CORC REPORT 1999-1

Approximately solving large-scale linear programs. I. Strengthening lower bounds and accelerating convergence*

Daniel Bienstock

Columbia University
New York, NY 10027

June, 1999

Abstract

This report describes computational experience with an implementation of Plotkin-Shmoys-Tardos and Grigoriadis-Khachiyan approximation algorithm for block-angular linear programs, together with a technique for improving Lagrangian relaxations. Our implementation produces fast approximate solutions to large pure multicommodity flow problems, and linear programs arising in network design.
A preliminary version of this report has been accepted for presentation at the 11th Annual SIAM-ACM Symposium on Discrete Algorithms (SODA 2000).

1 Introduction

In many applications one must handle linear programs with the general structure

$$\min \quad c^T x$$
s.t.
$$Ax \geq b \tag{1}$$
$$x \in P \tag{2}$$

where P is a polytope over which we can efficiently solve linear programs. We are especially interested in problems of this sort that arise as continuous relaxations of difficult mixed-integer programs with large duality gap. In such a case it is essential to obtain a better formulation quickly, and the standard cutting-plane framework provides a way to achieve this. However, when the linear programs are very difficult this approach may not be feasible.

In recent years, research has produced algorithms (see [6], [7], [11] and the references therein) that seek to produce *approximate* solutions to linear programs of this sort, by constructing an exponential potential function which serves as a surrogate for feasibility and optimality. These algorithms have provably good convergence properties for broad problem cases.

*This research was partially funded by NSF awards NCR-9706029, CDA-9726385 and 29132-5538 (CORC)

In this report, we describe some new ideas to make this basic methodology workable across various problem classes. Our contributions are

- A new lower bounding technique, which provably improves over the standard lower bounding method,
- A periodic "pure LP" step which accelerates convergences of the basic method and incorporates the lower bounding procedure, and
- A different way for conditioning the potential function.
- A general-purpose implementation with good computational results for large multicommodity flow problems and large LP-relaxations of network design problems.

This paper is organized as follows. In Section 2 we discuss a general method for improving lower bounds obtained by a Lagrangian procedure. In Section 3 we provide background on the exponential potential reduction method. In Section 4 we describe our implementation of the method. In Section 5 we present computational experience.

2 Improving Lagrangian Relaxations.

Consider the linear program

$$\textbf{(LP)} \qquad \begin{array}{rl} \min & c^T x \\ \text{s.t.} & \end{array}$$

$$Ax = b \tag{3}$$

$$x \in P \tag{4}$$

where P is a polytope. Let z^* denote its optimal objective value. Suppose we relax constraints (3) using multipliers π. This yields the *Lagrangian*

$$L(x, \pi) = (c^T - \pi^T A)x + \pi^T b \tag{5}$$

It is well known that for any π,

$$\min \{L(x, \pi) \, : \, x \in P\} \leq z^* \tag{6}$$

Suppose $\hat{x}$ minimizes the Lagrangian in (6) and let $L^*(\pi)$ be the resulting optimal value. The observation to be made is that if

$$c^T \hat{x} > L^*(\pi) \tag{7}$$

then we should be able to obtain a stronger lower bound on z^* than $L^*(\pi)$. In essence, when this situation arises the multipliers π are poor approximations to the optimal dual variables for constraints (3).In this section we discuss a simple procedure to obtain better multipliers, should this situation arise. To this effect, for *fixed* π, and any real B denote:

$$LP(\pi, B) \qquad \begin{array}{rl} F_\pi(B) = \min & L(x, \pi) \\ \text{s.t.} & \end{array}$$

$$c^T x \leq B \tag{8}$$

$$x \in P \tag{9}$$

The intuition here is that if B were an *upper* bound on z^* then constraint (8) should be redundant. Now we have

Lemma 2.1 *For any B,* $\min\{F_\pi(B), B\} \le z^*$.

Proof. Note that $F_\pi(z^*) \le z^*$, since with $B = z^*$, the linear program $LP(\pi, B)$ is simply a relaxation of LP. Now let $K = \min\{F_\pi(B), B\}$, and assume by contradiction that $K > z^*$. Then $F_\pi(B) \le F_\pi(K) \le F_\pi(z^*) \le z^*$, a contradiction. ∎

We are thus led to the problem of finding B^π that maximizes $\min\{F_\pi(B), B\}$ (we remind the reader that π is fixed). Since $F_\pi(B)$ is a continuous, monotonely nonincreasing function of B, we have that B^π exists, is unique and satisfies $F_\pi(B^\pi) = B^\pi$.

One approach to computing B^π would be to apply binary search (over B) to $F_\pi(B)$. However, computational experience suggests $LP(\pi, B)$ may be difficult to solve by direct methods even if we have a fast algorithm for solving linear programs over P. [8] describes a technique that we could use to approximately solve $LP(\pi, B)$. However, an alternative approach is that of (simultaneously) searching for B^π and the corresponding optimal dual variable for constraint (8), and this turns out to provide a much better insight.

Following this approach, for given B and $\gamma \ge 0$ (this value being a guess for the *negative* of the dual variable for (8)), we have that

$$\min_{x \in P} \left\{ L(x, \pi) + \gamma c^T x - \gamma B \right\} \quad \le \quad F_\pi(B), \tag{10}$$

and consequently

$$\min_{x \in P} \left\{ L(x, \pi) + \gamma c^T x \right\} \quad \le \quad F_\pi(B) + \gamma B, \tag{11}$$

or equivalently,

$$\min_{x \in P} \left\{ L(x, \frac{\pi}{1+\gamma}) \right\} \quad \le \quad \frac{F_\pi(B) + \gamma B}{1+\gamma}. \tag{12}$$

Since $F_\pi(B^\pi) = B^\pi$, we have that for any $\gamma \ge 0$,

$$\min_{x \in P} \left\{ L(x, \frac{\pi}{1+\gamma}) \right\} \quad \le \quad B^\pi. \tag{13}$$

Finally,

Lemma 2.2 *Let γ^π be the negative of the optimal dual variable for constraint (8) in $LP(\pi, B^\pi)$. Then*

$$\max_{\gamma \ge 0} \min_{x \in P} \left\{ L(x, \frac{\pi}{1+\gamma}) \right\} \quad = \quad \min_{x \in P} \left\{ L(x, \frac{\pi}{1+\gamma^\pi}) \right\} \; = \; B^\pi \; = \; \max_B \min\left\{F_\pi(B), B\right\}. \tag{14}$$

Proof. This follows from the fact that B^π and γ^π satisfy (10) with equality. ∎

In other words, we must maximize (as a function of nonnegative γ) the left-hand side of (13), which we denote $g_\pi(\gamma)$. Let v^j, $j \in J$ be the set of extreme points of P. We have that

$$g_\pi(\gamma) \quad = \quad \min_{j \in J} \left\{ c^T v^j + \frac{\pi^T}{1+\gamma}(b - Av^j) \right\}. \tag{15}$$

For each j, the quantity in the right-hand side of the above equation is, as a function of γ, either constant, or strictly increasing, or strictly decreasing. Thus, $g_\pi(\gamma)$ is a unimodal function of γ, and we can maximize it using binary search over γ, provided that an upper bound on γ^π is known. Here, each iteration in the binary search involves the solution of a linear program over P.

3 The exponential penalty function framework.

Consider a linear program of the form

$$\textbf{(LP2)} \qquad \begin{array}{lrcl} \min & c^T x & & \\ \text{s.t.} & & & \\ & Ax & \leq & b \qquad (16) \\ & x & \in & P. \qquad (17) \end{array}$$

Suppose we have an efficient algorithm for solving a linear program over the polytope P, but that constraints (16) make the combined problem difficult. The classical approach of Lagrangian relaxation, coupled with subgradient optimization, aims to effectively solve such a linear program by removing constraints (16) and instead penalyzing their violation (see [4] for recent developments and more complete references).

The approach described in [6], [11], on the other hand, seeks to solve LP2 to a desired precision ϵ, essentially by attempting to minimize a related *potential* function. In order to describe the methodology in [6], [11], suppose first that we are trying to find a feasible solution to a system of the form

$$Hx \leq d \qquad (18)$$

$$x \in P, \qquad (19)$$

for some matrix H with m rows and *nonzero* right-hand side vector d. Denote this system by **FEAS**. Given $\epsilon > 0$, if a feasible solution to this system exists, the approach in [6], [11] will find a vector $x \in P$ with $Hx \leq (1+\epsilon)d$ (such a vector is called ϵ-feasible). To this end, let the rows of H by $h_1, h_2, \cdots, h_m$. The potential function associated with FEAS is:

$$\Phi(x) = \sum_i e^{\alpha(\frac{h_i x}{d_i} - 1)} \qquad (20)$$

where $\alpha > 0$ and the sum is over the rows of H. The intuition is that an x feasible for FEAS will have low potential (at most m), but an $x \in P$ which violates $Hx \leq d$ will have high potential, provided that α is large enough. More precisely, if we choose α to be inversely proportional to ϵ, then points that are *not* ϵ-feasible will have high potential. The potential reduction iterates through a sequence of vectors $x^1, x^2, \cdots$ in P, essentially by applying steepest descent (or, more precisely, Frank-Wolfe iterations) to reduce the potential. Given an iterate x^k, the method solves the problem,

$$\textbf{(DIR)} \qquad \begin{array}{lrcl} \min & [\nabla\Phi(x^k)]^T y^k & & \\ \text{s.t.} & & & \\ & y^k & \in & P. \qquad (21) \end{array}$$

The next iterate, x^{k+1}, is chosen of the form

$$x^{k+1} = \beta y^k + (1-\beta)x^k. \qquad (22)$$

where $0 \leq \beta \leq 1$ is chosen so that x^{k+1} minimizes the potential in the segment $[x^k, y^k]$. In what follows we will describe an iteration of the form just described as a *potential reduction step.*

In [6], [11], it is shown the sequence $\{x^j\}$ will converge to an ϵ-feasible solution to FEAS, provided this system is feasible, or prove that FEAS is infeasible, in a polynomial number of potential reduction steps, under appropriate technical assumptions on H and P. Primarly, the number of iterations is $O(\epsilon^{-2})$. Note that

$$\begin{aligned} [\nabla\Phi(x^k)]^T &= v^T H \quad \text{where} && (23)\\ v_i &= \frac{\alpha}{d_i} e^{\alpha(\frac{h_i x^k}{d_i} - 1)}, && (24)\end{aligned}$$

and consequently we can view the overall algorithm as a Lagrangian relaxation procedure, with multipliers v being used in iteration k. However, the algorithm does not contain a subgradient optimization component, as is more properly a Dantzig-Wolfe algorithm. This has implications in terms of experimental behavior (discussed later).

The procedure just described handles feasibility problems. To approximately solve the linear program LP2 given above, we convert it into a sequence of feasibility problems: given a target "budget" B, the objective $c^T x$ is included as a budget constraint $c^T x \leq B$, yielding the system $FEAS(B)$:

$$\begin{aligned} c^T x &\leq B && (25)\\ Ax &\leq b && (26)\\ x &\in P, && (27)\end{aligned}$$

with the quantity B determined by binary search (a different approach is used in [7]).

Finally, the method can be extended to handle *homogeneous* inequalities. Suppose that we have an inequality

$$h_i x \leq 0 \qquad (28)$$

Then ([11]) we introduce a new variable y, replace (28) by

$$h_i x + ay \leq a, \qquad (29)$$

and replace P by

$$\hat{P} = P \times \{y \,:\, y = 1\}, \qquad (30)$$

where the constant $a > 0$ is chosen so that the left-hand side of (29) is nonnegative for any $x \in P$. Using this representation, the term in the potential function corresponding to (29) will be $e^{\alpha\frac{h_i x}{a}}$, i.e. we penalize the (absolute) violation scaled by a. See [11] for further details.

One final element to be discussed is that, in many practical applications (for example, multicommodity flows) the polyhedron P decomposes into a product of polyhedra $P^1 \times P^2 \times \cdots \times P^M$ – in other words, LP2 has a "block-angular" structure with "linking" constraints (16). For notational purposes, write $P^i \subset R^{P^i}$. In such a case, the direction-finding linear program **DIR** decomposes into M independent linear programs. More important (as shown in [6], [11]) there are variants of the basic potential reduction procedure (with theoretically good behavior) which restrict each iteration to a *single* block P^i. In other words, instead of solving, at each iteration, problem **DIR**, we pick a single block i $(1 \leq i \leq M)$ and restrict the step to the coordinates in block i only. More precisely, in the k^{th} step we solve the problem

$$\textbf{(DIRi)} \qquad \begin{aligned} &\min \quad [g^{k,i}]^T y^k \\ &\text{s.t.} \\ &y^k \in P^i, \end{aligned} \qquad (31)$$

where $g^{k,i}$ is the projection of $\nabla\Phi(x^k)$ to R^{P^i}. The next iterate, x^{k+1}, is obtained by using (22) restricted to coordinates in R^{P^i}, and by setting x^{k+1} equal to x^k for all remaining coordinates.

It can be shown (see the references for a complete discussion) that if this block i is chosen using either a round-robin, or random, rule, the overall algorithm is still guaranteed to terminate in polynomial time.

3.1 Outline of the basic method

In this section we outline a basic implementation of the exponential potential reduction algorithm for approximately solving a linear program of the form

$$\textbf{(LP3)} \qquad \begin{array}{c} \min \quad c^T x \\ \text{s.t.} \end{array}$$

$$Ax \leq b \tag{32}$$

$$x \in P^1 \times P^2 \times \cdots \times P^M. \tag{33}$$

for general A, and P^i (possibly $M = 1$).

Given $\epsilon > 0$ we seek to find a vector with relative infeasibility at most ϵ and which is at most a factor of $(1 + \epsilon)$ more expensive than the optimal value of the linear program, i.e. an ϵ-feasible, ϵ-optimal vector. The algorithm runs a sequence of inner iterations, for given $\delta > 0$ and budget B. The objective of an inner iteration is to determine a δ-feasible solution to the system $FEAS(B)$ (restated here)

$$c^T x \leq B \tag{34}$$

$$Ax \leq b \tag{35}$$

$$x \in P^1 \times P^2 \times \cdots \times P^M. \tag{36}$$

or to determine that the system is infeasible. The inner iterations are organized into B-phases, during which B is kept constant, but δ varies, starting from an initial value $\delta > \epsilon$ and is progressively decreased until reaching the desired value ϵ, or until we determine that $FEAS(B)$ is infesible.

***B*-phase**

Step 0. Initialization. We start with a vector $x \in P^1 \times P^2 \times \cdots \times P^M$, and $\delta > \epsilon$ with $Ax \leq (1+\delta)b$ and $c^T x \leq (1 + \delta)B$.

Step 1. Set $\delta = \max\left\{\frac{\delta}{2}, \epsilon\right\}$.

Step 2. Core Procedure. Run potential reduction steps to determine whether $FEAS(B)$ has a δ-feasible solution or is infeasible.

(a) If $FEAS(B)$ is infeasible, exit;

(b) Otherwise, if $\delta > \epsilon$ go to Step 1, and if $\delta = \epsilon$, exit.

END

The core procedure (Step 2) embodies the key elements described in Section 3. In our implementation, each potential reduction step is taken using a *single* block P^i, chosen randomly.

[Note that the core procedure may terminate in case **(a)** i.e. by deciding that $FEAS(B)$ is infeasible. In [11] the authors describe a method based on weak linear programming duality that will, if successful, prove that $FEAS(B)$ is infeasible. This method was used in [5]. In the next section we will describe a refinement of this method that is provably stronger. Also, [10] describes a different method, based on linearization of the potential function, to prove infeasibility of $FEAS(B)$. We have implemented an improvement of this method; however, in our latest testing this criterion was not particularly useful and we will not describe it here.]

The main algorithm for the approximate LP-solver runs B-phases for varying values of B. The algorithm begins with values $\underline{\mathrm{B}}$ and $\bar{B}$, such that (i) $FEAS(\underline{\mathrm{B}})$ is infeasible (i.e. $\underline{\mathrm{B}}$ is a lower bound to LP3), (ii) $FEAS(\bar{B})$ is known to be ϵ-feasible, and (iii) $\bar{B} \leq (1+\epsilon)\underline{\mathrm{B}}$.

Main Algorithm

Step 0. Initialization. We start with $\underline{\mathrm{B}} = -\infty$ and a valid $\bar{B}$.

Step 1. Set $B = 0.5(\underline{\mathrm{B}} + \bar{B})$. Run a B-phase to handle problem $FEAS(B)$.

(a) If $FEAS(B)$ is infeasible, set $\underline{\mathrm{B}} = B$.

(b) If $FEAS(B)$ is ϵ-feasible, set $\bar{B} = B$.

Step 2. If $\bar{B} \leq (1+\epsilon)\underline{\mathrm{B}}$, exit. Otherwise, go to Step 1.

END

To complete the description of the implementation, the initial value $\bar{B}$ is obtained by running a "pure feasibility" version of the Core Procedure, i.e. by using the potential reduction method to determine an ϵ-feasible $x \in P^1 \times P^2 \times \cdots \times P^M$.

4 Our implementation

In this section we describe the key details in which our implementation differs from the basic method described above.

4.1 Periodic "exact" restricted linear programs

As far as we know, to date all implementations of the potential reduction method have been for multicommodity flow problems (see the references). In such problems, the direction-finding subproblem **DIRi** (Section 3) is either a shortest path problem, or a minimum-cost flow problem. These implementations made heavy use of this fact, i.e., highly-tuned codes for minimum-cost flows were used for **DIRi**. In our implementation, we use a generic LP-solver to handle **DIRi**. Thus, keeping the number of iterations small becomes very important.

In our experimental testing, we found that two sperate, but related, aspects of the algorithm that could be improved, leading to dramatic reductions in the number of iterations: (i) the lower bounding procedure, and (ii) the "quality" of the primal solution.

We will consider (ii) first. Recall that in a potential reduction iteration, the next iterate, x^{k+1}, is a convex combination of x^k and another point y^k which is also in P. In other words, each iterate will be a convex combination of points in P. Suppose, for example, that problem LP3 is a multicommodity flow problem, where P describes the flow conservation equations for the different commodities and (16) describes joint capacity inequalities. Then it may well be the case that, for k large enough, x^k may contain flow-carrying cyles for some of the commodities. In general, removing those cycles will improve the cost of the solution and will at the very least not worsen the violation of inequalities (16). It is not clear how to extend this technique to general LP3.

Regarding (i), recall from the previous section that a new lower bound is proved by the algorithm whenever a system $FEAS(B)$ is proved infeasible. In [5] the authors essentially use the multipliers v_i in (24) as dual variables, in order to compute a lower bound on the largest $\delta > 0$ such that there is an $x \in P$ with $Ax \leq (1+\delta)b$ and $c^T x \leq (1+\delta)B$. In our experiments, we found that in order for either of these two lower bounding procedures to prove effective we needed to run many potential reduction iterations (e.g. the multipliers v_i did not quickly converge to good approximations of optimal dual variables).

In our testing we found that, even though the v_i did not quickly converge to good values, the x^k would quickly converge to a set of "good columns". Even though the number of columns in $LP3$ may be very large (of the order of one million, say) for relatively small k it would be the case that many of the components of x^k were very close to zero. This suggests looking at the linear program

$$\textbf{(LP3(k))} \qquad \min \; c^T x \quad \text{s.t.}$$

$$Ax \leq b \tag{37}$$

$$x \in P \tag{38}$$

$$x_j = 0 \text{ if } j \in Z(k) \tag{39}$$

where $Z(k) = \{j : |x_j^k| \leq \theta\}$ and θ is a parameter. In our testing we used $\theta = 1.0e^{-06} \sum_j |x_j^k|$. Thus $LP3(k)$ is the original linear program restricted to the "large" variables in x^k. More precisely, suppose we are considering a system $FEAS(B)$. Then we run the following procedure:

Procedure RESTRICTLP

Step 1.. Solve $LP3(k)$. If infeasible, quit RESTRICTLP. Otherwise, let y^k be its optimal solution and π^k the vector of optimal dual variables corresponding to inequalities (37).

Step 2. Set $x^{k+1} = \beta y^k + (1-\beta)x^k$, where $0 \leq \beta \leq 1$ is chosen so that x^{k+1} minimizes the potential in the segment $[x^k, y^k]$.

Step 3. Compute $B^k = \max_{\gamma \geq 0} \min_{x \in P} \left\{ L(x, \frac{\pi^k}{1+\gamma}) \right\}$. Where $\underline{\mathrm{B}}$ is the current lower bound on the value of $LP3$, we reset $\underline{\mathrm{B}} \leftarrow \max\{B^k, \underline{\mathrm{B}}\}$. Furthermore, if $B^k > B$, then system $FEAS(B)$ is declared infeasible.

END

In our implementation, we run RESTRICTLP each $1.5M$ potential reduction steps.

4.2 Proving lower bounds

Step 3 of RESTRICTLP may yield an improved lower bound on the value of $LP3$ using the techniques of Section 2. In our implementation, this is the computationally most effective lower bounding technique. However, similar techniques can be used to improve on the lower bound method of [11]. We discuss this next.

Consider a system $FEAS(B)$ handled by the Core Procedure, i.e. a system of the form

$$c^T x \leq B \tag{40}$$

$$Ax \leq b \tag{41}$$

$$x \in P, \tag{42}$$

Suppose that $b > 0$ and $B > 0$. One way to prove that this system is infeasible is to show that $\hat{\lambda}$, defined as the minimum value λ such that the system

$$c^T x \leq \lambda B \tag{43}$$

$$Ax \leq \lambda b \tag{44}$$

$$x \in P, \tag{45}$$

is feasible, satisfies $\hat{\lambda} > 1$. In [11] the following approach is used: let $v > 0$ be a scalar, and $\pi > 0$ be a vector of the same dimension as b. Define

$$\bar{\lambda} = \frac{\min_{x \in P} \{(vc^T + \pi^T A)x\}}{vB + \pi^T b}. \tag{46}$$

Then a simple analysis shows that $\hat{\lambda} \geq \bar{\lambda}$. Thus, if $\bar{\lambda} > 1$ we have that $FEAS(B)$ is infeasible, or, in other words, we will have that $\bar{\lambda}B$ is a lower bound on the value of LP3.

This method is used in [5] to prove lower bounds. The minimum in (46) is relatively expensive to compute since it involves all the blocks P^i. Thus, when $\bar{\lambda} > 1$, it is worth trying to prove an even better lower bound. Notice that (46) can be rewritten as

$$\bar{\lambda}(vB + \pi^T b) = \min_{x \in P} \left\{ (vc^T + \pi^T A)x \right\} \tag{47}$$

or

$$\bar{\lambda}(vB + \pi^T b) - \pi^T b = \min_{x \in P} \left\{ (vc^T + \pi^T A)x - \pi^T b \right\} \tag{48}$$

which, since $v > 0$, can be restated as

$$\bar{\lambda}B + (\bar{\lambda} - 1)\frac{\pi^T b}{v} = \min_{x \in P} \left\{ (c^T + \frac{\pi^T}{v}A)x - \frac{\pi^T}{v}b \right\} \tag{49}$$

The function minimized in the right-hand side of (49) is a Lagrangian, which therefore is a lower bound on the value of LP3. Since $\bar{\lambda} > 1$, and $\pi > 0$, $b > 0$ and $v > 0$ we have that this lower bound is *strictly stronger* than $\bar{\lambda}B$.

Finally, if we apply the technique in Section 2 to the dual variables $\frac{\pi}{v}$ we will prove a lower bound at least as good as $\bar{\lambda}B + (\bar{\lambda} - 1)\frac{\pi^T b}{v}$.

4.3 Fine-tuning the potential function

In the definition of the potential function (20) used to approximately solve **LP2** (Section 3), the contribution of the i^{th} constraint in $Ax \leq b$, is of the form $e^{\alpha s_i(k)}$ where $s_i(k)$ indicates the relative violation of constraint i by x^k. In order for the standard algorithm to have polynomial-time convergence, the constant α should be chosen inversely proportional to the desired accuracy ϵ, and proportional to the *logarithm* of the number of rows in A. In [10], the authors show that choosing $\alpha = \frac{ln(3m)}{\epsilon}$ guarantees polynomial-time convergence.

However, even a modest amount of experimentation will convince an implementor that the theoretically correct choice for α is much too large. In [1] we chose $\alpha = \frac{0.2}{\epsilon}$. In [5], the authors use a heuristic rule for setting α; typically this rule also sets α small in relation to $\frac{ln(3m)}{\epsilon}$ but tries to avoid setting it "too" small. In this Section we present a different method for choosing α.

One way to observe why a large value of α may impede practical convergence is to consider, is to recall that the exponential potential reduction method is essentially a steepest-descent (or better, a Frank-Wolfe) method, and therefore may be slow to converge when the Hessian matrix of the potential function is ill-conditioned. Consider a system

$$Hx \leq d \tag{50}$$

$$x \in P, \tag{51}$$

(where H has m rows and $d > 0$) for which we seek an ϵ-feasible solution. and the system

$$z \leq \mathbf{1} \tag{52}$$

$$(x, z) \in \hat{P} \tag{53}$$

where $\mathbf{1}$ is the vector of m 1's, and

$$\hat{P} = \left\{ (x, z) \,:\, x \in P,\, z_i = \frac{h_i x}{d_i},\ 1 \leq i \leq m \right\} \tag{54}$$

where h_i is the i^{th} row of H. For $x \in P$, z is simply the vector of relative violations of $Hx \leq d$. It can be seen that the exponential potential reduction method will generate the same sequence of iterates for both systems. Consider the second system at iteration k. Then it can be seen that the Hessian of the potential function is proportional to $diag \left\{ e^{\alpha z_1^k}, \cdots, e^{\alpha z_m^k} \right\}$. Consequently, we can expect that if there is a large disparity between violations of different constraints, a steepest-descent method may converge slowly.

A different way to look at this effect, which better explains what we observe experimentally, is to consider a "pure covering" linear program

$$\min \quad c^T x$$

s.t.

$$Ax \geq b \tag{55}$$

$$x \in P \tag{56}$$

where $A \geq 0$, $b > 0$, $c \geq 0$ and P is contained in the nonnegative orthant. When considering a feasibility system $FEAS(B)$ for current budget B arising from this linear program, the budget constraint $c^t x \leq B$ is "competing" against all the constraints in (55). A likely result is that after applying the potential reduction method for some iterations, the budget constraint will be the *most violated*, by far. If α is large, the direction finding problem **DIR** will next choose a direction which essentially will be the negative of c. However, after moving a small amount in this direction, the contribution to potential from the budget row will dramatically decrease, whereas the *combined* contribution from all the constraints (55) will become a factor. In other words, we will choose a small step size, leading to slow convergence. If α is too large, the chosen stepsize may be numerically zero, i.e. the algorithm has jammed.

At the other end of the spectrum, when α is chosen too *small*, then the minimizer for the potential function will be relatively deeply in the interior of the polyhedron defined by $Ax \geq b$ and will be far away from the LP optimum. Here we may see iterates of *increasing* objective value but which satisfy $Ax > b$ (and we have seen this experimentally).

To avoid the two extreme cases for α we use the following rule. When considering a system $FEAS(B)$ α is initialized to $\frac{ln(1.5m)}{\epsilon}$. After each run of RESTRICTLP, if this run found a strictly better feasible solution (i.e. feasible according to the underlying "exact" LP solver) than what we had before, we interpret this fact to mean that the exponential potential reduction method has "done its job" i.e. it has converged to the neighborhood of a good solution. In such a case, we *reduce* α by a factor of γ. We use $\gamma = 2$ if $\epsilon > 2\%$, thereafter we set $\gamma = 1.1$.

On the other hand, if at any point of the algorithm we observe that the objective value of the current iterate is *larger* than that of the previous iterate, while at the same time the average relative infeasibility of constraints other than the budget constraint, is negative, then we *increase* α by a factor of two.

4.4 Other implementation details

Our implementation uses CPLEX 6.5 to solve all linear programs with constraint set P^i as general *linear programs*, regardless of any special structure, using the primal simplex method with all options set to the default. This applies both to the linear programs arising in the direction step **DIRi** as well as those that arise when solving problems of the form $max_{\gamma \geq 0} \min_{x \in P} \left\{ L(x, \frac{\pi^k}{1+\gamma}) \right\}$, where each Lagrangian minimization breaks up into linear programs over the individual blocks. The linear programs **LP3(k)** handled in the RESTRICTLP procedure are solved using the barrier method until the gap between $\underline{B}$ and $\bar{B}$ is small, at which point we start using the steepest-edge primal simplex method, warm-started using the current iterate of the ϵ-approximation method (those variables that are not fixed at zero in **LP3(k)**); both methods used with default tolerances.

The implementation proves lower bounds using the technique in Section 2 as used in the RESTRICTLP steps described in Section 4.1. The upper bounds $\bar{B}$ are likewise produced in RESTRICTLP. In other words, we obtain upper bounds by running the underlying "exact" LP solver (in this case, CPLEX) on a restricted linear program; consequently our upper bounds are attained by solutions whose feasibility error is that of the underlying LP solver (i.e. much smaller than the values of ϵ we will use).

Our implementation reads a file with a formulation described in the ".lp" format, with the additional feature that the constraints describing each individual block P^i are listed consecutively; and that different blocks are delimited by the keyword "BLOCK". No other arguments, parameters, or configuration files are used. By default, the code attempts to achieve $\epsilon = 0.0001$.

5 Computational Experience

In this initial report we report on computational experience on problems of two classes: (a) relaxations of formulations of several network design problems, and (b) pure multicommodity-flow problems. Unless specifically stated, all experiments were carried out on a 667 Mhz ev5 Alpha computer, with 1GB of memory and 8MB of cache.

5.1 Network design problems.

The basic problem is described [3]. Given a directed graph G and fractional multicommodity demands, the mixed-integer program consists of installing integer capacities on the edges of the graph, at minimum-cost, so that the demands can be routed. The linear programming relaxations, strengthened with facet-defining inequalities, have the general structure

$$\min \quad \textstyle\sum_{ij} c_{ij} x_{ij}$$

s.t.

$$Ax \geq b \tag{57}$$

$$\sum_{(j,i)} f_{k,ji} - \sum_{(i,j)} f_{k,ij} = d_i^k \tag{58}$$

$$\sum_k f_{k,ij} - x_{ij} \leq a_{ij} \tag{59}$$

$$x_{ij} \geq 0, \ \ f_{k,ij} \geq 0 \quad \forall k, (i,j) \tag{60}$$

Here $f_{k,ij}$ is the flow of commodity k on edge (i,j), x_{ij} is the capacity we buy for edge (i,j); (58) are flow-conservation equations and (59) are capacity inequalities (where a_{ij} is the "available" capacity on edge (i,j)). Constraints (57) are facet-defining inequalities of several types. The simplest have the general structure

$$\sum_{(i,j)\in C} x_{ij} \geq k \tag{61}$$

where C is a set of edges (e.g. a directed cut) and k is an integer. Some of the examples we consider below include "survivability" modeling features, and here we have some inequalities of the form

$$\sum_{(i,j)\in C^1} x_{ij} + \sum_{(i,j)\in C^2} \frac{r+1}{r} x_{ij} \geq k \tag{62}$$

where $r > 0$ is an integer (see [2], [9]).

On these problems, we observed that CPLEX's barrier code was far superior to the dual simplex method, whenever the problem was large enough, as was the case for all the instances listed below.

Problem	m	n	nonz	Barrier time	Barrier 1% time	ϵ-approx 1% time
netd1	31317	270760	793143	1097	681	17
netd2	31363	270806	799783	1563	1383	81
netd3	31746	270866	837132	3697	628	21
netd4	32496	427085	1256593	16775	10000	34
netd5	32866	427180	1284466	17870	11047	387
netd6	32496	427085	1256593	25231	10336	60
netd7	33003	427215	1316879	44688	12305	47
netd8	33218	427215	1329112	51140	11976	59
netd9	33966	588618	1777359	89577	73166	343

Table 1: Network design problems

In Table 1, "n", "m" and "nonz" are, respectively, the number of columns, constraints and nonzeroes after the linear program has been preprocessed by CPLEX. "Barrier time" is the time that the barrier code required to solve the problem to default tolerances, in seconds, excluding the time required to compute the Cholesky factorization. "barrier 1% time" is the time, in seconds, that the barrier code ran until the half-sum of the "primal value" and "dual value" (as listed in CPLEX's output) was within 1% of the eventual optimum. "approx 1%" is the time it took our code to prove a lower bound $\underline{B}$ and an upper bound $\bar{B}$ such that $\bar{B} \leq 1.01\bar{B}$.

As stated above, the dual method (incl. using steepest edge pivoting) was inferior to the barrier code, particularly in the more difficult instances. For example, on instance **netd9**, after several days, dual simplex had computed a bound that was more than 20% away from the optimum. Also note that the time required by the barrier code to compute the initial ordering of columns is nontrivial; on **netd9** this was of the order of 800 seconds.

5.2 Multicommodity flow problems

There is an abundant bibliography concerning multicommodity flow problems; see [5]. These problems arise in the telecommunications industry, for example. Nevertheless, it is difficult to obtain "real" large-scale instances, and researchers in this area have traditionally relied on random problem generators.

In [5] the authors describe computational results with a multicommodity-flow code based on the exponential potential reduction method. To test their code, they used the generators "RMFGEN", "MULTIGRID" and "TRIPARTITE". The first two are multicommodity variations of well-known single-commodity network flow problems. The third one ([5]) generates problems that are difficult for the exponential potential reduction method. In this section, we test our code on instances produced by these three generators provided to us by the authors of [5] and on instances we obtained using these generators. It is important to note that the behavior of each generator is consistent in terms of the difficulty of the instances it produces; thus, we report on a sample of the problems only.

We note that each "block" (or, rather, each problem **DIRi**) reduces to a minimum-cost flow problem. In [5] these were handled using a network-simplex code, warm started. However, we stress that we solve these problems as general linear programs. In spite of this fact, we obtain nontrivial speed-ups over the code in [5]. We attribute this fact to the decreased iteration count, which itself is due to the better lower bounds and to the RESTRICTLP procedure.

The next table describes results with TRIPARTITE instances, the last two of which we generated, and were not run using the code in [5]. In this table, the column headed " ϵ-approx iter" shows the number of problems **DIRi** solved in this run. For completeness, we also show statistics for the first two runs using a recent version of the code in [5], run on a SUN Ultra 2 (approximately twice as slow as our machine). On TRIPARTITE problems the Barrier method appears to be much more effective than the dual simplex; however, it does require significantly more memory. The 1%-approximation Cplex running times using the barrier method follow the same convention as in Table 1; in the case the dual simplex method we report the run time when the method was used with a 1% optimality tolerance.

Problem	m	n	nonz	Barrier time	Barrier 1% time	ϵ-approx 1% time	GOPS 1% time*	ϵ-approx 1% iter	GOPS 1% iter
tripart1	3838	25632	75232	29	17	18	338	941	66072
tripart2	18741	127007	368526	237	176	509	6772	2831	180368
tripart3	37541	312034	917288	1126	875	1287	-	3272	-
tripart4	54769	783817	2322024	11624	9687	2370	-	5191	-

Table 2: TRIPARTITE instances. * = run on Sun Ultra 2, "-" = not run

he next table describes two instances created by RMFGEN, and an instance created by MULTIGRID. The first problem was run using the Barrier method; the other two using the steepest-edge Dual simplex method because of memory limitations.

Problem	m	n	nonz	Cplex time	Cplex 1% time	ϵ-approx 1% time	ϵ-approx 1% iter
rmfgen1	27352	116403	257443	187	133	115	3292
rmfgen2	156891	670052	2009281	9940	4868	1292	31840
multigrid1	271039	818943	2414742	$>$ 50000	1200	230	1923

Table 3: RMFGEN and MULTIGRID instances

References

[1] D. Bienstock, Experiments with a network design algorithm using ϵ-approximate linear programs (1996), submitted.

[2] D. Bienstock, G. Muratore, Strong inequalities for capacitated survivable network design problems (1997) to appear, *Math. Programming.*

[3] D. Bienstock, S. Chopra, O. Günlük and C. Tsai, Minimum Cost Capacity Installation for Multicommodity Network Flows, *Math. Programming* **81** (1998), 177-199.

[4] F. Barahona and R. Anbil, The Volume Algorithm: producing primal solutions with a subgradient method (1997), to appear, *Mathematical Programming.*

[5] A. Goldberg, J. Oldham, S. Plotkin and C. Stein, An Implementation of a Combinatorial Approximation Algorithm for Minimum Multicommodity Flow, IPCO 1998.

[6] M.D. Grigoriadis and L.G. Khachiyan (1991), Fast approximation schemes for convex programs with many blocks and couping constraints, *SIAM Journal on Optimization* **4** (1994) 86-107.

[7] M.D. Grigoriadis and L.G. Khachiyan, An exponential-function reduction method for block-angular convex programs, *Networks* **26** (1995) 59-68.

[8] M.D. Grigoriadis and L.G. Khachiyan, Approximate minimum-cost multicommodity flows in $\tilde{O}(\epsilon^{-2}KNM)$ time, *Mathematical Programming* **75** (1996), 477-482.

[9] T. Magnanti and Y. Wang, Polyhedral Properties of the Network Restoration Problem-with the Convex Hull of a Special Case (1997), to appear, *Math. Programming.*

[10] S. Plotkin and D. Karger, Adding multiple cost constraints to combinatorial optimization problems, with applications to multicommodity flows, In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing,* (1995), 18-25.

[11] S. Plotkin, D.B. Shmoys and E. Tardos, Fast approximation algorithms for fractional packing and covering problems, *Proc. 32nd Annual IEEE Symp. On Foundations of Computer Science,* (1991), 495-504.