JOURNAL OF OPTIMIZATION THEORY AND APPLICATIONS: Vol. 85, No. 3, pp. 705-726, JUNE 1995

Finite Method for a Nonlinear Allocation Problem[1]

A. R. WASHBURN[2]

Communicated by D. G. Luenberger

Abstract. The problem considered is as follows: m resources are to be allocated to n activities, with resource i contributing linearly to the potential for activity j according to the coefficient $E(i,j)$. The objective is to minimize some nonlinear function of the potentials. If the objective function is sufficiently well behaved, the problem can be solved in finitely many steps using the method described in this paper.

Key Words. Nonlinear allocation problems, networks, optimization.

1. Introduction

Consider the following nonlinear program:

$$\text{(NLP)} \quad \min \quad f(y_1, \ldots, y_n), \tag{1}$$

$$\text{s.t.} \quad \sum_{i=1}^{m} E(i,j)x_{ij} = y_j, \qquad 1 \le j \le n, \tag{2a}$$

$$\sum_{j=1}^{n} x_{ij} = b_i, \qquad 1 \le i \le m, \tag{2b}$$

$$x_{ij} \ge 0, \qquad 1 \le i \le m, \qquad 1 \le j \le n. \tag{2c}$$

It is assumed in (NLP) that

$$b_i \ge 0, \qquad \text{for all } i,$$

and

$$E(i,j) \ge 0, \qquad \text{for all } (i,j),$$

[1]The author thanks S. Toi Lawphongpanich for several helpful references and suggestions.
[2]Professor, Operations Research Department, Naval Postgraduate School, Monterey, California.

with at least one positive $E(i,j)$ in each row i. It is further assumed that $f(y)$ is continuous and decreasing in each of its arguments on some subset S of R^n that includes the nonnegative orthant. y represents the vector $(y_1, \ldots, y_n)$; similarly, x represents the collection of all mn of the x_{ij}, etc. The components of y will be called potentials, one for each column of E. Potentials are not logically necessary in defining (NLP), since the expressions defining y could simply be substituted into $f(y)$, but the generous notation will prove useful in the sequel. The components of b will be called resources, one for each row of E, and the components of x will be called allocations. Thus, (NLP) is the problem of allocating m resources to minimize a function of n potentials.

(NLP) could be interpreted as a search problem by assuming that the search is to be conducted over a fixed time period by m distinct types of search resources, with b_i units of search resource i available over the period. Specifically, assume that a single target is located with probability p_j in one of n regions, that region j has area A_j, and that the reward for finding the target in region j is V_j. Assume further that the amount of area swept per unit effort by a searcher of type i in region j is s_{ij}. If x_{ij} is the allocation of search effort of type i to region j, then the total area swept in region j is $\sum_{i=1}^{m} x_{ij} s_{ij}$. If the search in each region is random (Ref. 1), then the probability of no detection is $\exp(-y_j)$, where

$$y_j = \left(\sum_{i=1}^{m} x_{ij} s_{ij} \right) \Big/ A_j;$$

so, in this case

$$E(i,j) = s_{ij}/A_j.$$

The average value not found is then

$$f(y) = \sum_{j=1}^{n} V_j p_j \exp(-y_j),$$

to be minimized. The case $m = 1$ already has specially-tailored solution procedures (Ref. 2). This specialization of (NLP) where the objective function is a positively weighted sum of exponentials will be referred to as the exponential case. It will be the subject of the implementation described in Section 5.

The US Air Force's Heavy Attack model (Refs. 3–4) is also essentially the exponential case, with resources being aircraft sorties and potentials being the average number of targets of type j killed, were it not for the fact that some of the potential will be wasted in attacking targets already dead.

The exponential case seems to arise frequently in military problems where weapons are to be assigned to targets. Reference 5 is another example.

(NLP) has a continuous objective function and a compact, nonempty feasible set, so a solution certainly exists (Ref. 6). A variety of general techniques is available for solving problems like (NLP) where the constraints are linear (Refs. 7–10). By introducing an extra node to collect all of the potentials, the constraints of (NLP) can be expressed as those of a generalized network where $E(i,j)$ is the gain on edge (i,j), so specialized nonlinear techniques such as those of Ref. 11 are applicable. Rockafellar (Ref. 12) discusses several methods for solving (NLP) when $f(\cdot)$ is convex and separable. Reference 13 refers to the case $m=1$ as the resource allocation problem; several algorithms are offered, depending on the nature of $f(\cdot)$ and whether allocations are required to be integer valued.

Our interest here is in developing a solution procedure for (NLP) that is finite (unlike most of the methods above) in the sense of requiring only a finite number of solutions of certain generally nonlinear equations. In the exponential case, this will amount to saying that only a finite number of logarithms and exponentials is required. Charnes and Cooper (Ref. 14) describe such an algorithm for the exponential case when $m=1$. The exponential case is also studied by the authors of Ref. 15, who develop solution methods for $m=1$ and 2. Lebedev (Ref. 16) considers a generalization of (NLP) where the objective function contains linear terms independent of y; these extra terms are a valuable modeling capability, but would frustrate the efficient tree-based computational approach that we have in mind. Danskin (Ref. 17) offers an inductive algorithm for the case where $f(\cdot)$ is separable, and Ref. 18 deals with the existence of cycles and unique solutions in the separable case. Efficient solution techniques require restrictive assumptions about m and especially about $f(\cdot)$, as the above references attest and as the reader may observe in the sections that follow.

2. Basic Feasible Solutions

For any feasible x to (NLP) one can form an associated undirected bipartite graph G with $m+n$ nodes and an edge (i,j) connecting row node i to column node j if and only if $x_{ij}>0$. If G has no cycles (Ref. 19), then x will be called acyclic. If G has no edge (i,j) for which $E(i,j)=0$, then x will be called conservative.

Theorem 2.1. Given the assumptions of the first paragraph of Section 1, (NLP) has an optimal solution for which x is acyclic and conservative.

Proof. Assume that x is optimal. We will show that, if x has a cycle, there is an alternate optimal solution x' with one or more fewer positive variables than x. By repeating this operation, an acyclic solution must eventually be found. If at any point x is not conservative, simply shift the offending allocation x_{ij} to some other column J, where $E(i, J) > 0$; by assumption, such a column can always be found. This will not change the number of positive variables and cannot decrease any potential, so the result of this shift will still be optimal.

If the graph G associated with x has a cycle, then it must also have some subcycle for which the edges are all distinct. Let S be the set of edges in that subcycle, and let RN and CN be the sets of row nodes and column nodes, respectively, involved in S. For $i \in RN$, let C_i be the subset of S incident to i; for $j \in CN$, let R_j be the subset of S incident to j. For some column J in CN, consider the following linear program with a variable u_e for every edge $e \in S$:

$$
\begin{aligned}
\text{(LP)} \quad \max \quad & \sum_{e \in R_J} u_e E(e), \\
\text{s.t.} \quad & \sum_{e \in R_j} u_e E(e) = \sum_{e \in R_j} x_e E(e), \qquad j \in CN - J, \\
& \sum_{e \in C_i} u_e = \sum_{e \in C_i} x_e, \qquad i \in RN.
\end{aligned}
$$

The number of edges in S is necessarily at least 4 and even. Let it be $2L$, where $L \geq 2$. CN and RN can have at most L nodes each, so (LP) is a linear program with $2L$ variables and at most $2L - 1$ constraints. Therefore there is an optimal basic solution u_e of (LP) for which at least one variable is 0 (Ref. 20). Let x' be the result of allocating u_e for edges $e \in S$ or x_e otherwise. Then, x' is feasible in (NLP) and at least as good as x, since all potentials are the same except that possibly the potential for column J is improved. Therefore, x' is optimal in (NLP). Since x' has at least one less positive variable than x, we must eventually discover an optimal acyclic solution by this technique. □

A major advantage of the acyclic solutions is that there can be at most $n + m - 1$ positive allocations x_{ij}. To see this, consider that G must consist of a number of connected components $T_1, \ldots, T_K$, each of which is not connected to any of the others. There may be components that consist of a single isolated node and no edges. Since each component is connected and has no cycles, it is a tree and G is therefore a forest. If T_k has r_k nodes, it must have exactly $r_k - 1$ edges (Ref. 19), even if $r_k = 1$. Therefore, the total number of edges is $n + m - K$, which is at most $n + m - 1$.

Definition 2.1. Any bipartite graph G with the following properties is a basis for (NLP):

(i) the nodes of G are all of the $m+n$ row and column nodes;
(ii) row node I is isolated if and only if $b_I=0$;
(iii) if $E(i,j)=0$, then there is no edge connecting (i,j);
(iv) there are no cycles.

If the component trees of G are $T_1, \dots, T_K$, then R_k and C_k will denote the rows and columns of T_k. Thus, $R_1, \dots, R_K$ is a partition of $\{1, \dots, m\}$, and $C_1, \dots, C_K$ is a partition of $\{1, \dots, n\}$.

In this paper, a positive vector v will be a vector all of whose components are positive; in symbols, $v>0$. For each basis G, there are positive m-vectors and n-vectors λ and μ such that

$$\lambda_i = E(i,j)\mu_j, \qquad \text{for edges } (i,j) \text{ in } G. \tag{3}$$

These multiplier vectors can be easily computed by first arbitrarily defining $\mu_j>0$ or $\lambda_i>0$ for some node i or j in component k, and then using the contagion principle that (3) uniquely defines multipliers at nodes that are neighbors of nodes where multipliers are already defined [recall that $E(i,j)>0$ for edges (i,j) in G, so division by $E(i,j)$ is permissible in (3)]. The multiplier vectors are of use in dealing with the following question: "For what potentials y can feasible allocations x be found?" Lemma 2.1 provides an answer.

Lemma 2.1. Suppose that $T_1, \dots, T_K$ are the component trees of a basis G for (NLP), that λ and μ are associated multiplier vectors, and that $x_{ij}=0$ unless $(i,j)\in G$. Then:

(A) if (2a) and (2b) hold,

$$\sum_{j\in C_k} \mu_j y_j = \sum_{i\in R_k} \lambda_i b_i, \qquad k=1, \dots, K; \tag{4}$$

(B) if (4) holds, there is a unique solution x to (2a) and (2b).

Proof. Since $\lambda_i = \mu_j E(i,j)$, whenever $x_{ij}\neq 0$, substitute λ_i/μ_j for $E(i,j)$ in (2a) to obtain

$$\sum_{i\in R_k} \lambda_i x_{ij} = \mu_j y_j, \qquad j\in C_k. \tag{5}$$

Also, multiply both sides of (2b) by λ_i to obtain

$$\sum_{j\in C_k} \lambda_i x_{ij} = \lambda_i b_i, \qquad i\in R_k. \tag{6}$$

If x satisfies (5) and (6), it follows upon summing (5) for $i \in R_k$ and (6) for $j \in C_k$ that (4) must hold, since the sums are identical. This establishes Part (A). Since (5) and (6) for $(i,j) \in T_k$ are the constraints of a transportation problem with basis T_k, supplies $\lambda_i b_i$, and demands $\mu_j y_j$, the quantities $\lambda_i x_{ij}$ are uniquely determined (Ref. 20). Since $\lambda_i > 0$, the same is true of the allocations x_{ij}. □

Given a basis G and associated multiplier vectors λ and μ, consider the following nonlinear program, where (2a) and (2b) are replaced by (4):

$$(\mathrm{N}(G)) \quad \min \quad f(y),$$
$$\text{s.t.} \quad (4) \text{ and } y_j \geq 0, \qquad j = 1, \ldots, n.$$

The constraints of (N(G)) are compact, so a minimum certainly exists. (N(G)) is simpler than (NLP) because allocations are not involved, so one might hope to solve (NLP) by guessing a basis G, solving (N(G)) for y, and then solving (5)–(6) for x. For an arbitrary basis, some components of x might be negative, but nonetheless we have the following theorem.

Theorem 2.2. If $f(y)$ is convex on the nonnegative orthant of E_n and nonincreasing in each component of y, then there is some basis G and potentials y such that:

(i) y is an optimal solution of (N(G));
(ii) there is an x such that (x, y) is an optimal solution of (NLP).

Proof. Let (x, y) be an optimal, acyclic conservative solution of (NLP), let G be the corresponding basis, and let λ and μ be multipliers for G. By Lemma 2.1, y must be feasible in (N(G)). Let y' be optimal in (N(G)), so that $f(y') \leq f(y)$. By Lemma 2.1, there exist allocations x' on G such that (x', y') is feasible in (NLP), except that possibly x' does not satisfy the nonnegativity constraints. Let $\epsilon > 0$ be such that $\epsilon x' + (1-\epsilon)x$ is nonnegative; there exists such an ϵ, because by assumption $x > 0$ on G. Let

$$x^* = \epsilon x' + (1-\epsilon)x \quad \text{and} \quad y^* = \epsilon y' + (1-\epsilon)y.$$

Then (x^*, y^*) is feasible in (NLP), and

$$f(y^*) \leq \epsilon f(y') + (1-\epsilon) f(y),$$

by convexity. If $f(y') < f(y)$, then $f(y^*) < f(y)$, which contradicts the optimality of (x, y) in (NLP). Therefore $f(y') = f(y)$, and y is optimal in (N(G)), which completes the proof. □

Under the premises of Theorem 2.2, there is some basis G such that the optimal solution of (N(G)) corresponds in a direct way to an optimal solution of (NLP). The latter could therefore be solved by exhaustively considering all bases, solving (N(G)) for each one, and eliminating those for which there is no optimizing y with corresponding nonnegative allocations x. However, it would be useful to have a procedure for producing a sequence of improving bases that terminates with an optimal basis, the idea being to avoid consideration of many bases that are feasible but not optimal, as in the simplex method of linear programming. To accomplish this, it will be necessary to make further assumptions about $f(\cdot)$.

3. Pivoting Procedure for Differentiable Functions

From now on, we assume that $f(\cdot)$ has continuous first derivatives, in addition to being strictly convex and decreasing in the nonnegative orthant of E_n. (N(G)) then has a unique optimal solution y for each basis G on account of the strict convexity assumption, and by Lemma 2.1 the allocations x are also unique. If $x>0$ on G, then G itself will be called positive; and if (x, y) is optimal in (NLP), then G will be called optimal. If G is such that $x \geq 0$, then a positive basis G' can always be obtained from G by simply dropping all edges from G for which $x_{ij}=0$, and y will still be optimal in (N(G$'$)). The goal of SIMPLX is to find a positive basis that is optimal.

The Karush–Kuhn–Tucker (KKT) conditions are necessary and sufficient (Ref. 8) for optimality in either (NLP) or (N(G)) on account of the convexity assumption. Letting $f_j(y)$ be the derivative of $f(y)$ with respect to y_j, the KKT conditions for y to be an optimal solution of (N(G)) are that there should exist positive Lagrange multipliers α_k, one for each component T_k of G, such that y is feasible in (N(G)) and

$$f_j(y)+\alpha_k\mu_j \geq 0, \qquad j \in C_k, \qquad k=1, \ldots, K, \tag{7}$$

where λ and μ are multipliers for G, with equality holding if $y_j>0$. If G is a positive basis, then (7) can be strengthened to

$$f_j(y)+\alpha_k\mu_j=0, \qquad j \in C_k, \qquad k=1, \ldots, K. \tag{8}$$

Note that, when T_k consists of the isolated column j, the system (4), (8) requires $y_j=0$ [since R_k is empty in (4)] and $\alpha_k=-f_j(y)/\mu_j$.

Now, suppose that (α, y) is a solution to (4), (8) for some positive basis G, and let λ^* and μ^* be defined by

$$\lambda_i^*=\alpha_k\lambda_i, \qquad i \in R_k, \tag{9}$$

$$\mu_j^*=\alpha_k\mu_j, \qquad j \in C_k. \tag{10}$$

If x is the allocation corresponding to y, then (x, y, λ^*, μ^*) satisfy all of the KKT conditions for (NLP), with λ^* and μ^* being the Lagrange multipliers for (2b) and (2a), respectively, except possibly for the requirement that

$$\lambda_i^* \geq E(i,j)\mu_j^*, \qquad \text{for } 1 \leq i \leq m \text{ and } 1 \leq j \leq n.$$

If this requirement is also satisfied, then G is optimal.

Suppose then that

$$\lambda_I^* < E(I, J)\mu_J^*,$$

with I and J necessarily being such that (I, J) is not in G. If $b_I = 0$, then λ_I^* can simply be increased to reverse the inequality; so, assume $b_I > 0$. Since $\lambda_I^* - E(I, J)\mu_J^*$ is the derivative of the optimized objective function with respect to x_{IJ} (Ref. 8), hereafter the reduced cost on nonbasic edge (I, J), the objective function can be decreased by increasing x_{IJ} from 0 to some positive level Δ that certainly does not exceed b_I. This leads to consideration of programs (NLP(Δ)) and (N(G, Δ)), which are the same as (NLP) and (N(G)), except that x_{IJ} is fixed at Δ, rather than 0. The latter program is

$$\text{(N(G}, \Delta)) \quad \min \; f(y),$$

$$\text{s.t.} \quad y_j \geq \Delta E(I, J)\delta_{jJ}, \qquad j = 1, \ldots, n, \tag{11a}$$

$$\sum_{j \in C_k} \mu_j (y_j - \Delta E(I, J)\delta_{jJ}) = \sum_{i \in R_k} \lambda_i (b_i - \Delta\delta_{iI}), \qquad k = 1, \ldots, K, \tag{11b}$$

where δ_{ij} is 1 if $i = j$, otherwise 0. Equations (8) and (11) are the necessary and sufficient KKT conditions for y to be an optimal solution of (N(G, Δ)).

The SIMPLX algorithm for solving (NLP) begins with a positive basis G_1, and produces an improving sequence of positive bases $G_1, G_2, \ldots, G^*$ that terminates with an optimal basis. The process of obtaining G_{i+1} from G_i will be called a pivot. If G_i is not optimal, there is some edge (I, J) not in G_i with a negative reduced cost. Basically, what happens in a pivot is that one gradually increases Δ, dropping edges from the basis as allocations fall to 0, until finally edge (I, J) is added to the basis when $\Delta = b_I$ or when the reduced cost on edge (I, J) falls to 0. The algorithm below describes the pivoting operation precisely.

SIMPLX Algorithm.

Step 1. Let H_1 be the input positive basis, let $\Delta_1 = 0$, and let $p = 1$.

Step 2. With $G = H_p$, solve (8), (11) for (α, y), and let the solution be $(\alpha^p(\Delta), y^p(\Delta))$ for $\Delta_p \leq \Delta \leq b_I$.

Step 3. Use Lemma 2.1 to obtain $x^p(\Delta)$, the unique allocations to H_p that, in conjunction with $x_{IJ}=\Delta$, have potentials $y^p(\Delta)$.

Step 4. Let Δ^* be such that $x^p(\Delta)>0$ on H_p for $\Delta_p \leq \Delta \leq \Delta^*$, but $x_{AB}(\Delta^*)=0$ for some edge (A, B) in H_p.

Step 5. Let $RC^p(\Delta)=\alpha^p_{k1}(\Delta)\lambda_I-\alpha^p_{k2}(\Delta)E(I, J)\mu_J$, where I and J are in components $k1$ and $k2$ of H_p, for $\Delta_p \leq \Delta \leq \Delta^*$.

Step 6. If $RC^p(\Delta^*)<0$, let $\Delta'=\Delta^*$. Otherwise, let Δ' be the smallest Δ exceeding Δ_p such that $RC^p(\Delta)=0$.

Step 7. Delete from H_p all edges for which $x^p(\Delta')$ has null allocations to obtain H_{p+1}.

Step 8. Test if $RC(\Delta')<0$ and $\Delta'<b_I$. If so, let $\Delta_{p+1}=\Delta'$, increment p, and return to Step 3. If not, go to Step 9.

Step 9. Add edge (I, J) to H_{p+1} to obtain the output positive basis H^*, and stop.

Theorem 3.1. If $f(\cdot)$ is as described at the beginning of this section and if the input positive basis has a nonbasic edge (I, J) with negative reduced cost, then all of the operations of the above algorithm are well defined, and the output is a better positive basis than the input.

The proof of Theorem 3.1 can be found in the Appendix. Additional observations about pivoting are given below.

(a) The superscripts on x, y, α, RC, while useful in the proof of Theorem 3.1, are not logically necessary. In any given pivot, $x(\Delta)$, $y(\Delta)$, $\alpha(\Delta)$, $RC(\Delta)$ can be defined to be continuous functions over $[0, \Delta']$, where Δ' is the final value in Step 9. The superscripts will be omitted from here on except in the Appendix.

(b) The multipliers (λ, μ) for the input basis are multipliers for every basis H_p in a given pivot, but not for the output H^*, because $\lambda_I<E(I, J)\mu_J$. If J is in component T of the basis to which (I, J) is added, however, then a set of multipliers for H^* can be obtained by multiplying all multipliers for rows and columns of T by the factor $\lambda_I/(E(I, J)\mu_J$.

(c) Since the objective function strictly improves with each new basis considered, no basis can be considered twice. Cycling is impossible.

(d) The pivoting operation described above differs from the comparable linear programming operation in that the number of edges that have to leave the basis in order for (I, J) to enter is not always 1; it may be 0 or it may be larger than 1. It differs essentially in this respect from the convex simplex method (Ref. 7). In the CSM, the number of basic variables is constant, and every pivot replaces one basic variable with one nonbasic variable. When a local minimum is encountered in CSM, the corresponding variable remains nonbasic, but positive. In SIMPLX, the number of basic variables is not constant, but the idea that nonbasic variables are zero is preserved, along with the one-to-one relationship between bases and solutions. This relationship implies that the number of pivots is bounded by the number of bases in SIMPLX, whereas the number of pivots required by the CSM may be unbounded.

Theorem 3.1 and the above observations establish that (NLP) can be solved by repeated pivoting, with the number of bases considered in all pivots being bounded by the number of positive bases, itself finite. This fact is not in itself useful, however; after all, (NLP) could be solved in one step if the step were "find (x, y, λ^*, μ^*) such that the KKT for (NLP) are satisfied." The SIMPLX procedure will be of value only if the solution of the system of equalities (8), (11) for (α, y) is, for some reason, simple enough to justify repeated solution.

4. Separable Case

In this section, $f(\cdot)$ is assumed to be separable; that is,

$$f(y) = \sum_{j=1}^{n} h_j(y_j). \tag{12}$$

To meet the assumptions of Theorem 3.1, it is also assumed that each $h_j(\cdot)$ is a decreasing, differentiable, strictly convex function on $[0, \infty)$. Then,

$$f_j(y) = h_j'(y_j),$$

where $h_j'(\cdot)$ is the derivative of $h_j(\cdot)$. Being monotone, $h_j'(\cdot)$ has an inverse function $g_j(\cdot)$, and (8) can be written as

$$y_j = g_j(-\alpha_k \mu_j), \qquad j \in C_k \text{ and } k = 1, \ldots, K. \tag{13}$$

In the pivot that terminates when (I, J) is admitted to the basis, let

$$s_k = \mu_J E(I, J)\{J \in C_k\} - \lambda_I \{I \in R_k\}, \qquad k = 1, \ldots, K, \tag{14}$$

where the $\{\cdot\}$ factors are 1 if the enclosed statement is true, else 0, and where C_k and R_k are as usual the rows and columns of the kth component of the basis. Substitute (13) and (14) into (11b) to obtain

$$\sum_{j \in C_k} \mu_j g_j(-\alpha_k \mu_j) = \sum_{i \in R_k} \lambda_i b_i + s_k \Delta, \qquad k = 1, \ldots, K. \tag{15}$$

Solution of (15) for α amounts to solution of K equations, each a function of one variable. Solution of such equations is simple enough that SIMPLX might be an attractive method even if the equations had to be solved numerically, but of course the best case for SIMPLX will be when there is a closed-form solution for α. The best example of this is undoubtedly the exponential case.

5. Implementation in the Exponential Case

This is a special separable case where

$$h_j(y_j) = V_j \exp(-y_j);$$

any coefficients multiplying y_j can be absorbed into the coefficients $E(i,j)$. In this case,

$$g_j(-\alpha_k \mu_j) = \log(V_j/\mu_j) - \log(\alpha_k), \qquad 0 < \alpha_k \leq V_j/\mu_j.$$

Let

$$A_k = \sum_{j \in C_k} \mu_j, \qquad B_k = \sum_{j \in C_k} \mu_j \log(V_j/\mu_j), \qquad D_k = \sum_{i \in R_k} \lambda_i b_i.$$

Then $A_k > 0$, and the solution of (15) for α_k is

$$\log(\alpha_k(\Delta)) = (B_k - D_k - s_k\Delta)/A_k. \tag{16}$$

Substitution of (16) into (13) reveals that $y_j(\Delta)$ is a linear function of Δ, with the slope being s_k/A_k. It follows that $x_{ij}(\Delta)$ is also a linear function of Δ, so that Δ^* in pivot Step 4 can easily be computed by comparing ratios. Now, let

$$F = \lambda_I/(E(I, J)\mu_J)$$

be the factor needed to calculate new multipliers when edge (I, J) is finally added. Then, $RC(\Delta') = 0$ in Step 6 if and only if

$$\log(\alpha_{k1}(\Delta')) + \log(F) = \log(\alpha_{k2}(\Delta')), \tag{17}$$

a linear equation in Δ'. Equation (17) has no solution if $k^1 = k^2$, in which case $\Delta^* = \Delta'$ in Step 6. If $k^1 \neq k^2$, then the coefficient of Δ' is positive (negative) on the left (right) side of (17); so, solve (17) for Δ' and, if $\Delta' > \Delta^*$, change Δ' to Δ^*. These are all simple operations. In no step of the pivoting procedure is it necessary to solve anything more complicated than a linear equation in one unknown. In particular, no line searches on Δ are required.

A FORTRAN 77 program (also called SIMPLX) has been written to implement SIMPLX in the exponential case. SIMPLX exploits the forest

Table 1. Values of $E(i,j)$ for the example.

	$j=1$	$j=2$	$j=3$	$j=4$
$i=1$	1	2	3	4
$i=2$	3	2	2	1
$i=3$	0	1	0	1

Table 2. Multipliers and allocations for the sixth basis.

	$\mu=107.81$	$\mu=14.93$	$\mu=161.71$	$\mu=7.46$
$\lambda=29.85$		1.602		1.398
$\lambda=323.43$	0.743		1.257	
$\lambda=14.93$		1.000		

Table 3. Multipliers and allocations for the seventh basis.

	$\mu=21.73$	$\mu=48.89$	$\mu=32.59$	$\mu=24.55$
$\lambda=97.78$		1.009	0.890	1.101
$\lambda=65.19$	1.276		0.724	
$\lambda=48.89$		1.000		

Table 4. Multipliers and allocations for the eighth basis.

	$\mu=24.99$	$\mu=37.49$	$\mu=24.99$	$\mu=18.74$
$\lambda=74.97$		0.371	1.461	1.168
$\lambda=74.97$	1.230	0.770		
$\lambda=37.49$		1.000		

structure of the basis to minimize storage and facilitate computations. The forest is actually represented as a single tree by introducing a ground node to which all of the components of the basis are connected through a root node in each component. Most of the operations associated with tree manipulation are similar to those encountered in transshipment problems, well covered in Ref. 21. For example, the allocations are stored in array X, with $X(k)$ representing the allocation from node k to its predecessor if k is a row, or from the predecessor to k if k is a column. All arrays are of length $n+m+1$ (node length), with the extra node being column 0 (see below). The forest always has $n+m+1$ edges as well as $n+m+1$ nodes, since each component has an edge connecting it to ground.

The initial feasible basis consists of making nonzero allocations only to node 0, an artificial column with a very small value. This is equivalent

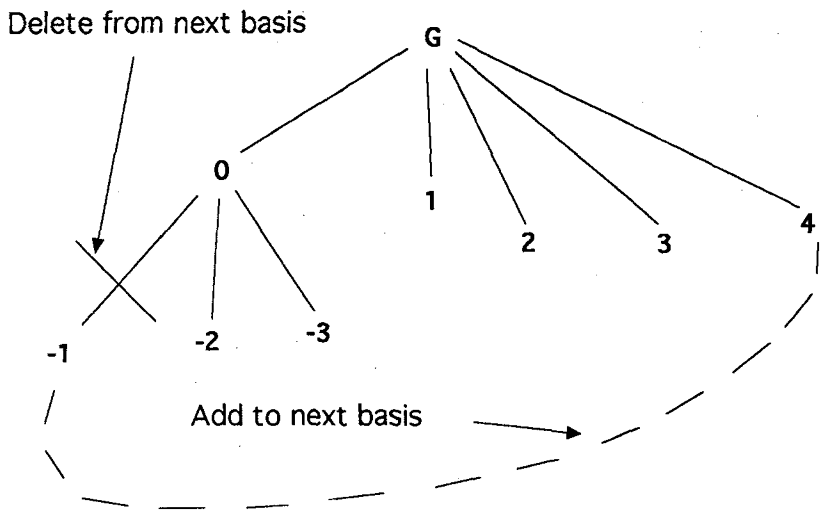

Fig. 1. Initial basis for the example.

to starting with all potentials for real columns being 0. The computational reward for beginning with a more advanced starting point appears to be small.

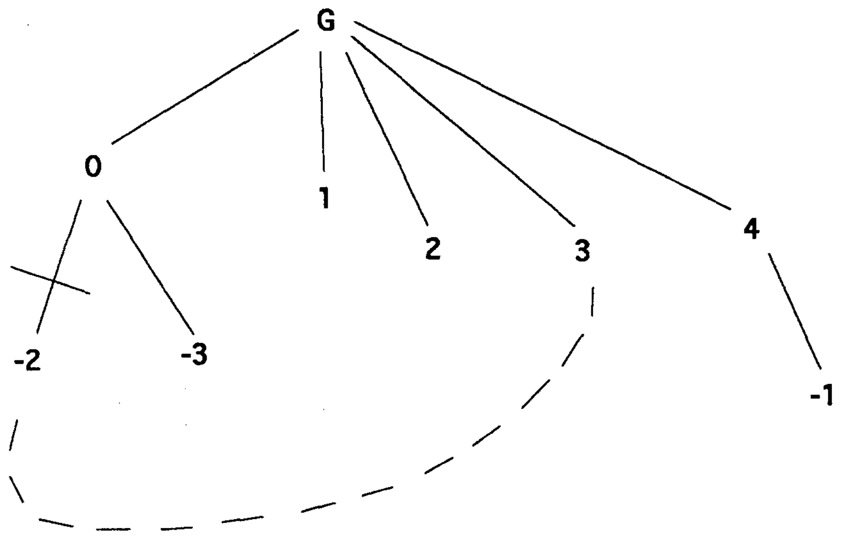

Fig. 2. Second basis for the example.

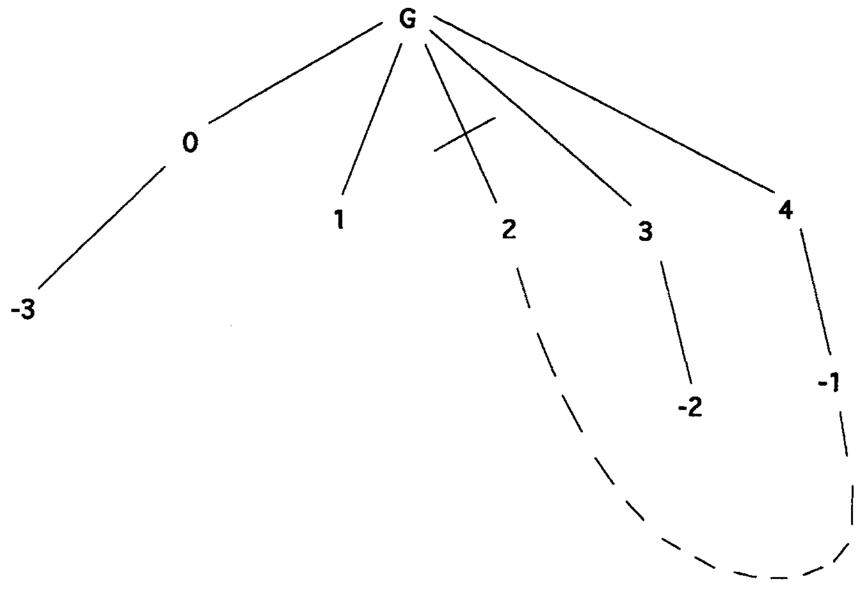

Fig. 3. Third basis for the example.

An example: suppose that

$$(b_i) = (3, 2, 1), (v_j) = (1000, 1000, 2000, 2000),$$

and that $E(i, j)$ is as shown in Table 1. The initial basis is shown in Fig. 1 as a tree where row nodes are distinguished from column nodes by the use of negative numbers, and where the ground node is represented by G. Application of the pivot algorithm results in deleting the $(-1, 0)$ arc and adding the $(-1, 4)$ arc, resulting in the basis shown in Fig. 2, etc. The process terminates after the eighth basis, which is optimal. The multipliers and allocations for the last three bases are shown in Tables 2–4. Although allocations are stored in a node-length array in SIMPLX, Tables 2–4 show them in the more familiar matrix form. Not counting bases involving the artificial node 0, there are 1336 bases when $m = 3$ and $n = 4$. SIMPLX considers four of these, plus four more involving column 0, before arriving at an optimal solution. The objective function values for the last three iterations are 291.9, 127.8, and 106.2 (the objective function is just the sum of all the μ_j's in the exponential case). Every pivot requires a single edge deletion in this example, as is typical of small problems.

SIMPLX has been tested on randomly generated problems where $E(i, j)$ is exponentially distributed with mean 1, V_j is uniformly distributed on [0, 1], and b_i is uniformly distributed on [0, B], where $B = N/M$. Making $B = N/M$

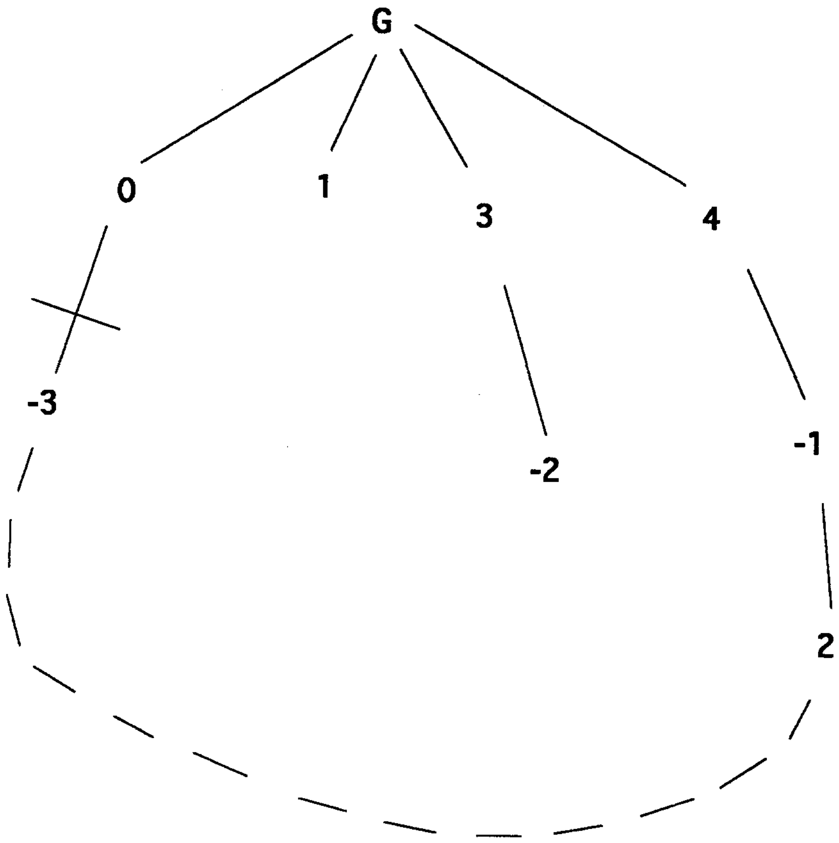

Fig. 4. Fourth basis for the example.

keeps the average potential at 1/2 when all resources are evenly divided over the real columns. When B is very small, the objective function is essentially linear; when B is very large, the objective becomes the maximization of the minimum potential; the selection $B=N/M$ was designed to avoid these linear extremes. M and N were varied from 4 to 100, with the number of nonlinear operations (logarithms and exponentials) NLE and the CPU time T on the Naval Postgraduate School Amdahl 5500 Model 700A Mainframe being recorded for each of 100 cases. NLE and T were then regressed logarithmically on M and N, the resulting regressions being

$$T=(2.0\,\mu\text{sec})M^{1.39}N^{1.42}, \qquad R^2=99\%, \tag{18}$$

$$\text{NLE}=(2.32)M^{0.534}N^{0.677}, \qquad R^2=99\%. \tag{19}$$

The largest problems considered were with $M=100$ and $N=100$, where the

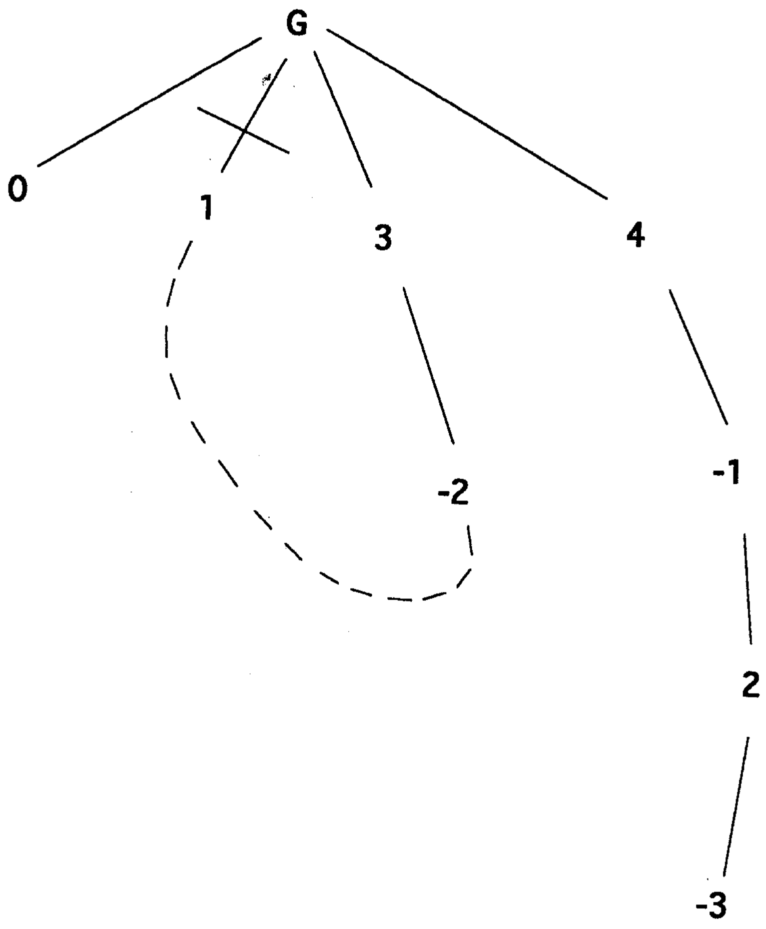

Fig. 5. Fifth basis for the example.

CPU time was in all cases less than a second. Such times are trivial if only one problem needs to be solved, but there are a variety of reasons why solution of (NLP) might be a frequently called subroutine in a larger program. The FAB algorithm for solving moving-target search problems (Ref. 22) is an example.

SIMPLX times were compared with the solution times of MINOS 5.2, a general purpose nonlinear solver, using the GAMS (Ref. 23) interface on the same computer. The MINOS CPU time was taken to be resource usage as reported by GAMS. On a variety of randomly generated test problems, the MINOS/SIMPLX time ratio varied from 24 on a problem with $(M, N) = (50, 90)$ to 78 on a problem with $(M, N) = (10, 10)$, with the average being about 40. The ratio depends on the method used to generate the

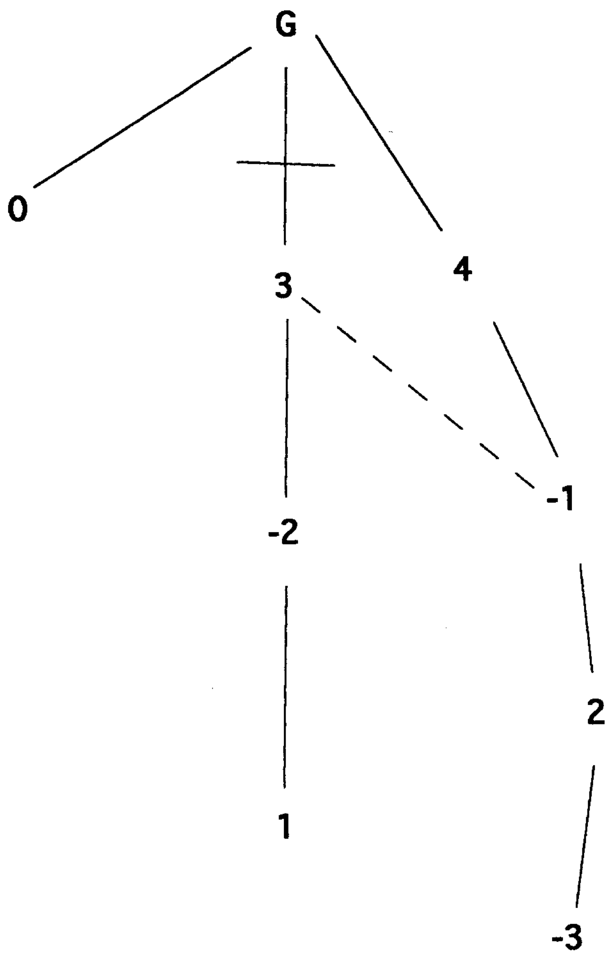

Fig. 6. Sixth basis for the example.

random coefficients and constraints, as well as *M* and *N*. MINOS does not exploit the network structure of the constraints, so the large ratios should not be surprising.

All of the above computations were made in 1992. In 1994, a Beta-Test version of a GAMS interface to the GENOS network optimizer (Ref. 11) was provided to the author by the GAMS Development Corporation. GENOS exploits the network structure of the constraints, so one would expect the GENOS/SIMPLX run time ratio to be substantially smaller than the MINOS/SIMPLX ratio. Tests were performed on a 486 PC running at 33 MHz with about 4 megabytes of free memory available. The largest

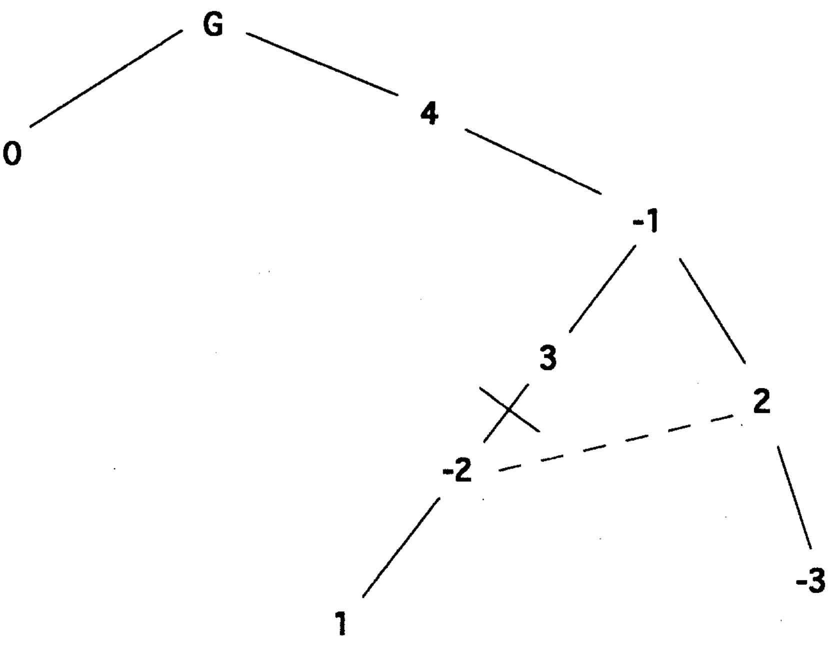

Fig. 7. Seventh basis for the example.

square $m \times m$ problem solvable by GAMS/GENOS on that computer is 23×23. Initial tests on such problems using the primal truncated Newton (PTN) option of GENOS with default optimization parameters resulted in final solutions on the order of 50% larger than optimal. After some trial and error, this problem was fixed by changing the GENOS parameter "NART" from its default of 2 to 50. Five randomly generated 23×23 problems were then solved by both SIMPLX and GENOS. In all cases, except the third, the two solutions agreed to within six significant figures; in the third case, the GENOS solution was 0.4% high. The solution times in seconds were (0.38, 0.33, 0.33, 0.33, 0.33) for SIMPLX and (2.30, 2.20, 6.87, 1.82, 2.09) for GENOS. The GENOS/SIMPLX ratio of average times is $3.06/0.34 = 9$, or only 6 if case 3 is omitted. Evidently, there is a large computational gain by GENOS in relation to MINOS in exploiting the network constraints, as well as by SIMPLX in relation to GENOS by exploiting the form of the objective function. GENOS also offers a simplicial decomposition option, but it did not seem to work as well as PTN; it is likely that the author simply failed to find the right combination of tuning parameters.

The reason for keeping track of the number of exponentials and logarithms in running SIMPLX in 1992 was that it was initially anticipated that

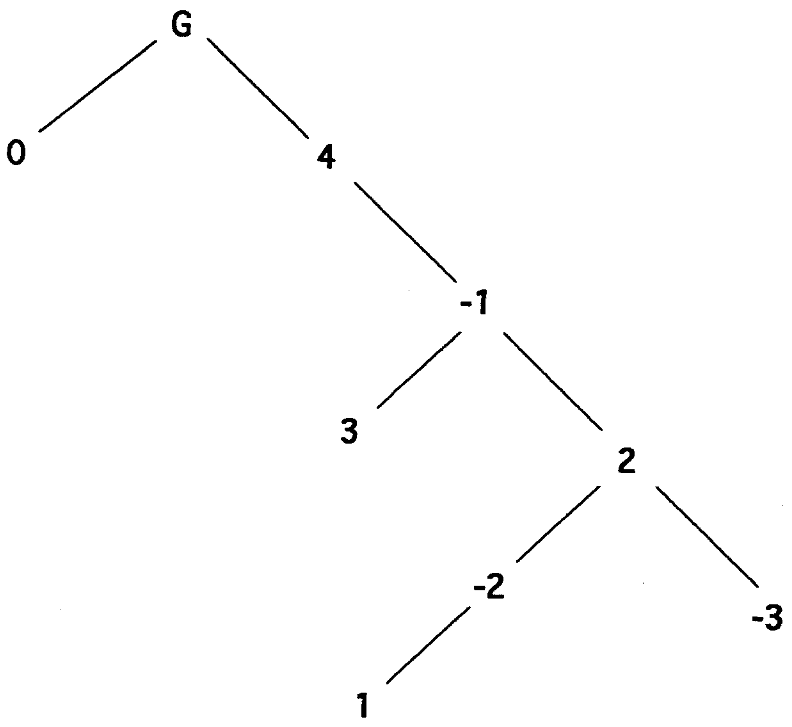

Fig. 8. Eighth, optimal basis for the example.

such nonlinear operations would require relatively large amounts of time. Actually, these operations account for only about 4% of the CPU time; apparently, SIMPLX is occupied mainly with manipulating and scaling the various arrays required to represent the current basis as a forest, rather than with nonlinear operations. This is encouraging for applications where the solution of the system (8), (11) might be more difficult than in the exponential case.

6. Appendix: Proof of Theorem 3.1

The index p is a pass counter, where by pass we mean the sequential execution of Steps 2–7. We will first show by induction that, on each pass,

(a) H_p is a basis;
(b) $x^p(\Delta_p) > 0$;
(c) $\Delta_p < b_I$;
(d) $RC^p(\Delta_p) < 0$.

(a)–(d) are true by assumption on the first pass, so we must show that (a)–(d) are true on pass $p+1$, assuming that there is one, if (a)–(d) are true on pass p.

The feasible set of $(\mathrm{N}(\mathrm{H_p}, \Delta))$ is not empty as long as $\Delta_p \leq \Delta \leq b_I$; so, since $f(\cdot)$ is strictly convex, there exist a unique $(\alpha^p(\Delta), y^p(\Delta))$ satisfying the KKT conditions (8), (11). Furthermore, $y^p(\Delta)$ is a continuous function of Δ on the interval $[\Delta_p, b_I]$, because $f(\cdot)$ is continuous (Ref. 24, Theorem 1.3.2), so the same can be said of $\alpha^p(\Delta)$ in Step 2, $x^p(\Delta)$ in Step 3, and $RC^p(\Delta)$ in Step 5. Since by assumption $x^p(\Delta_p) > 0$, $\Delta^* \geq \Delta_p$ in Step 3, and furthermore $\Delta^* \leq b_I$ because all allocations from row I must total 0 when $\Delta = b_I$. Thus, Δ^* is well defined, and $\Delta^* \in (\Delta_p, b_I]$. Since $RC^p(\Delta)$ is continuous and by assumption $RC^p(\Delta_p) < 0$, Δ' in Step 6 is also well defined and $\Delta' \in (\Delta_p, \Delta^*]$. The edge deletions in Step 7 cannot isolate row I as long as $\Delta' < b_I$, because the basic allocations in row I must sum to $b_I - \Delta'$, which is positive. The test condition after Step 7 thus assures that H_{p+1} is a basis, as well as $\Delta_{p+1} < b_I$, so (a) and (c) must hold on pass $p+1$ if (a)–(d) hold on pass p. It remains to show that (b) and (d) must also hold on pass $p+1$.

By construction $x^p(\Delta_{p+1}) > 0$ on H_{p+1}, but this is not the same thing as property (b), which requires $x^{p+1}(\Delta_{p+1}) > 0$ on H_{p+1}. To prove (b), it suffices to demonstrate that allocations are continuous across pass boundaries. The simplest way to do this is to first establish the same fact about potentials. First note that, by construction, $(x^p(\Delta_{p+1}), y^p(\Delta_{p+1}))$ is feasible in $(\mathrm{NLP}(\Delta_{\mathrm{p+1}}))$, and that H_{p+1} is the basis of $x^p(\Delta_{p+1})$. Therefore $y^p(\Delta_{p+1})$ is feasible in $(\mathrm{N}(\mathrm{H_{p+1}}, \Delta_{\mathrm{p+1}}))$; we omit the proof of this statement, which would closely parallel the proof of the first part of Lemma 2.1. But $(\mathrm{N}(\mathrm{H_p}, \Delta_{\mathrm{p+1}}))$ is a relaxation of $(\mathrm{N}(\mathrm{H_{p+1}}, \Delta_{\mathrm{p+1}}))$, so $y^p(\Delta_{p+1})$ must also be optimal in $(\mathrm{N}(\mathrm{H_{p+1}}, \Delta_{\mathrm{p+1}}))$; that is,

$$y^{p+1}(\Delta_{p+1}) = y^p(\Delta_{p+1}).$$

The allocations $x^p(\Delta_{p+1})$ are null on all edges except those in H_{p+1}, and together with $X_{IJ} = \Delta_{p+1}$, have potentials $y^{p+1}(\Delta_{p+1})$. Since the unique allocations with those properties are $x^{p+1}(\Delta_{p+1})$, it follows that $x^p(\Delta_{p+1}) = x^{p+1}\Delta_{p+1})$; therefore, $x^{p+1}(\Delta_{p+1}) > 0$ on H_{p+1}. Thus, (b) is true on pass $p+1$ if (a)–(d) are true on pass p.

Since row I is not isolated in H_{p+1}, there must be some column j such that $(I, j) \in H_{p+1}$. Edge (I, j) must also be in H_p, since every edge in H_{p+1} is also in H_p. Therefore, $RC^p(\Delta)$ in Step 5 can be rewritten as

$$RC^p(\Delta) = f_J(y^p(\Delta))E(I, J) - f_J(y^p(\Delta))\lambda_I/\mu_j,$$

an expression that is also true if $p+1$ is substituted for p. Since the potentials are continuous across pass boundaries, so is the reduced cost; that is,

$$RC^{p+1}(\Delta_{p+1}) = RC^p(\Delta_{p+1}).$$

The test in Step 8 thus assures that (d) is true on pass $p+1$ if (a)–(d) are true on pass p. This completes the inductive proof described earlier.

We next show that the addition of (I, J) to H_{p+1} in Step 9 does not introduce a cycle. If $\Delta' = b_I$ in Step 9, then there can be no arc (I, j) in H_{p+1}, because such an arc would have a positive allocation, so the addition of (I, J) to H_{p+1} in that case cannot cause a cycle. If $\Delta' < b_I$, then necessarily $RC^p(\Delta') = 0$ on account of the definition of Δ' in Step 6 and the entrance condition for Step 9. This equality could not happen if I and J were in the same component of H_p, since in that case $RC^p(\Delta') < 0$ because $\lambda_I - E(I, J)\mu_J < 0$. Therefore I and J must be in different components of H_p, as well as in the smaller basis H_{p+1}, so adding edge (I, J) to H_{p+1} cannot introduce a cycle. With this fact, it is easy to verify that the output H^* is a basis. Now, let x be the allocation to H^* that consists of $x^{p+1}(\Delta')$ to H_{p+1} and Δ' to edge (I, J), and let

$$y^{p+1}(\Delta_{p+1}) = y^p(\Delta')$$

be the potentials associated with x. Then $x > 0$ on H^*, so H^* is a positive basis. It remains only to verify that

$$f(y^{p+1}(\Delta_{p+1})) > f(y^p(\Delta_p)), \qquad p \geq 1.$$

Since $\Delta^* > \Delta_p$ in Step 4 and $RC^p(\Delta_p) < 0$ in Step 5, it follows that $\Delta_p < \Delta'$ in Step 6. Since $RC^p(\Delta) < 0$, for $\Delta_p \leq \Delta < \Delta'$, it follows that

$$\int_{\Delta_p}^{\Delta'} RC^p(\Delta)\, d\Delta < 0.$$

Since this integral is also $f(y^{p+1}(\Delta_{p+1})) - f(y^p(\Delta_p))$, it follows that the sequence $f(y^p(\Delta_p))$ is strictly decreasing. This completes the proof of Theorem 3.1. □

References

1. Koopman, B., *Theory of Search, Part 2*, Operations Research, Vol. 4, pp. 519–521, 1956.
2. Washburn, A., *Note on Constrained Maximization of a Sum*, Operations Research, Vol. 29, pp. 411–414, 1981.
3. Clasen, R., Graves, G., and Lu, J., *Sortie Allocation by a Nonlinear Programming Model for Determining a Munitions Mix*, RAND Report R-1411-DDPAE, 1974.
4. Bausch, D., and Brown, G., *NDP Fortran and Phar Lap Tools*, Interfaces, Vol. 15, pp. 20–25, 1988.
5. James, R., and Shaw, J., *Iline: An Iterative Algorithm for the Weapon Assignment Problem*, 28th ORSA/TIMS Meeting, New York, New York, 1989.

6. BAZARAA, M., and SHETTY, C., *Nonlinear Programming: Theory and Algorithms*, Wiley, New York, New York, 1975.
7. ZANGWILL, W., *Nonlinear Programming: A Unified Approach*, Prentice-Hall, Englewood Cliffs, New Jersey, Chapter 8, 1970.
8. SIMMONS, D., *Nonlinear Programming for Operations Research*, Prentice-Hall, Englewood Cliffs, New Jersey, Chapter 8, 1975.
9. VON HOHENBALKEN, B., *Simplicial Decomposition in Nonlinear Programming Problems*, Mathematical Programming, Vol. 13, pp. 49–68, 1977.
10. HEARN, D., LAWPHONGPANICH, S., and VENTURA, J., *Restricted Simplicial Decomposition: Computation and Extensions*, Mathematical Programming Study, Vol. 31, pp. 99–118, 1987.
11. AHLFELD, D., DEMBO, R., MULVEY, J., and ZENIOS, S., *Nonlinear Programming on Generalized Networks*, ACM Transactions on Mathematical Software, Vol. 13, pp. 350–367, 1987.
12. ROCKAFELLAR, R., *Network Flows and Monotropic Optimization*, Wiley, New York, New York, Chapter 11, 1984.
13. IBARAKI, T., and KATOH, N., *Resource Allocation Problems*, MIT Press, Cambridge, Massachusetts, 1988.
14. CHARNES, A., and COOPER, A., *The Theory of Search: Optimum Distribution of Search Effort*, Management Science, Vol. 5, pp. 44–50, 1958.
15. BAKHTIN, I., KRASNOSEL'SKII, M., and LEVIN, A., *Finding the Extremum of a Function on a Polyhedron*, USSR Computational Mathematics and Mathematical Physics, Vol. 3, pp. 533–546, 1963.
16. LEBEDEV, S., *A Finite Method for Solving Nonlinear Transportation Problems*, Ekonomika i Matematica Metody, Vol. 1, pp. 71–82, 1965.
17. DANSKIN, J., *The Theory of Max-Min*, Springer Verlag, New York, New York, pp. 85–100, 1967.
18. EINBU, J., *Optimal Allocations of Continuous Resources to Several Activities with a Concave Return Function: Some Theoretical Results*, Mathematics of Operations Research, Vol. 3, pp. 82–88, 1978.
19. BERGE, C., *The Theory of Graphs*, Wiley, New York, New York, p. 152, 1962.
20. BAZARAA, M., and JARVIS, J., *Linear Programming and Network Flows*, Wiley, New York, New York, 1977.
21. BRADLEY, G., BROWN, G., and GRAVES, G., *Design and Implementation of Large-Scale Primal Transshipment Problems*, Management Science, Vol. 24, pp. 1–34, 1977.
22. WASHBURN, A., *Search for a Moving Target: The FAB Algorithm*, Operations Research, Vol. 31, pp. 739–751, 1983.
23. BROOKE, A., KENDRICK, D., and MEERAUS, A., *GAMS: A User's Guide*, Scientific Press, Redwood City, California, 1988.
24. DANTZIG, G., FOLKMAN, J., and SHAPIRO, N., *On the Continuity of the Minimum Set of a Continuous Function*, Journal of Mathematical Analysis and Applications, Vol. 17, pp. 519–548, 1967.