LIDS-P-1791 July 1988

Relaxation Method for Large Scale Linear Programming using Decomposition

by

Paul Tseng †

Abstract

In this paper we propose a new decomposition method for large scale linear programming. This method dualizes an (arbitrary) subset of the constraints and then maximizes the resulting dual functional by dual ascent. The ascent directions are chosen from a finite set and are generated by a truncated version of the painted index algorithm [13]. Central to this method is the novel notion of (ε,δ)-complementary slackness ($\varepsilon \geq 0$, $\delta \in [0,1)$) which allows each relaxed problem to be solved only approximately with $O(\varepsilon\delta)$ accuracy and provides a lower bound of $\Omega(\varepsilon(1-\delta))$ on the amount of improvement per dual ascent. By dynamically adjusting ε, the relaxed problems can be solved with increasing accuracy. We show that (i) the method terminates finitely, provided that ε is bounded away from 0, (ii) the final solution produced by the method is feasible and is within $O(\varepsilon)$ in cost of the optimal cost, and (iii) the final solution produced by the method is optimal for all ε sufficiently small.

KEYWORDS: Linear programming decomposition, dual ascent, Tucker tableau, (ε,δ)-complementary slackness.

---

*Work supported by the National Science Foundation under grant NSF-ECS-8519058 and by the Army Research Office under grant DAAL03-86-K-0171. This paper is based in part on the technical report [17].

†The author is presently with the Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA 02139.

Acknowledgement: The author wishes to thank Professor D. P. Bertsekas, the honorary co-author, for his many helpful suggestions in this work.

1. Introduction

Consider linear programs with m ($m \geq 1$) bounded variables and n′ ($n' \geq 1$) equality constraints:

$$\begin{aligned} &\textit{Minimize} && a^T x && \textbf{(P)} \\ & && Ex = f, && (1) \\ &\textit{subject to} && Dx = b, && (2) \\ & && l \leq x \leq c, && (3) \end{aligned}$$

where a, l and c are given vectors in $\mathbb{R}^m$, f is a given vector in $\mathbb{R}^n$, b is a given vector in $\mathbb{R}^{n'-n}$ ($1 \leq n \leq n'$), E is a given $n \times m$ real matrix, D is a given $(n'-n) \times m$ real matrix, and the inequalities in (3) hold componentwise (here superscript T denotes transpose). We will denote the jth component of a, l, c and x by a_j, l_j, c_j and x_j respectively, the kth component of b and f by b_k and f_k respectively, and the (i,j)th entry of E by e_{ij}.

We have separated the equality constraints into the two subsets (1) and (2) to emphasize their decomposition nature. An example is the multicommodity network flow problem ([1], [6], [7], [13]) where (1) corresponds to the network flow constraints and (2) corresponds to the joint capacity constraints.

By assigning a Lagrange multiplier p_i to the ith constraint of (1) and by denoting p the vector whose ith component is p_i, we can write (cf. Lagrangian duality [4], [10], [13]) the dual of (**P**) as

$$\begin{aligned} &\textit{Maximize} && q(p) && \textbf{(D)} \\ &\textit{subject to} && p \in \mathbb{R}^n, \end{aligned}$$

where q denotes the dual functional (q is concave and piecewise linear)

$$q(p) = p^T f + \min\{\, (a - E^T p)^T x \mid Dx = b,\ l \leq x \leq c \,\}. \qquad (4)$$

We will call p the price vector and p_i the price of row i. The dual problem **(D)** has two nice properties – (i) it is unconstrained and (ii) the minimization problem (4) typically decomposes into many small subproblems (as in the case where **(P)** is a multicommodity network flow problem).

We make the following standing assumptions:

Assumption A: **(P)** is feasible, i.e. there exists $x \in \mathbb{R}^m$ satisfying (1), (2) and (3).

Assumption B: For each $p \in \mathbb{R}^n$, the minimization problem (4) has a unique optimal dual solution associated with the constraints $Dx = b$.

Assumption B has been made for simplicity. If Assumption B does not hold, we can either apply the classical technique of perturbation to b (see for example [22]) together with (redundant) constraint elimination or modify our method accordingly. For any $k \geq 1$, any $\xi \in \mathbb{R}^k$ (whose ith component we denote by ξ_i) and any subset B of $\{1,..,k\}$, we will denote by ξ_B the vector whose components are ξ_i, $i \in B$. For any matrix A with k columns, whose columns are indexed from 1 to k, and any subset B of $\{1,..,k\}$, we will denote by A_B the matrix comprising the columns of A whose indexes are in B. For any real vector x and square matrix A, we will denote by $\|x\|_\infty$ and $\|A\|_\infty$ the L_∞-norm of x and A respectively. We will say that a scalar quantity is $O(\lambda)$ ($\Omega(\lambda)$), for some scalar λ, if it is upper (lower) bounded by some positive constant (depending on the problem data only) times λ.

Solution of the Lagrangian dual **(D)** using either the subgradient method [21], [23]–[28] or the primal dual ascent method (e.g. steepest ascent) [26], [29]–[31] has been well studied. Each of these methods has some drawbacks – the subgradient method requires estimates of the optimal cost and heuristics to generate a primal feasible solution; the primal dual ascent method requires solving certain restricted primal problem to find each dual ascent direction (the ε-subgradient method in [21], [26], [31] further requires calculation involving the ε-subdifferential of q). In this paper we present a new dual ascent method for **(D)**, called the relaxation method, that is finitely convergent, does not require exact solution

of the subproblem (4) at each iteration, and is easy to implement. This method is based on the novel notion of (ε,δ)-complementary slackness that allows each subproblem to be solved with only $O(\varepsilon\delta)$ accuracy and provides a $\Omega(\varepsilon(1-\delta))$ lower bound on the improvement in q per dual ascent. The parameters ε and δ can be dynamically adjusted to control the accuracy of the solution and the computational effort per iteration. The dual ascent directions are generated from a finite set using a truncated version of the painted index algorithm (Rockafellar [13], 10G). Although we do not present any computational results with the new method here, its efficiency has been verified for the special case where $n' = n$, i.e. without decomposing the constraints ([3], [15]). One particular implementation, under which the coordinate directions have priorities as candidates for dual ascent, has been particularly successful on network flow problems ([2], [3]) and has motivated the name "relaxation" for the new method. Based on the experience in [2], [3] and [15], we are very hopeful that the new method will be competitive with other methods for large scale linear programming.

Our paper is organized as follows: in §2 we introduce (ε,δ)-complementary slackness and describe its relation to $O(\varepsilon\delta)$-optimality; in §3 we estimate the rate of dual ascents using (ε,δ)-complementary slackness; in §4 we describe the painted index algorithm for generating Tucker tableaus possessing certain sign patterns; in §5 and §6 we describe and prove finite convergence of the relaxation method, using the results of §3 and §4; in §7 we discuss algorithmic variations and practical implementation for the relaxation method; in §8 we specialize the relaxation method to network flow problems with side constraints; and in §9 we present our conclusion.

2. (ε,δ)-Complementary Slackness

For each $p\in\mathbb{R}^n$, let $\phi(p)$ denote the optimal dual solution in (4) associated with the constraints $Dx = b$. $\phi(p)$ is well defined since, for each fixed p, this optimal dual solution is unique by Assumption B. For each p, $q(p)$ equals the optimal dual cost for (4). By writing down the dual of (4), we obtain, after some algebra, that

$$q(p) = p^T f + \phi(p)^T b + \sum_{t_j < a_j} (a_j - t_j) l_j + \sum_{t_j > a_j} (a_j - t_j) c_j, \tag{5}$$

where $t = E^T p + D^T \phi(p)$.

Central to our method is the following notion of (ε,δ)-complementary slackness. Fix scalars $\varepsilon > 0$ and $\delta \in [0,1)$, and define, for each $p \in \mathbb{R}^n$ and each $\eta \in \mathbb{R}^{n'-n}$, column j to be

ε-inactive	if	$\tau_j < a_j - \varepsilon$,
ε-balanced	if	$a_j - \varepsilon \le \tau_j \le a_j + \varepsilon$,
ε-active	if	$\tau_j > a_j + \varepsilon$,

where $\tau = E^T p + D^T \eta$. Then a primal dual pair $x \in \mathbb{R}^m$ and $(p,\eta) \in \mathbb{R}^{n'}$ is said to satisfy (ε,δ)-complementary slackness ((ε,δ)-CS for short) if

$$x_j = l_j \qquad \forall\ \varepsilon\text{-inactive } j, \tag{6a}$$

$$l_j \le x_j \le c_j \qquad \forall\ \varepsilon\text{-balanced } j, \tag{6b}$$

$$x_j = c_j \qquad \forall\ \varepsilon\text{-active } j, \tag{6c}$$

$$Dx = b, \tag{7}$$

$$\|D^T \eta - D^T \phi(p)\|_\infty \le \varepsilon\delta. \tag{8}$$

In the special case where $n' = n$, the above definition reduces to that of ε-complementary slackness given in [3] and [15]. Note that any $x \in \mathbb{R}^m$ satisfying $Dx = b$, $l \le x \le c$, and $(p,\eta) \in \mathbb{R}^{n'}$ will satisfy (ε,δ)-CS if $\delta > 0$ and ε is taken sufficiently large. Also note that, for any x and (p,η) satisfying $(0,\delta)$-CS, if $Ex = f$, then x and (p,η) are respectively (cf. linear programming duality) optimal primal and dual solutions of **(P)**. Hence we sense that, for small ε, if a primal dual pair satisfying (ε,δ)-CS also satisfy $Ex = f$, then they should be near optimal. We make these notions precise below (compare with Proposition 6 in [15]):

Proposition 1 If $x \in \mathbb{R}^m$ and $(p,\eta) \in \mathbb{R}^{n'}$ satisfy (ε,δ)-CS then

$$0 \le a^T x + p^T(f - Ex) - q(p) \le \varepsilon \sum_{j=1}^{m} (c_j - l_j). \tag{9}$$

Proof: See Appendix A.

For each $p \in \mathbb{R}^n$, how do we compute x and η that satisfy (ε,δ)-CS together with p? By Assumption B, there exists a $\rho > 0$, depending on the problem data only, such that if

$$min\{\ (a - E^T p - D^T \eta)^T \xi \mid\ l \le \xi \le c\ \}\ ,$$

is within $\rho\varepsilon\delta$ of

$$min\{\ (a - E^T p)^T \xi \mid D\xi = b, l \le \xi \le c\ \}\ , \tag{10}$$

then η satisfies (8). This observation can be used to show that, if x and (p,η) satisfy (6a)-(6c) and (7), with ε replaced by $\varepsilon' = \rho\varepsilon\delta/\sum_j (c_j - l_j)$, then x and (p,η) satisfy (ε,δ)-CS. In the terminology of [15], such x and η satisfy ε'-complementary slackness and primal feasibility for the minimization problem (10). We can compute such x and η using either the method in [15] or <u>any</u> linear programming method that is suitably modified to generate primal and dual solutions satisfying ε'-complementary slackness and primal feasibility. [Note that the larger the ε', the easier it is to compute x and η.] If D has a simple structure (e.g. Dx = b are generalized upper bound constraints), x and η can be computed very easily.

If x and (p,η) satisfy (ε,δ)-CS <u>and</u> Ex = f, then (9) implies that

$$0 \ \le\ a^T x - q(p)\ \le\ \varepsilon \sum_{j=1}^{m} (c_j - l_j)\ , \tag{11}$$

in which case x and p can be thought of as near optimal solutions of **(P)** and **(D)** respectively. Moreover, it can be seen from (6a)-(6c) that x may be viewed as an optimal solution to a perturbed problem of **(P)**, 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 primal solution of the perturbed problem is also an optimal primal solution of **(P)**. Therefore, for ε sufficiently small and $\delta \in [0,1)$, any vector that is feasible for **(P)** and satisfies (ε,δ)-CS with some price vector is in fact optimal for **(P)**. The required size of for this to occur may be estimated a priori by

$$min\{a^T x - a^T x^* \mid x\ is\ a\ basic\ feasible\ solution\ of (\mathbf{P}),\ a^T x \ne a^T x^*\}\cdot \left(\sum_{j=1}^{m} (c_j - l_j)\right)^{-1},$$

where x^* denotes any optimal solution of **(P)**. Although such a theoretical bound is loose and difficult to compute, in practice, ε need not to be very small (typically $\varepsilon = .0001(\|E\|_\infty + \|D\|_\infty)\|c-l\|_\infty$ works well).

We remark that, for the case n' = n, (ε,δ)-CS possesses other properties [18], [19], [20] that are useful for improving method efficiency. The notion of (ε,δ)-CS also relates to that of ε-subgradient [10], [13], [21], [26], [31], but the methods based on ε-subgradient tend to be impractical, except in some special cases [13], [31].

3. Estimating the Rate of Dual Ascent

In this section we will use (ε,δ)-CS to derive a lower bound on the rate of dual ascent. This bound plays a key role in the generation of good dual ascent directions ("good" in the sense that $\Omega(\varepsilon(1-\delta))$ improvement in the dual functional q can be made along these directions; see Propositions 2 and 5).

Consider any $p \in \mathbb{R}^n$ and any $u \in \mathbb{R}^n$. Denote by q'(p;u) the directional derivative of q at p in the direction of u, i.e.

$$q'(p;u) \equiv \lim_{\lambda \downarrow 0} \frac{q(p+\lambda u)-q(p)}{\lambda} .$$

Using (5), q'(p;u) has the more explicit formula

$$q'(p;u) = u^T f + \psi^T b - \sum_{\substack{t_j < a_j \text{ or} \\ t_j = a_j,\, v_j < 0}} l_j v_j - \sum_{\substack{t_j > a_j \text{ or} \\ t_j = a_j,\, v_j > 0}} c_j v_j , \tag{12}$$

where

$$t = E^T p + D^T \phi(p), \quad v = E^T u + D^T \psi , \tag{13}$$

and ψ is the unique vector in $\mathbb{R}^{n'-n}$ satisfying

$$\phi(p+\lambda u) = \phi(p) + \lambda \psi \quad \textit{for } \lambda > 0 \textit{ sufficiently small}. \tag{14}$$

[It is shown in Appendix B that such ψ exists and is computable.]

Analogous to (12), (13), we define the following quantity. For any p and u in $\mathbb{R}^n$ and any η and π in $\mathbb{R}^{n'-n}$, denote

$$C^{\varepsilon}(p,\eta;u,\pi) = u^T f + \pi^T b - \sum_{\substack{a_j-\tau_j>\varepsilon \text{ or} \\ |a_j-\tau_j|\le\varepsilon,\, v_j<0}} v_j l_j - \sum_{\substack{a_j-\tau_j<-\varepsilon \text{ or} \\ |a_j-\tau_j|\le\varepsilon,\, v_j>0}} v_j c_j \,, \tag{15}$$

where

$$\tau = E^T p + D^T \eta, \quad v = E^T u + D^T \pi. \tag{16}$$

Note that $q'(p;u) = C^0(p,\phi(p);u,\psi)$, where ψ is given by (14). The relationship between $C^{\varepsilon}(p,\eta;u,\pi)$ and $q'(p;u)$ that is of particular interest to us is the following:

Proposition 2 For any p, u in $\mathbb{R}^n$ and any η, π in $\mathbb{R}^{n'-n}$ satisfying $\|D^T\eta - D^T\phi(p)\|_\infty \le \varepsilon\delta$,

$$q'(p+\alpha u;u) \;\ge\; C^{\varepsilon}(p,\eta;u,\pi) \quad \forall\, \alpha \in [0,\varepsilon(1-\delta)/\theta(u)],$$

where $\theta(u) \equiv \max\{\, \|v\|_\infty \mid$ v given by (13) and (14) for some p $\}$.

Proof: See Appendix C. Note that $\theta(u)$ is well defined since it can be seen from Appendix B that, for a fixed direction u, the number of distinct ψ's satisfying (14) with some p is finite.

Proposition 2 says that, if $C^{\varepsilon}(p,\eta;u,\pi) > 0$ for some η near $\phi(p)$ and some π, then u is a dual ascent direction at p. Furthermore, the line search stepsize in the direction u is at least $\varepsilon(1-\delta)/\theta(u)$. We will see in §5 that, for any (p,η) satisfying $\|D^T\eta - D^T\phi(p)\|_\infty \le \varepsilon\delta$, if there does not exist a x satisfying (ε,δ)-CS with (p,η) and Ex = f, then there exists (u,π) such that $C^{\varepsilon}(p,\eta;u,\pi) > 0$. A result analogous to Proposition 2 is obtained in [15] for the special case where n' = n.

For computational efficiency, let us express $C^{\varepsilon}(p,\eta;u,\pi)$ in a simpler form. Let x and (p,η) be any primal dual pair that satisfy (ε,δ)-CS and denote d = Ex–f. Then we have (using (15), (16) and the definition of (ε,δ)-CS)

$$C^{\varepsilon}(p,\eta;u,\pi) = u^T f + \pi^T b - v^T x - \sum_{\substack{|a_j-\tau_j|\le\varepsilon \\ v_j<0}} v_j(l_j - x_j) - \sum_{\substack{|a_j-\tau_j|\le\varepsilon \\ v_j>0}} v_j(x_j - c_j)$$

$$= u^T f + \pi^T b - (u^T E + \pi^T D)x - \sum_{\substack{|a_j-\tau_j|\le\varepsilon \\ v_j<0}} v_j(l_j - x_j) - \sum_{\substack{|a_j-\tau_j|\le\varepsilon \\ v_j>0}} v_j(x_j - c_j)$$

$$= -u^T d + \sum_{\substack{v_j < 0 \\ j\ \varepsilon\text{-balanced}}} v_j(x_j - l_j) + \sum_{\substack{v_j > 0 \\ j\ \varepsilon\text{-balanced}}} v_j(x_j - c_j), \tag{17}$$

where the last equality follows from the fact that $Dx-b=0$ and $d=Ex-f$. Computing $C^{\varepsilon}(p,\eta;u,\pi)$ using (17) is more efficient than using (15) since the number of ε-balanced columns is typically $O(n')$.

4. Tucker Tableaus and the Painted Index Algorithm

In §3 we saw (cf. Proposition 2) that, if (p,η) and (u,π) are such that $C^{\varepsilon}(p,\eta;u,\pi) > 0$ and η is near $\phi(p)$, then u is a dual ascent direction at p. To generate such (u,π) for a given (p,η) such that η is near $\phi(p)$, we will use two classical results from monotropic programming theory – Tucker tableaus and the painted index algorithm ([13], Ch. 10) – an algorithm which generates, via pivoting, a finite sequence of Tucker tableaus the last of which possesses certain special sign pattern. In this section we briefly review these two results.

Consider the linear homogeneous system

$$Tx = 0, \tag{18}$$

where T is a real matrix of full row rank. Let the jth column of T be indexed by j, and denote the set of indexes for the columns of T by J. Consider any partition of J into B and N such that T_B is an invertible matrix (this is possible since T has full row rank). The matrix

$$-(T_B)^{-1}T_N \tag{19}$$

is called a Tucker tableau representing (18) (note that the number of distinct tableaus is finite). The rows and the columns of $-T_B^{-1}T_N$ are indexed by the elements of B and N respectively, and every element of B (N) is said to be in row (column) position. With respect to a given tableau, an index is basic if it is in row position and nonbasic otherwise. The set of basic indexes is a basis.

Tucker tableaus play a fundamental role in the theory of monotropic programming. However, the following property of Tucker tableaus suffices for our purpose (its proof is straightforward using (19)):

Proposition 3 For a given Tucker tableau, let a_{ij} denote the entry of the tableau in the row indexed by basic index i and the column indexed by nonbasic index j. For any nonbasic j^*, the vector $z = (\cdots z_j \cdots)_{j \in J}$ whose jth component is

$$z_j = \begin{cases} 1 & \text{if } j = j^* \\ a_{jj^*} & \text{if } j \text{ is basic} \\ 0 & \text{else} \end{cases}, \tag{20a}$$

satisfies $Tz = 0$. For any basic index i^*, the vector $v = (\cdots v_j \cdots)_{j \in J}$ whose jth component is

$$v_j = \begin{cases} 1 & \text{if } j = i^* \\ -a_{i^*j} & \text{if } j \text{ is nonbasic} \\ 0 & \text{else} \end{cases}, \tag{20b}$$

satisfies $v^T = u^T T$, where u^T denotes the i^*th row of $(T_B)^{-1}$ and B denotes the set of basic indexes.

By a painting of the index set J we will 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 column compatible if the colour of s and the pattern of signs occuring in that column satisfies the requirements shown in Figure 1. Note that a column whose index is red is never compatible. The requirements for a compatible row

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

arb = arbitrary

inc = incompatible

Figure 1. Column compatibility for Tucker tableau.

are analgously shown in Figure 2.

	g	w	b	r	
r	0	0	0	arb	arb = arbitrary
b	0	≥0	≤0	arb	
w	0	≤0	≥0	arb	inc = incompatible
g	inc				

Figure 2. 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 generates a sequence of tableaus, the final one of which has 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 ([13], Ch. 10)

Start with any Tucker tableau. The given white or black index s may correspond to either a row or a column (s is 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 this 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 this column 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 repeat with the new tableau. If s corresponds to a column,

we act analogously to the above, with the words "column" and "row" interchanged and the words "basic" and "nonbasic" interchanged.

The Tucker tableau can be recursively updated after each pivot in a manner similar to that for simplex iterations. Finite termination of the painted index algorithm is ensured by using Bland's priority rule (i.e. arbitrarily assign priorities to the indexes and break ties in choosing the index to leave/enter basis by favouring the one with the highest priority), which we will hereon assume is used always.

An important observation we make regarding the painted index algorithm is that each intermediate tableau generates a vector v via (20b) with i^* equal to the lever index, and since the number of distinct Tucker tableaus is finite, the number of distinct v's that can be generated this way is also finite.

5. The Relaxation Iteration

For each primal vector $x \in \mathbb{R}^m$, we define the deficit of row i to be

$$d_i = \sum_{j=1}^{m} e_{ij} x_j - f_i .$$

Let d be the vector with components d_i (in vector form d = Ex–f). We define the total deficit of x to be

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

The total deficit is a measure of how close x is to satisfying Ex = f (Ex = f if and only if x has zero total deficit).

Based on the discussions in §3 and §4, we can now formally describe the relaxation iteration for (**P**) and (**D**). Each iteration begins with a primal dual pair x and (p,η) satisfying (ε,δ)-CS and Ex ≠ f, and returns another pair x′ and (p',η') satisfying (ε,δ)-CS for which either (i) $q(p') > q(p)$ or (ii) $(p',\eta') = (p,\eta)$ and (total deficit of x′) < (total deficit of x).

Relaxation Iteration

Step 0 Given a primal dual pair x and (p,η) satisfying (ε,δ)-CS and Ex ≠ f. Denote d = Ex–f and select some row s of E for which $d_s \neq 0$. In the description to follow we assume that $d_s < 0$. The case where $d_s > 0$ may be treated in an analogous manner.

Step 1 (Dual Primal Resolution)

Consider the linear homogeneous system (whose columns are indexed from 1 to n′ + m)

$$\begin{array}{ccccc} & {\scriptstyle 1,\ldots,n} & {\scriptstyle n+1,\ldots,n'} & {\scriptstyle n'+1,\ldots,n'+m} & \\ \left[\begin{array}{ccc} -I & 0 & E \\ 0 & -I & D \end{array}\right] & \left[\begin{array}{c} w \\ y \\ z \end{array}\right] & = 0, & & \end{array} \qquad (21)$$

with index i (corresponding to w_i), i = 1,...,n, painted

white if $d_i > 0$,

black if $d_i < 0$,

red if $d_i = 0$,

with index $n+k$ (corresponding to y_k), $k = 1,...,n'-n$, painted

red

and with index $n'+j$ (corresponding to z_j), $j = 1,...,m$, painted

green if j is ε-balanced and $l_j < x_j < c_j$,

black if j is ε-balanced and $l_j = x_j < c_j$,

white if j is ε-balanced and $l_j < x_j = c_j$,

red if j is not ε-balanced or if j is ε-balanced and $l_j = x_j = c_j$.

Let the initial Tucker tableau representing (21) be one for which s is basic. Let s be the lever index and assign the lowest priority to index s (this ensures that s is always basic, as is shown in Appendix B of [15]). We will call the row in the tableau indexed by s the lever row. Go to **Step** 1.1.

1.1 (Dual Ascent Check)

Denote by α_{hr} the tableau entry in lever row and the column indexed by r. Let the vectors u, π and v be given by:

$$u_i = \begin{cases} 1 & \text{if } i=s \\ -\alpha_{si} & \text{if } i \text{ is nonbasic} \\ 0 & \text{otherwise} \end{cases}, \quad i = 1,...,n, \tag{22}$$

$$\pi_k = \begin{cases} -\alpha_{s,k+n} & \text{if } k+n \text{ is nonbasic} \\ 0 & \text{otherwise} \end{cases}, \quad k = 1,...,n'-n, \tag{23}$$

$$v_j = \begin{cases} \alpha_{s,j+n'} & \text{if } j+n' \text{ is nonbasic} \\ 0 & \text{otherwise} \end{cases}, \quad j = 1,...,m. \tag{24}$$

If $C^e(p,\eta;u,\pi) > 0$, go to Step 2. Otherwise go to Step 1.2.

1.2 (Primal Rectification Check)

There is an entry in the lever row that fails the compatibility test. Let r be the index of any column containing such an entry. If this column is compatible, go to Step 3. Otherwise there is an entry in this column, indexed by some h, that fails the compatibility test. Pivot on (h,r) and go to Step 1.1.

Step 2 (Dual Ascent Step)

Let λ be any stepsize inside $[\varepsilon(1-\delta)/\theta(u),\lambda^*]$, where λ^* is any line search stepsize, i.e.

$$q(p+\lambda^* u) \;=\; max\,\{q(p+\alpha u) \mid \alpha > 0\}\,.$$

Set $p' \leftarrow p + \lambda u$ and compute x' and η' to satisfy (ε,δ)-CS with p'. Exit.

Step 3 (Primal Rectification Step)

It can be seen from Figure 1 (and the fact that the column indexed by r uses s) that r is either black or white. Let the vectors w^* and z^* be given by:

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

$$w_i^* = \begin{cases} 1 & if\ i = i^* \\ \alpha_{ir} & if\ i\ is\ basic \\ 0 & else \end{cases}, \qquad z_j^* = \begin{cases} \alpha_{j+n',r} & if\ j+n'\ is\ basic \\ 0 & else \end{cases}.$$

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

$$w_i^* = \begin{cases} \alpha_{ir} & if\ i\ is\ basic \\ 0 & else \end{cases}, \qquad z_j^* = \begin{cases} 1 & if\ j = j^* \\ \alpha_{j+n',r} & if\ j+n'\ is\ basic \\ 0 & else \end{cases}.$$

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

$$w_i^* = \begin{cases} -1 & if\ i = r^* \\ -\alpha_{ir} & if\ i\ is\ basic \\ 0 & else \end{cases}, \qquad z_j^* = \begin{cases} -\alpha_{j+n',r} & if\ j+n'\ is\ basic \\ 0 & else \end{cases}.$$

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

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

Let

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

and set $x' \leftarrow x + \mu z^*$, $p' \leftarrow p$, $\eta' \leftarrow \eta$. Exit.

Validity of the Relaxation Iteration

We show below that all steps in the relaxation iteration are well defined, that x' and (p',η') satisfy (ε,δ)-CS, and either (i) $q(p') > q(p)$ or (ii) $(p',\eta') = (p,\eta)$, (total deficit of x') < (total deficit of x).

Clearly Step 1.1 is well defined. To see that Step 1.2 is well defined, note that (cf. (20b) and (21)) u, π and v given by (22)-(24) satisfy $v = E^T u + D^T \pi$. Hence if the Tucker tableau is such that its lever row (row indexed by s) is compatible, then our choice of index painting, the hypothesis $d_s < 0$, and the definition of a compatible row (cf. Figure 2) would imply

$d_s < 0$ and $a_{si} d_i \geq 0$ for all i such that i is nonbasic,

$x_j = l_j$ for all j such that j is ε-balanced and $a_{s,j+n'} < 0$,

$x_j = c_j$ for all j such that j is ε-balanced and $a_{s,j+n'} > 0$,

which in view of (17), (22) and (24) implies that $C^{\varepsilon}(p,\eta;u,\pi) > 0$. Therefore the lever row is not compatible when we begin Step 1.2, so Step 1.2 is well defined. We know that the painted index algorithm, using Bland's priority rule, terminates finitely with either a compatible row using s or a compatible column using s. Hence Steps 1.1 and 1.2 can be repeated only a finite number of times before either Step 2 or Step 3 is entered.To see that Step 2 is well defined, it suffices to show that λ^* exists and $\varepsilon(1-\delta)/\theta(u) \leq \lambda^*$. If λ^* does not exist, then (cf. concavity of q)

$$q'(p+\alpha u;u) > 0 \quad \forall\, \alpha > 0,$$

which, in view of the piecewise linear nature of q, implies that q'(p+αu;u) is bounded away from 0. Therefore

$$\lim_{\alpha \to \infty} q(p+\alpha u) = +\infty,$$

and Assumption A is contradicted. That $\varepsilon(1-\delta)/\theta(u) \leq \lambda^*$ follows from Proposition 2. To see that Step 3 is well defined, note that (cf. (20a) and (21)) w^* and z^* defined in Step 3 satisfy $w^* = Ez^*$. Furthermore, our choice of index painting, together with compatibility of the column indexed by r, guarantee that (i) $Dz^* = 0$ and (ii) for $\mu > 0$ sufficiently small, $x+\mu z^*$ satisfies (ε,δ)-CS with (p,η) and $x+\mu z^*$ has strictly smaller total deficit than x.

(ε,δ)-CS is clearly maintained in Step 2. In Step 3, the only changes in either the primal or the dual vector occur in the components of the primal vector whose corresponding column is ε-balanced. Since the amount of change μ is chosen such that the new primal vector satisfies the capacity constraints (3), (ε,δ)-CS is maintained (the choice of μ is the largest for which (ε,δ)-CS is maintained and the deficit of each row is monotonically decreased in magnitude).

6. Finite Convergence of the Relaxation Method

The relaxation method that consists of successive iterations of the type described in §5 (for the moment assume that ε is fixed throughout) and terminates when (1) is satisfied is not guaranteed to terminate finitely. We distinguish the following two difficulties:

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

(b) An infinite number of dual ascents take place.

Difficulty (a) may be bypassed by choosing an appropriate priority assignment of the indexes and difficulty (b) is bypassed by choosing ε to be positive:

Proposition 4 If in the relaxation method the green indexes are assigned the highest priorities and the black and white indexes belonging to {1,...,n}, except for the lever index, are assigned the next highest priorities, then the number of primal rectification steps between successive dual ascent steps is finite.

Proof: This is a special case of Proposition 3 in [15] where the indexes n + 1 up to n' are always painted red.

Proposition 5 For any $\varepsilon > 0$ and $\delta \in [0,1)$, the number of dual ascent steps in the relaxation method is finite.

Proof: Since the number of distinct dual ascent directions u used by the relaxation method is finite (recall that u is given by the row entries of some Tucker tableau (cf. (22)) and the number of distinct Tucker tableaus is finite), the number of distinct values of rate of ascent (cf. (12), (13) and (14)) is finite and so it follows the rate of each dual ascent is lower bounded by a positive constant. By Proposition 2, the line search stepsize at each dual ascent step is lower bounded by the positive scalar $\varepsilon(1-\delta)/\theta$, where $\theta = \max\{\theta(u) \mid u \text{ given by (22)}\}$. Therefore the improvement in the dual functional q per dual ascent step is lower bounded by a positive constant (which depends on the problem data and $\varepsilon(1-\delta)$ only) and it follows that the total number of dual ascent steps is finite. Q.E.D.

Propositions 4 and 5 together yield the main result of this section:

Proposition 6 If the conditions of Proposition 4 are met, the relaxation method terminates finitely for any $\varepsilon > 0$ and $\delta \in [0,1)$.

Since (ε,δ)-CS holds at all iterations of the relaxation method and the method terminates only if the total deficit reaches zero, the final primal dual pair produced by the method must satisfying (ε,δ)-CS and (1). We can then compute the respective cost of this primal dual pair to see if they are close enough (an

a priori bound on this "closeness" is given by (11)) and, if not, to reduce ε and repeat. As shown in the discussion immediately following Proposition 1, if ε is sufficiently small, then the final primal vector is optimal for **(P)**.

8. Algorithmic Variations

For simplicity, we thus far have not paid much attention to refinements that improve the efficiency of the relaxation method. In this section we consider a few such refinements:

1. (Computation of $C^{\varepsilon}(p,\eta;u,\pi)$)

   Since (u, π, v) given by (22), (23) and (24) satisfies $v = E^{T}u + D^{T}\pi$, we can compute $C^{\varepsilon}(p,\eta;u,\pi)$ in Step 1.1 using u and v only (cf. (17)). For sparse problems, $C^{\varepsilon}(p,\eta;u,\pi)$ can even be updated instead of recomputed each time the Tucker tableau changes. Alternatively, we can check say every third tableau (instead of every tableau) for a dual ascent direction. Experimentation shows that it is typically beneficial to check frequently in the first few iterations.

2. (Dual ascent stepsize)

   Perhaps the most efficient scheme for implementing the stepsize rule in Step 2 is to move along the breakpoints of q, in the ascent direction u, until either the stepsize is sufficiently large or the directional derivative becomes nonpositive. At each such breakpoint, we can update the directional derivative (cf. (12), (13) and (14)), as well as the distance to the next breakpoint, using sensitivity analysis on the cost for the subproblem (4). Some amount of overrelaxation (i.e. allowing the stepsize to exceed λ^{*}) is possible, provided that the increase in the dual functional q is bounded away from zero. The quantity $\theta(u)$ is typically difficult to estimate, except for special cases such as network flow (in fact, it can be shown that $\theta(u) \leq$ maximum magnitude of an entry in a Tucker tableau representing (21)).

3. (Dynamically adjusting ε and δ)

For simplicity we have fixed ε and δ throughout the relaxation method, but it can be seen that both ε and δ can be changed immediately after each dual ascent (ε and δ can also differ from one component to another). Finite termination will still hold, provided that the sequence of ε's is bounded away from zero and the sequence of δ's is bounded away from 1. In this case, the x and η satisfying (ε,δ)-CS with the current price vector needs to be recomputed after each decrease in either ε or δ, which may or may not be expensive, depending on the structure of D. Alternatively, we can fix ε at a large value, solve using the relaxation method, decrease ε and then resolve, and so on (while using the same δ all the time). This technique closely relates to ε-scaling [18], which improves the complexity of network algorithms in the case where the cost coefficients have large magnitudes.

4. (Coordinate ascent implementation)

One particular choice of the initial Tucker tableau (in Step 1 of the relaxation iteration) that has been particularly successful for the case $n' = n$ is

$$\begin{bmatrix} E \\ D \end{bmatrix},$$

for which the indexes 1 to n' are basic [3], [15], [16]. Since the direction associated with the lever row s of this tableau (cf. (22)) is the sth coordinate vector in $\mathbb{R}^n$, this implementation may be viewed as a generalized coordinate ascent or relaxation implementation whereby coordinate directions are given priorities as candidate for dual ascent. Computational tests showed that, on network flow problems without side constraints, the coordinate directions typically contribute between 80 to 90 percent of the improvements in the dual functional [16].

5. (Convergence for $\varepsilon = 0$)

In general, if ε is set to zero, the number of dual ascent steps can be infinite, as shown by the example in [16], Appendix G. The only exception is when $n' = n$, E is the node-arc incidence matrix

for an ordinary network, and both the cost vector and the initial price vector are rational. Alternatively, we can place additional restrictions on the relaxation iteration in a manner reminiscent of the out-of-kilter method ([13], §11K):

(a) Use only dual ascent directions those given by a compatible lever row, and then move along the dual ascent direction only as far as the first breakpoint of q encountered.

(b) Assign higher priorities to black or white indexes of the form i, $i \in \{1,...,n\}$, except for the lever index, over black or white indexes of the form $n'+j$, $j \in \{1,...,m\}$.

(c) If the previous iteration terminated with a dual ascent, use the same lever index as in the previous iteration and use the final tableau from the previous iteration as the initial tableau.

It can be shown that, under the conditions of Proposition 4 and the above modification, the relaxation method terminates finitely for $\varepsilon = 0$ (the proof of this is however quite long [17]). Experimentation for the special case where $n' = n$ [15] suggests that it is typically beneficial to implement this modification only after a sufficiently large number of dual ascent steps have been performed. We remark that the modified relaxation method differs from the out-of-kilter method in that it does not require the same lever index (which always corresponds to a row of E in our case) be used at successive iterations until the corresponding deficit reaches zero, but in its place it requires modification (b).

6. (Tucker tableau storage)

For large problems, it would be inefficient to store and update the entire Tucker tableau representing (21) in Steps 1.1 and 1.2. Below we present three techniques for reducing this storage: (a) column elimination, (b) product form of the inverse, and (c) row factorization:

(a) Since no entry in a Tucker tableau column whose index is red can fail the row compatibility test (see Figure 2), this column can be eliminated from consideration until a dual ascent is made. Since the number of non-red indexes is at most n + (number of ε-balanced columns) and the number of ε-balanced columns is typically O(n'), for ε small (see [15], [16]), we would typically need to store only O(n') columns of each Tucker tableau.

(b) Analogous to the revised simplex method [4], [22], each sequence of pivots can be represented by a sequence of $n' \times n'$ matrices, each of which differs from the identity in only one column. By storing this sequence of matrices (which can be stored compactly by storing only the nonzero entries), we only have to store and update the lever row of each Tucker tableau. The tableau column whose index is to enter the basis (namely index r) can be computed by multiplying the corresponding column in the matrix appearing in (21) by the above sequence of matrices. Moreover, only those entries in rows whose corresponding index is non-green need to be computed (since no entry in a Tucker tableau row whose index is green can fail the column compatibility test (see Figure 1)). Since the number of pivots (until either Step 2 or Step 3 is entered) is typically much smaller than n' [15], [16], storing the above sequence of matrices is quite efficient.

(c) Suppose that D has relatively few rows and the Tucker tableaus representing the reduced system

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

can be stored and updated very efficiently (perhaps using the techniques in (a) and (b)). Then we can apply a factorization scheme of Rockafellar ([13], Ch. 10F), whereby pivoting is done only on those rows of the Tucker tableaus representing (21) that correspond to D and on the Tucker tableaus representing the above reduced system. We will discuss this scheme in greater detail in §8, together with an application to network flow with side constraints.

8. Network Flow with Side Constraints

In this section we review a factorization scheme of Rockafellar ([13], Ch. 10F) for decomposing Tucker tableaus by row and we apply this scheme to the special case of **(P)** where E is the node-arc incidence matrix for a directed network. By exploiting the network structure of E, we show that most of the Tucker tableaus can be stored and updated using the data structure for this network only.

Consider the following two tier linear homogeneous system

$$\begin{bmatrix} F \\ F_0 \end{bmatrix} x = 0, \tag{25}$$

where

$$F_0 \; x = 0 \tag{26}$$

is the auxiliary system. Consider any Tucker tableau representing (25)

$$\begin{bmatrix} A' \\ A'' \end{bmatrix},$$

where we have partitioned the tableau row-wise to correspond to a partition of its basis such that the indexes corresponding to the rows of A" form a basis for the auxiliary system (26). Thus if we let B", B' and N denote the subset of indexes that are respectively basic for (26), nonbasic for (26) but basic for (25), and nonbasic for (25), then

$$x_{B'} = A' x_N \;, \quad x_{B''} = A'' x_N \;. \tag{27a}$$

Furthermore, let

$$[\; A'_0 \quad A_0 \;]$$

denote the Tucker tableau representing (26) for which B" is a basis and whose columns are partitioned into A_0' and A_0 to correspond to B' and N respectively. Then

$$x_{B''} = A'_0 x_{B'} + A_0 x_N \,. \tag{27b}$$

Combining the first equality in (27a) with (27b) we obtain that

$$x_{B''} = (A'_0 A' + A_0) x_N \,,$$

which, together with the second equality in (27a), imply

$$A'' = A'_0 A' + A_0 \,. \tag{28}$$

Eq. (28) essentially states the Factorization Theorem in Ch. 10F of [13]. It allows us, instead of maintaining A' and A", to maintain A', A_0' and A_0 and compute the columns of A" only when needed. This is computationally attractive when A" is dense while $[\, A_0' \;\; A_0 \,]$ is sparse. Note that if an element of B' becomes nonbasic during a pivot, only A' needs to be updated, while if an element of B" becomes nonbasic during a pivot, then, after updating A', it will be necessary to also update $[\, A_0' \;\; A_0 \,]$. In the latter case, it may be necessary to also exchange an element of B' with the index just pivoted into B" in order that B" remains as a basis for the auxiliary system (26). We will apply the factorization (28) to the system (21) with

$$F = [\, 0 \;\; -I \;\; D \,], \qquad F_0 = [\, 0 \;\; -I \;\; E \,].$$

This choice of factorization allows us to efficiently store $[\, A_0' \;\; A_0 \,]$ in terms of a spanning tree on the original network and to avoid storing A", which has a large number of rows and is not necessarily sparse. Alternatively, we may apply the factorization scheme with F = [-I 0 E] and F_0 = [0 -I D], but this can be computationally expensive (since E typically has many more rows than D) unless A' is stored using, say, the product form of the inverse (cf. note 6 in §7).

We will assume that E is the node-arc incidence matrix for a connected, directed ordinary network G, whose nodes are numbered from 1 to n and whose arcs are numbered from n'+1 to n'+m. In other words,

$$e_{ij} = \begin{cases} 1 & \text{if there exists an arc } n'+j \text{ leaving node } i, \\ -1 & \text{if there exists an arc } n'+j \text{ entering node } i, \\ 0 & \text{otherwise.} \end{cases}$$

Let us add to G a node n+1 and an arc i joining node n+1 to each node i (i = 1,2,...,n) and call the resulting network G' (we will refer to the nodes and the arcs by their numbers). We first give a network characterization of $[A_0' \;\; A_0]$. Since $[A_0' \;\; A_0]$ is a Tucker representation of

$$\begin{bmatrix} -I & 0 & E \end{bmatrix} \begin{bmatrix} w \\ y \\ z \end{bmatrix} = 0 \tag{29}$$

(the columns of [-I 0 E] we index from 1 to n'+m) and its dual, whose basis is characterized by a spanning tree T on G' rooted at node n+1 (see [13], Ch. 10C), we can compute the nonzero columns of $[A_0' \;\; A_0]$ directly using T. More precisely, the arcs in T form B" and the arcs not in T form (B'UN)\{n+1,...,n'}.

Entries in the nonzero columns of $[A_0'\ A_0]$ are given by (letting a_{ij}^0 denote the entry correspond to arc i in T and arc j not in T)

$$a_{ij}^0 = \begin{cases} 1 & \text{if } i \text{ is in unique cycle of } T\cup\{j\} \text{ and is oriented in same direction as } j, \\ -1 & \text{if } i \text{ is in unique cycle of } T\cup\{j\} \text{ and is oriented in opposite direction as } j, \\ 0 & \text{otherwise.} \end{cases}$$

We will assume that T is stored in an array whose kth entry records the arc joining node k to its immediate predecessor in T. To compute a_{ij}^0 for each arc i in T and each arc j not in T requires tracing backwards along the path from the root to each end node of j in T to see if the unique cycle of $T\cup\{j\}$ contains i. The greatest work however comes from performing a pivot where we exchange an element of B" for an element of N and the resultant B" does not form a basis for the auxiliary system (29), in which case we have to find an element of B" and an element of B' whose exchange would make B" a basis. More precisely, let i (j) denote the arc of G' that is to be pivoted out of B" into N (out of N into B") and suppose that the unique cycle in $T\cup\{j\}$ does not contain i. Then there exist an arc k in B' that connects the two components of $T\setminus\{i\}$ (since $(B'\cup B''\cup\{j\})\setminus\{i\}$ forms a basis for (25), it must contain a subset that is a basis for (29)). Then $(B''\cup\{k\})\setminus\{i\}$ forms a basis for (29) and $(B'\cup B''\cup\{j\})\setminus\{i\}$ forms a basis for (25) (see Figure 3). However, to find such k requires searching through the elements of B' to find an

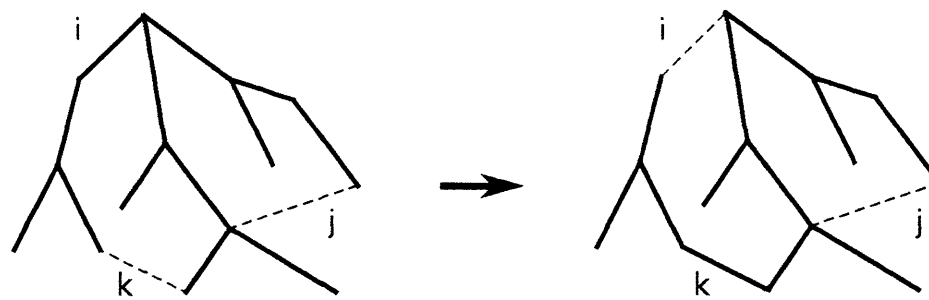

Figure 3. If the unique cycle in $T\cup\{j\}$ does not contain i, we must find some k in B' to replace i in T instead of j (the dark arcs comprise T).

arc that joins the two components of $T\setminus\{i\}$. A simple scheme for this is to, for each element k in B', trace backwards along the path in T from the root to each end node of arc k to see if the unique cycle of $T\cup\{k\}$ contains i (if yes, then we use k). However, if n'–n = (cardinality of B') is large or if T has large depth then more sophisticated data structures and algorithms would be needed. Techniques used by

dual simplex methods may be useful here since finding a k that connects the two components of $T\backslash\{i\}$ is analogous to a dual simplex pivot.

We remark that D sometimes possesses special structure which makes solving (4) easier. One such example is the multicommodity network flow problem [1], [7], ([13], Ch. 10F) for which E has the form

$$E = \begin{bmatrix} E'' & & & \\ & E'' & & \\ & & \ddots & \\ & & & E'' \end{bmatrix},$$

where E", say n" x m", is the node-arc incidence matrix for some directed network and D has the form

$$D = \begin{bmatrix} I & I & \ldots & I \end{bmatrix},$$

where I denotes the m" x m" identity matrix. Subproblem (4) then decomposes into m" disjoint subproblems, where the jth (j = 1,...,m") subproblem has the form

$$\begin{aligned} \textit{Minimize} \quad & \sum_{r=1}^{K} (a_j^r - (E_j'')^T p^r) x_j^r \\ \textit{subject to} \quad & x_j^1 + x_j^2 + \ldots + x_j^K = b_j \\ & l_j^r \le x_j^r \le c_j^r \quad , \; r = 1,\ldots,K, \end{aligned}$$

and K = m/m" denotes the total number of distinct commodities, E_j'' denotes the jth column of E", x_j^r denotes the flow of the rth commodity on arc j, p^r denotes the price vector associated with the constraint $E''x^r = 0, \ldots$, etc. In general, if D is block diagonal with K blocks along its diagonal then (4) decomposes into K subproblems.

9. Conclusion and Extensions

We have presented a new dual ascent method for linear programming, based on (ε,δ)-CS, that decomposes the problem in a manner similar to Lagrangian relaxation. This method uses inexact line search, terminates finitely for any $\varepsilon > 0$, $\delta \in [0,1)$, and produces a primal dual pair within $O(\varepsilon)$ of

optimality. For the special case of network flow problems with side constraints, it can be efficiently implemented using network data structures.

The relaxation method also extends directly to the case where some of the constraints are inequality constraints. In this case the dual variables associated with the inequality constraints are constrained to be nonnegative. This method perhaps also extends to monotropic programming problems [13], but more work are needed. Although it may be interpreted as a primal dual method, the relaxation method differs quite substantially from either the primal dual ascent [29]–[31] or the out-of-kilter method [13]. The relaxation method generates dual ascent directions more rapidly, solves each subproblem to within $O(\varepsilon\delta)$ accuracy, and owes its convergence to an $\Omega(\varepsilon(1-\delta))$ improvement in the dual cost per iteration. Furthermore, experimentation on ordinary network flow problems [3], [16] suggests that the relaxation method may be an order of magnitude faster than the primal dual ascent method.

Appendix A

In this appendix we show that, for any $x \in \mathbb{R}^m$ and $(p,\eta) \in \mathbb{R}^{n'}$ satisfing (ε,δ)-CS,

$$0 \;\le\; a^T x + p^T(f - Ex) - q(p) \;\le\; \varepsilon \sum_{j=1}^{m} (c_j - l_j). \tag{A.1}$$

Proof: Since (cf. (7)) $Dx = b$,

$$a^T x \;= a^T x \;- \eta^T D x + \eta^T b\,. \tag{A.2}$$

Let

$$\bar{a} \;= a - E^T p - D^T \eta\,, \qquad \bar{\beta} \;= a - E^T p - D^T \phi(p)\,.$$

Then (A.2) and (6a)-(6c) imply

$$a^T x - p^T E x \;=\; \eta^T b \;+ \sum_{\bar{a}_j > \varepsilon} \bar{a}_j l_j + \sum_{\bar{a}_j < -\varepsilon} \bar{a}_j c_j + \sum_{-\varepsilon \le \bar{a}_j \le \varepsilon} \bar{a}_j x_j\,. \tag{A.3}$$

Since (cf. (8)) $\|D^T\eta - D^T\phi(p)\|_\infty \le \varepsilon\delta \le \varepsilon$, we have

$$\bar{\beta}_j > 0 \qquad \text{if } \bar{a}_j > \varepsilon,$$
$$\bar{\beta}_j < 0 \qquad \text{if } \bar{a}_j < -\varepsilon.$$

This together with (cf. (5))

$$q(p) \;= p^T f + \phi(p)^T b + \sum_{\bar{\beta}_j > 0} \bar{\beta}_j l_j + \sum_{\bar{\beta}_j < 0} \bar{\beta}_j c_j + \sum_{\bar{\beta}_j = 0} \bar{\beta}_j y_j\,,$$

where $y = (\ldots y_j \ldots)$ is any primal vector satisfying (0,0)-CS with $(p,\phi(p))$ (so $b - Dy = 0$), imply

$$q(p) \;= p^T f + \phi(p)^T b + \sum_{\bar{a}_j > \varepsilon} \bar{\beta}_j l_j + \sum_{\bar{a}_j < -\varepsilon} \bar{\beta}_j c_j + \sum_{-\varepsilon \le \bar{a}_j \le \varepsilon} \bar{\beta}_j y_j$$

$$= p^T f + \eta^T b + \sum_{\bar{a}_j > \varepsilon} \bar{a}_j l_j + \sum_{\bar{a}_j < -\varepsilon} \bar{a}_j c_j + \sum_{-\varepsilon \le \bar{a}_j \le \varepsilon} \bar{a}_j y_j\,, \tag{A.4}$$

where the last equality follows from $(\eta - \phi(p))^T(b - Dy) = 0$. Combining (A.3) with (A.4), we obtain

$$a^T x + p^T(f - Ex) \;- q(p) \;= \sum_{-\varepsilon < \bar{a}_j \le \varepsilon} \bar{a}_j (x_j - y_j)\,,$$

from which it follows that

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

and the right hand inequality in (A.1) is proven. To prove the left hand inequality, we note that, since Dx = b and l ≤ x ≤ c, (4) implies that

$$q(p) \;=\; p^T f + \min\{(a - E^T p)^T \xi \mid D\xi = b,\ l \le \xi \le c\} \;\le\; p^T f + (a - E^T p)^T x .$$

Q.E.D.

Appendix B

In this appendix we show that, for each $p \in \mathbb{R}^n$ and $u \in \mathbb{R}^n$, there exists a (n'–n) x n real matrix L from a finite set that satisfies

$$\phi(p+\lambda u) = \phi(p) + \lambda(Lu) \qquad \textit{for } \lambda > 0 \ \textit{ sufficiently small,}$$

where $\phi(p)$ denotes the optimal dual solution to the subproblem (cf. (4))

$$\begin{aligned} &\textit{Minimize} && (a - E^T p)^T x \\ &\textit{subject to} && D x = b\,, \quad l \le x \le c \end{aligned} \tag{Q(p))}$$

Proof: Let B denote the optimal basis for Q(p') , where p' is p perturbed in the direction u. Then, for sufficiently small positive λ, B is an optimal basis for $Q(p+\lambda u)$ and the corresponding optimal dual solution is

$$\phi(p+\lambda u) = (D_B^{-1})^T r_B\,, \tag{B.1}$$

where r denotes the cost vector of $Q(p+\lambda u)$, i.e.

$$r = a - E^T(p+\lambda u). \tag{B.2}$$

Combining (B.1) with (B.2), we obtain

$$\phi(p+\lambda u) = (D_B^{-1})^T(a_B - E_B^T(p+\lambda u)) = (D_B^{-1})^T(a_B - E_B^T p) - \lambda(D_B^{-1})^T E_B^T u = \phi(p) - \lambda(E_B D_B^{-1})^T u\,.$$

It follows that

$$L = -(E_B D_B^{-1})^T\,.$$

[Note that the above analysis suggests a way to update $\phi(p+\lambda u)$ as λ changes by updating the optimal basis for $Q(p+\lambda u)$.]

Appendix C

In this appendix we show that, for any p, u in $\mathbb{R}^n$ and η, π in $\mathbb{R}^{n'-n}$ such that $\|D^T\eta - D^T\phi(p)\|_\infty \le \varepsilon\delta$, ($\varepsilon \ge 0$, $\delta \in [0,1)$),

$$q'(p+\alpha u;u) \;\ge\; C^\varepsilon(p,\eta;u,\pi) \quad \forall\, \alpha \in [0,\varepsilon(1-\delta)/\theta(u)],$$

where $\theta(u) \equiv \max\{\|v\|_\infty \mid$ v given by (13) and (14) for some p $\}$.

Proof: Let y be a primal vector that satisfies (ε,η)-CS with p, let α be a fixed stepsize in $[0,\varepsilon(1-\delta)/\theta(u)]$, and let x be a primal vector that satisfies (0,0)-CS with $(p+\alpha u,\phi(p+\alpha u))$. For notational simplicity denote $p' = p+\alpha u$, $t' = E^T p' + D^T\phi(p')$ and $t = E^T p + D^T\eta$. Then (cf. (15) and (16))

$$C^\varepsilon(p,\eta;u,\pi) = u^T f + \pi^T b - \sum_{a_j - t_j > \varepsilon} v_j l_j - \sum_{\substack{|a_j - t_j| \le \varepsilon \\ v_j < 0}} v_j l_j - \sum_{a_j - t_j < -\varepsilon} v_j c_j - \sum_{\substack{|a_j - t_j| \le \varepsilon \\ v_j > 0}} v_j c_j \,, \tag{C.1}$$

where

$$v = E^T u + D^T \pi \,, \tag{C.2}$$

and also (cf. (12), (13), (14))

$$q'(p';u) = u^T f + \Psi^T b - \sum_{a_j - t_j > 0} v_j l_j - \sum_{\substack{a_j - t_j = 0 \\ v_j < 0}} v_j l_j - \sum_{a_j - t_j < 0} v_j c_j - \sum_{\substack{a_j - t_j = 0 \\ v_j > 0}} v_j c_j \,, \tag{C.3}$$

where

$$\upsilon = E^T u + D^T \Psi \,, \tag{C.4}$$

and ψ satisfies

$$\phi(p'+\lambda u) = \phi(p') + \lambda\Psi \qquad \textit{for } \lambda > 0 \textit{ sufficiently small}.$$

Using (C.1), (C.2) and the fact

$$y_j = \begin{cases} l_j & \forall j \ni a_j - t_j > \varepsilon \\ c_j & \forall j \ni a_j - t_j < -\varepsilon \end{cases} \qquad \textit{and} \quad Dy - b = 0 \,, \tag{C.5}$$

we obtain that

$$C^\varepsilon(p,\eta;u,\pi) = u^T f + \pi^T b - \sum_{a_j - t_j > \varepsilon} v_j l_j - \sum_{|a_j - t_j| \le \varepsilon} v_j y_j - \sum_{a_j - t_j < -\varepsilon} v_j c_j$$

$$- \sum_{\substack{|a_j - t_j| \le \varepsilon \\ v_j < 0}} v_j (l_j - y_j) - \sum_{\substack{|a_j - t_j| \le \varepsilon \\ v_j > 0}} v_j (c_j - y_j)$$

$$= u^T f - \sum_{a_j - t_j > \varepsilon} (E_j^T u) l_j - \sum_{|a_j - t_j| \le \varepsilon} (E_j^T u) y_j - \sum_{a_j - t_j < -\varepsilon} (E_j^T u) c_j$$

$$- \sum_{\substack{|a_j - t_j| \le \varepsilon \\ v_j < 0}} v_j (l_j - y_j) - \sum_{\substack{|a_j - t_j| \le \varepsilon \\ v_j > 0}} v_j (c_j - y_j) . \qquad \text{(C.6)}$$

On the other hand, x satisfies (0,0)-CS with (p',Φ(p')) and therefore

$$x_j = \begin{cases} l_j & \forall j \ni a_j - t'_j > 0, \\ c_j & \forall j \ni a_j - t'_j < 0, \end{cases} \qquad and \quad Dx - b = 0 . \qquad \text{(C.7)}$$

From the definition of $\theta(u)$ we have

$$\| t' - (E^T p + D^T \Phi(p)) \|_\infty \le \alpha \theta(u)$$

which, together with the hypothesis $\|D^T\eta - D^T\Phi(p)\|_\infty \le \varepsilon\delta$ and $\alpha \le \varepsilon(1-\delta)/\theta(u)$, imply that

$$\| (a - t') - (a - t) \|_\infty \le \alpha\theta(u) + \varepsilon\delta \le \varepsilon(1-\delta) + \varepsilon\delta = \varepsilon.$$

It follows that

$$a_j - t'_j > 0 \quad if \ a_j - t_j > \varepsilon,$$
$$a_j - t'_j < 0 \quad if \ a_j - t_j < -\varepsilon,$$

which together with (C.7) imply

$$x_j = \begin{cases} l_j & \forall j \ni a_j - t_j > \varepsilon \\ c_j & \forall j \ni a_j - t_j < -\varepsilon \end{cases} \qquad and \quad Dx - b = 0 . \qquad \text{(C.8)}$$

Then combining (C.3), (C.4) with (C.8), we have

$$q'(p';u) = u^T f - \sum_{a_j - t_j > \varepsilon} (E_j^T u) l_j - \sum_{|a_j - t_j| \le \varepsilon} (E_j^T u) x_j - \sum_{a_j - t_j < -\varepsilon} (E_j^T u) c_j . \qquad \text{(C.9)}$$

Combining (C.6) with (C.9), we obtain

$$C^\varepsilon(p,\eta;u,\pi) - q'(p';u) = - \sum_{|a_j - t_j| \le \varepsilon} (E_j^T u)(y_j - x_j) - \sum_{\substack{|a_j - t_j| \le \varepsilon \\ v_j < 0}} v_j (l_j - y_j) - \sum_{\substack{|a_j - t_j| \le \varepsilon \\ v_j > 0}} v_j (c_j - y_j) .$$

Since b–Dy = 0 and b–Dx = 0, we have D(y–x) = 0, so $\pi^T D(y-x) = 0$. This together with (C.5) and (C.8) imply

$$0 = \sum_{|a_j - t_j| \le \varepsilon} (D_j^T \pi)(y_j - x_j). \tag{C.10}$$

Adding (C.10) to $C^{\varepsilon}(u,\pi,p) - q'(p';u)$ gives

$$C^{\varepsilon}(p,\eta;u,\pi) - q'(p';u) = \sum_{|a_j - t_j| \le \varepsilon} v_j(x_j - y_j) + \sum_{\substack{|a_j - t_j| \le \varepsilon \\ v_j < 0}} v_j(y_j - l_j) + \sum_{\substack{|a_j - t_j| \le \varepsilon \\ v_j > 0}} v_j(y_j - c_j)$$

$$= \sum_{\substack{|a_j - t_j| \le \varepsilon \\ v_j < 0}} v_j(x_j - l_j) + \sum_{\substack{|a_j - t_j| \le \varepsilon \\ v_j > 0}} v_j(x_j - c_j). \tag{C.11}$$

The right hand side of (C.11) is nonpositive and therefore

$$q'(p';u) \ge C^{\varepsilon}(p,\eta;u,\pi).$$

Q.E.D.

References

[1] Assad, A. A., "Multicommodity Network Flows - A Survey," *Networks*, Vol. 8, pp. 37-91 (1978).

[2] Bertsekas, D. P., "A Unified Framework for Minimum Cost Network Flow Problems," *Mathematical Programming*, Vol. 32, pp. 125-145 (1985).

[3] 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; revised September 1986), *Operations Research J.*, Vol. 36, pp. 93-114 (1988).

[4] Dantzig, G. B., *Linear Programming and Extensions*, Princeton Univ. Press, Princeton, N.J. (1963).

[5] Golden, B. and Magnanti, T.L., *Network Optimization*, currently in draft form.

[6] Jewell, W. S., "A Primal Dual Multicommodity Flow Algorithm," Operations Research Center Report 66-24, University of California, Berkeley (1966).

[7] Kennington, J. L., "A Survey of Linear Cost Multicommodity Network Flows," *Operations Research*, Vol. 26, No. 2, pp. 209-236 (1978).

[8] Magnanti, T., "Optimization for Sparse Systems," in *Sparse Matrix Computations* (J. R. Bunch and D. J. Rose, eds), Academic Press, New York, pp. 147-176 (1976).

[9] Magnanti, T. and Golden, B., "Deterministic Network Optimization: A Bibliography," *Networks*, Vol. 7, pp. 149-183 (1977).

[10] Rockafellar, R. T., *Convex Analysis*, Princeton Univ. Press (1970).

[11] 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).

[12] Rockafellar, R. T., "The Elementary Vectors of a Subspace of R^N," in *Combinatorial Mathematics and Its Applications*, by R. C. Bose and T. A. Dowling (eds.), The Univ. of North Carolina Press, Chapel Hill, N. C., pp. 104-127 (1969).

[13] Rockafellar, R. T., *Network Flows and Monotropic Programming*, Wiley-Interscience (1983).

[14] Tarjan, R. E., *Data Structures and Network Algorithms*, Society for Industrial and Applied Mathematics, Philadelphia (1983).

[15] Tseng, P. and Bertsekas, D. P., "Relaxation Methods for Linear Programs," *Mathematics of Operations Research*, Vol. 12, pp. 1-28 (1987).

[16] Tseng, P., "Relaxation Methods for Monotropic Programming Problems," Ph. D. Thesis, Operations Research Center, MIT (1986).

[17] Tseng, P. and Bertsekas, D. P., "Relaxation Methods for Linear Programs with Side Constraints," LIDS-P-1696, Laboratory for Information and Decision Systems, MIT (1987).

[18] Bertsekas, D.P. and Eckstein, J., "Distributed Asynchronous Relaxation Methods for Linear Network Flow Problems," LIDS-P-1606, Laboratory for Information and Decision Systems, MIT (1987); to appear in *Proceedings of IFAC '87*, Pergamon Press (1987).

[19] Tardoz, E. "A Strongly Polynomial Minimum Cost Circulation Algorithm,"*Combinatorica*, Vol. 5, pp. 247-256 (1985).

[20] Tseng, P. and Bertsekas, D. P., "Relaxation Methods for Monotropic Programs," LIDS-P-1697, Laboratory for Information and Decision Systems, MIT (1987).

[21] Bertsekas, D. P. and Mitter, S. K., "A Descent Numerical Method for Optimization Problems with Nondifferentiable Cost Functionals," *SIAM J. Control*, Vol. 11, pp. 637-652 (1973).

[22] Murty, K.G., *Linear Programming*, Wiley & Sons (1983).

[23] Polyak, B.T., "A General Method for Solving Extremal Problems," *Sov. Math. Dokl.*, Vol. 8, pp. 593-597 (1967).

[24] Polyak, B.T., "Minimization of Unsmooth Functionals," *USSR Comput. Math.*, pp. 509-521 (1969).

[25] Demyanov, V.F., "Algorithms for Some Minimax Problems," *J. Comp. Syst. Sci.*, Vol. 2, pp. 342-380 (1968).

[26] Shor, N.Z., *Minimization Methods for Non-Differentiable Functions*, Springer-Verlag (1985).

[27] Held, M., Wolfe, P. and Crowder, H., "Validation of Subgradient Optimization," *Mathematical Programming*, Vol. 6, pp. 62-88 (1974).

[28] Oettli, W., "An Iterative Method, Having Linear Rate of Convergence, for Solving a Pair of Dual Linear Programs," *Mathematical Programming*, Vol. 3, pp. 302-311 (1972).

[29] Grinold, R.C., "Steepest Ascent for Large Scale Linear Programs," *SIAM Review*, Vol. 14, pp. 447-464 (1972).

[30] Fisher, M.L., Northup, W.D., and Shapiro, J.F., "Using Duality to Solve Discrete Optimization Problems: Theory and Computational Experience," in *Mathematical Programming Study 3: Nondifferentiable Optimization* (Balinski and Wolfe eds.), pp. 56-94, North-Holland (1975).

[31] Lemarechal, C., "An Algorithm for Minimizing Convex Functions," in *Information Processing* (Rosenfeld, J.L. ed.), pp. 552-556 (1974).

[32] Murty, K.G., *Linear Programming*, John Wiley & Sons (1983).