J. Indian Inst. Sci., March–April 1998, **78**, 131–151.
© Indian Institute of Science

The search for optimal Lagrange multipliers

VIJAY CHANDRU* AND MICHAEL A. TRICK†
*Department of Computer Science and Automation, Indian Institute of Science, Bangalore 560 012, India
email: chandru@csa.iisc.ernet.in
†Graduate School of Industrial Administration, Carnegie–Mellon University, Pittsburgh, PA 15213, USA
email: trick@gsia.cmu.edu

Received on April 15, 1998.

Abstract

Lagrangean constraint relaxation methods are a widely used approximation technique for NP-hard discrete optimization problems. The computational complexity of the Lagrangean relaxation technique is examined in this paper. It is shown that the optimization of Langrangean multipliers is polynomial-time equivalent to optimization of the Lagrangean subproblems. The main technique used to derive this result is the ellipsoid method for optimizing linear functions over convex bodies.

Consequences, both theoretical and computational, of this result are developed. A direct polynomial-time reduction of `integer knapsack` to `integer group knapsack` is shown to follow via an elementary treatment of (a Lagrangean dual-based) super group embedding of the knapsack problem. The computational message of the paper is that multiplier search using the ellipsoid method is an attractive alternative to first-order subgradient methods. With this perspective, some preliminary computational experiments on 1-tree Lagrangean relaxation of the travelling salesman problem were conducted and the results are reported.

Keywords: Lagrangean relaxation, computational complexity, integer programming, subgradient method, ellipsoid method.

1. Introduction

Lagrangean constraint relaxation methods are multiplier methods for mathematical programming problems that are dual-based, approximation methods. For convex programming problems, these are exact dual methods. In the presence of non-convexities, the approximation achieved by these methods is equivalent to a simple convexification[1, 2]. In the context of integer programming, Lagrangean relaxation has been used in a variety of applications with generally favorable results. Two features of these methods have contributed to their success. The first is that judicious choice of the constraints to be relaxed leads to useful exploitation of special substructures that exist in most applications. The second feature is that these methods often provide excellent bounds on the optimal objective value. These are critical in restricting the combinatorial explosion inherent in partial enumeration methods, such as branch and bound, for these discrete optimization problems.

Lagrangean decomposition[3, 4] is a related idea which involves making copies of variables and splitting the integer program into two or more subproblems. Lagrangean decomposition is used when there are two or more exploitable substructures in the integer program. In principle, Lagrangean decomposition is a special case of Lagrangean relaxation applied to a reformulation of the integer program.

A fundamental problem in both Lagrangean relaxation and decomposition is the search for optimal Lagrangean multipliers. This involves the maximization of an implicitly defined piecewise linear and concave function. In current practice, subgradient optimization techniques are normally used to solve this non-differentiable optimization problem. However, these techniques are not pure ascent methods and are not even finite algorithms in the strict sense. They are heuristically terminated and uncertainty as to whether the optimal multipliers have been found is unavoidable.

The ellipsoid method was proposed by Shor[5] of the erstwhile Soviet Union as a space dilation method for non-differentiable convex programming. Haĉijan[6] (pronounced Khachiyan) later observed that with minor technical modifications this algorithm solves linear programming problems in polynomial time. A large body of work on the ellipsoid method exists. The survey paper by Bland *et al.*[7] gives an early account of these developments.

In this paper, we propose the use of the ellipsoid algorithm to compute optimal Lagrange multipliers. Our justification of this proposal stems from both theoretical and computational perspectives. It may be noted that the use of the ellipsoid algorithm to solve the multiplier search problem was advocated by Chandru[8] in 1982. It has also been briefly discussed in the compendium of Schrijver[9].

In the following section, we review some background results in Lagrangean relaxation and decomposition as well as the optimization/separation equivalence implied by the ellipsoid algorithm. In Section 3, we show that the ellipsoid algorithm provides a polynomial-time (Turing) reduction of the multiplier search problem to the Lagrangean subproblem. As a theoretical consequence of this theorem, we show in Section 4 that a general integer linear program can be reduced to a group knapsack problem. We present some preliminary computational evidence, in Section 5, that the ellipsoid method is practical and robust for obtaining good Lagrange multipliers quickly. The computational results are for the 1-tree relaxation of randomly generated travelling salesman problems (TSP).

2. Preliminaries

In what follows, we briefly review the results in Lagrangean relaxation that are of relevance to later sections. We also present the main result in connection with the ellipsoid algorithm that we shall need. Several survey papers[10–13] have been written on Lagrangean relaxation. Throughout this paper we shall assume some familiarity with computational complexity theory at the level found in the treatise by Garey and Johnson[14].

The problems of interest to use are discrete optimization problems of the form $\max\{cx : x \in S\}$ where S is a discrete set belonging to the collection of d-vectors of 0s and 1s or some positive integer values. The complexity of the procedures we shall discuss will be relative to the input length L of this problem. For example, when we discuss the TSP on a complete graph, the input length is with respect to a binary encoding of the distance matrix and *not* with respect to an integer programming formulation of the TSP.

2.1. *Lagrangean relaxation and decomposition*

Consider a representation of a discrete optimization problem in the form:

$$\text{(P) } z = \max\{cx : Ax \geq b, x \in X \subseteq \mathbb{R}^n\}.$$

We assume throughout that

(A1) The explicit constraints ($Ax \leq b$) are *small* in number and dimension. More precisely, the dimensions of the matrix A are bounded by some polynomials in L.

(A1) The implicit constraints, embodied in X, have a *finite* description. By this we mean that X can be replaced by a finite list $\{x^1, x^2, \ldots x^T\}$.

Note that assumption (A2) is much milder than it may appear at first reading. For example, the results of Gathen and Sieveking[15] imply that any integer linear program or mixed-integer linear program can meet this assumption.

Definitions 2.1. The following definitions are with respect to (P)

Lagrangean: $L(u, x) = u(b - Ax) + cx$

Lagrangean subproblem: $\max_{x \in X}\{\tilde{c}x = (c - uA)x\}$

Lagrangean dual function: $\mathcal{L}(u) = \max_{x \in X} L(u,x)$

Lagrangean dual problem: $d = \min_{u \geq 0} \mathcal{L}(u)$ (D)

It is easily shown that (D) satisfies a weak duality relationship with respect to (P), i.e. $z \leq d$. The assumption (A2) also implies that $\mathcal{L}(u)$ is a piece-wise linear and convex function. In fact, both these properties may be observed from the equivalence of (D) to the large-scale linear program:

$$\begin{aligned} d = \min_{\eta,u}\ & \eta \text{(DLP)} \\ \text{s.t.}\quad & \eta - u(b - Ax^i) \geq cx^i \; i = 1, 2, \ldots, T \\ & u \geq 0. \end{aligned}$$

The usual linear programming dual of (DLP) is given by:

$$\begin{aligned} z_{LP} = \max & \sum_{i=1}^{T} (cx^i)\lambda_i \text{(PLP)} \\ \text{s.t.} & \sum_{i=1}^{T} (Ax^i - b)\lambda_i \leq 0 \\ & \sum_{i=1}^{T} \lambda_i = 1 \\ & \lambda_i \geq 0 \text{ for } i = 1, 2, \ldots, T. \end{aligned}$$

Many interesting characteristics of Lagrangean relaxation such as convexification, integrality property, etc. may easily be derived from these formulations (see Shapiro[13]). In practice, the constraints X are chosen such that the evaluation of the Lagrangean dual function $\mathcal{L}(u)$ is easily made (i.e. the Lagrangean subproblem, $\max_{x \in X}\{\tilde{c}x\}$ is easily solved).

Lagrangean decomposition is used when the underlying integer program has two or more exploitable substructures. To illustrate the idea, let us consider the integer program:

$$\begin{aligned} \max\quad & cx \\ \text{s.t. } & D_1x \le d_1 \\ & D_2x \le d_2 \\ & x \ge 0, \text{ integer} \end{aligned} \tag{P$_2$}$$

The first step is to formulate (P$_2$) as follows:

$$\begin{aligned} z_2 = \max\ & cx^1 \\ \text{s.t. } & D_1x^1 \le d_1 \\ & D_2x^2 \le d_2 \\ & x^1 - x^2 = 0 \\ & x^1, x^2 \ge 0, \text{ integer} \end{aligned} \tag{P$_2'$}$$

The linking constraints ($x^1 - x^2 = 0$) are now treated as the explicit constraints yielding the dual problem:

$$d_2 = \min_{\mu} \max_{x^1,x^2} \left\{ cx^1 + \mu\left(x^1 - x^2\right) \middle| \begin{array}{l} D_1x^1 \le d_1, \quad D_2x^2 \le d_2 \\ x^1 \ge 0 \text{ integer}, x^2 \ge 0 \text{ integer} \end{array} \right\} \tag{D$_2'$}$$

Note that (D$_2'$) is the usual Lagrangean dual problem associated with (P$_2'$). The dual variables μ are unrestricted in sign in this case since the explicit constraints ($x^1 - x^2 = 0$) are equality constraints. The problem again is to find the optimal Lagrangean multipliers. Because Lagrangean decomposition can be seen as Lagrangean relaxation of a suitably reformulated problem, any method to find the multipliers for Lagrangean relaxation can also be used for Lagrangean decomposition.

2.2. *Ellipsoid algorithm*

The most commonly used general method of finding the optimal multipliers in Lagrangean relaxation is subgradient optimization. Subgradient optimization is the non-differentiable counterpart of steepest descent methods. Given a dual solution u^k, the iterative rule for creating a sequence of solutions is given by:

$$u^{k+1} = u^k + t_k\gamma(u^k)$$

where $x(u^k)$ is a maximizer of $\max_{x \in X} L(u^k, x)$ t_k is an appropriately chosen step size.

Subgradient optimization has proven effective in practice for a variety of problems. It is possible to choose the step sizes $\{t_k\}$ to guarantee convergence to the optimal solution. Unfortunately, the method is not finite, in that the optimal solution is attained only in the limit. Further, it is not a pure descent method and bounds on the suboptimality of the current iterate are not generally available. In practice, the method is heuristically terminated and the best solution in the generated sequence is recorded. In the context of non-differentiable optimization, the ellipsoid algorithm was devised by Shor[5] to overcome precisely some of these difficulties with the subgradient method. The ellipsoid algorithm may be viewed as a scaled subgradient method in much the same way as variable metric methods may be viewed as scaled steepest descent methods (cf. Goffin[16]).

The ellipsoid algorithm of Shor[5] gained prominence in the late 1970s when Hačijan[6] showed that this convex programming method specializes to a polynomial-time algorithm for linear programming problems. This theoretical breakthrough naturally led to intense study of this method and its properties. The survey paper by Bland *et al.*[7] and the monograph by Akgül[17] attest to this fact. Direct theoretical consequences for combinatorial optimization problems was independently documented by Padberg and Rao[18], Karp and Papadimitriou[19] and Grötschel *et al.*[20] For an elegant treatment of the many deep theoretical consequences of the ellipsoid algorithm, the reader is directed to the monograph of Lovász[21] and the book by Grötschel *et al.*[20]

Computational experience with the ellipsoid algorithm, however, showed a disappointing gap between the theoretical promise and practical efficiency of this method in the solution of linear programming problems. Dense matrix computations as well as slow average-case convergence properties are the reasons most often cited for the behaviour of the ellipsoid algorithm. On the positive side, though, it has been noted (cf. Ecker and Kupferschmid[22]) that the ellipsoid method is competitive with the best-known algorithms for (nonlinear) convex programming problems.

We first give a brief description of a very simple form of the algorithm so as to be able to describe later how we will use it in solving the Lagrangean dual problem.

Let us consider the problem of testing if a polyhedron $Q \in \mathbb{R}^d$, defined by linear inequalities, is non-empty. For technical reasons let us assume that Q is rational, i.e. all extreme points and rays of Q are rational vectors or equivalently that all inequalities in some description of Q involve only rational coefficients. The ellipsoid method (in contrast with the simplex method and Karmarkar's algorithm[23]) does not require the linear inequalities describing Q to be explicitly specified. It suffices to have an oracle representation of Q. Several different types of oracles can be used in conjunction with the ellipsoid method[18–20, 24]. We will use the *strong separation oracle* described below.

Oracle: Strong separation (Q, y)

```
Given a vector $y \in \mathbb{R}^d$, decide whether $y \in Q$, and if not find a
hyperplane that separates $y$ from $Q$; more precisely, find a
vector $c \in \mathbb{R}^d$ such that $c^T y < \min\{c^T x \mid x \in Q\}$.
```

The ellipsoid algorithm initially chooses an ellipsoid large enough to contain a part of the polyhedron Q if it is non-empty. This is easily accomplished because we know that if Q is non-empty then it has a rational solution with values bounded by a function of the largest coefficient in the linear program and the dimension of the space.

The centre of the ellipsoid is a feasible point if the separation oracle tells us so. In this case, the algorithm terminates with the coordinates of the centre as a solution. Otherwise, the separation oracle outputs an inequality that separates the centre point of the ellipsoid from the polyhedron Q. We translate the hyperplane defined by this inequality to the centre point. The hyperplane slices the ellipsoid into two halves, one of which can be discarded. The algorithm now creates a new ellipsoid that is the minimum volume ellipsoid containing the remaining half of the old one. The algorithm questions if the new centre is feasible and so on. The key is that the new ellipsoid has substantially smaller volume than the previous one. When the volume of the current ellipsoid shrinks to a sufficiently small value, we are able to conclude that Q is empty. This fact is used to show the polynomial time convergence of the algorithm. The details are as follows.

Ellipsoids in $\mathbb{R}^d$ are denoted as $E(A, y)$ where A is a $d \times d$ positive definite matrix and $y \in \mathbb{R}^d$ is the centre of the ellipsoid $E(A, y)$.

$$E(A, y) = \{x \in \mathbb{R}^d | (x - y)^T A^{-1}(x - y) \leq 1\}$$

The ellipsoid algorithm is described on the iterated values, A_k and x^k which specify the underlying ellipsoids $E_k(A_k, x^k)$.

```
Procedure: Ellipsoid ($Q$)
    0. Initialize:
    • $N: = N(Q)$ (comment: iteration bound)
    • $R: = R(Q)$ (comment: radius of the initial ellipsoid/sphere $E_0$)
    • $A_0: = R^2 I$
    • $x_0: = 0$ (comment: centre of $E_0$)
    • $k: = 0$
    1. Iterative step:
        while $k < N$
        call strong separation ($Q$, $x^k$)
        if $x^k \in Q$ halt
        else hyperplane $\{x \in \mathbb{R}^d \mid c^T x = c_0\}$ separates $x^k$ from $Q$
        Update
```

$$b := \frac{1}{\sqrt{c^T A_k c}} A_k c$$

$$x^{k+1} := x^k - \frac{1}{d+1} b$$

$$A_{k+1} := \frac{d^2}{d^2 - 1}\left(A_k - \tfrac{2}{d+1} b b^T\right)$$

$$k := k + 1$$

```
        endwhile
    2. Empty polyhedron:
```

- halt and declare "Q is empty"

3. End

The crux of the complexity analysis of the algorithm is on the a priori determination of the iteration-bound $N(Q)$. This in turn depends on three factors. The volume of the initial ellipsoid E_0 (determined by $R(Q)$), the rate of volume shrinkage $\left(\frac{vol(E_{k+1})}{vol(E_k)} < e^{-\frac{1}{(2d)}}\right)$ and the volume threshold at which we can safely conclude that Q must be empty. The assumption of Q being a rational polyhedron is used to argue that Q can be modified into a full-dimensional polytope without affecting the decision question ("Is Q non-empty?"). After careful accounting for all these technical details and a few others (for example, compensating for the round-off errors caused by the square root computation in the algorithm) it is possible to establish the following fundamental result.

Theorem 2.2[18, 19, 24]. *There exists a polynomial $g(d, \phi)$ such that procedure* `Ellipsoid` (Q) *runs in time bounded by T, $g(d, \phi)$ where ϕ is an upper bound on the size of linear inequalities in some description of Q and T is the maximum time required by the oracle* `strong separation`(Q, y) *on inputs y of size at most $g(d, \phi)$.*

Consider a *linear optimization* problem of the form:

$$\text{OPT} \min\{dy: y \in Q\}$$

when Q is as described above. Optimizing the linear function dy over the convex body Q can be done by the 'sliding objective function' variant of the ellipsoid algorithm as described in Bland *et al.*[7] If a feasible point y' is found, the method adds an inequality $(dy \leq dy' - \delta)$ for some positive δ. Since y' violates this constraint, it solves the separation problem for this iteration. As a consequence, we get the following corollaries to Theorem 2.2.

Corollary 2.3: *The optimization problem (OPT) is solvable in polynomial-time if and only if Q has a polynomial-time generator of separating inequalities.*

Corollary 2.4: *OPT is NP-hard if and only if the separation problem is also NP-hard.*

In combinatorial optimization, these results are often used to indicate polynomial solvability of various problems and to provide an approach for NP-hardness classification. We shall give an illustration of this idea in Section 4.

3. Ellipsoid method solves the Lagrangean dual

This is obtained by applying Theorem 2.2 and its corollaries to the optimization problem (DLP) for choosing optimal Lagrange multipliers.

Lemma 3.1. *The generator of separation inequalities in DLP and the Lagrangean subproblem $\max_{x\, m\, X} L(u, x)$ are polynomial-time equivalent.*

Proof: Given a solution $(\overline{\eta}, \overline{u})$ the separation problem is to determine whether $\overline{\eta}$ is an upper bound on the dual functional value $\mathcal{L}(\overline{u}) = \max_{x\text{in}X} L(\overline{u}, x)$. If not, an obvious separating inequality is given by $(\overline{\eta} - \mathcal{L}(\overline{u}) \geq 0)$. Thus, the separation problem is easily solved once we have a method for solving the Lagrangean subproblem. Conversely, if we are given a separation oracle and a fixed $\tilde{u}$, it is possible to obtain $\mathcal{L}(\tilde{u})$ by performing a binary search on the range of values that η can take to obtain the least upperbound on $\mathcal{L}(\tilde{u})$. $\square$

Thus, we have

Theorem 3.2: *The Lagrangean dual problem is polynomial-time solvable if and only if the Lagrangean subproblem is. Consequently, the Lagrangean dual problem is NP-hard if and only if the Lagrangean subproblem is.*

The theorem suggests that in practice if we set up the Lagrangean relaxation so that the subproblem is tractable, then the search for optimal Lagrangean multipliers is also tractable. Of course, this search problem may be solved by the ellipsoid method and our computational results reported in Section 6 indicate the viability of this approach. A less direct implication is that we may be able to reformulate the Lagrangean relaxation-bound calculation as an optimization problem that can be solved by non-ellipsoid methods in polynomial-time. Results of Martin[25] provide some evidence that variable redefinition methods can be used to solve the Lagrangean dual problem as a compact (polynomial-size) linear program in many cases.

In practice, however, the Lagrangean relaxation is often set up so that the resulting subproblem is NP-hard but not 'pathologically' hard. For example, the relaxation of general assignment problems by Fisher[10] yields a subproblem that is a collection of (0-1) knapsack problems. Knapsack problems are NP-hard but do admit pseudo-polynomial time algorithms that are extremely efficient if the coefficients are small. Theorem 3.2 may be specialized for such situations as follows.

Proposition 3.3. *If the Lagrangean subproblem admits a pseudo-polynomial algorithm then so does the Lagrangean dual problem.*

Another issue of importance from a practical perspective is that of bounds on the Lagrangean function. Since the Lagrangean relaxation method is usually embedded within a branch and bound framework, early termination of the ellipsoid algorithm can be useful in curtailing the amount of computational effort expended at each node of the branch and bound tree. Early termination may be employed if we had a technique for computing good lower bounds on the optimal Lagrangean dual value d. The next proposition shows that such a technique does indeed exist.

Proposition 3.4. *Let E be the ellipsoid at any iteration of the ellipsoid algorithm (applied to (D)) with centre (u^k, η^k). Let $x(u^k)$ be an optimal solution to the Lagrangean subproblem at u^k. If $E \cap \{u : u \geq 0\} \neq \phi$, then,*

$$d \geq \min_{u \in E, u \geq 0} u(b - Ax(u^k)).$$

Proof: Since E contains a feasible u, E also contains the optimal value for u. (D) is equivalent to

$$d = \min_{u \in E, u \geq 0} \min_{x \in X} L(u,x).$$

Restricting X to the single value $\{x(u^k)\}$ yields a valid lower bound on d. $\square$

It is not difficult to optimize a linear function over the intersection of the ellipsoid and non-negative orthant. All of the calculations needed are required in the ellipsoid algorithm to update the ellipsoid. Thus, the lower bound is essentially a byproduct.

Finally, we also note that analogous results can be obtained for the Lagrangean dual problem (D_2') that results when Lagrangean decomposition approach is taken. The interesting result from a computational perspective is that if all the subproblems admit polynomial time algorithms then so does the Lagrangean dual problem (D_2').

4. A Turing reduction

The theory of NP-completeness implies that any two NP-complete decision problems are polynomially equivalent. The existence of polynomial reductions is the motivation to look for more natural or direct equivalences between such problems. In this section, we will provide such an equivalence theorem relating integer knapsack problems (KP) to integer group knapsack problems. The construction demonstrating the equivalence of these two NP-complete problems will involve Lagrangean relaxation and the polynomiality will follow from results of the previous section.

An integer knapsack problem is an optimization problem of the form:

$$z = \max\left[c_0 y_0 + \sum_{j=1}^{n} c_j x_j\right]$$

$$\text{s.t. } a_0 y_0 + \sum_{j=1}^{n} a_j x_j = b \qquad \text{(KP)}$$

$$y_0, x_j \geq 0 \text{ and integer for } j = 1, \ldots n$$

where $b \geq a_j > 0$ for all $j = 0, 1, 2, \ldots, n$. Assume the ordering

$$(c_0/a_0) \geq (c_1/a_1) \geq \ldots \geq (c_n/a_n).$$

To obtain a relaxation of KP we first substitute for y_0 the linear form $(x_0 + s^\Theta)$ where s is taken to be a positive integer and both x_0 and Θ are non-negative, integer-valued variables. Relaxation of the non-negativity restriction on Θ yields the formulation:

$$\max \sum_{j=1}^{n} c_j x_j + (c_0 / a_0)[b - \Sigma_{j=0}^{n} a_j x_j] \qquad \text{(GP)}$$

$$\text{s.t.} \sum_{j=1}^{n} a_j x_j - b \equiv 0 (\text{modulo } s a_0)$$

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

GP is the usual group knapsack relaxation of KP. The modulo constraint enforces integral-ity of the Θ variables which has been substituted by $[b - \sum_{j=0}^{n} a_j x_j](1/a_0 s)$ in GP. A Lagrangean elaxation may be realized by associating a scalar non-negative multiplier u with the relaxed onstraint ($sa_0 \geq 0$). This yields the form:

$$L(u,x) = (c_0 + u)(b / a_0) + \sum_{j=0}^{n} (c_j - (c_0 + u)(a_j / a_0))x_j$$

vith the Lagrangean dual problem defined as

$$d = \min_{u \geq 0} \max_{x \in X} L(u,x)$$

$$X = \{(x_0, x_{1,..,}x_n): \sum_{j=0}^{n} a_j x_j - b \equiv 0(\text{mod}\, sa_0) x_j \geq 0 \text{ and integer}\}$$

As usual, we have the weak duality relationship that z is no larger than d. It is somewhat sur-prising therefore that the duality gap $(d - z)$ can be set to zero by choosing a large enough value or s, the modulus parameter. This is the essence of the 'convergent duality' results of Bell[26] ind Bell and Shapiro[27]. We shall now prove such a theorem and furthermore show that s does 10t have to be too large (i.e. s is polynomial-space bounded) for the convergent duality relation o hold. First we need some technical lemmas.

Lemma 4.1. For any non-negative value of the multiplier u we have $c_j - (x_0 + u)(a_j/a_0) \leq 0$ for each $j = 0, 1, \ldots, n$.

Proof:

$$\begin{aligned} c_j - (c_0 + u)\,(a_j/a_0) &= c_j - c_0(a_j/a_0) - u(a_j/a_0) \\ &\leq -u(a_j/a_0) \\ &\leq 0 \end{aligned}$$

where the first inequality follows form the non-decreasing order $(c_0/a_0) \geq (c_1/a_1) \ldots \geq (c_n/a_n)$. ■

Lemma 4.2. *We may restrict the feasible set X to contain only irreducible elements, i.e.* $X = \{x^1, x_2, \ldots, x_t\}$ *where* x^t *satisfies*

$$\sum_{j=0}^{n} a_j x_j^t \equiv b(\text{mod}\, sa_0), x^t \geq 0 \text{ integer}$$

ind $x_j^t \leq sa_0$ for each $j = 0, 1, \ldots, n$. *Further, this restriction of X to irreducible elements does 10t affect the quality of the upper bound d on z.*

Proof: From the previous lemma it follows that all coefficients of the Lagrangean subproblems $\max_{x \in X} L(\mu, x)$ are non-positive and therefore an irreducible solution is always optimal. ■

Henceforth, we assume that X has been restricted to its irreducible set. As noted in Section 2, the dual problem can be represented as a large-scale linear program whose linear programming dual is given by:

$$z_{LP} = \max \sum_{t=1}^{T} \left(c_0 b + \sum_{j=0}^{n} \left(c_j a_0 - c_0 a_j \right) x_j^t \right) \lambda_t \qquad \text{(PLP)}$$

$$\text{s.t.} \sum_{t=1}^{T} (b - \sum_{j=0}^{n} a_j x_j^t) \lambda_j \geq 0$$

$$\sum_{t=1}^{T} \lambda_t = 1$$

$$\lambda_t \geq 0 \text{ for all } t = 1,2,\ldots, T.$$

It is useful to note here that if an optimal solution to PLP is integer valued (i.e. $\lambda_t{}^* = 1$ for some t^*, then we will have an optimal solution x^{t^*}) for the KP. Now, we go on to construct a *threshold value* S for the modulus parameter s, beyond which the linear program (PLP) is forced to have an integer-valued optimal solution.

Lemma 4.3. $(b + 1)$ is a strict upper bound on the sum of non-negative numbers $\{x_j\} j = 1,2,\ldots, n$ which satisfy

$$\sum_{j=0}^{n} a_j x_j \leq b.$$

Proof: Recall that $b \geq a_j \geq 1$ for all $j = 1,2,\ldots, n$. Therefore, maximum value attainable by the sum of x_j variable satisfying the said sign and inequality constraints is the maximum of the ratios (b/a_j) for $j = 1,2,\ldots, n$. This number is always strictly less than $(b + 1)$. □

Threshold modulus:

Define

$$S' = \max \left| \sum_{j=1}^{n} a_j x_j - b \right|$$

$$\text{s.t.} \sum_{j=1}^{n} x_j \leq b + 1$$

$$x_j \geq 0 \text{ for } j = 1,\ldots n.$$

The threshold value is then taken to be

$$S = \lceil (S'/a_0) \rceil + 1$$

which is no larger than $O(b^2)$ by inspection.

Consider the partition (X_1, X_2) of the irreducible set X.

$$X_1 = \left\{ x^t \in X: \sum_{j=1}^{n} x_j < b+1 \right\}$$

$$X_2 = X \setminus X_1.$$

Let (T_1, T_2) be the associated partition of the index set $\{1,2,\ldots, T\}$.

Lemma 4.4. *Beyond the threshold modulus (i.e. $s \geq S$) and for all t in T_1 we have the inequality, $\Sigma_{j=0}^{n} a_j x_j^t \geq b$.*

Proof: Since x^t is in X for all $t \in T_1$, we have

$$a_0 x_0^t + \sum_{j=1}^{n} a_j x_j^t - b \equiv 0(\operatorname{mod} s a_0).$$

By the definition of T_1 and the threshold value S we also know that

$$\left| \sum_{j=1}^{n} a_j x_j^t - b \right| < s a_0 \text{ for } t \in T_1 \text{ and } s \geq S.$$

The lemma now follows since both a_0 and x_0^t are non-negative in value. □

Lemma 4.5. *Beyond the threshold modulus ($s \geq S$) and for any feasible solution $\overline{\lambda}$ in PLP we must have $\overline{\lambda}_t = 0$ for all $t \in T_2$.*

Proof: Suppose $\overline{\lambda}$ is a feasible solution in PLP, then

$$\sum_{t\in T_1} \left(b - \sum_{j=0}^{n} a_j x_j^t \right) \overline{\lambda}_t \sum_{t\in T_2} \left(b - \sum_{j=1}^{n} a_j x_j^t \right) \overline{\lambda}_t \geq 0.$$

The first term is non-positive from the observations that $\overline{\lambda}$ is non-negative and the previous lemma implies that each of the parenthetic expressions is non-positive. Therefore,

$$\sum_{t\in T_2} \left(b - \sum_{j=0}^{n} a_j x_j^t \right) \overline{\lambda}_t \geq 0.$$

Excluding the non-negative terms, $a_0 x_0^t$ yields

$$\sum_{t\in T_2}\left(b-\sum_{j=1}^{n} a_j x_j^t\right)\overline{\lambda}_t \geq 0.$$

Now suppose the lemma is false, i.e. $\sum_{t\in T_2} \overline{\lambda}_t = \gamma > 0$. Consider

$$\overline{x} = (1/\gamma)\sum_{t\in T_2} \lambda_t x_j^t.$$

Clearly, $\overline{x} \geq 0$ and further

$$\left(b-\sum_{j=1}^{n} a_j \overline{x}_j\right) = 1/\gamma \sum_{t\in T_2}\left(\left(b-\sum_{j=1}^{n} a_j x_j^t\right)\overline{\lambda}_t\right) \geq 0.$$

Lemma 4.3. implies that

$$\sum_{j=1}^{n} \overline{x}_j < (b+1).$$

But, from the definition of T_2 (and X_2), we have

$$\sum_{j=1}^{n} x_j^t \geq (b+1)$$

for each t in T_2. Thus a contradiction has been reached. □

The lemmas can now be combined to give the main result.

Theorem 4.6. *KP is polynomial-time reducible to the group knapsack.*

Proof: Given an instance of KP we set up the Lagrangean relaxation as described above to obtain a Lagrangean dual problem whose associated subproblem is the group knapsack problem (GP). Assume further that the modulus parameter s is taken to be above the threshold value. Note that the threshold value S is $O(b^2)$ and is hence polynomial-space bounded. This ensures that the set X is succinct and finite as needed for assumption (A2) (Section 2) to hold.

Theorem (3.2) now implies that the Lagrangean dual for this set up is polynomial time equivalent to the group knapsack problem. To complete the proof we need to now argue that Lagrangean dual problem solves the integer KP.

Suppose λ^* is an optimal solution for PLP. From lemma (4.6), we have:

$$\sum_{t\in T_1}\left(b-\sum_{j=1}^{n} a_j x_j^t\right)\lambda_t^* \geq 0$$

$$\sum_{t\in T_1} \lambda_t^* = 1$$

$$\lambda_t^* \geq 0 \text{ for all } t \in T_1.$$

From lemma (4.5) we know that

$$\left(b-\sum_{j=1}^{n} a_j x_j^{t^*}\right) \geq 0 \text{ for each } t \in T_1.$$

Therefore, for each $t^* \in T_1$ for which $\lambda_t^* > 0$ we must have

$$\sum_{j=0}^{n} a_j x_j^{t^*} = b$$

and hence $x_j^{t^*}$ is optimal in KP with $(y_0 = x_0^{t^*})$. Note also that PLP will be infeasible if and only if KP is infeasible. Thus, PLP is equivalent to KP. But the Lagrangean dual problem is exactly the linear programming dual of PLP. Hence the theorem. □

This theorem may be viewed as an extension of results in polynomial time aggregation of integer programming problems. The most general of these results was proved by Kannan[28].

Theorem 4.7. *The general integer programming problem*

$$\max\ \{cx : Ax = b, x \geq \text{ and integer}\}$$

is polynomial-time reducible to a knapsack problem.

This result does not assume explicit bounds on the variables. The two theorems can therefore be combined to yield.

Theorem 4.8. *The general integer programming problem is polynomial-time reducible to a group knapsack problem.*

As was stated in the introductory paragraph of this section, all these theorems are also consequences of the fact that the three problems—integer programming, knapsack and group knapsack—are NP-hard with NP-complete decision analogues. The new result is that *direct* reductions are possible. The reduction of knapsack to group knapsack—as described in this section—is a *Turing* reduction in that an instance of the knapsack problem is to be solved. It would be interesting to find an 'instance-to-instance' reduction (Karp or many-to-one reduction[14]) for these problems. Finally, we note that the Lagrangean constructs of this section can be used to directly prove Theorem 4.8. This was shown in Chandru[8] and involves, in addition to the ellipsoid algorithm, some preprocessing using the polynomial-time Smith normal form algorithm of Kannan and Bachem[30].

5. Optimizing over valid inequalities

We use the results of the previous section to provide an alternative method for optimizing over classes of valid inequalities of integer polyhedra. This method has the advantage using optimization algorithms rather than separation algorithms. For many classes of valid inequalities, the corresponding optimization algorithm is more efficient than the separation algorithm.

Consider a discrete optimization problem:

$$\max\{cx : x \in S\} \tag{Q}$$

where S is some (very large) set of points. An inequality $ax \leq b$ is valid if $ax' \leq b$ for all $x' \in S$. For many important problems, certain classes of inequalities have been shown to be valid inequalities.

Given classes of valid inequalities $A_1x \leq b_1, A_2x \leq b_2, \ldots A_kX \leq b_k$, S can be relaxed to:

$$\max\{cx : A_1x \leq b_1, A_2x \leq b_2, \ldots A_kX \leq b_k\} \tag{F}$$

A solution to (F) gives an upper bound on the value of Q.

In many problems, the classes of valid inequalities found can contain an exponential number of constraints. If the separation problem is polynomially solvable for each class, then F is polynomially solvable by the ellipsoid algorithm. Furthermore, if the separation problem is solvable for a class of valid inequalities, then the optimization problem over that class alone is solvable. For instance, if for any x' it is possible to find a constraint of $A_1x \leq b_1$ that is violated by a x', or determine that none exist, then

$$\max\{cx : A_1x \leq b_1\}$$

is also solvable in polynomial time by the ellipsoid algorithm. In some cases, however, the optimization problem is much easier than the separation algorithm. An example of this is the subtour elimination constraints of the TSP. The number of such constraints is exponential in the number of cities to be visited but optimizing over the constraints is easy. The optimization problem is essentially a spanning tree problem (as shown in Held and Karp[31]). The separation problem is a series of minimum cut problems and is more time consuming (though still polynomial) than just finding an optimal spanning tree.

We can exploit these more efficient optimization algorithms by combining Lagrangean decomposition with the ellipsoid algorithm.

Theorem 5.1. *If the optimization problem for each of these classes of valid inequalities $A_1x \leq b_1, \ldots, A_kX \leq b_k$ is polynomially solvable, then the optimal value of F can be found in polynomial time.*

Proof: Consider the following, equivalent, formulation of A:

$$\max\{cx^1 : A_i x^i \le b_i, x^i = x^{i+1}\ \forall i. \qquad (F')$$

This can be solved by repeated applications of Lagrangean decomposition, so Theorem 3.2 provides a polynomial bound. □

We illustrate our results on a relaxation of the TSP. An extensive review of previous results of TSP can be found in Lawer *et al.*[31] Let $A_1x \le b_1$ be the 2-matching constraints which force exactly two edges to meet every node. Let $A_2x \le b_2$ be the subtour elimination constraints which force the tour to be a single loop. Both sets have an exponential number of constraints. The linear programming relaxation consisting of just A_1 and A_2 (along with upper and lower bounds on the variables) does not necessarily give an optimal solution to the TSP, for there may be fractional optimal solutions. Previously, applying the ellipsoid algorithm directly was the only polynomial algorithm for solving this relaxation.

(Primal ellipsoid): Solve F with the ellipsoid algorithm, solving the matching separation and subtour elimination separation problem at each iteration.

Our results give an alternative polynomial method.

(Dual ellipsoid): Solve F′ with the ellipsoid algorithm and Lagrangean decomposition solving a matching optimization problem and a subtour elimination optimization problem at each iteration.

The separation algorithms required by the primal ellipsoid algorithm are quite complex. The subtour elimination separation algorithm is a series of minimum cuts of a graph. The matching separation algorithm is even more complex and, generally, heuristics are used to identify violated constraints.

In contrast, the optimization algorithms required by the dual ellipsoid are quite simple. The subtour elimination optimization algorithm is very efficient, being simply the greedy algorithm. The matching optimization algorithm is another well-known algorithm (Edmonds[33]) with efficient implementations widely available.

It does not seem possible to determine a priori which of the alternatives is the most efficient for a given set of valid inequalities, as the number of iterations each algorithm requires is not known. In cases where the optimization algorithms are very effective, however, the dual ellipsoid method is a promising alternative.

6. Computational results

In this section, we give computational results for finding optimal multipliers for the one-tree relaxation of the TSP. The ellipsoid algorithm finds very good, but not optimal, solutions very quickly. This makes the ellipsoid algorithm very attractive in applications like branch and bound where good solutions are required quickly and the marginal advantage of optimal solutions is small. However, the ellipsoid algorithm is slower by a factor of two to three in finding the optimal multipliers on the problems examined, making the algorithm inferior in cases where the optimal solution is required. The ellipsoid algorithm also has the practical advantage of being almost parameterless. Subgradient optimization, in contrast, has numerous parameters that must be 'fine-tuned' for efficient operation.

First, we give a brief description of the one-tree relaxation for the TSP. A more detailed description is in Held and Karp[30,33].

For an undirected graph G, with costs on each edge, the TSP is to find a minimum cost set H of edges of G such that:

1. exactly two edges of H are adjacent to each node, and
2. H forms a connected, spanning subgraph of G.
   Condition 1 certainly implies:
3. exactly two edges of H are adjacent to node 1, and
4. H has n edges, where n is the number of nodes in G.

We now relax Condition 1. The problem of satisfying Conditions 2, 3 and 4 is the minimum cost one-tree problem, for any graph that satisfies these conditions is a spanning tree plus an extra edge.

We can easily find the minimum cost one-tree as follows:

Step 1. find the minimum spanning tree in G-{1}.

Step 2. find the two smallest cost edges incident to node 1.

The edges in the spanning tree in Step 1 together with the edges in Step 2 form the optimal one-tree.

If the constraints associated with Condition 1 are placed in the objective with Lagrangean multipliers, we get the one-tree Lagrangean relaxation for the TSP. We will use this relaxation to determine the effectiveness of the ellipsoid algorithm for finding Lagrangean multipliers.

We use straightforward implementation of the ellipsoid algorithm with just a single parameter: the radius of the initial ellipsoid.

In contrast, we had to do extensive testing to determine the best method of implementing subgradient optimization for the problems we were interested in. The details of the choice of step size are in the appendix. Four parameters are required:

(a) an overestimate of the optimal cost;
(b) an initial step multiplier;
(c) an iteration limit; and
(d) a step length divisor.

We chose values for b, c and d by experimentation. The overestimate of the optimal cost was based on the historical optimal costs generated by our problem generators.

We created two problem generators: first, the random generator assigns costs randomly drawn from a uniform distribution; second, the Euclidean generator places nodes randomly in a square with the cost on the edge between two nodes equalling the distance between the nodes. All testing was done on an IBM PC/XT* with a math co-processor.

*These experiments were carried out when the authors were attending a summer school and the only machine available to them was an IBM PC/XT. Clearly, with today's superfast machines, the CPU timings would shrink to a small factor of those reported here. However, the relative performance of the methods tested, i.e. the subgradient and ellipsoid methods, would remain unchanged.

Table I
Time to optimality

Problem type	*Average time to optimality (s)*		
	Ellipsoid	*Subgradient*	*Ratio*
Random–20 nodes	219	68	3.23
Random–50 nodes	2916	937	3.11
Euclidean–20 nodes	105	83	1.27
Euclidean–50 nodes	2677	898	2.98

Two sets of problems were created by each generator, one with 20-node problems and another with 50. Five problems were generated in each set, and the results were averaged over the five. Table I gives the time required to find the optimal solution, defined to be within one of the optimal solutions. The ellipsoid method was two to three times slower than the subgradient optimization.

The ellipsoid method seems to have great difficulty in the final stages of the algorithm. While it found very good solutions quickly, it required a large number of iterations to find the optimal solution. Subgradient optimization did poorer initially but once good solutions were created, it found optimal solutions quickly. This is shown in Figs 1 (20-node) and 2 (50-node). The vertical axis is best objective function found, expressed as a percentage of the optimal solution. The horizontal axis represents time. The results shown are for the random problems; the Euclidean results are similar.

In some applications, getting the optimal solution is not as important as getting very good solutions quickly. For instance, in branch and bound, the objective function is used to identify search directions that do not need to be explored further. In general, a small difference in the objective function will not significantly affect the decision on whether or not completely search the tree. In this case, it may be better to stop quickly with suboptimal solution rather than spend excessive time finding an optimal solution which will rarely affect the search decision. The

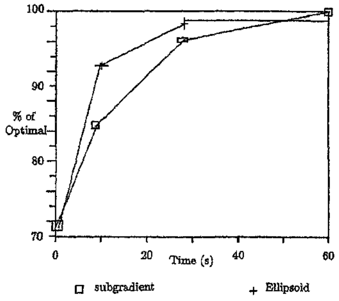

FIG. 1. 20-node results.

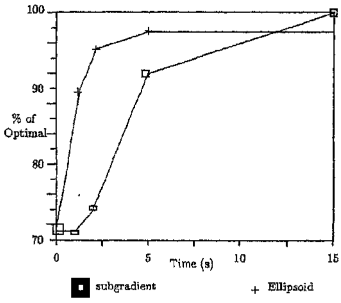

FIG. 2. 50-node results.

ellipsoid algorithm is a good approach in these cases. For example, in our random 50-node problems, terminating the ellipsoid algorithm after two minutes gives solutions that are 95% of the optimal value. Terminating subgradient optimization at that time gives only 75% of the optimal value. We would have to use subgradient optimization for more than six minutes to get a 95% solution.

In other applications, getting the optimal solution is more important. In those applications, subgradient optimization may be better.

One further advantage of the ellipsoid method is the lack of parameters. Computational times for subgradient optimization depend heavily on the choice of values for the parameters. For instance, choosing the number of iterations to be 25 rather than 10 almost doubled the average time to optimality for the 50-node problems in our tests. The results given here use finely tuned parameters. A poor choice of parameters will degrade the effectiveness of subgradient optimization. In contrast, the ellipsoid method has just one parameter: the radius of the initial ellipsoid. If this parameter is too small, then the optimal multipliers will occur on the boundary of the initial ellipsoid. This condition can easily be checked and re-optimization can take place. Providing this value is large enough, the algorithm is relatively insensitive to specific values.

An interesting possibility is to combine the best features of the two approaches. A hybrid algorithm that begins with ellipsoid algorithm to get a very good solution and then uses subgradient optimization to find an optimal solution might dominate each algorithm individually.

7. Conclusion

In this paper we have examined the theoretical (worst-case) complexity of the Lagrangean relaxation method. The main result shown is that the ellipsoid method is a theoretically efficient method for choosing optimal multiplier values. This result is shown to have both theoretical and practical implications. As topics for further investigation we present two recommendations.

The first is to identify Lagrangean constructions that involve polynomial time-solvable subproblems and to attempt reformulation of the Lagrangean dual problems with the objective of obtaining small-size linear programming descriptions. Some preliminary results along these lines have been obtained by Martin[25]. The second recommendation concerns computational testing along the lines of Section 6 for other examples of Lagrangean relaxation. The results of this paper were addressed to general-purpose methods for finding good Lagrangean multipliers. Additional testing on problems other than the TSP is needed to ascertain the true contribution of the framework presented in this paper to practical computation in discrete optimization.

References

1	MAGNANTI, T. L., SHAPIRO, J. F. AND WAGNER, M. H.	Generalized linear programming solves the dual, *Mgmt Sci.*, 1976, **22**, 1195–1203.
2.	SHAPIRO, J. F.	*Mathematical programming; structures and algorithms*, Wiley, 1979.

3. GLOVER, F. AND KLINGMAN, D. *Layering strategies for creating exploitable structure in linear and integer programs*, Technical Report CBDA 119, Graduate School of Business, University of Texas, 1985.

4. GUIGNARD–SPIELBERG, M. *Lagrangean decomposition: An improvement over Lagrangean and surrogate duals*, Technical Report #62, Wharton School, University of Pennsylvania, 1984.

5. SHOR, N. Z. Convergence rate of the gradient descent method with dilation of the space, *Cybernetics*, 1970, **6**, 102–108.

6. HAČIJAN, L. G. A polynomial algorithm in linear programming, *Sov. Math. Dokl.*, 1979, **20**, 191–194.

7. BLAND, R. G., GOLDFARB, D. AND TODD, M. J. The ellipsoid method: A survey, *Op. Res* , 1981, **29**, 1039–1091.

8. CHANDRU, V. *The complexity of the super-group approach to integer programming*, Ph. D. Dissertation, O.R.Center, MIT, 1982.

9. SCHRIJVER, A. *The theory of linear and integer programming*, Wiley, 1986.

10. FISHER, M. L. The Lagrangean relaxation method for solving integer programming problems, *Mgmt Sci.*, 1981 **27**, 1–18.

11. FISHER, M. L. An applications oriented guide to Lagrangean relaxation, *Interfaces*, 1985, **15**, 10–21.

12. GEOFFRION, A M. Lagrangean relaxation for integer programming, *Math. Prog. Study*, 1974, **2**, 82–114.

13. SHAPIRO, J. F. A survey of Lagrangean techniques for discrete optimization, *Ann. Discrete Math.*, 1979, **5**, 113–138.

14. GAREY, M. R. AND JOHNSON, D. S. *Computers and intractability: A guide to the theory of NP-completeness*, W.H. Freeman, 1979.

15. GATHEN, J. AND SIEVEKING, M. A bound on solutions of equalities and inequalities, *Proc. Am. Math. Soc.*, 1978, **72**, 155–158.

16. GOFFIN, J. L. Variable metric relaxation methods, Part II: The ellipsoid method, *Math. Programming*, 1984, **30**, 147–162.

17. AKGUL, M. *Topics in relaxation and ellipsoidal methods, Research Notes in Mathematics*, Pitman, 1984.

18. PADBERG, M. W. AND RAO, M. R. *The Russian method; I, II, III. Working Papers*, Graduate School of Business Administration, New York University, 1980.

19. KARP, R. M. AND PAPADIMITRIOU, C. H. On linear characterizations of combinatorial optimization problems, *Proc. Foundations of Computing Science*, IEEE, 1980.

20. GROTSCHEL, M., LOVÀSZ, L. AND SCHRIJVER, A. *Geometric algorithms and combinatorial optimization*, Springer-Verlag, 1988.

21. LOVÀSZ, L *An algorithmic theory of numbers, graphs and convexity*, SIAM Press, 1986.

22. ECKER, J. G. AND KUPFERSCHMID, M. An ellipsoid algorithm for nonlinear programming, *Math. Programming*, 1983, **27**, 83–106.

23. KARMARKAR, N. K. A new polynomial-time algorithm for linear programming, *Combinatorica*, 1984, **4**, 373–395.

24. GROTSCHEL, M., LOVÀSZ, L. AND SCHRIJVER, A. The ellipsoid method and its consequences in combinatorial optimization, *Combinatorica*, 1981, **1**, 169–197, (Corrigendum: 1984, **4**, 291–295).

25. MARTIN, R. K. Using separation algorithms to generate mixed integer model reformulations, *Op. Res. Lett.*, 1991, **10**, 119–128.

26. BELL, D. E. Constructive group relaxation for integer programs, *SIAM J. Appl. Math.*, 1976, **30**, 708–719.

27. BELL, D. E. AND SHAPIRO, J. F. A convergent duality theory for integer programming, *Op. Res.*, 1977, **25**, 419–434.

28. KANNAN, R. Polynomial-time aggregation of integer programming problems, *J.A.C.M.*, 1983, **30**, 133–145.

29. KANNAN, R. AND BACHEM, A. Polynomial algorithms for computing the Smith and Hermite normal forms of an integer matrix, *SIAM J. Computing*, 1979, **8**, 499–507.

30. HELD, M. AND KARP, R. M. The travelling-salesman problem and minimum spanning trees, *Op. Res.*, 1970, **18**, 1138–1162.

31. LAWLER, E. L., LENSTRA, J. K., RINNOOY KAN, A. H. G. AND SHMOYS, D. B. (EDS) *The travelling salesman problem*, Wiley, 1985.

32. EDMONDS, J. Paths, trees and flowers, *Can. J. Math.*, 1965, **17**, 449–467.

33. HELD, M. AND KARP, R. M. The travelling-salesman problem and minimum spanning trees: Part II, *Math. Programming*, 1970, **1**, 6–25.

Appendix 1: The step size

As mentioned in an earlier section, subgradient optimization requires a sequence of step sizes $\{t_j\}$ to determine the sequence of multipliers $\{\mu_j\}$. The most effective method we found was:

(a) Let μ^o be the zero vector and define x^j to be the optimal solution to $D_{\mu j}$ (the Lagrangean subproblem associated with μ^j);

(b) Let w^* be an overestimate of $v(D)$ and $\{\pi_j\}$ a sequence of scalars such that $0 < \pi_j \leq 2$;

(c) Let

$$t_{j+1} = \pi_j \frac{\omega^* - v(D_\mu j)}{|b_2 - A_2 x^j|^2}$$

where

$v(\cdot)$ = the optimal value of problem $(\cdot)$,
$|\cdot|$ = the Euclidean norm of $\cdot$,
$A_2 x = b_2$ = the relaxed constraints.

We tried various alternatives for π_j before deciding on setting $\pi_0 = 1$ and decreasing π_j by a factor of two whenever a fixed number of iterations has passed without an improvement in the objective (this is based on a suggestion in Fisher[10]). The best value for this number of iterations was 10 for all problems we examined.