B}{n} \rfloor]$, and constant in $[\lfloor \frac{B}{n} \rfloor, B]$. Solving this problem is then equivalent to determining in n

arrays of increments, $\{\Delta f_j(k)\}$, $k = 0, \ldots, \lfloor \frac{B}{n} \rfloor - 1$ of length $\lfloor \frac{B}{n} \rfloor$ each, the last entry of value $\geq c$. Since the arrays are independent, the information theory lower bound is $\Omega(n \log \lfloor \frac{B}{n} \rfloor)$. Similarly, for the case of an inequality constraint the same lower bound applies for the problem on n variables,

$$\max \sum_{j=1}^{n} f_j(x_j),$$

$$\sum_{j=1}^{n} x_j \leq B,$$

$$x_i \geq 0, \quad \text{integer } j = 1, \ldots, n,$$

since x_{n+1} can simply be viewed as the slack and $c = 0$.

3.2. *The algebraic-tree model.* One might criticize the choice of the comparison model for this problem as being too restrictive. Indeed, the use of arithmetic operations may help to reduce the problem's complexity. This is the case for the quadratic simple allocation problem, which is solvable in linear time, $O(n)$, (Brucker 1984). The lower bound here demonstrates that such success is not possible for other nonlinear functions. The computation model that is used hereafter allows the arithmetic operations $+, -, \times, \div$ as well as comparisons and branching based on any of these operations. It is demonstrated that the nature of the lower bound is unchanged even if the floor operation is permitted as well.

We rely on Renegar's (1987) lower bound proof in this arithmetic model of computation for finding ϵ-accurate roots of polynomials of fixed degree ≥ 2. In particular, the complexity of identifying an ϵ-accurate single real root in an interval $[O, R]$, $\Omega(\log \log(R/\epsilon))$ even if the polynomial is monotone in that interval. Let $p_1(x), \ldots, p_n(x)$ be n polynomials each with a single root to the equation $p_i(x) = c$ in the interval $[0, B/n]$, and each $p_i(x)$ a monotone decreasing function in this interval. Since the choice of these polynomials is arbitrary, the lower bound on finding the n roots of these n polynomials is $\Omega(n \log \log(B/n\epsilon))$. Let $f_j(x_j) = \int_0^{x_j} p_j(x)\, dx$. The f_j's are then polynomials of degree ≥ 3. The problem,

$$(P_\epsilon) \qquad \max \sum_j f_j(x_j \cdot \epsilon) + c \cdot x_{n+1} \cdot \epsilon,$$

$$\sum_{j=1}^{n+1} x_j = \frac{B}{\epsilon},$$

$$x_j \geq 0, \qquad x_j \text{ integer},$$

has an optimal solution $\mathbf{x}$ such that $\mathbf{y} = \epsilon \cdot \mathbf{x}$ is the $(n\epsilon)$-accurate vector of roots solving the system

$$\begin{cases} p_1(y_1) = c, \\ p_2(y_2) = c, \\ \vdots \\ p_n(y_n) = c. \end{cases}$$

This follows directly from the Kuhn-Tucker conditions of optimality, and the fact that an optimal integer solution to the scaled problem with a scaling constraint, $s, \mathbf{x}^*$, and the optimal solution to the continuous problem $\mathbf{y}^*$ satisfy $\|\mathbf{x}^* - \mathbf{y}^*\|_\infty \leq ns$. This

proximity between the optimal integer and the optimal continuous solutions was proved in Hochbaum and Shanthikumar (1990) for a general constraint matrix. The right-hand side is ns when the constraint matrix is totally unimodular. (A tighter proximity is proved in Corollary 4.3.) Hence, a lower bound for the complexity of solving (P_ϵ) is $\Omega(n \log\log(B/n^2\epsilon))$. For $\epsilon = 1$, we get the desired lower bound for the integer problem.

In Mansour, Schieber and Tiwari (1991) there is a lower bound proof for finding ϵ-accurate square roots that allows also the floor, $\lfloor\ \rfloor$, operation. In our notation the resulting lower bound for our problem is $\Omega(\sqrt{\log\log(B/\epsilon)})$, hence even with this additional operation the problem cannot be solved in strongly polynomial time. Again, the quadratic objective is an exception and the algorithms for solving the quadratic objective simple resource allocation problems rely on solving for the continuous solution first, then rounding down, using the floor operation, and proceeding to compute the resulting integer vector to feasibility and optimality using fewer than n greedy steps. See for instance Ibaraki and Katoh (1988) for such an algorithm. Since the lower bound result applies also in the presence of the floor operation, it follows that the "ease" of solving the quadratic case is indeed due to the quadratic objective and not to this, perhaps powerful, operation.

4. A proximity theorem. Consider the scaled problem, GAPs,

$$\text{(GAPS)} \qquad \max \sum_{j\in E} f_j(sx_j),$$

$$\sum_{j\in E} x_j = \frac{B}{s},$$

$$\sum_{j\in A} x_j \le \frac{r(A)}{s}, \qquad A \subset E,$$

$$x_j \ge \frac{l_j}{s} \quad \text{and integer } j \in E.$$

A direct application of the algorithms in Hochbaum and Shanthikumar (1990) calls for finding an optimal integer solution to this scaled problem. Yet, the running time depends on the largest subdeterminant of the constraint matrix, which may not be polynomial. Here we employ a different approach that relies on tighter proximity thus resulting in polynomial running time. We use an algorithm, *greedy*(s), that compares the increments Δ_j as in *greedy*, (rather than $\Delta_j^{(s)}$) but the increase of the selected component is s units, when such increase is feasible. If such increase is infeasible, yet a positive increase is feasible, *greedy*(s) increments the variable for the last time by one unit. The proximity theorem proves that only the last increment made in *greedy*(s) to each variable may be "incorrect."

Procedure greedy(s)
Step 0: $x = \mathbf{l}$, $B = B - \mathbf{l}\cdot\mathbf{e}$, $E = \{1, 2, \ldots, n\}$.
Step 1: Find i such that $\Delta_i(x_i) = \max_{j\in E}\{\Delta_j(x_j)\}$.
Step 2: (Feasibility check), is $\mathbf{x} + \mathbf{e}^i$ infeasible?
If yes, $E \leftarrow E - \{i\}$, and $\delta_i = s$.
Else, is $\mathbf{x} + s\cdot\mathbf{e}^i$ infeasible?
If yes, $E \leftarrow E - \{i\}$, $x_i \leftarrow x_i + 1$, $B \leftarrow B - 1$, and $\delta_i = 1$.
Else, $x_i \leftarrow x_i + s$, and $B \leftarrow B - s$.
Step 3: If $B = 0$, or $E = \phi$, stop, output $\mathbf{x}$.
Otherwise go to step 1.

Let the output of *greedy*(s) be $\mathbf{x}^{(s)}$, and the output of *greedy* which is an optimal solution to (GAP), be $\mathbf{x}^*$. The vector $\boldsymbol{\delta}$ records the last increments executed by *greedy*(s). Note that $\delta_i = 1$ or s; hence $\mathbf{x}^{(s)}$ is equal to 0 or 1 modulo s. All algorithmic results reported here would hold if only increments of size s permitted in *greedy*(s). This feature of allowing single unit increments as the last increment in a variable is only to strengthen the statement of the proximity theorem. In general, $\mathbf{x}^{(s)}$ may not satisfy the equality constraint, $\sum_{j \in E} x_j = B$, if $B > 0$ at the termination of the procedure.

PROXIMITY THEOREM (THEOREM 4.1). *If there is a feasible solution to (GAP) then there exists an optimal solution* $\mathbf{x}^*$ *such that* $\mathbf{x}^* > \mathbf{x}^{(s)} - \boldsymbol{\delta} \geq \mathbf{x}^{(s)} - s \cdot \mathbf{e}$. (*where the inequalities are component-wise*).

DISCUSSION. This theorem is of similar flavor as the one that has been used by Edmonds and Karp (1972) to solve the minimum cost network flow problem via scaling where the solution obtained at each iteration bounds the solution at the next iteration, and the optimal solution, from below.

PROOF. In the proof we need the output of *greedy*(s) to be feasible and satisfy the equality constraint. In order to achieve that, we introduce another greedy algorithm *greedy*$'(s)$ with output $\mathbf{x}'^{(s)}$. $\mathbf{x}'^{(s)}$ is a solution derived from $\mathbf{x}^{(s)}$ by applying *greedy* to it, until the equality constraint is satisfied. Note that *greedy*$'(s)$ is not a polynomial algorithm. The proof will show that $\mathbf{x}^* > \mathbf{x}'^{(s)} - \boldsymbol{\delta}'$ where $\boldsymbol{\delta}'$ is the vector of last increments in *greedy*$'(s)$. Since (see claim 4.2 below) $\mathbf{x}'^{(s)} - \boldsymbol{\delta}' \geq \mathbf{x}^{(s)} - \boldsymbol{\delta}$, the theorem will follow for $\mathbf{x}^{(s)}$ as well.

greedy$'(s)$ differs from *greedy*(s) in step 3 where "stop" is replaced by "go to step 4." Step 4 is essentially an application of *greedy* with the initial solution $\mathbf{x}^{(s)}$.

Step 4a: $\boldsymbol{\delta}' = \boldsymbol{\delta}$ $E = \{1, 2, \ldots, n\}$.

Step 4b: Find i such that $\Delta_i(x_i) = \max_{j \in E}\{\Delta_j(x_j)\}$.

Step 4c: (Feasibility check), is $\mathbf{x} + \mathbf{e}^i$ infeasible?
If yes, $E \leftarrow E - \{i\}$.
Else, $x_i \leftarrow x_i + 1$, $B \leftarrow B - 1$, and $\delta'_i = 1$.

Step 4d: If $B = 0$, stop, output $\mathbf{x}$. (If $E = \phi$ while $B > 0$, then GAP is infeasible).
Otherwise go to step 4b.

Note that unlike the output of *greedy*(s), the vector $\mathbf{x}'^{(s)}$ satisfies the equality constraint. We now prove the claim.

Claim 4.2. $\mathbf{x}^{(s)} + (s-2)e \geq \mathbf{x}'^{(s)} \geq \mathbf{x}^{(s)}$ and $\mathbf{x}'^{(s)} - \boldsymbol{\delta}' \geq \mathbf{x}^{(s)} - \boldsymbol{\delta}$,

PROOF. Obviously $\mathbf{x}'^{(s)} \geq \mathbf{x}^{(s)}$. If $x_j^{(s)} < x_j'^{(s)}$ then an increment of further $s-1$ units of $x_j^{(s)}$ is infeasible; hence, $\mathbf{x}^{(s)} + (s-2)e \geq \mathbf{x}'^{(s)}$. Now $\delta'_j \geq \delta_j$ for all j. This is because δ_j is either 1 or s, whereas δ'_j is either equal to δ_j or is 1. □

One corollary of this claim that will be used in Theorem 5.1 is that $\sum_{j \in E} x_j^{(s)} \geq \sum_{j \in E} x_j'^{(s)} - (s-2)n = B - (s-2)n$. The rest of the theorem is proved for $\mathbf{x}'^{(s)}$. To simplify the notation, without risking ambiguity, we shall use the notation $\mathbf{x}^{(s)}$ for the output of *greedy*$'(s)$ and $\boldsymbol{\delta}$ for the vector of last increments.

Let $\mathbf{x}^{**}$ be an optimal solution to GAP. Let the vector, $\bar{\mathbf{x}}$, be defined by $\bar{x}_j = \min\{x_j^{**}, x_j^{(s)}\}$. Consider the problem GAP restricted to solutions satisfying $\mathbf{x} \geq \bar{\mathbf{x}}$. Since $\mathbf{x}^{**} \geq \bar{\mathbf{x}}$ applies, the modified problem has the same objective value. Applying the optimality of greedy solution to the submodular function $\bar{r}(A) = r(A) - \sum_{j \in A} \bar{x}_j$, and $\bar{B} = B - \sum_{j=1}^{n} \bar{x}_j$, we get that starting the greedy algorithm from $\bar{\mathbf{x}}$ it finds an optimal solution, which is denoted by $\mathbf{x}^*$. Now run this greedy algorithm, *greedy* (of §2), with choosing i such that $x_i < x_i^{(s)} - \delta_i$ whenever possible. By the

concavity of the functions f_j and the greedy choices of *greedy'*(s) applied to (GAPs), we get that if $x_i < x_i^{(s)} - \delta_i$ and $x_k \geq x_k^{(s)}$, then $\Delta_i \geq \Delta_k$.

Therefore, as long as $\mathbf{x} \geq \mathbf{x}^{(s)} - \boldsymbol{\delta}$ is not satisfied, we must have that $\mathbf{x} \leq \mathbf{x}^{(s)}$. Recall that $\mathbf{x}^*$ denotes an optimal solution obtained by *greedy*, beginning with $\bar{\mathbf{x}}$, such that whenever $\mathbf{x}^* \geq \mathbf{x}^{(s)} - \boldsymbol{\delta}$ is not satisfied, we have $\mathbf{x}^* \leq \mathbf{x}^{(s)}$. But $\mathbf{x}^* \leq \mathbf{x}^{(s)}$ implies that $\mathbf{x}^* = \mathbf{x}^{(s)}$, since $\sum_{j=1}^n x_j^* = \sum_{j=1}^n x_j^{(s)} = B$. Hence, whenever $\mathbf{x}^* \geq \mathbf{x}^{(s)} - \boldsymbol{\delta}$ is not satisfied, $\mathbf{x}^* = \mathbf{x}^{(s)} \geq \mathbf{x}^{(s)} - \boldsymbol{\delta}$, a contradiction. Therefore, $\mathbf{x}^* \geq \mathbf{x}^{(s)} - \boldsymbol{\delta}$. □

We conclude from this theorem, a proximity result on the distance between an optimal integer and optimal continuous solutions to GAP. Such a result is not useful in finding optimal integer solutions to the problem unless the continuous problem is particularly easy to solve. This is the case for the continuous *quadratic* allocation problem, and potentially for other cases of the general quadratic allocation problem.

COROLLARY 4.3. *For an integer optimal solution to GAP,* $\mathbf{z}^*$, *there is a continuous optimal solution to GAP,* $\mathbf{x}^*$, *such that,*

$$\mathbf{z}^* - \mathbf{e} < \mathbf{x}^* < \mathbf{z}^* + n\mathbf{e}.$$

and, vice versa, for a continuous optimal solution to GAP, $\mathbf{x}^*$, *there is an integer optimal solution to GAP,* $\mathbf{z}^*$, *such that,*

$$\mathbf{z}^* - \mathbf{e} < \mathbf{x}^* < \mathbf{z}^* + n\mathbf{e}.$$

PROOF. Let $\mathbf{x}(\epsilon)$ be an ϵ-accurate solution to the continuous problem, i.e., it satisfies for $\mathbf{x}^*$, $\|\mathbf{x}^* - \mathbf{x}(\epsilon)\|_\infty \leq \epsilon$. Obviously, $\mathbf{x}^* = \lim_{\epsilon \to 0} \mathbf{x}(\epsilon)$.

Given any $\epsilon > 0$, by rescaling ϵ to 1, an optimal integer solution, in integer multiplies of ϵ is derived from a procedure *greedy*$(1/\epsilon)$. Hence the proximity Theorem 4.1 applies,

$$\mathbf{z}^* - \mathbf{e} < \mathbf{x}(\epsilon).$$

Taking $\epsilon \to 0$, this becomes

$$\mathbf{z}^* - \mathbf{e} < \mathbf{x}^*,$$

Also, since $\mathbf{z}^* \cdot \mathbf{e} = \mathbf{x}^* \cdot \mathbf{e} = B$ it follows that $\mathbf{x}^* < \mathbf{z}^* + n\mathbf{e}$. □

In particular, $\|\mathbf{z}^* - \mathbf{x}^*\|_\infty < n$. This is a tighter proximity theorem than the one existing in the literature for constrained linear (Cook, et al. 1986), quadratic (Granot and Skorin-Kapov 1990) and nonlinear (Hochbaum and Shanthikumar 1990) optimization problems, all of which have $\|\mathbf{z}^* - \mathbf{x}^*\|_\infty < n\Delta$, where Δ is the largest subdeterminant of the constraint matrix. The result stated here could be viewed as effectively considering the largest subdeterminant of a polymatroid to be 1, although it could be in fact exponentially large.

A potential use of this result is to produce more efficiently integer solutions to the quadratic cases of GAP, where the continuous solution is relatively easy to derive from Kuhn-Tucker conditions (all of which are linear for quadratic objective function).

5. The main algorithm. Given a general resource allocation problem, GAP, with $\mathbf{l}$ a feasible solution. The following algorithm is based on scaling and proximity. It solves the GAP problem in $O(n(\log n + F)\log(B/n))$ operations, with F denoting the running time (in *greedy* or *greedy*(s)) required to verify that an increment in one of the vector's component is feasible. Note that *greedy* is identified with *greedy*(1).

Algorithm GAP
Step 0: Let $s = \lceil B/2n \rceil$.
Step 1: If $s = 1$, call *greedy*. The output is $\mathbf{x}^*$. Stop. $\mathbf{x}^*$ is an optimal solution.
Step 2: Call *greedy*(s). Let $\mathbf{x}^{(s)}$ be the output.
Set $\mathbf{l} = \mathbf{x}^{(s)} - s\mathbf{e}$
Set $s \leftarrow \lceil s/2 \rceil$.
Go to step 1.

THEOREM 5.1. (a) *Algorithm GAP is valid*;
(b) *The running time of Algorithm GAP is* $O(n(\log n + F)\log(B/n))$.

PROOF. (a) The validity is a direct corollary of the proximity theorem. Since the vector $\mathbf{x}^{(s)} - s \cdot \mathbf{e}$ bounds an optimal solution from below, setting the lower bound constraints to $\mathbf{x} \geq \mathbf{x}^{(s)} - s \cdot \mathbf{e}$ does not change the optimal value of GAP.

(b) *greedy*(s) is called $\log\lceil B/2n \rceil$ times and *greedy* is called once. Each time a call is made there are at most $\lceil (B - \mathbf{l} \cdot \mathbf{e})/s \rceil$ increments to be executed. Using Claim 4.2,

$$\mathbf{l} \cdot \mathbf{e} = \sum_{i=1}^{n} l_i = \sum_{i=1}^{n} \left(x_i^{(s)} - s \right) = \sum_{i=1}^{n} x_i^{(s)} - sn \geq B - (s-2)n - sn > B - 2sn.$$

Hence, at each iteration $\lceil (B - \mathbf{l} \cdot \mathbf{e})/(s/2) \rceil \leq 4n$. Thus, there are not more than $O(n)$ increments at each call to be executed by *greedy*(s), or *greedy*. The amount of work required for each increment is $O(\log n + F)$, where $O(\log n)$ comparisons are needed to maintain the sorted vector of up to n potential increments and F steps to check the feasibility of a selected increment. Note that if an increment in component j is not feasible, that component is removed from consideration. Consequently, there are at most n such "failed" feasibility tests. □

The next section describes specialized implementations of Algorithm GAP that are more efficient than Algorithm GAP in the feasibility checking phase or in maintaining the increments' array.

6. Faster implementations.

6.1. *The simple resource allocation problem* (*SRA*). The fastest algorithm known for this problem is an $O(n \log(B/n))$ algorithm by Frederickson and Johnson (1982). This algorithm is optimal in the comparison model (see §3.1). Their procedure uses among others, a subroutine called CUT. Here we show how incorporating CUT at each iteration of the main Algorithm GAP yields the same running time while avoiding other complex procedures of the FJ algorithm. Given an SRA problem with a constraint, $\sum_{j=1}^{n} x_j = B$, each variable could potentially take any integer value in $[O,B]$. Given the n monotone nonincreasing arrays of increments $\{f_j(i) - f_j(i-1)\}_{i=1}^{B}$, $j = 1, \ldots, n$, CUT removes all but $O(B)$ largest increments. Effectively CUT provides new upper bounds to each variable $x_i \leq u_i$ that are satisfied by an optimal solution. The procedure CUT works in linear time, $O(n)$. CUT is used in Frederickson and Johnson (1982) as a preprocessing step followed by an algorithm that finds the Bth largest entry in the remaining array of size $O(B)$. We use CUT in the scaled problem where it is followed by a median selection algorithm among $O(n)$ entries.

The main algorithm here makes $O(\log(B/n))$ calls to *greedy*(s). Each call for *greedy*(s) generates a feasible solution $\mathbf{x}^{(s)}$ to the constraints $\sum_{j=1}^{n} x_j^{(s)} = s\lceil B/s \rceil$, with $x_j^{(s)}$ nonnegative integers, for $j = 1, \ldots, n$. This vector $\mathbf{x}^{(s)}$ is generated by considering the increments of one unit at points on a grid of granularity s. The array describing such $\lceil B/s \rceil$ increments for each of the n variables is of length $O(n)$ since $s = \lceil B/2n \rceil$

(in fact $\lceil B/s \rceil \leq 2n + 1$). So the total number of entries in the n arrays is $O(n^2)$. We apply CUT, thus removing all but $O(n)$ entries from the n arrays. Of these $O(n)$ entries we need to find the $(2n + 1)$st ranking element and the implied partition of elements to those smaller than that element and those larger. Such a selection procedure can be done in linear time in the size of the array (Blum, et al. 1972). Hence, each call to *greedy*(s) works in $O(n)$ time. The total running time is thus $O(n \log(B/n))$.

6.2. *The generalized upper bounds resource allocation problem (GUB).* The GUB problem is easier to handle once we observe that it is polynomially equivalent to a simple resource allocation problem where each variable has an upper bound constraint:

$$\text{(UB)} \qquad \max \sum_{j=1}^{n} f_j(x_j),$$

$$\sum_{j=1}^{n} x_j = B,$$

$$x_j \leq u_j \quad \text{and nonnegative integers } j = 1, \ldots, n.$$

This observation is proved in Lemma 6.2.1.

Consider the set S_i, the constraint $\sum_{j \in S_i} x_j \leq b_i$, and the following simple resource allocation problem restricted to S_i:

$$(\text{SRA}_i) \qquad \max \sum_{j \in S_i} f_j(x_j),$$

$$\sum_{j \in S_i} x_j = b_i,$$

$$x_j, \quad \text{nonnegative integers } j \in S_i.$$

LEMMA 6.2.1. *Let the solution to SRA$_i$ be $\{x_j^{(i)}\}_{j \in S_i}$. There exists an optimal solution to GUB, $\mathbf{x}^*$, satisfying $x_j^* \leq x_j^{(i)}$ $j \in S_i$.*

PROOF. If there is no such optimal solution, then choose among all optimal solutions, the one $\mathbf{x}^*$ such that $\delta = \sum_{j \in S_i} \max\{x_j^* - x_j^{(i)}, 0\}$ is minimum. Let $j_1 \in S_i$ be such that $x_{j_1}^* > x_{j_1}^{(i)}$; then $\exists j_2 \in S_i$ such that $x_{j_2}^* < x_{j_2}^{(i)}$, otherwise $\sum_{j \in S_i} x_j^* > b_i$.

From the optimality of $\mathbf{x}^{(i)}$, $\Delta_{j_2}(x_{j_2}^{(i)} - 1) \geq \Delta_{j_1}(x_{j_1}^{(i)})$.

From the optimality of $\mathbf{x}^*$, $\Delta_{j_2}(x_{j_2}^*) \leq \Delta_{j_1}(x_{j_1}^* - 1)$.

From the concavity,

$$\Delta_{j_2}(x_{j_2}^*) \geq \Delta_{j_2}\left(x_{j_2}^{(i)} - 1\right) \quad \text{and}$$

$$\Delta_{j_1}\left(x_{j_1}^{(i)}\right) \leq \Delta_{j_1}\left(x_{j_1}^* - 1\right).$$

Hence all these increments are equal, and the solution $\mathbf{x}'^*$,

$$x_j'^* = \begin{cases} x_j^*, & j \neq j_1, j_2, \\ x_{j_1}^* - 1, & j = j_1, \\ x_{j_2}^* + 1, & j = j_2, \end{cases}$$

satisfies $\sum_{j \in S_i} \max\{x_j'^* - x_j^{(i)}, 0\} = \delta - 1$, which contradicts the minimality of this expression for $\mathbf{x}^*$. □

The algorithm for GUB applies scaling as before. Each of the $\log(B/n)$ scaled problems involve n variables (arrays) with at most $2n + 1$ values each. Now each of the *scaled* problems SRA_i is solved by first applying CUT and then median finding in $O(n_i)$ time. Once the problems $\text{SRA}_1, \ldots, \text{SRA}_m$ are solved, the (scaled) upper bounds on the variables are available. Those upper bounds are used to trim the arrays to length u_j each $j = 1, \ldots, n$. CUT is then applied, in $O(n)$ time, to the truncated arrays and removes all but $O(n)$ entries. The $O(n)$th element if then found in linear time. Hence, each scaled problem is solved in linear time. The total running time of the algorithm is thus $O(n \log(B/n))$. This running time is optimal for this problem and matches its concrete complexity in the comparison model.

6.3. *The nested problem.* Unlike the problem SRA and GUB, here we actually execute the algorithm *greedy*(s) in order to obtain a feasible scaled solution. Compared to the general algorithm, the faster implementation is due entirely to more efficient feasibility checking procedure. The running time of the general algorithm is $O(n(\log n + F)\log(B/n))$ with F the running time needed to determine whether for a feasible solution $\mathbf{x}, \mathbf{x} + \mathbf{e}^i$ is feasible. An obvious way of performing the feasibility checking is to check each of the m constraints in $O(m)$ total time per each of the $O(n)$ feasibility checks. We show how to perform that check more efficiently in $O(1)$ time on the average, i.e., in a total of $O(n)$ time for the $O(n)$ greedy steps.

For notational convenience, we shall write the nested system:

$$\begin{aligned} \sum_{j \in S_1} x_j &\le b_1 \\ \sum_{j \in S_2} x_j &\le b_2 \\ &\vdots \\ \sum_{j \in S_m} x_j &\le b_m \end{aligned}$$

with $S_1 \subset S_2 \subset \cdots \subset S_m$ as

$$\sum_{j=1}^{i} x_j \le b_{k(i)}, \qquad i = 1, \ldots, |S_m|.$$

Where $k(i)$ is such that $S_{k(i)-1} \subset \{x_1, \ldots, x_i\} \subseteq S_{k(i)}$. Consequently, if S_i contains, say, 3 more variables than S_{i-1} then there will be three constraints each adding one of these variables with the same right-hand side. In this new formulation we set $S_i = \{1, \ldots, i\}$. We now associate with each variable (or set, as in the new formulation each set S_i corresponds to a variable x_i) a nonnegative "slack" s_i, initialized as the value s_i^a, $s_i^o = b_i - b_{i-1}$. (b_0 is assumed to be 0). Some of these slacks s_i^o may already be zero at the outset if $b_i = b_{i-1}$. Slacks are defined so the following invariant property is satisfied. If a variable with a positive slack is to be increased by one unit, then this increase is feasible, and a variable x_i with zero slack can be feasibly increased if any of the preceding slacks $s_1, \ldots, s_{i-1}$ is positive. More precisely, we claim that the following feasibility checking algorithm initialized with slacks $s_i = s_i^o$ is

valid. The input is a feasible vector $\mathbf{x}$, the index i of the variable to be increased, the present slacks vector, and a $0-1$ labeling vector l.

$$l_i = \begin{cases} 1 & \text{if } s_i > 0, \\ 0 & \text{if } s_i = 0. \end{cases}$$

Feasibility check $(\mathbf{x}, i; s_1, \ldots, s_n; l_1, \ldots, l_n)$
Set $s_0 = 1$.
Let $k(i) = \max\{j | 0 \le j \le i, s_j > 0\}$.
If $k(i) > 0$, set $s_{k(i)} \leftarrow s_{k(i)} - 1$, $l_{k(i)} \leftarrow \min\{l_{k(i)}, s_{k(i)}\}$
output "feasible," and $(s_1, \ldots, s_n)$,
else output "infeasible."
end.

LEMMA 6.3.1. *The feasibility check algorithm is valid.*

PROOF. In order for an increment vector $\boldsymbol{\delta}$ of a feasible solution $\mathbf{x}$ to be feasible, $\boldsymbol{\delta}$ has to satisfy:

$$\begin{aligned} \delta_1 &\le b_1 - x_1, \\ \delta_1 + \delta_2 &\le b_2 - x_1 - x_2, \\ &\vdots \\ \sum_{j=1}^{n} \delta_j &\le b_n - \sum_{j=1}^{n} x_j. \end{aligned}$$

As discussed earlier, the right-hand side in a nested system can always be monotone nondecreasing. Hence we set the right-hand sides, $\bar{\mathbf{b}}$ to be $\bar{b}_i = \min\{b_i - \sum_{j=1}^{i} x_j, \bar{b}_{i+1}\}$, where $\bar{b}_n$ is set equal to $b_n - \sum_{j=1}^{n} x_j$. The algorithm maintains the vector $\bar{\mathbf{b}}$ monotone nondecreasing. When, for an index $\mathbf{i}$, it is found that $s_{k(i)} > 0$, then $\bar{b}_i = \sum_{j=1}^{i} s_j > 0$, hence the value of x_i can indeed be incremented by one unit (assuming $\mathbf{b}$ is an integer vector). One can also deduce that $\bar{b}_{k(i)} = \cdots = \bar{b}_i$. The updating of the slacks corresponds to setting the new values of $\bar{b}_{k(i)}, \ldots, \bar{b}_i$ to be all equal to $\bar{b}_i - 1$, which corresponds to the right-hand sides of the system the increment vector needs to satisfy following the increment of x_i by one unit. □

The implementation of the feasibility check requires the labeling of constraints (or variables) with a $0-1$ label, l_i, and the identification of the index $k(i)$. This can be done in linear time as follows.

If we view inclusion as a partial order where S_1 precedes S_2 if $S_1 \subseteq S_2$, the search is for the nearest predecessor with label 1. Since the partial order is linear, we can maintain the 0 labeled indices on the line, in intervals separated by 1's. Each interval is represented by pointers to its left-end and right-end points. The left-end point of an interval has a pointer to the nearest preceding right-end point of the previous interval. With this data structure, each iteration consists of finding the left-end point of an interval to which i belongs (that will provide the index $k(i)$), or updating the sets of intervals by a merger when a label becomes 0. This series of n union-find operations can be executed in $O(n)$ time using Gabow and Tarjan's algorithm (Gabow and Tarjan 1985).

The total complexity of the algorithm for the *nested* problem is then $O((n \log n + n) \cdot \log(B/n)) = O(n \log n \log(B/n))$.

6.4. *The tree constrained problem.* Recall from §2 that the tree structure is such that each set is decomposed into disjoint subsets. Consequently, the number of nodes in the tree, m, is $O(n)$. For the sake of convenience we assume the existence of an upper bound constraint for each variable. If such a constraint does not exist for a variable, then we can add it by setting the right-hand side to be the same as in the leaf constraint (the smallest set) in which this variable appears. These additional constraints are redundant and do not change the set of feasible solutions. As a result, the tree can be assumed to have n leaf nodes, corresponding to each of the variables.

A straight-forward way in which to implement the feasibility check works as follows. The depth of the tree is n at most. For each feasibility check of increase in x_i, all the nodes on the path from the root to the leaf containing index i, are inspected and the slack in each such constraint (the current difference between the right-hand side and the left-hand side) is updated. Hence in two passes of length $O(n)$ each, the feasibility check and the update of the slacks is accomplished. The total running time is therefore $O(n(\log n + n)\log(B/n) = O(n^2 \log(B/n))$. This running time is equivalent to the best running time reported in Ibaraki and Katoh (1988) (at the time when these results were initially reported). Recently, an algorithm by Dyer and Frieze (1990), improved on that running time to $O(n \log^2 n \log B)$. We show here that implementing the feasibility check using the path compression approach, yields an algorithm with running time$O(n \log n \log(B/n))$.

We label each node in the tree with the value of its slack—the difference between the value of the right-hand side, and the value of the left-hand side for the current solution. Initially the labels are the right-hand sides of the respective constraints. The feasibility check, as described above, consists of two phases. The first phase is to identify the minimum label on the path from the leaf node corresponding to the variable to be incremented, to the root, r. If the value of this minimum label is positive, then the increment is feasible. The other phase is the updating of the values of the slacks (reduce each by one) on the same path once the feasibility of an increment has been confirmed.

For the first phase, the operator of minimum of two values is associative, therefore the path compression technique described in Tarjan (1979) is applicable and can be used to evaluate the feasibility. Such operation is called EVAL in the terminology of Tarjan (1979). Since there are only $O(n)$ EVAL operations, this part can be accomplished in $O(n \log n)$, as proved in Theorem 1 in Tarjan (1979). The update operation is trickier. First, it consists of a different operator (subtraction of 1); hence the result on path compression is not applicable, as it requires the same operator to be used in EVAL and the update. Furthermore, the path compression algorithm counts the number of *nodes* that are being updated. Since there could be $O(n^2)$ updates, the application of the path compression will take longer running time than the straight-forward technique. To overcome this difficulty, we maintain the labels in a special data structure that allows us to perform an update of a path in the tree as a single operation. Also EVAL is performed simultaneously with the update; i.e., as the value of the minimum is computed from the leaf to some node on the path to the root, the values of the slacks along this path have already been updated. Although such simultaneous update is not correct, in the sense that when an increment is not feasible, some of the labels on the path have been updated, as if the increment has been performed, resulting in slack labels that are not the difference between the right-hand side and left-hand side as required. Yet, the algorithm remains correct by interrupting the EVAL process once a 0 label has been encountered. This works since if some node in the tree has the label 0, then all leaves of the subtree rooted at that node, cannot be feasibly incremented, and the value of the minimum on the path from these leaves to the root (which is 0) remains unchanged.

The labels of all nodes in the tree are maintained in an array, SLACK, that can also be read as a single word. Since all right-hand sides in each phase of the scaling algorithm are $O(n)$, the length of SLACK is up to $O(n \log n)$ bits. The array SLACK is created following each scaling in linear time. In addition, there are $O(n)$ such arrays created, SUBARRAY(r, v), one for each node v in the tree. SUBARRAY(r, v) is an array of the same length as SLACK, and it consists only of 0 or 1 values, 1 for all entries corresponding to nodes on the unique path from r to v, and 0 in all other entries. Those values are padded with 0's so that this array is of the same length as SLACK. SUBARRAY's are created once in a preprocessing stage, and maintained unchanged throughout. The SUBARRAY's are created using depth-first-search or breadth-first-search in linear time.

Implementing EVAL using the path compression technique results in a modified tree, called the *virtual tree*, with the property that a parent of a node in the virtual tree, is an ancestor in the original tree. Updating the labels requires updating in the original tree. The updating of the path between a node v, and its immediate ancestor, *parent*(v) in the virtual tree, is done by setting:

$$\text{SLACK} \leftarrow \text{SLACK} - \text{SUBARRAY}(r, v) + \text{SUBARRAY}(r, \text{parent}(v)).$$

The EVAL operation in Tarjan (1979) is implemented by proceeding from leaf to root in the virtual tree and applying EVAL(v) for each node v on that path. We modify this operation as follows:

procedure EVAL(v);
  if *parent*$(v) = r$ *then* EVAL := label(v) Stop.
    else if *label*$(v) = 0$ *then* EVAL = 0 Stop.
      else COMPRESS(v); EVAL := min{label(parent(v)), label(v)}
      SLACK := SLACK − SUBARRAY(r, v) + SUBARRAY$(r,$ parent$(v))$**fi**;

The procedure COMPRESS(v) is the same as in Tarjan (1979). The change in the EVAL(v) procedure is only in the interruption when EVAL is 0 and the added update of SLACK. The total number of operations of this new EVAL procedure is only a constant factor more than the original EVAL procedure. Hence the total running time for the $O(n)$ feasibility checks *and* updates is $O(n \log n)$. The running time of the scaling algorithm for solving the tree case is hence $O(n \log n \log(B/n))$.

7. Concluding remarks. Among the questions still left open in this paper is the tightness of the different lower bounds. Can one find optimal algorithms with respect to the comparison model lower bound for the nested case and the tree case of the allocation problem? In order to do that, one would have to devise an approach of maintaining the array of the increments without sorting.

Another question is on the tightness of the lower bounds of the algebraic computation tree model. In particular it may be possible, when the objective function is a separable polynomial, to reduce the running time to a function of $\log_2 \log_2 B$. Such algorithms will probably require a refining of the existing Newton methods for finding roots of polynomials.

The quadratic problem takes a special place in terms of its complexity among the nonlinear problems. The lower bound in the algebraic computation tree model does not apply to the quadratic objective function. Hence, strongly polynomial algorithms could potentially be developed for quadratic optimization problems. In particular, the linear transportation problem is known solvable in strongly polynomial time, where no such algorithm is known for the quadratic transportation problem. Recently, this issue has been partially resolved by demonstrating a strongly polynomial algorithm for the quadratic transportation problem with a fixed number of sources or sinks

(Cosares and Hochbaum 1990). Also, recently, the quadratic cases of the allocation problem—the generalized upper bounds, nested and tree—were shown to be solvable by strongly polynomial algorithms of complexity $O(n)$, $O(n \log n)$ and $O(n \log n)$ respectively (Hochbaum and Hong 1992).

Acknowledgement. I wish to thank George J. Shanthikumar for suggesting the study of the allocation problem and for many helpful discussions. Thanks are extended to A. Tamir for pointing out important relevant literature, especially the reference (Mansour, Schieber and Tiwari 1991). I am also indebted to R. Shamir for his detailed comments and to S. P. Hong for identifying an error in an early version of the proof of Theorem 4.1. Finally, I am grateful to an anonymous referee for suggestions that resulted in improving the substance and presentation of the paper and simplifying significantly the proof of Theorem 4.1. The author was supported in part by the Office of Naval Research under Grant N00014-88-K-0377 and by Grant ONR N00014-91-J1241 and by the National Science Foundation under Grant ECS-85-01988.

References

Bitran, G. R. and Hax, A. C. (1981). Disaggregation and Resource Allocation Using Convex Knapsack Problems with Bounded Variables. *Management Sci.* **27** 431–441.

Blum, M. Floyd, R. W. Pratt, V. R. Rivest, R. L. and Tarjan, R. E. (1972). Time Bounds for Selection. *J. Comput. System Sci.* **7** 448–361.

Brucker, P. (1984). An $O(n)$ Algorithm for Quadratic Knapsack Problems. *Oper. Res. Lett.* **3** 163–166.

Charnes, A. and Cooper, W. W. (1958). The Theory of Search: Optimal Distribution of Effort. *Management Sci.* **5** 44–49.

Cook, W. Gerards, A. M. H. Schrijver, A. and Tardos, E. (1986). Sensitivity Results in Integer Linear Programming. *Math. Programming* **34** 251–264.

Cosares, S. and Hochbaum, D. S. (1990). Strongly Polynomial Algorithms for the Quadratic Transportation Problem with Fixed Number of Sources. *Math. Oper. Res.*, (to appear).

Dyer, M. E. and Frieze, A. M. (1990). On an Optimimization Problem with Nested Constraints. *Discrete Appl. Math.* **26** 159–173.

_____ and Walker, J. (1987). An Algorithm for a Separable Integer Programming Problem with Cumulatively Bounded Variables. *Discrete Appl. Math.* **16** 135–149.

Edmonds, J. and Karp, R. M. (1972). Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems. *J. Assoc. Comput. Mach.* **19** 248–264.

Elton, E. J. Gruber, M. J. and Padberg, M. W. (1976). Simple Criteria for Optimal Portfolio Selection. *J. Finance* **31** 1341–1357.

Federgruen, A. and Groenevelt, H. (1986a). The Greedy Procedure for Resource Allocation Problems: Necessary and Sufficient Conditions for Optimality. *Oper. Res.* **34** 909–918.

_____ and _____ (1986b). Optimal Flows in Networks with Multiple Sources and Sinks, with Applications to Oil and Gas Lease Investment Programs. *Oper. Res.* **34** 218–225.

Fox, B. L. (1966). Discrete Optimization via Marginal Analysis. *Management Sci.* **13** 210–216.

Frederickson, G. N. and Johnson, D. B. (1982). The Complexity of Selection and Ranking in $X + Y$ and Matrices with Sorted Columns. *J. Comput. System Sci.* **24** 197–208.

Gabow, H. N. and Tarjan, R. E. (1985). A Linear-Time Algorithm for a Special Case of Disjoint Set Union. *J. Comput. System Sci.* **30** 209–221.

Galperin, A. and Waksman, Z. (1981). A Separable Integer Programming Problem Equivalent to its Continual Version. *J. Comput. Appl. Math.* **7** 173–179.

Granot, F. and Skorin-Kapov, J. (1990). Some Proximity and Sensitivity Results in Quadratic Integer Programming. *Math. Programming* **47** 259–268.

Grötchel, M. Lovász, L. and Schrijver, A. (1981). The Ellipsoid Method and its Consequences in Combinatorial Optimization. *Combinatorica* **1** 169–197.

Gross, O. (1956). A Class of Discrete Type Minimization Problems. RM-1644, Rand Core.

Groenevelt, H. (1985). *Two Algorithms for Maximizing a Separable Concave Function over a Polymatroidal Feasible Region*. Technical Report, The Graduate School of Management, University of Rochester.

Hochbaum, D. S. and Hong, S. P. (1992). About Strongly Polynomial Time Algorithms for Quadratic Optimization over Submodular Constraints. Manuscript, University of California Berkeley, CA.

_____ and Shanthikumar, J. G. (1990). Convex Separable Optimization is not much Harder than Linear Optimization. *J. Assoc. Comput. Mach.* **37** 843–862.

Ibaraki, T. and Katoh, N. (1988). *Resource Allocation Problems: Algorithmic Approaches.* MIT Press, Boston, MA.

Koopman, B. O. (1953). The Optimum Distribution of Effort. *Oper. Res.* **1** 52–63.

_____ (1957). The Theory of Search: Part III. The Optimum Distribution of Searching Effort. *Oper. Res.* **5** 613–626.

Luss, H. and Gupta, S. K. (1975). Allocation of Effort Resources Among Competing Activities. *Oper. Res.* **23** 360–366.

Mansour, Y. Schieber, B. and Tiwari, P. (1991). Lower Bounds for Computations with the Floor Operations. *SIAM J. Comput.* **20** 315–327.

Nemhauser, G. L. and Wolsey, L. A. (1988). *Integer and Combinatorial Optimization.* John Wiley and Sons,

Renegar, J. (1987). On the Worst Case Arithmetic Complexity of Approximation Zeroes of Polynomials, *J. Complexity* **3** 9–113.

Sanathanan, L. (1970). On an Allocation Problem with Multistage Constraints. *Oper. Res.* **18** 1647–1663.

Tamir, A. (1980). Efficient Algorithms for the Selection Problem with Nested Constraints and its Application to a Production—Sales Planning Model. *SIAM J. Control Optim.* **3** 282–287.

Tardos, E. (1986). A Strongly Polynomial Algorithm to Solve Combinatorial Linear Problems. *Oper. Res.* **34** 250–256.

Tarjan, R. E. (1979). Applications of Path Compression on Balanced Trees. *J. Assoc. Comput. Mach.* 690–715.

Yao, D. D. and Shanthikumar, J. G. (1987). The Optimal Input Rates to a System of Manufacturing Cells. *INFOR* **25** 57–65.

Ziegler, H. (1982). Solving Certain Singly Constrained Convex Optimization Problems in Production Planning. *Oper. Res. Lett.* **1** 246–252.

Zipkin, P. (1980). Simple Ranking Methods for Allocation or One Resource. *Management Sci.* **26** 34–43.

D. S. Hochbaum: School of Business Administration and IEOR Department, University of California, Berkeley, California 94720; e-mail:dorit@hochbaum.berkeley.edu