April 1986 LIDS-P-1553

Relaxation Methods for Linear Programs

by

Paul Tseng †

Dimitri P. Bertsekas †

Abstract

In this paper we propose a new method for solving linear programs. This method may be viewed as a generalized coordinate descent method whereby the descent directions are chosen from a finite set. The generation of the descent directions are based on results from monotropic programming theory. The method may be alternately viewed as an extension of the relaxation method for network flow problems [1], [2]. Node labeling, cuts, and flow augmentation paths in the network case correspond to respectively tableau pivoting, rows of tableaus, and columns of tableaus possessing special sign patterns in the linear programming case.

Key words. Elementary vectors, Tucker representations, Painted Index algorithm, epsilon-complementary slackness.

---

*Work supported by the National Science Foundation under grant NSF-ECS-3217668.

†The authors are with the Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA 02139.

1. Introduction

Consider the general linear programming problem with m variables and n homogeneous equality constraints. We denote the constraint matrix for the equality constraints by E (E is a $n \times m$ real matrix), the jth variable by x_j, and the per unit cost of the jth variable by a_j. The problem then has the form :

$$\text{Minimize} \quad \sum_{j=1}^{m} a_j x_j \tag{LP}$$

$$\text{subject to} \quad \sum_{j=1}^{m} e_{ij} x_j = 0 \qquad \forall\ i=1,2,\ldots,n \tag{1}$$

$$l_j \le x_j \le c_j \qquad \forall\ j=1,2,\ldots,m \tag{2}$$

where the scalars l_j and c_j denote respectively the lower and the upper bound for the jth variable and e_{ij} denotes the (i,j)th entry of E . We make the standing assumption that **(LP)** is feasible.

We consider an unconstrained dual problem to **(LP)** : Let p_i denote the Lagrange multiplier associated with the ith constraint of (1). Denoting by x and p the vectors with entries x_j, j = 1,2,...,m, and p_i, i = 1,2,...,n respectively we can write the corresponding Lagrangian function

$$L(x,p) = \sum_{j=1}^{m} (a_j - \sum_{i=1}^{n} e_{ij} p_i) x_j$$

The unconstrained dual problem is then

$$\begin{array}{c} \text{Minimize} \quad q(p) \\ \text{subject to no constraints on } p \end{array} \tag{3}$$

where the dual functional q is given by

$$q(p) \quad = \quad - \min_{l_j \le x_j \le c_j} \quad L(x,p) \tag{4}$$

$$= \quad \sum_{j=1}^{m} \max_{l_j \le x_j \le c_j} \quad \left(\sum_{i=1}^{n} e_{ij} p_i - a_j\right) x_j$$

We will call x the <u>primal vector</u> and p the <u>price vector</u>. The vector t with coordinates

$$t_j \quad = \quad \sum_{i=1}^{n} e_{ij} p_i \qquad \forall\ j = 1,2,\ldots,m \tag{5}$$

is called the <u>tension vector</u> corresponding to p. In what follows we will implicitly assume that the relation (5) holds where there is no ambiguity.

For any price vector p we say that the column index j is :

Inactive	if	$t_j < a_j$
Balanced	if	$t_j = a_j$
Active	if	$t_j > a_j$

For any primal vector x the scalar

$$d_i \quad = \quad \sum_{j=1}^{m} e_{ij} x_j \qquad \forall\ i = 1,2,\ldots,n \tag{6}$$

will be referred to as the <u>deficit</u> of row index i.

The optimality conditions in connection with **(LP)** and its dual given by (3), (4) state that (x, p) is a primal and dual optimal solution pair if and only if

$x_j = l_j$ for each inactive column index j (7)

$l_j \leq x_j \leq c_j$ for each balanced column index j (8)

$x_j = c_j$ for each active column index j (9)

$d_i = 0$ for each row index i (10)

Conditions (7) - (9) are the complementary slackness conditions. Let d be the vector with coordinates d_i (in vector form d = Ex) . We define the total deficit for x to be

$$\sum_{i=1}^{n} |d_i| .$$

The total deficit is a measure of how close x is to satisfying the linear homogeneous constraints (1).

The dual problem (3) can be easily seen to be an unconstrained convex programming problem, and as such its solution motivates the use of nonlinear programming methods. One such method is the classical coordinate descent method whereby at each iteration a descent is made along one of the coordinate directions. This method does not work in its pure form when the cost function is nondifferentiable. We bypass this difficulty by occasionally using directions other than the coordinate directions. The idea is illustrated in the example of Figure 1 where a multi-coordinate direction is used only when coordinate descent is not possible.

To develop the mechanism for generating the multi-coordinate descent directions we will view the problem of this paper in the context of monotropic programming theory [7], [8]. We can write **(LP)** as

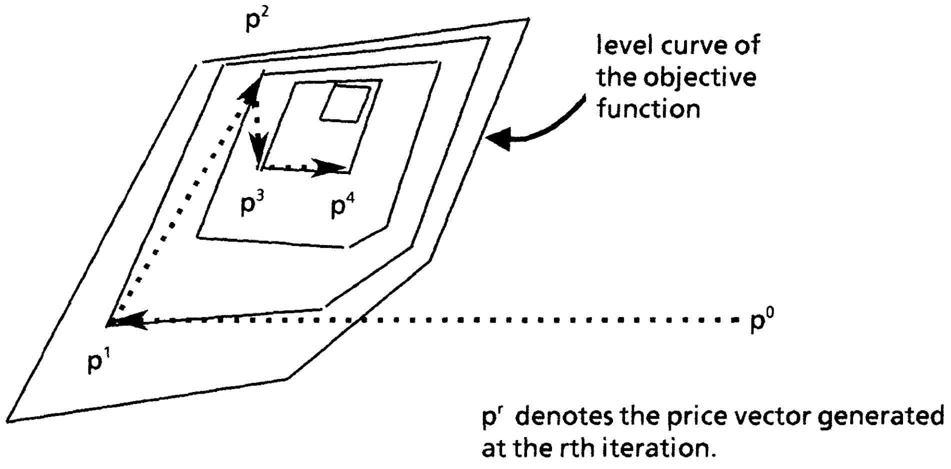

p^2
level curve of the objective function
p^3
p^4
p^0
p^1
p^r denotes the price vector generated at the rth iteration.

Figure 1 Example of convergence using multi-coordinate descents.

$$\text{Minimize} \quad \sum_{j=1}^{m} f_j(x_j) + \sum_{i=1}^{n} \delta(d_i) \tag{P}$$

$$\text{subject to} \qquad (-d,\, x) \in C$$

where $f_j : R \rightarrow (-\infty,\infty]$ is the convex function

$$f_j(x_j) = \begin{cases} a_j x_j & \text{if} \quad l_j \leq x_j \leq c_j \\ +\infty & \text{if} \quad x_j < l_j \text{ or } x_j > c_j \end{cases}$$

$\delta : R \rightarrow (-\infty,\infty]$ is the convex function

$$\delta(\zeta) = \begin{cases} 0 & \text{if } \zeta = 0 \\ \infty & \text{else} \end{cases}$$

and C is the extended circulation space

$$C = \left\{ (-d,\, x) \;\middle|\; \sum_{j=1}^{m} e_{ij} x_j = d_i \quad , \forall\, i = 1,2,\ldots,n \right\} \tag{11}$$

From (4) we see that the dual functional q(p) can be written explicitly as

$$q(p) = g(E^T p) \tag{12}$$

where

$$g(t) = \sum_{j=1}^{m} g_j(t_j) \tag{13}$$

and the convex, piecewise linear functions g_j are given by

$$g_j(t_j) = \begin{cases} (t_j - a_j)l_j & \text{if } t_j \le a_j \\ (t_j - a_j)c_j & \text{if } t_j > a_j \end{cases} \tag{14}$$

(see Figure 2).

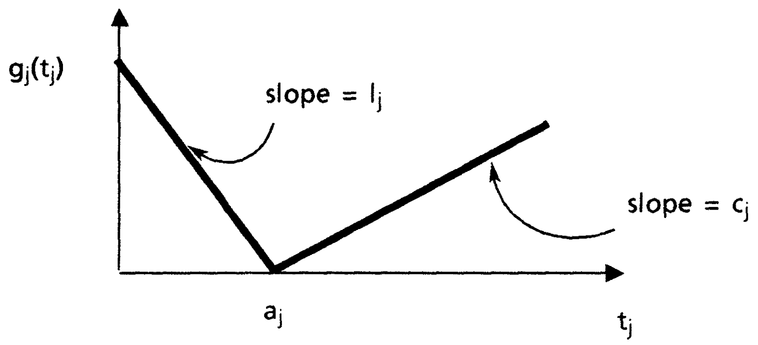


Figure 2 Graph of g_j

Actually g_j is the conjugate convex function of f_j (in the usual sense of convex analysis [6])

$$g_j(t_j) = \sup_{x_j} \{t_j x_j - f_j(x_j)\}$$

as the reader can easily verify (see also [8]).

We now write the dual problem (3) in a form which is symmetric to **(P)**

$$\textit{Minimize} \quad \sum_{j=1}^{m} g_j(t_j) \tag{D}$$

$$\textit{subject to} \quad (p, t) \in C^{\perp}$$

where $C^{\perp}$ is the subspace

$$C^{\perp} = \left\{ (p, t) \;\middle|\; t_j = \sum_{i=1}^{n} e_{ij} p_i \right\} \tag{15}$$

Problems (**P**) and (**D**) are symmetric in that they both involve minimization of a separable function over a subspace, C and $C^{\perp}$ can be easily verified to be orthogonal subspaces, f_j and g_j are conjugates of each other, while the conjugate convex function of δ is the zero function. In fact (**P**) and (**D**) constitute a pair of dual monotropic programming problems as introduced in Rockafellar [7]. It was shown there in a more general setting that these programs have the same optimal value and their solutions are related via the conditions (7)-(10). An important special property of these programs is that at each nonoptimal point it is possible to find descent directions among a finite set of directions -- the elementary vectors of the subspace C [in the case of (**P**)] or the subspace $C^{\perp}$ [in the case of (**D**)]. The notion of an elementary vector of a subspace is dealt with extensively in [8] (see also [7]) where it is defined as a vector in the subspace having minimal signed support (i.e. a minimal number of nonzero coordinates). We are interested in the elementary vectors because they can be very efficiently generated by a tableau pivoting technique and because they provide us with the necessary generalization of coordinate vectors in the price space. In the special case of network flow problems for which the tableau pivoting may be implemented by means of labeling, the generalized coordinate descent approach yields an algorithm that is superior to the primal simplex method, which for many years has been considered as the most efficient method for linear network flow problems [1], [2].

In the next section we give an overview of the relationship between the elementary vectors and certain tableaus, called the Tucker tableaus, and describe a pivoting algorithm, called the Painted Index algorithm, for generating Tucker tableaus with special sign patterns [8]. In Section 3 we characterize the descent directions in terms of the Tucker tableaus and show how to use the Painted Index algorithm to generate dual descent directions. In Section 4 we introduce a class of generalized coordinate descent algorithms for solving (**D**) where descent directions are generated by the Painted Index algorithm. A numerical example is given at the end of Section 4.

In Section 5 we address the issue of finite convergence of these algorithms. In Section 6 we report on computational experience.

2. Tucker Tableau and the Painted Index algorithm

In order to use elementary vectors in our algorithm we need a suitable characterization of the elementary vectors of the extended dual subspace $C^{\perp}$ and a method for generating them. In the special case of network constraints, the elementary vectors of $C^{\perp}$ are characterized by the cutsets of the network. In the general case of arbitrary linear constraints, the elementary vectors of C and $C^{\perp}$ are characterized by the Tucker representations of C and $C^{\perp}$ and to generate them we will use a generalization of node labeling for network problems called the Painted Index algorithm (see [8],Chap. 10) .

We will first give a brief overview of Tucker tableaus and then discuss the algorithm for generating them. Consider a linear homogeneous system

$$Tx = 0$$

where T is a matrix of full row rank. Each column of T has an index and we denote the set of indexes for the columns of T by *J*. Since T has full row rank, we can partition the columns of T into [B N] , where B is an invertible matrix. Then Tx = 0 can be expressed as

$$x_B = -B^{-1}Nx_N \qquad where \ x = \begin{bmatrix} x_B \\ x_N \end{bmatrix}$$

This way of expressing $Tx = 0$ is a Tucker representation of S, where the subspace S is given by $S = \{ x \mid Tx = 0 \}$. Similarly,

$$t_N = (B^{-1}N)^T t_B \qquad \textit{where } t = \begin{bmatrix} t_B \\ t_N \end{bmatrix}$$

is a Tucker representation of $S^\perp$, where $S^\perp$ is the orthogonal complement of S, given by $S^\perp = \{ t \mid t = T^T p \text{ for some } p \}$. The matrix $-B^{-1}N$ is a Tucker tableau. The columns of $-B^{-1}N$ are indexed by the indexes of the columns of N. The rows of $-B^{-1}N$ are indexed by the indexes of the columns of B. (see Figure 3) With respect to a given tableau, an index is basic if its corresponding

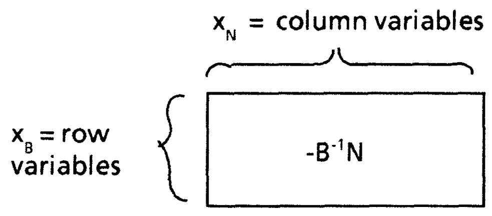


Figure 3 Tucker tableau corresponding to a partition of $Tx = 0$ into $Bx_B + Nx_N = 0$

variable is a row variable and nonbasic otherwise. Clearly the number of distinct tableaus is finite. Furthermore, starting from any tableau, it is possible to generate all Tucker representations of S and $S^\perp$ by a sequence of simplex method-like pivots (see Appendix C for the pivoting rule).

A fundamental relationship exists between the Tucker representations and the elementary vectors of S and $S^\perp$: Each column of a Tucker tableau yields in a certain way an elementary vector of S, and conversely, every elementary vector of S is obtainable from some column of some Tucker tableau. In a similar way, rows of Tucker tableaus correspond to elementary vectors of the dual subspace $S^\perp$:

<u>Proposition 1</u> ([8],Chap.10) For a given Tucker tableau and for each basic index i and nonbasic index j let a_{ij} denote the entry of the tableau in the row indexed by i and the column index by j . The elementary vector of S corresponding to column j* of the given tableau has the normalized form

$$z = (\ldots z_j \ldots)_{j\in J} \qquad where \qquad z_j = \begin{cases} 1 & if\ j=j^* \\ a_{jj^*} & if\ j\ is\ basic \\ 0 & else \end{cases} \tag{16}$$

The elementary vector of $S^{\perp}$ corresponding to row i* of the given tableau has the normalized form

$$v = (\ldots v_j \ldots)_{j\in J} \qquad where \qquad v_j = \begin{cases} 1 & if\ j=i^* \\ -a_{i^*j} & if\ j\ is\ nonbasic \\ 0 & else \end{cases} \tag{17}$$

By a <u>painting</u> of the index set J we mean a partitioning of J into four subsets (some possibly empty) whose elements will be called "green", "white", "black", and "red", respectively.

For a given tableau, a column, indexed by say s, of the tableau is said to be <u>column compatible</u> if the colour of s and the pattern of signs occuring in that column satisfies the requirements shown in Figure 4. Note that a column whose index is red is never compatible. The requirements

	g	w	b	r
r	0	0	0	inc
b	0	≤ 0	≥ 0	inc
w	0	≥ 0	≤ 0	inc
g	arb	arb	arb	inc

arb = arbitrary

inc = incompatible

Figure 4 Column compatibility for Tucker tableau

for a compatible row are analgously shown in Figure 5.

	g	w	b	r
r	0	0	0	arb
b	0	≥ 0	≤ 0	arb
w	0	≤ 0	≥ 0	arb
g		inc		

arb = arbitrary

inc = incompatible

Figure 5 Row compatibility for Tucker tableau

The Painted Index algorithm takes any painting of the index set J and any initial Tucker tableau and performs a sequence of pivoting steps to arrive at a final tableau that contains either a compatible column or a compatible row. More explicitly, for any given index s that is black or white, the algorithm produces a final tableau having either a compatible column "using" s or a compatible row "using" s (we say that a column (row) uses s if s is either the index of the column (row) or the index of some row (column) whose entry in that column (row) is nonzero). We describe the algorithm below :

Painted Index algorithm ([8], Chap. 10)

Start with any Tucker tableau. Let s be a white or black index that corresponds to either a row or a column (s will be called the lever index).

If s corresponds to a row, check whether this row is compatible. If yes, we terminate the algorithm. Otherwise there is an entry in the s row that fails the compatibility test. Let j be the index of any column containing such an entry, and check whether this column is compatible. If yes, we terminate the algorithm. Otherwise, there is an entry in column j

that fails the compatibility test. Let k be the index of any row containing such an entry. Pivot on (k,j) (i.e. make j basic and k nonbasic) and return to the beginning of the procedure.

If s corresponds to a column, we act analogously to the above, with the word "column" and "row" interchanged.

The Tucker tableau can be recursively updated after each pivot in a manner similar to that in the simplex method. This updating procedure is described in Appendix A. When the algorithm terminates, either a compatible row using s is found or a compatible column using s is found. The number of distinct Tucker tableaus is finite, thus implying that the number of distinct compatible columns or rows is also finite. Finite termination of the algorithm is guaranteed when Bland's priority rule is used [8] : Give to the elements of J distinct numbers (priorities), and whenever there is more than one index that can be selected as j or k , select the one whose priority is highest.

3. Dual Descent Directions and the Modified Painted Index algorithm

For a given price vector p and a direction u, the directional derivative of q at p in the direction of u is given by [cf. (12)]

$$q'(p;u) = g'(t;v)$$

where $v = E^T u$ and by definition

$$q'(p;u) = \lim_{\lambda \downarrow 0} \frac{q(p+\lambda u) - q(p)}{\lambda} \qquad , \qquad g'(t;v) = \lim_{\lambda \downarrow 0} \frac{g(t+\lambda v) - g(t)}{\lambda}$$

Since g is separable we obtain [cf. (13)]

$$g'(t;v) = \sum_{j \ni v_j < 0} g_j^-(t_j)v_j + \sum_{j \ni v_j > 0} g_j^+(t_j)v_j$$

where g_j^+ and g_j^- respectively denote the right and the left derivative of g_j. Therefore the work in evaluating directly q'(p;u) is roughly proportional to the size of the support of v. Thus we see that by using the elementary vectors of $C^\perp$ as descent directions we are in part minimizing the effort required to evaluate q'(p;u). Since each g_j has the form (14), then g_j^- and g_j^+ have the form

$$g_j^+(t_j) = \begin{cases} l_j & \text{if } j \text{ inactive} \\ c_j & \text{if } j \text{ active or if } j \text{ balanced} \end{cases}, \quad g_j^-(t_j) = \begin{cases} l_j & \text{if } j \text{ inactive or if } j \text{ balanced} \\ c_j & \text{if } j \text{ active} \end{cases} \tag{18}$$

from which it follows that

$$q'(p;u) = C(v,t)$$

where $t = E^Tp$, $v = E^Tu$, and

$$C(v,t) = \sum_{\substack{v_j < 0 \\ a_j \geq t_j}} l_j v_j + \sum_{\substack{v_j < 0 \\ a_j < t_j}} c_j v_j + \sum_{\substack{v_j > 0 \\ a_j > t_j}} l_j v_j + \sum_{\substack{v_j > 0 \\ a_j \leq t_j}} c_j v_j \tag{19}$$

It follows that $q'(p;u) < 0$ if and only if $C(v,t) < 0$. Furthermore since q(p) is a piecewise linear convex function we have the following result :

<u>Proposition 2</u> For a given vector (p, t) in $C^\perp$ and a direction (u, v) in $C^\perp$ there holds

$$q(p+\lambda u) = q(p) + \lambda C(v,t) \qquad \forall\ \lambda \in [0,\alpha)$$

where α is given by

$$\alpha = \min_{v_j(a_j-t_j)>0} \left\{ \frac{a_j-t_j}{v_j} \right\} = \inf \left\{ \min_{v_j<0\ j\,active} \frac{a_j-t_j}{v_j}\ ,\ \min_{v_j>0\ j\,inactive} \frac{a_j-t_j}{v_j} \right\} \qquad (20)$$

(α is the stepsize at which some column index becomes balanced.)

<u>Proof</u> :

For a fixed v, the quantity C(v,t) depends only on the following four index sets

$$\{ j \mid a_j > t_j , v_j \neq 0 \}\ ,\ \{ j \mid a_j < t_j , v_j \neq 0 \}\ ,\ \{ j \mid a_j = t_j , v_j > 0 \}\ ,\ \{ j \mid a_j = t_j , v_j < 0 \}$$

as can be seen from (19). By our choice of α it can be easily verified that these four index sets do not change for all tension vectors on the line segment between t and $t+\alpha v$, excluding the end point $t+\alpha v$. It follows that

$$C(v,t+\tau v) = C(v,t) \qquad \forall\ \tau \in [0,\alpha)$$

This combined with the fact

$$q'(p+\lambda u) = q(p) + \int_0^{\lambda} q'(p+\tau u;u)\,d\tau = q(p) + \int_0^{\lambda} C(v,t+\tau v)d\tau \qquad \forall\ \lambda \in [0,\alpha)$$

give the desired result. Q.E.D.

An alternative formula for C(v,t) that turns out to be particularly useful for network problems [2] is obtained as follows. Let x be a primal vector satisfying CS with p. Then by first expanding the terms in C(v,t) and then using the definition of (CS) we obtain

$$C(v,t) = \sum_{\substack{v_j<0 \\ a_j \geq t_j}} l_j v_j + \sum_{\substack{v_j<0 \\ a_j < t_j}} c_j v_j + \sum_{\substack{v_j>0 \\ a_j > t_j}} l_j v_j + \sum_{\substack{v_j>0 \\ a_j \leq t_j}} c_j v_j$$

$$= \sum_{a_j \neq t_j} x_j v_j + \sum_{\substack{v_j < 0 \\ a_j = t_j}} l_j v_j + \sum_{\substack{v_j > 0 \\ a_j = t_j}} c_j v_j$$

$$= \sum_{j=1}^{m} x_j v_j + \sum_{\substack{v_j < 0 \\ a_j = t_j}} (l_j - x_j) v_j + \sum_{\substack{v_j > 0 \\ a_j = t_j}} (c_j - x_j) v_j$$

Let d be the deficit vector corresponding to x (i.e. $d = Ex$). Then using the identity $u^T d = v^T x$ we obtain

$$C(v,t) = d^T u + \sum_{\substack{v_j < 0 \\ j : balanced}} (l_j - x_j) v_j + \sum_{\substack{v_j > 0 \\ j : balanced}} (c_j - x_j) v_j \tag{21}$$

In a practical implementation it is possible to use a data structure which maintains the set of balanced indices j. Since the number of balanced indices is typically around n or less, it follows that C(v,t) can be typically evaluated using (21) in O(n) arithmetic operations. This represents a substantial improvement over the O(m) arithmetic operations required to evaluate C(v,t) using (19), particularly if $m \gg n$. In the case where E is sparse, the use of (21) may allow further economies as for example in network flow problems (see [2]).

We now describe a particular way to apply the Painted Index algorithm to determine if a price vector p is dual optimal, and if p is not dual optimal to either a) generate an elementary vector (u, v) of $C^{\perp}$ such that u is a dual descent direction of q at p , or b) change the primal vector x so as to reduce the total deficit.

Modified Painted Index Algorithm

Let $x \in R^m$ satisfy (CS) with p and let $d = Ex$. If $d = 0$, then (x, p) satisfies the optimality conditions and p is then dual optimal. If $d \neq 0$, then we select some index s with $d_s \neq 0$. In the description that follows we assume that $d_s < 0$. The case where $d_s > 0$ may be treated in an analogous manner.

We apply the Painted Index algorithm, with s as the lever index and using Bland's anticycling rule, to the extended linear homogeneous system

$$\begin{array}{cc} {\scriptstyle 1,2,\ldots,n} & {\scriptstyle n+1,\ldots,n+m} \\ \end{array}$$
$$\begin{bmatrix} -I & E \end{bmatrix} \begin{bmatrix} w \\ z \end{bmatrix} \tag{22}$$

where index i (corresponding to w_i), $i = 1,2,\ldots,n$, is painted

white	if	$d_i > 0$
black	if	$d_i < 0$
red	if	$d_i = 0$

and index $j + n$ (corresponding to z_j), $j = 1,2,\ldots,m$, is painted

green	if	j balanced and $l_j < x_j < c_j$
black	if	j balanced and $l_j = x_j < c_j$
white	if	j balanced and $l_j < x_j = c_j$
red	if	j not balanced
	or if	j balanced and $l_j = x_j = c_j$

Furthermore, we (i) use as the initial tableau one for which s is basic (one such choice is E for which the indexes $n + 1$ to $n + m$ are basic) ; (ii) assign the <u>lowest</u> priority to index s (this ensures that s is always basic , see Appendix B for proof of this fact). The key feature of the algorithm is that at <u>each</u> intermediate tableau we check to see if a dual descent direction can be obtained from the tableau row corresponding to s. This is done as follows :

We denote

$a_{sj} =$ entry in s row of tableau corresponding to column variable z_j.

$a_{si} =$ entry in s row of tableau corresponding to column variable w_i.

Applying (17) to the extended linear homogeneous system (22) we obtain that the elementary vector (u, v) of $C^{\perp}$ using s that corresponds to this tableau is given by

$$u_i = \begin{cases} 1 & \text{if } i=s \\ -a_{si} & \text{if } w_i \text{ is a column variable} \\ 0 & \text{otherwise} \end{cases} \tag{23}$$

$$v_j = \begin{cases} a_{sj} & \text{if } z_j \text{ is a column variable} \\ 0 & \text{otherwise} \end{cases} \tag{24}$$

For this choice of (u, v) we obtain (using (21)) that

$$C(v,t) = d_s - \sum_i a_{si} d_i + \sum_{\substack{a_{sj} > 0 \\ j+n \text{ green} \\ \text{or black}}} (c_j - x_j) a_{sj} + \sum_{\substack{a_{sj} < 0 \\ j+n \text{ green} \\ \text{or white}}} (l_j - x_j) a_{sj} \tag{25}$$

If $C(v,t) < 0$ then the direction u is a dual descent direction and the algorithm terminates. Note from (25) that if the tableau is such that the sth row is compatible, then u is a dual descent direction since our choice of index painting and the definition of a compatible row implies that

$d_s < 0$ and $a_{si} d_i \geq 0$ for all i such that w_i is a column variable

and $x_j = c_j$ for all j such that z_j is a column variable, j + n green or black, and $a_{sj} > 0$

and $x_j = l_j$ for all j such that z_j is a column variable, j + n green or white, and $a_{sj} < 0$

which in view of (25) implies that $C(v,t) < 0$.

We know that the Painted Index algorithm terminates with either a compatible row using s or a compatible column using s. Thus we must either terminate by finding a dual descent direction corresponding to a tableau for which $C(v,t) < 0$ [cf. (25)] or else terminate with a compatible column using s. In the latter case, an incremental change towards primal feasibility is performed as follows :

Let r^* denote the index of the compatible column.

Let a_{ir^*} denote the entry in the compatible column corresponding to row variable w_i and let a_{jr^*} denote the entry in the compatible column corresponding to row variable z_j.

Case 1 If $r^* = i$ for some $i \in \{1,\dots,n\}$ and r^* is black then set

$$w_i^* \leftarrow \begin{cases} 1 & \text{if } i = r^* \\ a_{ir^*} & \text{if } i \text{ is basic} \\ 0 & \text{else} \end{cases} \qquad z_j^* \leftarrow \begin{cases} a_{jr^*} & \text{if } n+j \text{ is basic} \\ 0 & \text{else} \end{cases}$$

Case 2 If $r^* = n + j$ for some $j \in \{1,\dots,m\}$ and r^* is black then set

$$w_i^* \leftarrow \begin{cases} a_{ir^*} & \text{if } i \text{ is basic} \\ 0 & \text{else} \end{cases} \qquad z_j^* \leftarrow \begin{cases} 1 & \text{if } n+j = r^* \\ a_{jr^*} & \text{if } n+j \text{ is basic} \\ 0 & \text{else} \end{cases}$$

Case 3 If $r^* = i$ for some $i \in \{1,\dots,n\}$ and r^* is white then set

$$w_i^* \leftarrow \begin{cases} -1 & \text{if } i = r^* \\ -a_{ir^*} & \text{if } i \text{ is basic} \\ 0 & \text{else} \end{cases} \qquad z_j^* \leftarrow \begin{cases} -a_{jr^*} & \text{if } n+j \text{ is basic} \\ 0 & \text{else} \end{cases}$$

Case 4 If $r^* = n + j$ for some $j \in \{1,\dots,m\}$ and r^* is white then set

$$w_i^* \leftarrow \begin{cases} -a_{ir^*} & \text{if } i \text{ is basic} \\ 0 & \text{else} \end{cases} \qquad z_j^* \leftarrow \begin{cases} -1 & \text{if } n+j = r^* \\ -a_{jr^*} & \text{if } n+j \text{ is basic} \\ 0 & \text{else} \end{cases}$$

That w^* and z^* so defined satisfy $w^* = Ez^*$ follows from applying (16) to the extended linear homogeneous system (22). Furthermore, our choice of index painting, together with column compatibility of the column indexed by r^*, guarantees that, for $\mu > 0$ sufficiently small, $x + \mu z^*$ satisfies (CS) with p and that $x + \mu z^*$ has strictly smaller total deficit than x.

Given the above discussion, we see that the modified Painted Index algorithm will either produce a dual descent direction u given by (23) that can be used to improve the dual cost, or produce a primal direction z^* as defined above that can be used to reduce the total deficit.

The special case where the initial tableau is E and its sth row yields a dual descent direction is of particular interest and leads to the coordinate descent interpretation of our method. In this case the dual descent direction is [cf. (23)]

$$w_i = \begin{cases} 1 & \text{if } i=s \\ 0 & \text{otherwise} \end{cases}$$

so the algorithm will improve the dual cost by simply increasing the sth price coordinate while leaving all other coordinates unchanged. If the dual cost were differentiable then one could use exclusively such single coordinate descent directions. This is not true in our case as illustrated in Figure 1. Nonetheless the method to be described in the next section generates single coordinate descent directions very frequently for many classes of problems. This appears to contribute substantially to algorithmic efficiency since the computational overhead for generating single coordinate descent directions is very small. Indeed computational experimentation (some of which reported in [1], [2]) indicates that the use of single coordinate descent direction is the factor most responsible for the efficiency of the relaxation method for minimum cost network flow problems.

4. The Relaxation Algorithm for Linear Programs

Based on the discussions in Sections 2 and 3, we can now formally describe our algorithm. The basic relaxation iteration begins with a primal dual pair (x, p) satisfying (CS) and returns another pair (x', p') satisfying (CS) such that either (i) $q(p') < q(p)$ or (ii) $q(p') = q(p)$ and (total deficit of x') $<$ (total deficit of x).

Relaxation Iteration

Step 0 Given $(-d, x) \in C$ and $(p, t) \in C^{\perp}$ such that (x, p) satisfy (CS).

Step 1 If $d = 0$ then x is primal feasible and we terminate the algorithm. Otherwise choose a row index s such that d_s is nonzero. For convenience we assume that $d_s < 0$. The case where $d_s > 0$ may be analogously treated.

Step 2 We apply the modified Painted Index algorithm with s as the lever index to the extended system

$$\begin{bmatrix} -I & E \end{bmatrix} \begin{bmatrix} w \\ z \end{bmatrix}$$

as described in Section 3. If the algorithm terminates with a dual descent direction u we go to Step 4. Otherwise the algorithm terminates with a compatible column using s, in which case we go to Step 3.

Step 3 (Primal Rectification Step)

Compute :

$$\mu = \min \left\{ \min_{z_j^* > 0} \frac{c_j - x_j}{z_j^*} , \min_{z_j^* < 0} \frac{l_j - x_j}{z_j^*} , \min_{w_i^* \neq 0} \frac{-d_i}{w_i^*} \right\}$$

where z^*, w^* are as described in Section 3. Set

$$x \leftarrow x + \mu z^* \quad , \quad d \leftarrow d + \mu w^*$$

and go to the next iteration. (The choice of μ above is the largest for which (CS) is maintained).

Step 4 (Dual Descent Step)

Determine a stepsize λ^* such that

$$q(p+\lambda^* u) = \min_{\lambda > 0} \{q(p+\lambda u)\}$$

Set $(p, t) \leftarrow (p, t) + \lambda^*(u, v)$, update x to maintain (CS) with p , and go to the next iteration.

Validity and Finite Termination of the Relaxation Iteration

We will show that all steps in the Relaxation iteration are executable, that the iteration terminates in a finite number of operations, and that (CS) is maintained. Since the modified Painted Index algorithm (with the priority pivoting rule) is finitely terminating, the relaxation iteration then must terminate finitely with either a primal rectification step (Step 3) or a dual descent step (Step 4). Step 3 is clearly executable and finitely terminating. Step 4 is executable since a dual descent direction has been found. If Step 4 is not finitely terminating then there does not exist a stepsize λ^* achieving the line minimization in the direction of u. It follows from the convexity of q that

$$q'(p+\lambda u;u) \;<\; 0 \qquad for\,all\ \lambda > 0$$

which implies that

$$\sum_{v_j<0} l_j v_j \;+\; \sum_{v_j>0} c_j v_j \;<\; 0$$

in which case the assumption that **(LP)** is feasible is violated. (CS) is trivially maintained in Step 4. In Step 3, the only change in the primal or dual vectors comes from the change in the value of some primal variables whose indexes are balanced. Since the stepsize μ is chosen such that these primal variables satisfy the capacity constraints (2) we see that (CS) must be maintained.

Implementation of the line search in Step 4

It appears that usually the most efficient scheme for implementing the line search of Step 4 is to move along the breakpoints of q, in the descent direction u, until the directional derivative becomes nonnegative. This scheme also allows us to efficiently update the value of C(v,t). Algorithmically it proceeds as follows :

```
Step 4a   Start with p and u such that $C(v,t) < 0$.
Step 4b   If $C(v,t) \geq 0$ then exit (line search is complete). Else compute $\alpha$ using
          (20) ( $\alpha$ is the stepsize to the next breakpoint of q from p in the
          direction u ) . Then move p in the direction u using stepsize $\alpha$ and
          update t and x as follows:
               Increase $p_i$ by $\alpha u_i$ $\forall i$
               Set $x_j \leftarrow l_j$ $\forall$ balanced j such that $v_j<0$
               Set $x_j \leftarrow c_j$ $\forall$ balanced j such that $v_j>0$
```

Increase t_j by αv_j $\forall j$

Update C(v,t) by

$$C(v,t) \leftarrow C(v,t) - \sum_{\substack{a_j = t_j \\ v_j < 0}} (x_j - l_j) v_j - \sum_{\substack{a_j = t_j \\ v_j > 0}} (x_j - c_j) v_j$$

Return to Step 4b.

It is straightforward to check that the updating equation for C(v,t) is correct and that (CS) and the condition $t = E^T p$ are maintained.

Numerical Example

We now give a numerical example for the relaxation algorithm just described. To simplify the presentation we will make no explicit use of Bland's Priority pivoting rule. Consider the following linear program :

$$Min \quad x_1 + x_2 - x_3 + 2x_4 - x_5$$

$$subject\ to \qquad \begin{bmatrix} 2 & -1 & 0 & 1 & 0 \\ 0 & 1 & -1 & 0 & 1 \end{bmatrix} x = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$0 \le x_1 \le 1 \ , \ 1 \le x_2 \le 2 \ , 1 \le x_3 \le 2 \ , 1 \le x_4 \le 2 \ , \ -1 \le x_5 \le 0$$

The cost vector for this example is a = (1, 1, -1, 2, -1). Let the initial price vector be the zero vector. We obtain the following sequence of relaxation iterations :

Iteration 1

p = (0, 0) $\quad$ t = E^Tp = (0, 0, 0, 0, 0) $\quad$ a-t = (1, 1, -1, 2, -1) $\quad$ x = (0, 1, 2, 1, 0) $\quad$ d = (0, -1)

Initial Tucker tableau :

$$\begin{array}{lll|ccccc|}
 & & & z_1 & z_2 & z_3 & z_4 & z_5 \\
 & & & r & r & r & r & r \\
\hline
 & w_1 & r & 2 & -1 & 0 & 1 & 0 \\
\text{lever row} \longrightarrow & w_2 & b & 0 & 1 & -1 & 0 & 1 \\
\hline
\end{array}$$

r = red

w = white

b = black

g = green

Row 2 is compatible, so a dual descent step is possible with descent direction given by :

$u = (0, 1)$ $v = (0, 1, -1, 0, 1)$

and stepsize given by :

$$\alpha \leftarrow \min\{ (a_2 - t_2)/v_2 , (a_3 - t_3)/v_3 \} = 1 .$$

x is unchanged. The new price vector and tension vector are :

$p \leftarrow p + \alpha u = (0, 1)$ $t \leftarrow t + \alpha v = (0, 1, -1, 0, 1)$

<u>Iteration 2</u>

$p = (0, 1)$ $t = E^T p = (0, 1, -1, 0, 1)$ $a - t = (1, 0, 0, 2, -2)$ $x = (0, 1, 2, 1, 0)$ $d = (0, -1)$

Initial Tucker tableau :

$$\begin{array}{lll|ccccc|}
 & & & z_1 & z_2 & z_3 & z_4 & z_5 \\
 & & & r & b & w & r & r \\
\hline
 & w_1 & r & 2 & -1 & 0 & 1 & 0 \\
\text{lever row} \longrightarrow & w_2 & b & 0 & 1 & -1 & 0 & 1 \\
\hline
\end{array}$$

Row 2's compatibility is violated in columns 2 and 3. We pivot on row 1, column 2 :

Next Tucker tableau :

$$\begin{array}{rcc|ccccc|}
 & & & z_1 & w_1 & z_3 & z_4 & z_5 \\
 & & & r & r & w & r & r \\
\cline{4-8}
 & z_2 & b & 2 & -1 & 0 & 1 & 0 \\
\text{lever row} \longrightarrow & w_2 & b & 2 & -1 & -1 & 1 & 1 \\
\cline{4-8}
\end{array}$$

Row 2's compatibility is violated in column 3 and column 3 is compatible. The primal rectification direction is then given by :

$$z^* = (0, 0, -1, 0, 0) \qquad w^* = (0, 1)$$

and the capacity of rectification is given by :

$$\mu \leftarrow \min\{ (l_3-x_3)/z^*_3 \, , \, -d_2/w^*_2 \} = 1$$

p and t are unchanged. The new primal vector and deficit vector are :

$$d \leftarrow d + \mu w^* = (0, 0) \qquad x \leftarrow x + \mu z^* = (0, 1, 1, 1, 0)$$

The algorithm then terminates. The optimal price vector is (0, 1). The optimal primal vector is (0, 1, 1, 1, 0).

5. Finite Convergence of the Relaxation Algorithm

The relaxation algorithm that consists of successive iterations of the type described in the previous section is not guaranteed to converge to an optimal dual solution when applied to general linear programs due to the following difficulties :

(a) Only a finite number of dual descent steps may take place because all iterations after a finite number end up with a primal rectification step.

(b) An infinite number of dual descent steps takes place, but the generated sequence of dual costs does not converge to the optimal cost.

Difficulty (a) may be bypassed by choosing an appropriate priority assignment in the relaxation algorithm and showing that the number of primal rectification steps between successive dual descent steps is finite under the chosen assignment.

Proposition 3 If in the relaxation algorithm the green indexes are assigned the highest priorities and the black and white indexes belonging to $\{1,2,\ldots,n\}$, except for the lever index, are assigned the second highest priorities, then the number of primal rectification steps between successive dual descent steps is finite.

Proof : See Appendix C.

Proposition 3 is similar to Rockafellar's convergence result for his primal rectification algorithm ([8], Chap.10). However his algorithm is an out-of-kilter implementation and requires, translated into our setting, that each row index once chosen as the lever index must remain as the lever index at successive iterations until the corresponding deficit reaches zero value. We do not require this in our algorithm.

Difficulty (b) above can occur as shown in an example given in [9]. To bypass difficulty (b) we employ the ε-complementary slackness idea which we introduced in [2] for network flow problems. For any fixed positive number ε and any tension vector t define each column index $j \in \{1,2,\ldots,m\}$ to be

<u>ε-inactive</u>	if	$t_j < a_j - \varepsilon$
<u>ε-balanced</u>	if	$a_j - \varepsilon \le t_j \le a_j + \varepsilon$
<u>ε-active</u>	if	$t_j > a_j + \varepsilon$

Then for a given primal dual pair (x, p) and $t = E^Tp$ we say that x and p satisfy <u>ε-complementary slackness</u> if

$$\begin{aligned} x_j &= l_j && \forall\ \varepsilon\text{-inactive arcs } j \\ l_j \le x_j &\le c_j && \forall\ \varepsilon\text{-balanced arcs } j \qquad\qquad (\varepsilon\text{-CS}) \\ x_j &= c_j && \forall\ \varepsilon\text{-active arcs } j \end{aligned}$$

When $\varepsilon = 0$ we recover the definition of (CS). Define

$$C^{\varepsilon}(v,t) = \sum_{j:\varepsilon-inactive} l_jv_j + \sum_{\substack{v_j<0 \\ j:\varepsilon-balanced}} l_jv_j + \sum_{j:\varepsilon-active} c_jv_j + \sum_{\substack{v_j>0 \\ j:\varepsilon-balanced}} c_jv_j$$

For computational purposes we may alternately express $C^{\varepsilon}(v,t)$ in a form analogous to (21) : For a given p let x satisfy ε-CS with p and let d = Ex. Then using an argument similar to that used to derive (21) we obtain

$$C^{\varepsilon}(v,t) = d^Tu + \sum_{\substack{v_j<0 \\ j:\varepsilon-balanced}} (l_j - x_j)v_j + \sum_{\substack{v_j>0 \\ j:\varepsilon-balanced}} (c_j - x_j)v_j \qquad (26)$$

We note that, for a fixed v and t, $C^{\varepsilon}(v,t)$ is monotonically increasing in ε and that $C(v,t) = C^0(v,t)$.

<u>Proposition 4</u> If in the relaxation iteration of Section 4 we replace (CS) by (ε-CS) and C(v,t) by $C^{\varepsilon}(v,t)$ then the number of dual descent steps in the relaxation algorithm is finite.

Proof :

First we will show that each iteration of the modified relaxation algorithm is executable. Let (x, p) denote the primal dual pair that satisfies ε-CS at the beginning of the iteration and let $t = E^T p$. It is straightforward to verify, using (26), that with (CS) replaced by (ε-CS) in the painting of the indexes every compatible row yields a (u, v) [cf. (23), (24)] that satisfies $C^{\varepsilon}(v,t) < 0$. Therefore the iteration must terminate with either a compatible column or a (u, v) such that $C^{\varepsilon}(v,t) < 0$. In the former case we can perform a primal rectification step identical to that described in Section 4. In the latter case, since $C(v,t) = C^{0}(v,t) \le C^{\varepsilon}(v,t)$, it follows that u is a dual descent direction at p so that the dual descent step (Step 4) can be executed.

Next we will show that the line minimization stepsize in the dual descent step is bounded from below. Using the definition of $C^{\varepsilon}(v,t)$ we have that a dual descent is made when

$$C^{\varepsilon}(v,t) = \sum_{a_j - t_j > \varepsilon} l_j v_j + \sum_{\substack{v_j < 0 \\ |a_j - t_j| \le \varepsilon}} l_j v_j + \sum_{a_j - t_j < -\varepsilon} c_j v_j + \sum_{\substack{v_j > 0 \\ |a_j - t_j| \le \varepsilon}} c_j v_j < 0 \qquad (27)$$

Let

$$\varepsilon' = \frac{\varepsilon}{\max_j |v_j|} \qquad (28)$$

and let $p' = p' + \varepsilon' u$, $t' = t + \varepsilon' v$. Let x' be a primal vector satisfying (CS) with p'. Then (28) implies that

$$a_j - t_j > \varepsilon \Rightarrow a_j - t'_j > 0 \quad and \quad a_j - t_j < -\varepsilon \Rightarrow a_j - t'_j < 0$$

so that

$$a_j - t_j > \varepsilon \Rightarrow x'_j = l_j \quad and \quad a_j - t_j < -\varepsilon \Rightarrow x'_j = c_j \qquad (29)$$

Using (29) and (19) we obtain that

$$C(v,t') = \sum_{a_j - t_j > \varepsilon} l_j v_j + \sum_{\substack{v_j < 0 \\ |a_j - t_j| \le \varepsilon}} x'_j v_j + \sum_{a_j - t_j < -\varepsilon} c_j v_j + \sum_{\substack{v_j > 0 \\ |a_j - t_j| \le \varepsilon}} x'_j v_j \qquad (30)$$

Subtracting (27) from (30) we obtain that

$$C(v,t') - C^{\varepsilon}(v,t) = \sum_{\substack{v_j < 0 \\ |a_j - t_j| \le \varepsilon}} (x'_j - l_j) v_j + \sum_{\substack{v_j > 0 \\ |a_j - t_j| \le \varepsilon}} (x'_j - c_j) v_j \qquad (31)$$

Since the right hand side of (31) is nonpositive it follows that $C(v,t') \le C^{\varepsilon}(v,t) < 0$ so that u is a dual descent direction at $p + \varepsilon' u$, implying that the line minimization stepsize is bounded from below by ε'.

Consequently the line minimization stepsize at each dual descent step is bounded from below by εL where L is a positive lower bound for $1/\max\{|v_j| \mid j \in \{1,2,..,m\}\}$ as v ranges over the finite number of elementary vectors (u, v) of $C^{\perp}$ that can arise in the algorithm. Since the rate of dual cost improvement over these elementary vectors is bounded in magnitude from below by a positive number we see that the cost improvement associated with a dual descent (Step 4) is bounded from below by a positive scalar (which depends only on ε and the problem data). It follows that the algorithm cannot generate an infinite number of dual descent steps. Q.E.D.

Using Proposition 4 we obtain the following convergence result :

Proposition 5 If the conditions of Propositions 3 and 4 are satisfied, then the relaxation algorithm terminates finitely with a primal dual pair (x, p) satisfying ε-complementary slackness and $Ex = 0$.

Proof :

That the relaxation algorithm terminates finitely follows from Propositions 3 and 4 (note that the introduction of ε-CS does not destroy the validity of Proposition 3). That the final primal dual pair satisfies ε-complementary slackness follows from the observation that ε-complementary slackness is maintained at all iterations of the relaxation algorithm. That the final primal vector satisfies the flow conservation constraints (1) follows from the observation that the relaxation algorithm terminates only if the deficit of each row indexe is zero. Q.E.D.

The next proposition provides a bound on the degree of suboptimality of a solution obtained based on ε-CS.

Proposition 6 If (x, p) satisfies ε-complementary slackness and $Ex = 0$ then

$$0 \le f(x) + q(p) \le \varepsilon \sum_{j=1}^{m} (c_j - l_j)$$

Proof :

Let $t = E^T p$. Since $Ex = 0$ we have that

$$a^T x = (a - t)^T x \tag{32}$$

Using (32) and the definition of ε-complementary slackness we obtain

$$a^T x = \sum_{a_j - t_j > \varepsilon} (a_j - t_j) l_j + \sum_{a_j - t_j < -\varepsilon} (a_j - t_j) c_j + \sum_{-\varepsilon \le a_j - t_j \le \varepsilon} (a_j - t_j) x_j \tag{33}$$

On the other hand we have [cf. (12), (13), and (14)]

$$q(p) \;=\; -\sum_{a_j - t_j > 0} (a_j - t_j) l_j + \sum_{a_j - t_j < 0} (a_j - t_j) c_j \tag{34}$$

Combining (33) with (34) and we obtain

$$a^T x + q(p) \;=\; \sum_{0 < a_j - t_j \le \varepsilon} (a_j - t_j)(x_j - l_j) + \sum_{-\varepsilon \le a_j - t_j < 0} (a_j - t_j)(x_j - c_j)$$

from which it follows that

$$a^T x + q(p) \;\le\; \varepsilon \sum_{j=1}^{m} (c_j - l_j)$$

and the right hand inequality is established. To prove the left hand inequality we note that by definition

$$-\,q(p) \;=\; \begin{array}{ll} \textit{Minimize} & (a-t)^T \xi \\ \textit{subject to} & l \le \xi \le c \end{array}$$

from which it follows that

$$-\,q(p) \;\le\; (a-t)^T x \;=\; a^T x$$

where the inequality holds since $l \le x \le c$ and the equality holds since $Ex = 0$. Q.E.D.

A primal dual pair satisfying the conditions of Propositon 6 may be viewed as an optimal solution to a perturbed problem where each cost coefficient a_j is perturbed by an amount not exceeding ε. Since we are dealing with linear programs, it is easily seen that if ε is sufficiently small then every optimal solution of the perturbed primal problem is also an optimal solution of the original primal problem. Therefore, for sufficiently small ε, the modified relaxation algorithm based on ε-CS terminates in a finite number of iterations with an optimal primal solution. It is not difficult to see that the required size of ε for this to occur may be estimated by

$$\min\ \{\ a^T x - a^T x^* \mid x \text{ a basic feasible solution of } (\mathbf{LP})\ ,\ a^T x - a^T x^* \ne 0\ \}$$

where x^* is any optimal primal solution. However such an estimate is in general not computable apriori.

6. Computational Experience

To assess the computational efficiency of the relaxation algorithm we have written three relaxation codes in FORTRAN and compared their performances to those of efficient FORTRAN primal simplex codes. The three relaxation codes are : RELAX for ordinary network flow problems, RELAXG for positive gain network flow problems, and LPRELAX for general linear programming problems. All three codes accept as input problems in the following form

$$\text{Minimize} \quad \sum_{j=1}^{m} a_j x_j$$

$$\text{subject to} \quad \sum_{j=1}^{m} e_{ij} x_j = b_i \qquad , i=1,2,\ldots,n$$

$$0 \leq x_j \leq c_j \qquad , j=1,2,\ldots,m$$

For RELAXG, the matrix E is required to have in each column exactly one entry of +1, one negative entry, while the rest of the entries are all zeroes. For RELAX, the negative entries are further required to be -1. Three primal simplex codes - RNET [11] for ordinary network flow problems, NET2 [10] for positive gain network flow problems, and MINOSLP (Murtagh and Saunders) for general linear programming problems - were used to provide the basis for computational comparison. The test problems were generated using three random problem generators - NETGEN [13] for ordinary network problems, NETGENG [12] (an extended version of NETGEN) for positive gain network problems, and LPGEN for general linear programming problems. NETGEN and NETGENG are standard public domain generators, while LPGEN is a generator that we wrote specifically for the purpose of testing LPRELAX and MINOSLP. All codes were written in standard FORTRAN and, with the exception of RNET, were compiled on a VAX11/750 (operating system VMS 4.1). They were all ran under identical system load condition (light load, sufficient incore memory to prevent large page faults). For RNET we only obtained an object code that was compiled under an earlier version of VMS. The timing routine was the VAX11/750 system time routine LIB$INIT_TIMER and LIB$SHOW_TIMER. The solution times did not include the time to set up the problem data structure. In both RELAX, RELAXG, and LPRELAX, the initial price vector was set to the zero vector.

RELAX is an ordinary network flow code that uses a linked list to store the network topology. It implements the modified Painted Index algorithm by means of a labeling technique similar to Ford-Fulkerson labeling. Detailed description of RELAX is given in [2]. RNET is a primal simplex code developed at Rutgers University over a span of many years. In RNET the FRQ parameter was set at 7.0 as suggested by its authors. Preliminary testing with RELAX and RNET showed that RELAX performs about as well as RNET on uncapacitated transhipment problems but outperforms RNET on assignment problems, transportation problems, and capacitated transhipment problems (up to 3 to 4 times faster). Here we give the times for the first 27 NETGEN benchmark problems in Table 1 (computational experience with other NETGEN problems is reported in [2] and [9]). The superiority of RELAX over RNET is less pronounced on very sparse problems where the ratio m / n is less than 5. This may be explained by the fact that sparsity implies a small number of basic feasible solutions. Although the results presented are only for those problems generated by NETGEN we remark that similar results were obtained using a problem generator that we wrote called RANET. Since RANET uses a problem generating scheme quite different from that used by NETGEN, our computational results seem to be robust with respect to the type of problem generator used. Typically, the number of single coordinate descent steps in RELAX is from 2 to 5 times that of the number of multi-coordinate descent steps while the contribution made by the single coordinate descent steps in improving the dual cost is anywhere from 9 (for tightly capacitated transhipment problems) to 20 (for uncapacitated transportation problems) times that made by the multi-coordinate descent steps (see [9], Tables 2.2 and 2.3). Yet the single coordinate descent step is computationally very cheap. In the range of problems tested, the average number of coordinates involved in a multi-coordinate descent is found to be typically between 4 and 8 implying that even in the multi-coordinate descent steps the computational effort is small. Furthermore this number seems to grow very slowly with the problems size.

RELAXG is a positive gain network code developed from RELAX. It implements the modified Painted Index algorithm by means of a labeling technique similar to that used by Jewell [5]. The total storage requirement for RELAXG is : five m-length INTEGER*4 arrays, five n-length INTEGER*4 arrays, five m-length REAL*4 arrays, four n-length REAL*4 arrays, and two LOGICAL*1 arrays. Line minimization in the dual descent step is implemented by moving along successive breakpoints in the dual functional. Labeling information is discarded after each iteration. When the number of nodes (corresponding to row indexes) of nonzero deficit falls below a prespecified threshold TP, RELAXG switches to searching for elementary descent direction of "maximum" rate of descent and using as stepsize that given by (20), but with "active", "inactive" replaced by "ε-active", "ε-inactive" respectively.

To measure the efficiency of the gain network algorithm we compared RELAXG with the code NET2 of Currin [10]. NET2 is a FORTRAN primal simplex code developed on a CDC Cyber 170/175 computer operating under NOS 1.4 level 531/528. In the computational study conducted by its author [10] - experimenting with different data structures, initial basis schemes, potential updating and pivoting rules - NET2 was found to be on average the fastest (NET2 uses forward star representation). In addition to NETGENG we also tested RELAXG and NET2 on problems generated by our own random problem generator RANETG - an extension of RANET for generalized networks. The times with RANETG are roughly the same as with NETGENG - which shows that our computational results are robust with respect to the type of problem generator used. Table 2 contains the specification of the NETGENG benchmark problems described in [10] and [12] and the corresponding solution times from RELAXG and NET2. The fourth benchmark problem turned out to be infeasible in our case (as verified by both NET2 and RELAXG) - perhaps because we used a slightly different version of NETGENG or because the random number generator in NETGENG is machine dependent, as was the case with NETGEN. Table 3 contains the specification and the solution time for additional NETGENG problems. The solution times quoted for both NET2 and RELAXG do not include the time to read the input data and the time to initialize the data structures (these times were in general less than 8% of the total solution time).

Initial testing shows that out of the 18 benchmark problems RELAXG (with TP set to (#sources + #sinks)/2) is faster than NET2 on 11 of them. However out of the 7 problems where RELAXG1 performed worse, it sometimes performed very badly (see problem 9 of Table 2). Overall it appears that RELAXG tends to perform worse than NET2 on lightly capacitated asymmetric (the number of sources is either much greater or much smaller than the number of sinks) problems while RELAXG outperforms NET2 considerably on symmetric transportation and capacitated transhipment problems (see Tables 2 and 3). However it should be noted that NET2 was written on a different machine and under a different operating system. Computational experience with RELAXG and NET2 on other NETGENG problems is reported in [9].

LPRELAX is the relaxation code for general linear programming problems. LPRELAX does not use any sparsity information and is therefore more suited to dense problems with a small number of rows. At each iteration, LPRELAX first checks if the lever index corresponds to a single coordinate descent direction and performs a dual descent step with line search accordingly. It then applies the Painted Index algorithm to find either a compatible row or a compatible column using the lever index. In the former case a dual descent step, with stepsize given by (20) where "active" is replaced by "ε-active" and "inactive" is replaced by "ε-inactive", is performed. In the latter case a primal rectification step is performed. Experimentation showed that using the tableau left from the previous iteration as the initial tableau for the current iteration (an additional pivot may sometimes be required to make the lever index basic) is computationally beneficial and this was implemented in LPRELAX. To avoid unnecessary computation LPRELAX works with the reduced linear homogeneous system

$$[-I \quad E'\,] \begin{bmatrix} w \\ z' \end{bmatrix} = 0$$

where E′ consists of the columns of E whose indexes are ε-balanced and z′ consists of the entries of z whose indexes are ε-balanced. The only time that the columns that are not ε-balanced are used is during a dual descent step to compute v_j (v_j given by (24)) for all j not ε-balanced. However this computation can be done at the beginning of the dual descent step using the fact that

$$v_j = \sum_{i=1}^{n} e_{ij} u_i \qquad , \qquad u_i = \begin{cases} 1 & \textit{if } i=s \\ -a_{si} & \textit{if } i \neq s \textit{ and } i \textit{ is basic} \\ 0 & \textit{otherwise} \end{cases}$$

where a_{si} denotes the entry in the s row of the Tucker tableau (representing the above reduced system) corresponding to row variable w_i ; and s is the lever index.

The most critical part of the code, both in terms of numerical stability and efficiency, is the procedure for Tucker tableau updating. The current version attempts to identify numerical instability by checking, after every pivot, for unusually large entries appearing in the tableau and then backtracking if such an entry is identified. The backtracking scheme requires storing the set of indexes that were basic in the previous tableau. The threshold value for determining whether a tableau entry is zero was set at .0005 (it was found that if the threshold value was set too low then the pivots can cycle). For sparse problems some technique for preserving and exploiting the sparsity structure during pivoting would be needed to make the code efficient. LPRELAX has a total storage requirement of one n x m REAL*4 array (to store the constraint matrix), one n x 2n REAL*4 array (to store the reduced Tucker tableau), 5 m-length REAL*4 arrays, 4 n-length REAL*4 arrays, and 2 n-length arrays.

MINOSLP is a primal simplex code for linear programs developed by B. A. Murtagh and M. A. Saunders at the Systems Optimization Laboratory of Stanford University as a part of the FORTRAN package called MINOS for solving linear programming and nonlinear programming problems (the 1985 version of MINOS has MINOSLP in a module by itself). To generate the test problems we wrote a problem generator called LPGEN. Given a number of rows and columns, LPGEN generates the entries of the constraint matrix, the cost coefficients, the right hand side, and the upper bound on the variables randomly over a prespecified range. Since MINOSLP has a sparsity mechanism that LPRELAX does not have, in the tests we generated only dense problems so that the times will more accurately reflect the relative efficiency of the algorithms themselves. Note that since the relaxation algorithm uses tableau pivoting it can readily adopt any sparsity technique used by the primal simplex method. In both LPRELAX and MINOSLP we count the time from when the first iteration begins to when the last iteration ends (the time to read in the problem data is not counted).

Initial testing shows that LPRELAX is roughly 10% faster than MINOSLP on problems where the ratio m/n is greater than 10 but two to three times slower if m/n is less than 5 (see Table 4 for problem specifications and solution times). On the larger problems MINOSLP experienced severe problems with page faults - the reason of which is not yet understood. We also considered other measures of performance - in column nine and twelve of Table 4 we give the total number of pivots excuted by LPRELAX and MINOSLP respectively. However since LPRELAX does not work with the full $n \times m$ tableau we considered another measure, denoted by PB. For LPRELAX, PB is simply the number of columns in the reduced tableau summed over all pivots (so that $PB \times n$ is the total number of times that LPRELAX updates a tableau entry). For MINOSLP, PB is the number of iterations times n (so that $PB \times n$ is roughly the total number of times that the revised primal simplex method updates a tableau entry). In essence PB provides us with a measure of the relative efficiency of relaxation and revised primal simplex, assuming that tableau updating is the most time consuming task in either method. The cost of the primal solutions generated by LPRELAX and MINOSLP always agreed on the first six digits and on capacitated problems the cost of the dual solution generated by LPRELAX (with $\varepsilon = .1$) agreed with the primal cost on the first four or five digits (i.e. duality gap is less than 0.1%). However on uncapacitated problems, even with ε taken very small (around .01) this dual cost is typically very far off from the primal cost which is somewhat surprising given that the corresponding primal cost comes very near to the optimal cost. Decreasing ε sometimes increases the solution time and sometimes decreases the solution time. The total dual cost improvement contributed by the single coordinate descent steps is between 50 to 75 percent of the total on the set of problems tested (n between 20 and 50, m between 80 and 500) - a significant reduction from the 93 to 96 percent observed for the ordinary network code RELAX.

In terms of alternate implementations for LPRELAX, we may consider working with only a subset of the rows in the Tucker tableau (for example, the rows of green indexes may be ignored in then modified Painted Index algorithm and be subsequently reconstructed only when a primal rectification step is made), or checking the lever row in the Tucker tableau every few pivots for a dual descent direction, or using line search in a multi-coordinate descent step if the number of coordinates involved in the descent is below a certain threshold. There is also freedom in selecting the lever index at each iteration of the relaxation algorithm - for example, we may choose to use the previous lever index if the previous iteration terminated with a dual descent step.

Our computational experience can be summarized as follows : on ordinary network problems the relaxation method is superior to the primal simplex method; on gain network problems the relaxation method is at least as efficient as the primal simplex method except for asymmetric lightly capacitated problems; on dense linear programming problems the relaxation method is at least as efficient as the primal simplex method for problems where $m > 5n$. However given that both RELAXG and LPRELAX are codes still in the initial stage of development we have hopes that their solution times will be reduced further with improved coding and data structure.

	No. nodes	No. arcs	RELAX	RNET
Trans-portation Problems	200	1300	1.79	3.24
	200	1500	1.87	3.76
	200	2000	1.67	4.43
	200	2200	2.22	5.05
	200	2900	2.48	7.25
	200	3150	3.73	9.34
	200	4500	4.53	12.59
	200	5155	4.63	15.10
	200	6075	5.45	18.65
	200	6300	3.73	16.76
Assign-ment Problems	400	1500	1.11	4.82
	400	2250	1.27	6.57
	400	3000	1.69	8.80
	400	3750	2.29	9.82
	400	4500	2.50	9.94
Uncap-acitated & Lightly Capaci-tated Problems	400	1306	2.44	2.82
	400	2443	2.48	3.42
	400	1306	2.15	2.62
	400	2443	2.38	3.61
	400	1416	3.00	3.06
	400	2836	3.03	4.50
	400	1416	2.82	2.86
	400	2836	4.57	4.56
	400	1382	1.17	2.69
	400	2676	1.83	5.95
	400	1382	1.98	2.53
	400	2676	1.93	3.58

Table 1 Times for Benchmark NETGEN problems with arc cost $\in [1,100]$. Time in CPU seconds. RELAX compiled under VMS 4.1. RNET compiled under an earlier version of VMS. Both methods ran under identical conditions.

Pro-blem	Nodes	Sources	Sinks	Transhipment		Arcs	Multiplier Range	Total Supply	Percent Capaci-tated	Bound Range	Optimal Value	RELAXG*	NET2
				Sources	Sinks								
1	200	100	100	0	0	1500	.5-1.5	100000	0	-	1887171	12.46	15.38
2	200	100	100	0	0	2000	.5-1.5	100000	100	1000-2000	1871263	18.25	18.50
3	200	5	195	0	0	4000	.5-1.5	100000	100	1000-2000	2295799	7.73	8.09
4	200	15	50	10	20	6000	.25-.95	100000	100	1000-2000	-	infeasible	infeasible
5	300	150	150	0	0	1500	.5-1.5	100000	0	-	2407775	18.88	23.37
6	300	5	295	0	0	2000	.5-1.5	100000	0	-	2617790	23.62	17.93
7	300	135	165	135	130	4000	.5-1.5	100000	0	-	2197394	31.36	43.22
8	300	10	40	5	15	6000	.5-1.5	100000	0	-	2637187	27.48	15.79
9	300	2	50	1	20	7000	.5-1.5	100000	0	-	2970418	72.26	8.02
10	400	200	200	0	0	2000	.5-1.5	200	0	-	4554	22.47	36.28
11	400	3	397	0	0	4000	.2-1.4	100000	0	-	4471458	52.33	19.89
12	400	20	100	5	50	5000	.3-1.7	100000	0	-	3102228	23.65	24.52
13	400	25	70	10	20	6000	.5-1.5	100000	100	1000-2000	4171243	41.30	70.17
14	400	30	50	0	0	7000	.5-1.5	100000	100	1000-2000	3292428	38.71	100.59
15	1000	5	995	0	0	4000	.2-6.0	200000	100	4000-6000	6812918	357.9	144.68
16	1000	20	100	5	20	6000	.4-1.4	200000	100	4000-6000	17413428	229.23	91.50
17	1000	20	50	20	50	6500	.5-1.5	200000	0	-	15072130	40.06	71.99
18	1000	30	400	10	30	7000	.7-1.2	200000	0	-	11602427	654.06	171.41

Table 4 Problem specifications for NETGENG benchmark problems (random number seed = 13502460 , percent high cost = 0). Times (in seconds) for RELAXG and NET2 on NETGENG benchmark problems. *TP = (#sources + #sinks)/2.

		No. nodes	No. arcs	Optimal Value	RELAXG*	NET2
Symmetric Capacitated Transhipment cap ∈ [500,1000]	No. nodes fixed	400	2000	72244687	20.93	63.90
		400	6000	40627756	42.82	132.60
		400	8000	35215416	75.85	180.95
		400	10000	28376690	103.14	166.21
Symmetric Capacitated Transhipment cap ∈ [500,1000]	No. arcs fixed	400	7000	36574188	54.88	155.77
		600	7000	72651152	77.12	244.74
		800	7000	109997296	128.55	369.09
		1200	7000	205287424	229.34	656.09
Symmetric Uncapacitated Transportation	No. nodes fixed	400	2000	47332000	25.08	36.19
		400	6000	22869084	45.76	56.04
		400	8000	17202804	53.90†	70.99
		400	10000	16052708	130.27‡	72.96
Symmetric Unapacitated Transportation	No. arcs fixed	400	7000	18789882	49.74	60.21
		600	7000	43720964	75.35	110.67
		800	7000	64200112	97.98	164.52
		1200	7000	128071952	218.57†	336.72

Table 6 Times (in seconds) for RELAXG and NET2. Problems are created using NETGENG with SEED = 13502460, arc gains ∈[.5,1.5], supply = 500 × (No. nodes), and arc cost ∈ [1,1000]. *TP = (No. nodes)/2.

†Number of page faults exceeds 2500. ‡Number of page faults exceeds 10000.

Problem	n	m	Cost Range	RHS Range	Bound Range	Optimal Value	LPRELAX			MINOSLP		
							CPU Time	# Pivots	PB	CPU Time	# Pivots	PB
1	30	200	-1000 - 1000	-100 - 100	100 - 500	-11739484	151.33	1581	18309	76.68	318	9540
2	30	300	-1000 - 1000	-100 - 100	100 - 500	-19901881	232.58	4171	47028	138.36	447	13410
3	30	400	-1000 - 1000	-100 - 100	100 - 500	-25056870	188.81	1382	16171	213.32	580	17400
4	30	500	-1000 - 1000	-100 - 100	100 - 500	-32402651	266.90	2535	29545	296.14	670	20100
5	30	200	1 - 1000	-200 - 200	100 - 500	55695	58.31	972	11255	26.28	112	3360
6	30	300	1 - 1000	-200 - 200	100 - 500	24817	37.29	236	3759	35.01	115	3450
7	30	400	1 - 1000	-200 - 200	100 - 500	20034	48.05	476	6162	43.82	119	3570
8	30	500	1 - 1000	-200 - 200	100 - 500	11207	66.09	750	7523	64.57	150	4500
9	30	200	1 - 1000	-200 - 200	large	55695	58.10	975	11327	26.57	112	3360
10	30	300	1 - 1000	-200 - 200	large	24817	36.56	236	3759	34.90	115	3450
11	30	400	1 - 1000	-200 - 200	large	20034	49.09	476	6162	44.18	119	3570
12	30	500	1 - 1000	-200 - 200	large	1249	46.86	487	5962	52.43	118	3540
13	40	200	-100 - 100	-200 - 200	100 - 500	-1018866	298.45	2268	40218	115.53	343	13720
14	40	300	-100 - 100	-200 - 200	100 - 500	-1680779	393.68	3920	66388	217.93	498	19920
15	40	400	-100 - 100	-200 - 200	100 - 500	-2570187	278.08	4400	69986	301.48†	690	27600
16	40	500	-100 - 100	-200 - 200	100 - 500	-3497055	439.58	3064	55516	557.24‡	786	31440

Table 4 Times (in seconds) for LPRELAX and MINOSLP. † Number of page faults exceeds 5000. ‡ Number of page faults exceeds 100,000. Entries of constraint matrix are randomly generated in the range [-5,5].

References

[1] Bertsekas, D. P., "A Unified Framework for Minimum Cost Network Flow Problems", LIDS Report P-1245-A, Mass. Institute of Technology, October 1982; also Mathematical Programming, Vol. 32, No. 2, pp. 125-145, 1985.

[2] Bertsekas, D. P. and Tseng, P., "Relaxation Methods for Minimum Cost Ordinary and Generalized Network Flow Problems", LIDS Report P-1462, Mass. Institute of Technology, May 1985.

[3] Dantzig, G. B., Linear Programming and Extensions, Princeton Univ. Press, Princeton, N.J., 1963.

[4] Ford, L. R., Jr., and Fulkerson, D. R., Flows in Networks, Princeton Univ. Press, Princeton, N. J., 1962.

[5] Jewell, W. S., "Optimal Flow Through Networks with Gains", Operations Research, Vol. 10, No. 4, pp. 476-499, 1962.

[6] Rockafellar, R. T., Convex Analysis, Princeton Univ. Press, 1970.

[7] Rockafellar, R. T., "Monotropic Programming: Descent Algorithms and Duality", in Nonlinear Programming 4, by O. L. Mangasarian, R. Meyer, and S. Robinson (eds.), Academic Press, pp. 327-366, 1981.

[8] Rockafellar, R. T., Network Flows and Monotropic Programming, Wiley-Interscience, 1983.

[9] Tseng, Paul, "Relaxation Methods for Monotropic Programs", Ph. D. Thesis, M.I.T., 1986 (to appear).

[10] Currin, D. C., "A Comparative Evaluation of Algorithms for Generalized Network Problems", NRIMS Technical Report TWISK 289, Pretoria, South Africa, 1983.

[11] Grigoriadis, M. D. and Hsu, T., "The Rutgers Minimum Cost Network Flow Subroutines", (RNET documentation), Dept. of Computer Science, Rutgers University, November 1980.

[12] Hultz, J., "Algorithms and Applications for Generalized Networks", Unpublished Dissertation, University of Texas at Austin, 1976.

[13] Klingman, D., Napier, A., and Stutz, J., "NETGEN - A Program for Generating Large Scale (Un)capacitated Assignment, Transportation and Minimum Cost Network Problems", Management Science, Vol. 20, pp. 814-822, 1974.

Appendix A

In this appendix, we explain and describe the rule for Tucker tableau pivoting, as given in [8] Chap. 10. Tucker tableau pivoting is similar to simplex tableau pivoting - we partition the linear homogeneous system of full row rank

$$Tx = 0$$

into

$$Bx_B + Nx_N = 0$$

where B is invertible. The Tucker tableau given by this particular partitioning is

$$-B^{-1}N$$

where x_B is called the row variable and x_N is called the column variable. Let a_{ij} denote the (i,j)th entry of the above tableau. If the pivoting column is l and the pivoting row is k (a_{kl} is necessarily nonzero) then the new tableau after the pivoting operation has x_k as a column variable and x_l as a row variable.

Let a_{ij} denote the (i,j)th entry of the old tableau $-B^{-1}N$ and let b_{ij} denote the (i,j)th entry of the new tableau. Then the entries in the new tableau are obtained from those in the old tableau by the following pivoting rule :

$$b_{ij} = \begin{cases} 1/a_{kl} & \text{if } i=k\,,\, j=l \\ a_{il}/a_{kl} & \text{if } i\neq k\,,\, j=l \\ -a_{kj}/a_{kl} & \text{if } i=k\,,\, j\neq l \\ a_{ij}-a_{il}a_{kj}/a_{kl} & \text{if } i\neq k\,,\, j\neq l \end{cases}$$

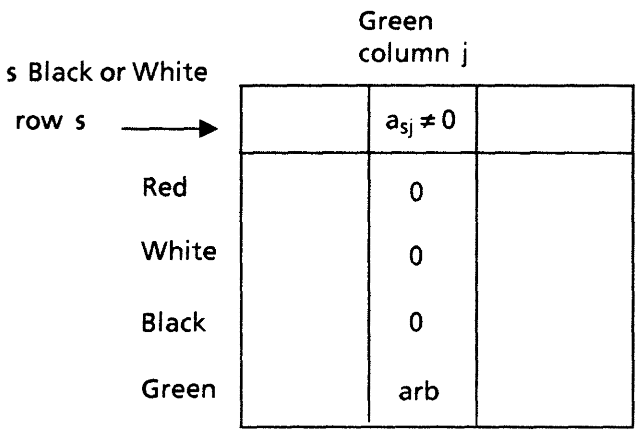
In other words, the new tableau is obtained by performing row operations to $-B^{-1}N$ to make the (k,l)th entry of the tableau a 1 and all other entries in the lth column of the tableau 0's , and then replacing the lth column of the resulting tableau by the kth column of the identity matrix to which the same row operations have been performed.

Appendix B

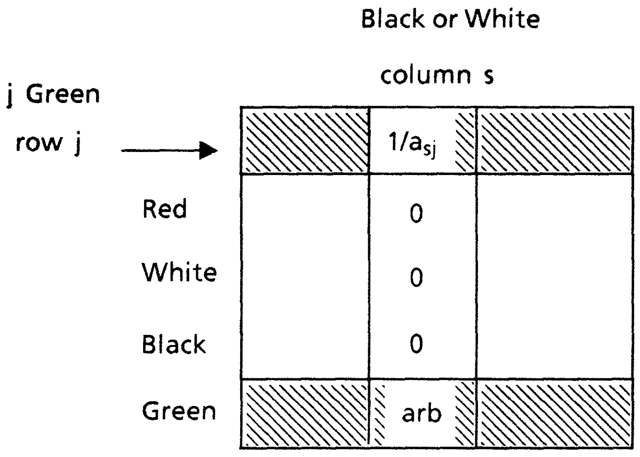
In this appendix, we show that if (i) the lever index s is painted black or white and (ii) s is in the row of the initial Tucker tableau and (iii) s is assigned the <u>lowest</u> priority (in the context of Bland's priority pivoting rule), then the (modified) Painted Index algorithm with Bland's pivoting rule either a) keeps s in the row of the Tucker tableau throughout the algorithm or b) produces a compatible column using s immediately after pivoting s into the column of the Tucker tableau.

<u>Proof</u> :

If s remains in the row of the Tucker tableau throughout the algorithm then we have case a). Else we examine the Tucker tableau just before s is pivoted into the column for the first time. Let j denote the index of the pivoting column. By the pivoting rule, j must be green (if j is black or white, then j can be chosen as a pivoting column index only if the sign of a_{sj} violates the row compatibility of row s, in which case s <u>cannot</u> be chosen as the pivoting row index since the sign of a_{sj} then does not violate the column compatibility of column j). Then by our assignment of lowest priority to s, the row entries of column j whose indices are red, white, or black (except for the entry indexed by s) must all have zero value. Therefore the tableau must have the following form (a_{sj} denotes the entry in column j and row s of the tableau)



The next tableau, after pivoting on a_{sj}, is then of the form



where the shaded areas indicate those entries whose value have been changed during the pivot. In this tableau, column s is clearly compatible and therefore we have case b).

Q.E.D.

Appendix C

In this appendix we prove Proposition 3 - that the number of primal rectification steps between successive dual descent steps is finite in the relaxation algorithm if the green indexes are assigned the highest priorities and the black and white indexes belonging to {1,2,..,n} are assigned the second highest priorities.

Before beginning the main body of the proof it would be helpful to briefly review how each primal rectification step comes about. At the beginning of the relaxation iteration we have on hand a primal vector x satisfying complementary slackness with p. We pick a row s such that $d_s \neq 0$ as the lever index and apply the modified Painted Index algorithm, using Bland's priority pivoting rule, to the following linear homogeneous system (we index the columns of $[-I \;\; E]$ from 1 to $n+m$)

$$\begin{array}{cc} 1,2,\ldots,n & n+1,\ldots,n+m \\ \left[\begin{array}{c|c} -I & E \end{array}\right. & \left.\right] \end{array} \begin{bmatrix} w \\ z \end{bmatrix} . \tag{C.1}$$

The initial Tucker tableau is chosen such that s is initially basic and the index colouring rule is :

For $1 \leq i \leq n$,	i	red if	$d_i = 0$
	i	white if	$d_i > 0$
	i	black if	$d_i < 0$
For $1 \leq j \leq m$,	$j+n$	red if	j is not balanced
	$j+n$	white if	j is balanced and $x_j = c_j$
	$j+n$	black if	j is balanced and $x_j = l_j$
	$j+n$	green if	j is balanced and $l_j < x_j < c_j$

and the priority assignment is : the green indexes are given the highest priorities, the white or black indexes belonging to {1,2,...,n} except for s are given the next highest priorities, s is given the lowest priority. A primal rectification step is taken when the modified Painted Index algorithm terminates with a Tucker tableau that contains a compatible column using the lever index, in which case a primal rectification direction, say z^*, is computed from the compatible column [cf. discussion at the end of Section 3] and z^* satisfies

$$\begin{matrix} w_i^* < 0 \Rightarrow d_i > 0 \\ w_i^* > 0 \Rightarrow d_i < 0 \end{matrix} \quad , \quad \begin{matrix} z_j^* < 0 \Rightarrow x_j > l_j \\ z_j^* > 0 \Rightarrow x_j < c_j \end{matrix} \quad , \quad w_s^* \neq 0$$

where $w^* = Ez^*$. The primal rectification step is then effected by setting $x \leftarrow x + \mu z^*$, $d \leftarrow d + \mu w^*$, where the capacity μ is given by

$$\mu = \min \left\{ \min_{z_j^* < 0} \frac{l_j - x_j}{z_j^*} , \min_{z_j^* > 0} \frac{c_j - x_j}{z_j^*} , \min_{w_i^* \neq 0} \frac{-d_i}{w_i^*} , \right\}$$

The deficit vector d $(d = Ex)$ is monotonically decreased in magnitude during the primal rectification step.

We now proceed to prove that the number of primal rectification steps between successive dual descent steps is finite : We will argue by contradiction. Suppose that there exists an infinite succession of relaxation iterations each of which terminates with a primal rectification step. In what follows we will be considering only these iterations. We first note that the set of red indexes is <u>fixed</u> (since index i is red either because $i \leq n$ and $d_i = 0$, in which case i will remain red during primal variable changes; or because $i = j + n$ and j is not balanced , in which case i will remain red since j does not become balanced during the primal rectification steps). Also we note that if $i \leq n$ and i is white (black) then after a while i always remains white (black).

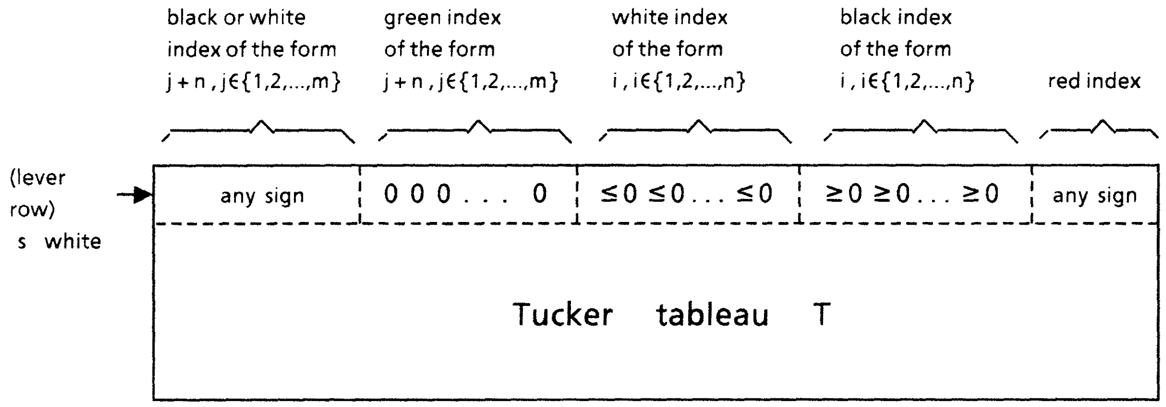
This is true because if i changes colour, i must become red first (corresponding to $d_i = 0$) , in which case i will remain red from then on.

Now, each primal rectification direction z^* has either (i) $j+n$ green for all j such that $z_j^* \neq 0$, or (ii) $j+n$ black or white for some j such that $z_j^* \neq 0$.

<u>Proposition C.1</u> Case (ii) can occur in only a finite number of primal rectification steps.

<u>Proof</u> :

Suppose that the modified Painted Index algorithm generates a primal rectification direction z^* such that, for some j, index $j+n$ is black or white and $z_j^* \neq 0$. Since the initial tableau (namely E) has $j+n$ nonbasic, this implies that $j+n$ must be the pivoting column in some intermediate Tucker tableau T. Then, given that the black and white indexes in $\{1,2,\dots,n\}$ and the green indexes all have priorities higher than that of $j+n$, T must have the following sign pattern (without loss of generality we will assume that the lever index, say s , is white) :


black or white index of the form j + n , j∈{1,2,...,m}
green index of the form j + n , j∈{1,2,...,m}
white index of the form i , i∈{1,2,...,n}
black index of the form i , i∈{1,2,...,n}
red index
(lever row) s white
any sign
0 0 0 . . . 0
≤0 ≤0 . . . ≤0
≥0 ≥0 . . . ≥0
any sign
Tucker tableau T

Denote the entry in T corresponding to row s and column index k by a_{sk} . Then

$$d_s = \sum_{\substack{k=j+n \\ k \text{ black or white}}} a_{sk}x_j + \sum_{\substack{k=i \\ k \text{ white}}} a_{sk}d_i + \sum_{\substack{k=i \\ k \text{ black}}} a_{sk}d_i + \sum_{\substack{k=j+n \\ k \text{ red}}} a_{sk}x_j$$

or equivalently

$$d_s + \sum_{\substack{k=i \\ k \text{ white}}} -a_{sk}d_i + \sum_{\substack{k=i \\ k \text{ black}}} -a_{sk}d_i = \sum_{\substack{k=j+n \\ k \text{ black or white}}} a_{sk}x_j + \sum_{\substack{k=j+n \\ k \text{ red}}} a_{sk}x_j \quad \text{(C.2)}$$

Since $x_j = c_j$ if j + n is coloured white , $x_j = l_j$ if j + n is coloured black, and $x_j = c_j$ or l_j if j + n is coloured red, then the right hand side of (C.2) can assume only a finite number of distinct values. The left hand side of (C.2) however is strictly decreased after each primal rectification step with s as the lever index (also note that since s is white, every term on the left hand side of (C.2) is nonnegative and is nonincreasing). Therefore the number of primal rectification steps in which case (ii) occurs must be finite. Q.E.D.

Proposition C.1 says that if the number of primal rectification steps is infinite, then after a while only primal rectification directions of the form (i) can appear. In other words, the relaxation method must produce an infinite sequence of successive primal rectification directions $\{z^t\}_{t=0,1,..}$ such that, for $t = 0,1,\ldots$, $z^t \neq 0$ and j + n is green for all j such that $z_j^t \neq 0$. However this is not possible since in this sequence, after each primal rectification step, some green index of the form j + n ($j \in \{1,2,..,m\}$) must become white or black (since after a while no index of the form i , $i \in \{1,2,..,n\}$, changes colour). This index then cannot be involved in any subsequent primal rectification steps. So in at most m primal rectification steps, every index of the form j + n , $j \in \{1,2,..,m\}$, must be coloured black or white, implying that the following primal rectification direction z ($z \neq 0$) cannot have j + n green for all j such that $z_j \neq 0$, a contradiction.

Q.E.D