A Newton Method for Linear Programming

O. L. Mangasarian
Computer Sciences Department
University of Wisconsin
1210 West Dayton Street
Madison, WI 53706
olvi@cs.wisc.edu

Abstract

A fast Newton method is proposed for solving linear programs with a very large ($\approx 10^6$) number of constraints and a moderate ($\approx 10^2$) number of variables. Such linear programs occur in data mining and machine learning. The proposed method is based on the apparently overlooked fact that the dual of an asymptotic exterior penalty formulation of a linear program provides an *exact* least 2-norm solution to the dual of the linear program for *finite* values of the penalty parameter but *not* for the primal linear program. Solving the dual for a finite value of the penalty parameter yields an exact least 2-norm solution to the dual, but *not* a primal solution unless the parameter approaches zero. However, the exact least 2-norm solution to dual problem can be used to generate an accurate primal solution if $m \geq n$ and the primal solution is unique. Utilizing these facts, a fast globally convergent finitely terminating Newton method is proposed. A simple prototype of the method is given in eleven lines of MATLAB code. Encouraging computational results are presented such as the solution of a linear program with two million constraints that could not be solved by CPLEX 6.5 on the same machine.

1 Introduction

The method proposed here is motivated by the effectiveness of a finitely terminating Newton method proposed in [23] for the unconstrained minimization of strongly convex piecewise quadratic functions arising from quadratic programs and utilized successfully for classification problems in [10]. To apply this approach to linear programs, a reasonable choice is the least 2-norm formulation [18, 24, 25, 11] of a linear program as a strongly convex quadratic program, which gives an exact least 2-norm solution for finite parameter values. This has been done in [14] where, a finite Newton method was proposed but without any computational results and without giving an exact solution to the dual of the least 2-norm solution. In our approach here, we make no assumption about our linear program other than solvability and an implied uniqueness condition, which are explained in detail in Section 2. In Section 3 we state our Newton algorithm with an Armijo stepsize and give its global convergence. In Section 4 we give encouraging comparative test results with CPLEX 6.5 [5] on a class of synthetically generated sparse linear programs with as many as two million constraints as well as on six publicly available machine learning classification problems. We also give brief MATLAB codes for a test problem generator as well as a simple version of our Newton solver without an Armijo stepsize. This code is used to obtain all of our numerical results. On five out of seven synthetic test problems, the proposed method was faster than CPLEX by a factor in the range of 2.8 to 34.2. On the remaining two problems, CPLEX ran out of memory. On the six smaller machine learning classification problems, CPLEX was faster but LPN gave higher training and testing set classification correctness.

We note that under the implied uniqueness assumption of our paper, all that is needed is that the number of constraints m of the primal linear program (2) be no less than the number of variables n of the problem. However the principal effectiveness of the proposed method is to problems where m is much bigger than n as evidenced by the numerical examples of Section 4. Other approaches to linear programs with very large number of constraints are given in [4, 15]. Another related approach that uses similar ideas to those presented here is given in [29], but does not recover the primal solution from the dual as we do.

A word about our notation. All vectors will be column vectors unless transposed to a row vector by a prime superscript $'$. For a vector x in the n-dimensional real space R^n, the *plus function* x_+ is defined as $(x_+)_i = \max\,\{0, x_i\}$, $i = 1, \ldots, n$, while x_* denotes the subgradient of x_+ which is the step function defined as $(x_*)_i = 1$ if $x_i > 0$, $(x_*)_i = 0$ if $x_i < 0$, and $(x_*)_i \in [0, 1]$ if $x_i = 0$, $i = 1, \ldots, n$. The scalar (inner) product of two vectors x and y in the n-dimensional real space R^n will be denoted by $x'y$. The 2-norm of x will be denoted by $\|x\|$, while $\|x\|_1$ and $\|x\|_\infty$ will denote the 1-norm and ∞-norm respectively. For a matrix $A \in R^{m\times n}$, A_i is the ith row of A which is a row vector in R^n and $\|A\|$ is the 2-norm of A: $\max\limits_{\|x\|=1} \|Ax\|$. If $S \subset \{1, \ldots, m\}$, then A_S is the submatrix of A consisting of rows $A_{i\in S}$. A column vector of ones of arbitrary dimension will be denoted by e and the identity matrix of arbitrary order will be denoted by I. If f is a real valued function defined on the n-dimensional real space R^n, the gradient of f at x is denoted by $\nabla f(x)$ which is a column vector in R^n and the $n \times n$ matrix of second partial derivatives of f at x is denoted by $\nabla^2 f(x)$. For a piecewise quadratic function such as, $f(x) = \frac{1}{2}||(Ax-b)_+||^2$, where $A \in R^{m\times n}$ and $b \in R^m$ the ordinary Hessian does not exist because its gradient, the $n \times 1$ vector $\nabla f(x) = A'(Ax-b)_+$, is not differentiable. However, one can define its **generalized Hessian** [12, 7, 23] which is the $n \times n$ symmetric positive semidefinite matrix:

$$\partial^2 f(x) = A' diag(Ax-b)_* A, \tag{1}$$

where $diag(Ax-b)_*$ denotes an $m \times m$ diagonal matrix with diagonal elements $(A_i x - b_i)_*$, $i = 1, \ldots, m$. The generalized Hessian (1) has many of the properties of the regular Hessian [12, 7, 23] in relation to $f(x)$. Throughout this work, the notation ":=" will denote definition and "s.t." will stand for "such that".

2 Equivalence of Primal Exterior Penalty LP to Dual Least 2-Norm LP

We give in this section an apparently overlooked, but implied [18, 30, 14], result that a parametric exterior penalty formulation of a linear program for any sufficiently small but *finite* value of the penalty parameter, provides an *exact* least 2-norm solution to the dual linear program. We begin with the primal linear program:

$$\min_{x\in R^n} c'x \;\; s.t. \;\; Ax \le b, \tag{2}$$

where $c \in R^n$, $A \in R^{m\times n}$ and $b \in R^m$, and its dual:

$$\max_{u\in R^m} \; -b'u = -\min_{u\in R^m} b'u \;\; s.t. \;\; A'u + c = 0, \;\; u \ge 0. \tag{3}$$

We write the parametric exterior penalty formulation of the primal linear program for a fixed positive value of the penalty parameter ϵ as the unconstrained minimization problem:

$$\min_{x\in R^n} f(x) \tag{4}$$

where f is the penalty function:

$$f(x) := \min_{x \in R^n} \epsilon c'x + \frac{1}{2}\|(Ax-b)_+\|^2. \tag{5}$$

The positive penalty parameter ϵ needs to approach zero in order to obtain a solution to the original linear program [8, 2] from (4). However, this will not be the case if we look at the least 2-norm formulation [25, 18] of the dual linear program (3) :

$$-\min_{v \in R^m} (b'v + \frac{\epsilon}{2}v'v) \ \ s.t. \ \ A'v + c = 0, \ v \geq 0. \tag{6}$$

That (6) leads to a least 2-norm solution of the dual linear program (3) follows from the fact established in [25, 18] that for any positive ϵ such that $\epsilon \in (0, \bar{\epsilon}]$ for some positive $\bar{\epsilon}$, the minimization problem (6) picks among elements of the solution set of the dual linear program (3), that which minimizes the perturbation term $\frac{v'v}{2}$. Because the objective function of (6) is strongly convex, its solution $\bar{v}$ is unique. The necessary and sufficient Karush-Kuhn-Tucker optimality conditions for problem (6) are that there exists a $y \in R^n$ such that:

$$\epsilon v \geq 0, \ \epsilon v + b - Ay \geq 0, \ \epsilon v'(\epsilon v + b - Ay) = 0, \ A'v + c = 0, \tag{7}$$

or equivalently:

$$\epsilon v = (Ay - b)_+, \ A'v + c = 0. \tag{8}$$

That is:

$$v = \frac{1}{\epsilon}(Ay-b)_+, A'(Ay-b)_+ + \epsilon c = 0. \tag{9}$$

Defining $f(y)$ as in (5), the optimality conditions (9) for the dual least 2-norm solution become:

$$v = \frac{1}{\epsilon}(Ay-b)_+, \nabla f(y) = A'(Ay-b)_+ + \epsilon c = 0. \tag{10}$$

That is:

$$v = \frac{1}{\epsilon}(Ay-b)_+, \ y \in arg \min_{y \in R^n} f(y) = arg \min_{y \in R^n} \frac{1}{2}\|(Ay-b)_+\|^2 + \epsilon c'y, \tag{11}$$

which is precisely the necessary and sufficient condition that y be a minimum solution of the parametric exterior penalty function $f(y)$ for the primal linear program (2), for any positive value of the penalty parameter ϵ. Hence, solving the exterior penalty problem (4) for any positive ϵ provides a solution $v = \frac{1}{\epsilon}(Ay - b)_+$ to (6). If in addition, $\epsilon \in (0, \bar{\epsilon}]$ for some positive $\bar{\epsilon}$, it follows, by [25, Theorem 1], that v is a least 2-norm solution to the dual linear program (3). We have thus established the following.

Proposition 2.1 Equivalence of Least 2-Norm Dual to Finite-Parameter Penalty Primal *The unique least 2-norm solution to the dual linear program (3) is given by:*

$$v = \frac{1}{\epsilon}(Ay - b)_+, \tag{12}$$

where y is a solution of the primal penalty problem:

$$\min_{y \in R^n} f(y) = \frac{1}{2}\|(Ay - b)_+\|^2 + \epsilon c'y, \tag{13}$$

for any finite value of the penalty parameter $\epsilon \in (0, \bar{\epsilon}]$ for some positive $\bar{\epsilon}$.

We note that the gradient:

$$\nabla f(y) = A'(Ay - b)_+ + \epsilon c, \tag{14}$$

which is Lipschitz continuous with constant $\|A'\|\,\|A\|$, is not differentiable. However, as stated in the Introduction, a generalized Hessian [12, 7, 23] with many of the properties of the ordinary Hessian can be defined, which is the following $n \times n$ symmetric positive semidefinite matrix:

$$\partial^2 f(y) = A' diag(Ay - b)_* A, \tag{15}$$

where $diag(Ay - b)_*$ denotes an $m \times m$ diagonal matrix with diagonal elements $(A_i y - b_i)_*$, $i = 1, \ldots, m$. The step function $(\cdot)_*$, is defined in the Introduction and is implemented here with $(0)_* = 0$. The matrix $\partial^2 f(y)$ will be used to generate our Newton direction, or more precisely a modified Newton direction, since the generalized Hessian may be singular in general. In fact the direction that will be used is the following one:

$$d := -(\delta I + \partial^2 f(y))^{-1} \nabla f(y), \tag{16}$$

where δ is some small positive number. With this direction and a variety of step sizes [20, Example 2.2] we can establish global convergence. A key empirical computational property of this direction appears to be global convergence for a class of linear programs with $m \gg n$ from any starting point *without* any step size at all. The exact least 2-norm solution $\bar{v}$ for the dual linear program (3) is obtained by first solving the primal exterior penalty problem (13) for any $\epsilon \in (0, \bar{\epsilon}]$, and then using (12) to determine the unique least 2-norm dual solution $\bar{v}$. This exact dual solution can be utilized to generate an exact solution to the original primal linear program (2) by solving the constraints of the linear program (2) corresponding to positive components of $\bar{v}_j$ as equalities, that is:

$$A_j z = b_j, \ \ j \in S := \{j \mid \bar{v}_j > 0\}. \tag{17}$$

We note that this system of linear equations must always have a solution z as a consequence of the complementarity condition: $A_j z - b_j = 0$ for $\bar{v}_j > 0$, $j = 1, \ldots, m$. In fact (17) yields an exact solution of the primal linear program (2) if we make the additional assumption that the submatrix:

$$A_S \ \text{ has linearly independent columns,} \tag{18}$$

where S is defined in (17). This assumption which implies that that the solution of the linear system (17) is unique, is sufficient but not necessary for generating an exact primal solution as will be shown in Section 3.

We turn now to our algorithmic formulation and its convergence.

3 Linear Programming Newton Algorithm: LPN

Our proposed algorithm consists of solving (13) for an approximate solution y of the linear program (2), computing an exact least 2-norm solution v to the dual linear program (3) from (12), and finally computing an exact solution z to the primal linear program (2) from (17). In order to guarantee global convergence, we utilize an Armijo stepsize [1, 16] and need to make the linear independence assumption (18) on the least 2-norm solution of the dual linear program (3). We now state our algorithm.

Algorithm 3.1 LPN: Linear Programming Newton Algorithm *Set the parameter values ϵ, δ and tolerance tol (typically: 10^{-3}, 10^{-4}, & 10^{-12} respectively). Start with any $y^0 \in R^n$ (typically $y^0 = (\bar{A}'\bar{A} + \epsilon I)^{-1}\bar{A}'\bar{b}$, where $\bar{A}$ is an arbitrary $n \times n$ subset of A and $\bar{b}$ is the corresponding $n \times 1$ subset of b). For $i = 0, 1, \ldots$*

(I) $y^{i+1} = y^i - \lambda_i(\partial^2 f(y^i) + \delta I)^{-1}\nabla f(y^i) = y^i + \lambda_i d^i$,
where the Armijo stepsize $\lambda_i = \max\{1, \frac{1}{2}, \frac{1}{4}, \ldots\}$ is such that:

$$f(y^i) - f(y^i + \lambda_i d^i) \geq -\frac{\lambda_i}{4}\nabla f(y^i)' d^i, \tag{19}$$

and d^i is the modified Newton direction:

$$d^i = -(\partial^2 f(y^i) + \delta I)^{-1} \nabla f(y^i). \tag{20}$$

(II) Stop if $\|y^i - y^{i+1}\| \leq tol$. Else, set $i = i + 1$ and go to (I).

(III) Define the least 2-norm dual solution v as:

$$v = \frac{1}{\epsilon}(Ay^{i+1} - b)_+ \tag{21}$$

and a solution z of the primal linear program by:

$$A_j z = b_j, \;\; j \in S := \{j \mid v_j > 0, \; j = 1, \ldots, m\}. \tag{22}$$

We state a convergence result for this algorithm now.

Theorem 3.2 *Each accumulation point $\bar{y}$ of the sequence $\{y^i\}$ generated by Algorithm 3.1 solves the exterior penalty problem (4). The corresponding $\bar{v}$ obtained by setting y to $\bar{y}$ in (12) is the exact least 2-norm solution to the dual linear program (3), provided ϵ is sufficiently small. An exact solution $\bar{z}$ to the primal linear program (2) is obtained from (22) by solving for z with $v = \bar{v}$, provided that the submatrix $A_{\bar{S}}$ of A has linearly independent columns where:*

$$\bar{S} := \{j \mid \bar{v}_j > 0, \; j = 1, \ldots, m\}. \tag{23}$$

Proof Let $\epsilon > 0$. That each accumulation point $\bar{y}$ of the sequence $\{y^i\}$ solves the minimization problem (13) follows from standard results such as [20, Theorem 2.1, Examples 2.1(i), 2.2(iv)] and the facts that the direction choice d^i of (20) satisfies:

$$-\nabla f(y^i)' d^i = \nabla f(y^i)'(\delta I + \partial^2 f(y^i))^{-1} \nabla f(y^i) \geq (\delta + \|A'A\|)^{-1} \|\nabla f(y^i)\|^2, \tag{24}$$

and that we are using an Armijo stepsize (19). Now let $\epsilon \in (0, \bar{\epsilon}]$. Then by Proposition 2.1 the corresponding $\bar{v}$ obtained by setting y to $\bar{y}$ in (12) is the exact least 2-norm solution to the dual linear program (3). If the submatrix $A_{\bar{S}}$ has linearly independent columns, then the solution of $\bar{z}$ of (22) with $v = \bar{v}$ is unique and must be a solution of the primal linear program (2). □

Remark 3.3 Choice of ϵ *Determining the size of $\bar{\epsilon}$, such that the solution v of the quadratic program (6) for $\epsilon \in (0, \bar{\epsilon}]$, is the least 2-norm solution of the dual problem (3), is not an easy problem theoretically. However, computationally this does not seem to be critical and is effectively addressed as follows. By [17, Corollary 3.2], if for two successive values of ϵ: $\epsilon^1 > \epsilon^2$, the corresponding solutions of the ϵ-perturbed quadratic programs (6): u^1 and u^2 are equal, then under certain assumptions, $u = u^1 = u^2$ is the least 2-norm solution of the dual linear program (3). This result is implemented computationally by using an ϵ, which when decreased by a factor of 10 yields the same solution to (6).*

We note that the assumptions of Theorem 3.2, and in particular uniqueness of the solution of (22) with $v = \bar{v}$, imply the uniqueness of the solution to the primal linear program (2). This follows from the fact that the least 2-norm multiplier $\bar{v}$ is a valid optimal multiplier for *all* optimal solutions of the primal linear program (2) and hence all primal solutions must equal the solution of the uniquely solvable system $A_{\bar{S}}z = b_{\bar{S}}$. We term our condition of *linear independence of the columns of $A_{\bar{S}}$* as a *strong uniqueness* condition, because it implies primal solution uniqueness but is not implied by it. Example 3.4 below has a unique but not strongly unique solution. However, it is still solvable by our proposed Newton method despite the fact that its solution is not strongly unique.

Example 3.4 *For the primal and dual problems:*

$$\min_{x \in R^2} x_1 \ \ s.t. \ \ -x_1 + x_2 \leq -1, \ x_1 - x_2 \leq 1, \ -x_1 \leq 0, \tag{25}$$

$$\max_{0 \leq u \in R^3} u_1 - u_2 \ \ s.t. \ \ u_1 - u_2 + u_3 = 1, \ -u_1 + u_2 = 0, \tag{26}$$

The unique primal solution is $x_1 = 0, \ x_2 = -1$. The dual solution set is $\{u \in R^3 \mid u_3 = 1, \ u_1 = u_2 \geq 0\}$ and the least 2-norm dual solution is $\bar{u}_1 = \bar{u}_2 = 0, \ \bar{u}_3 = 1$. Thus $A_{\bar{S}} = A_3 = [-1 \quad 0]$ and the solution $x_1 = 0, \ x_2 = -1$ is not strongly unique because the columns of A_3 are not linearly independent. However, Algorithm 3.1 solved this problem in 5 iterations using the MATLAB Code 4.2 with the given defaults and without an Armijo stepsize and yielding $z = [0 \ \ -1]'$ and $v = [0 \ \ 0 \ \ 1]'$.

We note further, that Algorithm 3.1 can possibly handle linear programs with even non-unique solutions as evidenced by the following simple example.

Example 3.5 *For the primal and dual problems:*

$$\min_{x \in R^2} x_1 + x_2 \;\; s.t. \;\; -x_1 - x_2 \leq -1, \; -x_1 \leq 0, \; -x_2 \leq 0, \tag{27}$$

$$\max_{0 \leq u \in R^3} u_1 \;\; s.t. \;\; u_1 + u_2 = 1, \; u_1 + u_3 = 1, \tag{28}$$

the primal solution set is $\{x \in R^2 \mid x_1 + x_2 = 1, \; x_1 \geq 0, \; x_2 \geq 0\}$, *while the dual solution set consists of the unique point:* $u_1 = 1, \; u_2 = u_3 = 0$. *When Algorithm 3.1 is applied to this example using the MATLAB Code 4.2 with the given default values, the exact primal* $z = [1 \;\; 0]'$ *and dual* $v = [1 \;\; 0 \;\; 0]'$ *solutions were obtained in two iterations without using the Armijo stepsize.*

We give now a sufficient (but in general not necessary) condition for the sequence $\{y^i\}$ of Algorithm 3.1 to converge.

Corollary 3.6 *The sequence* $\{y^i\}$ *is bounded and hence has an accumulation point and the corresponding sequence* $\{v^i\}$ *converges to the least 2-norm solution of the dual linear program (3), provided that* R^n *is contained in the conical hull of the rows of* $\begin{bmatrix} c' \\ A \end{bmatrix}$. *That is, for each* $p \in R^n$*:*

$$\zeta c' + s'A = p, \; (\zeta, s) \geq 0, \; \textit{has a solution} \; (\zeta, s) \in R^{1+n}. \tag{29}$$

Proof We shall prove that the sequence $\{y^i\}$ is bounded, hence it has an accumulation point. Since all accumulation points of the sequence $\{v^i\}$, that correspond to accumulation points of $\{y^i\}$, are all equal to the unique least 2-norm solution $\bar{v}$ of (3), the sequence $\{v^i\}$ must converge to $\bar{v}$.

We show now that the sequence $\{y^i\}$ is bounded by exhibiting a contradiction if it were unbounded and if assumption (29) holds.

Suppose that $\{y^i\}$ is unbounded, then for some subsequence (for simplicity we drop the subsequence index) we have by Algorithm 3.1 that:

$$f(y^i) \leq f(y^0), \;\; y^i \to \infty. \tag{30}$$

Dividing by $\|y^i\|^2$ gives:

$$\frac{1}{2}\|(A\frac{y^i}{\|y^i\|} - \frac{b}{\|y^i\|})_+\|^2 + \epsilon c' \frac{y^i}{\|y^i\|^2} \leq \frac{f(y^0)}{\|y^i\|^2}. \tag{31}$$

Letting $y^i \to \infty$ and denoting by $\hat{y}$ an accumulation point of the bounded sequence $\frac{y^i}{\|y^i\|}$, we have that:

$$\frac{1}{2}\|(A\hat{y})_+\|^2 \leq 0. \tag{32}$$

Hence

$$A\hat{y} \leq 0, \quad \hat{y} \neq 0. \tag{33}$$

From (31) multiplied by $\|y^i\|$ and noting that its first term is nonnegative we have that:

$$\epsilon c' \frac{y^i}{\|y^i\|} \leq \frac{f(y^0)}{\|y^i\|}, \tag{34}$$

from which, upon letting $y^i \to \infty$, we get:

$$c'\hat{y} \leq 0. \tag{35}$$

Combining (33) and (35) gives:

$$A\hat{y} \leq 0, \quad c'\hat{y} \leq 0 \quad \hat{y} \neq 0. \tag{36}$$

This however contradicts assumption (29) if we set $p = \hat{y}$ in (29) as follows:

$$0 = \zeta c'\hat{y} + s'A\hat{y} - \hat{y}'\hat{y} \leq -\hat{y}'\hat{y} < 0. \ \square \tag{37}$$

Remark 3.7 *We note that condition (29) of Corollary 3.6 is equivalent to the primal linear program having a bounded level sets, which is similar to the assumptions made in [14]. To see this we have by the Farkas theorem [19, Theorem 2.4.6] that (29) is equivalent to:*

$$Ax \leq 0, \ c'x \leq 0, \ p'x > 0, \ \textit{has no solution}\, x \in R^n, \ \textit{for each}\, p \in R^n. \tag{38}$$

This in turn is equivalent to:

$$Ax \leq 0, \ c'x \leq 0, \ x \neq 0 \ \textit{has no solution}\, x \in R^n. \tag{39}$$

This is equivalent to the boundedness of the level set $\{x \mid Ax \leq b, \ c'x \leq \alpha\}$ of the linear program (2), for any real number α.

We proceed now to numerical testing of our algorithm.

4 Numerical Tests

The objectives of the preliminary numerical tests presented here are to display the simplicity of the proposed LPN algorithm, its competitiveness with a state-of-the-art linear programming code for special types of problems, and its ability to provide exact answers for a class of linear programs with a very large number of constraints and such that $m \gg n$. Typically, $m \geq 10n$. We also demonstrate the effectiveness of the LPN algorithm by testing it and comparing it with CPLEX on standard classification test problems: four from the University of California Machine Learning Repository [27] and two publicly available datasets [28].

4.1 Very large synthetic datasets

We first test LPN on very large synthetically generated test problems. To display the simplicity of LPN we give two MATLAB m-file codes below. The first, "lpgen", is a linear programming test problem generator. The second, "lpnewt1" is an implementation of Algorithm 3.1 without the apparently unnecessary Armijo stepsize for the class of problems tested here. A sufficient well conditioned property that eliminates the Armijo stepsize requirement [23, Theorem 3.6] apparently is not needed in all the problems tested here. Both m-files are given below as Code 4.1 and Code 4.2, and are available at: www.cs.wisc.edu/math-prog/olvi.

The test problem generator lpgen generates a random constraint matrix A for a given m, n and density d. The elements of A are uniformly distributed between -50 and +50. A primal random solution x with elements in [-10,10], approximately half of which are zeros, and a dual solution u with elements in [0,10], approximately $3n$ of which are positive, are first specified. These solutions are then used to generate an objective function cost vector c and a right hand side vector b for the linear program (2). The number $3n$ of positive dual variables is motivated by support vector machine applications [22, 6, 31] where the number of positive multipliers corresponding to support vectors is often a few multiples of the dimensionality n of the input space.

To test LPN we solved a number of linear programs generated by the lpgen Code 4.1 on LOCOP2, a 400 Mhz Pentium II machine with a maximum of 2 Gigabytes of memory running Windows NT Server 4.0, and compared it with CPLEX 6.5 [5]. LPN used only the default values given in Code 4.2, which of course can be overridden. The results are presented in Table 1.

Seven problems generated by the lpgen Code 4.1 were solved by the LPN Code 4.2. Problem size varied between 10^5 to 10^7 in the number of nonzero elements of the constraint matrix A. LPN solved all these problems in 11 to 26 iterations to an accuracy better than 10^{-13}. Comparing times for the problems solved, CPLEX ran out of memory on two problems, and was 2.8 to 34.2 times slower on the remaining five problems with one problem, the third in Table 1, very poorly solved by CPLEX with a primal solution error of 7.0. The dual CPLEX option was used throughout the numerical test problems because it gave the best results for the type of problems tested. The other options, primal and barrier, did worse. For example on the fifth test problem in Table 1 with dual CPLEX time of 238.4 seconds, the primal CPLEX took 2850.8 seconds while the barrier CPLEX ran out of memory.

In order to show that LPN can also handle linear programs with nonunique solutions, as requested by one of the referees who also noted that the linear programs of Table 1 as generated by lpgen had unique feasible points as well, we perturbed the right hand side b of (2) so that the number of active constraints at the solutions obtained by LPN and CPLEX were less than n. These results are presented in Table 2 and show that the accuracy of LPN has decreased. However, it is still capable of solving these problems to a reasonable accuracy and can still handle the two largest problem which CPLEX ran out of memory on.

Code 4.1 lpgen MATLAB lp test problem generator

```
%lpgen: Generate random solvable lp: min c'x s.t. Ax =< b; A:m-by-n
%Input: m,n,d(ensity); Output: A,b,c; (x,u): primal-dual solution
pl=inline('(abs(x)+x)/2');%pl(us) function
tic;A=sprand(m,n,d);A=100*(A-0.5*spones(A));
u=sparse(10*pl(rand(m,1)-(m-3*n)/m));
x=10*spdiags((sign(pl(rand(n,1)-rand(n,1)))),0,n,n)*(rand(n,1)-rand(n,1));
c=-A'*u;b=A*x+spdiags((ones(m,1)-sign(pl(u))),0,m,m)*10*ones(m,1);toc0=toc;
format short e;[m n d toc0]
```

Code 4.2 lpnewt1 MATLAB LPN Algorithm 3.1 without Armijo

```
%lpnewt1: Solve primal LP: min c'x s.t. Ax=<b
%Via Newton for least 2-norm dual LP: max -b'v s.t. -A'v=c, v>=0
%Input: c,A,b,epsi,delta,tol,itmax;Output: v (l2norm dual sol), z primal sol
epsi=1e-3;tol=1e-12;delta=1e-4;itmax=100;%default inputs
pl=inline('(abs(x)+x)/2');%pl(us) function
tic;i=0;z=0;y=((A(1:n,:))'*A(1:n,:)+epsi*eye(n))\(A(1:n,:))'*b(1:n);%initial y
while (i<itmax & norm(y-z,inf)>tol & toc<1800)
    df=A'*pl((A*y-b))+epsi*c;
    d2f=A'*spdiags(sign(pl(A*y-b)),0,m,m)*A+delta*speye(n);
    z=y;y=y-d2f\df;
    i=i+1;
end
toc1=toc;v=pl(A*y-b)/epsi;t=find(v);z=A(t,:)\b(t);toc2=toc;
format short e;[epsi delta  tol i-1 toc1 toc2 norm(x-y,inf) norm(x-z,inf)]
```

4.2 Machine learning test problems

In this section we test and compare LPN with CPLEX on four classification datasets from the UCI Machine Learning Repository [27] and two publicly available datasets [28]. Again, we use LOCOP2.

Table 1: **Comparison of LPN Algorithm 3.1 and CPLEX 6.5 [5]. "oom" denotes "out of memory". LPN Time is total time, i.e. toc2 from Code 4.2**

	LPN			Dual CPLEX 6.5	
Problem Size× Density $m \times n \times d$	Time Seconds	Iter. #	Accuracy $\|x - z\|_\infty$	Time Seconds	Accuracy $\|x - r\|_\infty$
$(2 \times 10^6) \times 100 \times 0.05$	1041.8	26	1.1×10^{-14}	oom	oom
$(1.5 \times 10^6) \times 100 \times 0.05$	787.0	26	8.8×10^{-15}	oom	oom
$(1.0 \times 10^5) \times 1000 \times 0.1$	840.5	14	5.8×10^{-14}	28716.6	7.0
$(1.0 \times 10^5) \times 100 \times 1.0$	228.7	15	8.9×10^{-15}	905.0	7.5×10^{-14}
$(1.0 \times 10^5) \times 100 \times 0.1$	50.7	18	8.9×10^{-15}	238.4	5.3×10^{-12}
$(1.0 \times 10^4) \times 1000 \times 0.1$	417.5	11	5.1×10^{-14}	3513.2	5.4×10^{-12}
$(1.0 \times 10^4) \times 100 \times 0.1$	4.9	17	7.3×10^{-15}	13.8	1.3×10^{-11}

The classification model that we shall employ here consists of a linear support vector machine [6, 32, 3, 22] which we can state as the following linear program:

$$\begin{array}{ll} \min\limits_{(w,\gamma,s,\epsilon),\in R^{q+1+q+1}} & e's + \nu\epsilon \\ s.t. & D(Bw - e\gamma) + e\epsilon \geq e, \quad -s \leq w \leq s, \quad \epsilon \geq 0, \quad r'w \geq 2. \end{array} \tag{40}$$

Here, e is a column vector of ones and ν is a positive parameter that balances the model's ability to generalize to new unseen data (small ν) versus empirical data fitting (large ν). The matrices $B \in R^{p\times q}$ and $D \in R^{p\times p}$ constitute the problem data. The p rows of B represent p given data points in the input space R^q, with each point belonging to class $+1$ if the corresponding diagonal element of the diagonal matrix D is $+1$, or to class -1 if the corresponding diagonal element is -1. The vector r is the difference between the mean of the points in class $+1$ and the mean of the points in class -1. The linear program (40) generates a separating plane:

$$x'w = \gamma, \tag{41}$$

which approximately separates the points of classes $+1$ and -1. This separating plane lies midway between the parallel bounding planes:

$$\begin{array}{l} x'w = \gamma + 1, \\ x'w = \gamma - 1. \end{array} \tag{42}$$

Table 2: **Comparison of LPN Algorithm 3.1 and CPLEX 7.1 [5] on linear programs with nonunique solutions. "oom" denotes "out of memory".**

	LPN				Dual CPLEX 7.1	
Problem Size× Density $m \times n \times d$	Time Seconds	Iter. #	Accuracy $\lvert c'z + b'v \rvert$ $\lVert(-Az+b)_+\rVert_\infty$	Act.Constr. #	Time Seconds	Act.Constr. #
$(2 \times 10^6) \times 100 \times 0.05$	3280.0	5	3.8×10^{-5} 4.1×10^{-6}	34	oom	oom
$(1.5 \times 10^6) \times 100 \times 0.05$	1899.1	5	1.3×10^{-4} 3.3×10^{-6}	33	oom	oom
$(1 \times 10^5) \times 1000 \times 0.1$	3014.5	7	5.7×10^{-5} 2.5×10^{-6}	240	15335.0	102
$(1 \times 10^5) \times 100 \times 1.0$	190.0	6	8.7×10^{-5} 2.5×10^{-6}	32	317.5	57
$(1 \times 10^5) \times 100 \times 0.1$	52.3	6	2.0×10^{-10} 7.3×10^{-7}	21	95.6	89
$(1 \times 10^4) \times 1000 \times 0.1$	48.2	8	1.4×10^{-7} 1.6×10^{-6}	43	2018.4	105
$(1 \times 10^4) \times 100 \times 0.1$	4.5	7	2.5×10^{-6} 1.8×10^{-6}	22	6.8	86

With a maximum error of ϵ, the points of class $+1$ lie in closed halfspace:

$$\{x \mid x'w \geq \gamma + 1\}. \tag{43}$$

Similarly, the points of class -1 lie, with maximum error of ϵ, in the closed halfspace:

$$\{x \mid x'w \leq \gamma - 1\}. \tag{44}$$

The error ϵ is minimized by the linear program (40) relative to $e's = \|w\|_1$. The last constraint of the linear program (40) excludes the degenerate solution $w = 0$ by implying the usually satisfied condition that the 1-norm distance $\frac{2}{\|w\|_\infty}$ [21] between the two parallel bounding planes (42) does not exceed the 1-norm distance $\|r\|_1$ between the means of the two classes as follows:

$$\|w\|_\infty \cdot \|r\|_1 \geq r'w \geq 2. \tag{45}$$

The linear program (40) was set up in the form of the linear program (2) with $n = 2q + 2$ and $m = 2p + 2q + 2$. The linear program (2) was solved by both LPN and CPLEX for six standard test problems varying in size between 297 points to 4192 points, and input spaces of dimensionality varying between 6 and 14. The results are given in Table 3. Because these are relatively small problems, CPLEX solved these problems in less time than LPN and with much greater accuracy as measured by the relative infeasibility of a solution point to (2):

$$\frac{\|(Ax - b)_+\|_1}{\|Ax\|_1}. \tag{46}$$

However, correctness of the classifying separating plane on both the original data (training correctness) and on the tenfold testing (testing correctness) obtained by leaving one tenth of the data out for testing, repeating ten times and averaging, were higher for LPN than those for CPLEX on all six test problems. In addition, the LPN correctness values were comparable to those obtained by other methods [13, 10, 26]. These test correctness values are the key properties by which machine learning classifiers are evaluated and are not always best when a very accurate solution to the linear program (40) is obtained. The explanation for this apparent paradox is that a highly accurate solution causes overfitting and may bias the classifier towards the given empirical data at the expense of unseen data and hence results in poor generalization which translates into lower test set correctness as seen here for the CPLEX results. Application of Newton type methods have yielded some of the best test correctness results for classification problems [16, 10, 9]. The parameter value for ν for both LPN and CPLEX was the same and chosen to optimize training set correctness. For all problems in Table 3, the value $\nu = 10^5$ was used. All parameters of LPN used were set at the default values given in the MATLAB Code 4.2.

5 Conclusion

We have presented a fast Newton algorithm, LPN, for solving a class of linear programs. Computational testing on test problems with $m \gg n$ and $m \geq n$ demonstrate the effectiveness of the proposed method in comparison with a state-of-the-art linear programming solver for this class of linear programs. Future work might include the study of sharp theoretical conditions under which LPN converges without a stepsize and in a finite number of steps, as it does in all the numerical examples presented in this work. Computational

Table 3: **Comparison of LPN Algorithm 3.1 and CPLEX 6.5 [5] on six publicly available machine learning problems. $m \times n$ is the size of the matrix A of (2). $p \times q$ is the size of the matrix B of (40). "Infeas" denotes solution infeasibility (46). "Train" is training set correctness. "Test" is ten-fold testing set correctness.**

	LPN		Dual CPLEX 6.5	
Problem $m \times n$, $p \times q$	Seconds Infeas	Train Test	Seconds Infeas	Train Test
Bupa Liver 359×14, 345×6	0.11 8.10e-3	68.12% 66.71%	0.05 3.08e-19	64.64% 63.92%
Pima Indians 786×18, 768×8	0.21 4.30e-3	76.69% 76.20%	0.13 1.29e-18	73.83% 73.39%
Cleveland Heart 325×28, 297×13	0.14 1.31e-2	87.54% 84.74%	0.09 5.44e-18	83.16% 81.69%
Housing 534×28, 506×13	0.24 9.80e-3	86.96% 85.97%	0.13 1.02e-17	83.60% 83.72%
Galaxy Bright 2492×30, 2462×14	6.23 8.03e-4	99.63% 99.38%	0.79 5.37e-15	99.15% 99.16%
Galaxy Dim 4222×30, 4192×14	13.86 1.50e-3	94.58% 94.43%	1.57 1.81e-18	90.96% 90.71%

improvements might include alternate ways of generating a primal solution from the dual least 2-norm solution thus enabling the method to handle all types of linear programs.

Acknowledgments

The research described in this Data Mining Institute Report 02-02, March 2002, was supported by National Science Foundation Grants CCR-9729842, CCR-0138308 and CDA-9623632, by Air Force Office of Scientific Research Grant F49620-00-1-0085 and by the Microsoft Corporation. I am indebted to Robert R. Meyer for suggesting the MATLAB command SPONES to help generate very large test problems, to Jinho Lim for providing a CPLEX mex file for MATLAB that measures actual CPLEX cpu time and with options for primal simplex, dual simplex, network and barrier methods, and to Steve Wright for suggesting the use of various CPLEX options. I am indebted to my PhD student Michael Thompson for the numerical results presented in Table 3, and to two referees and an Associate Editor for constructive suggestions. Revised August and December 2002.

References

[1] L. Armijo. Minimization of functions having Lipschitz-continuous first partial derivatives. *Pacific Journal of Mathematics*, 16:1–3, 1966.

[2] D. P. Bertsekas. *Nonlinear Programming.* Athena Scientific, Belmont, MA, second edition, 1999.

[3] P. S. Bradley and O. L. Mangasarian. Feature selection via concave minimization and support vector machines. In J. Shavlik, editor, *Machine Learning Proceedings of the Fifteenth International Conference(ICML '98)*, pages 82–90, San Francisco, California, 1998. Morgan Kaufmann. ftp://ftp.cs.wisc.edu/math-prog/tech-reports/98-03.ps.

[4] K. L. Clarkson. Las vegas algorithms for linear and integer programming. *Journal of the Association for Computing Machinery*, 42(2):488–499, march 1995.

[5] CPLEX Optimization Inc., Incline Village, Nevada. *Using the CPLEX(TM) Linear Optimizer and CPLEX(TM) Mixed Integer Optimizer (Version 2.0)*, 1992.

[6] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines.* Cambridge University Press, Cambridge, MA, 2000.

[7] F. Facchinei. Minimization of SC^1 functions and the Maratos effect. *Operations Research Letters*, 17:131–137, 1995.

[8] A. V. Fiacco and G. P. McCormick. *Nonlinear Programming: Sequential Unconstrained Minimization Techniques.* John Wiley & Sons, New York, NY, 1968.

[9] G. Fung and O. L. Mangasarian. A feature selection newton method for support vector machine classification. Technical Report 02-03, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, September 2002. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/02-03.ps. Computational Optimization and Applications, to appear.

[10] G. Fung and O. L. Mangasarian. Finite Newton method for Lagrangian support vector machine classification. Technical Report 02-01, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, February 2002. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/02-01.ps. Neurocomputing, to appear.

[11] A. I. Golikov and Yu. G. Evtushenko. Search for normal solutions in linear programming. *Computational Mathematics and Mathematical Physics*, 14(12):1694–1714, 2000.

[12] J.-B. Hiriart-Urruty, J. J. Strodiot, and V. H. Nguyen. Generalized hessian matrix and second-order optimality conditions for problems with C^{L1} data. *Applied Mathematics and Optimization*, 11:43–56, 1984.

[13] T. Joachims. Making large-scale support vector machine learning practical. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, pages 169–184, Cambridge, MA, 1999. MIT Press.

[14] C. Kanzow, H. Qi, and L. Qi. On the minimum norm solution of linear programs. Preprint, University of Hamburg, Hamburg, 2001. http://www.math.uni-hamburg.de/home/kanzow/paper.html. Journal of Optimization Theory and Applications, to appear.

[15] K. Krishnan and J. Mitchell. A linear programming approach to semidefinite programming problems. Working paper, RPI, Troy, NY, May 2001.

[16] Y.-J. Lee and O. L. Mangasarian. SSVM: A smooth support vector machine. *Computational Optimization and Applications*, 20:5–22, 2001. Data Mining Institute, University of Wisconsin, Technical Report 99-03. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/99-03.ps.

[17] S. Lucidi. A new result in the theory and computation of the least-norm solution of a linear program. *Journal of Optimization Theory and Applications*, 55:103–117, 1987.

[18] O. L. Mangasarian. Normal solutions of linear programs. *Mathematical Programming Study*, 22:206–216, 1984.

[19] O. L. Mangasarian. *Nonlinear Programming*. SIAM, Philadelphia, PA, 1994.

[20] O. L. Mangasarian. Parallel gradient distribution in unconstrained optimization. *SIAM Journal on Control and Optimization*, 33(6):1916–1925, 1995. ftp://ftp.cs.wisc.edu/tech-reports/reports/1993/tr1145.ps.

[21] O. L. Mangasarian. Arbitrary-norm separating plane. *Operations Research Letters*, 24:15–23, 1999. ftp://ftp.cs.wisc.edu/math-prog/tech-reports/97-07r.ps.

[22] O. L. Mangasarian. Generalized support vector machines. In A. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 135–146, Cambridge, MA, 2000. MIT Press. ftp://ftp.cs.wisc.edu/math-prog/tech-reports/98-14.ps.

[23] O. L. Mangasarian. A finite Newton method for classification problems. Technical Report 01-11, Data Mining Institute, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, December 2001. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/01-11.ps.*Optimization Methods and Software* 17, 2002, 913-929.

[24] O. L. Mangasarian and R. De Leone. Error bounds for strongly convex programs and (super)linearly convergent iterative schemes for the least 2-norm solution of linear programs. *Applied Mathematics and Optimization*, 17:1–14, 1988.

[25] O. L. Mangasarian and R. R. Meyer. Nonlinear perturbation of linear programs. *SIAM Journal on Control and Optimization*, 17(6):745–752, November 1979.

[26] O. L. Mangasarian and D. R. Musicant. Lagrangian support vector machines. *Journal of Machine Learning Research*, 1:161–177, 2001. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/00-06.ps.

[27] P. M. Murphy and D. W. Aha. UCI machine learning repository, 1992. www.ics.uci.edu/∼mlearn/MLRepository.html.

[28] S. Odewahn, E. Stockwell, R. Pennington, R. Humphreys, and W. Zumach. Automated star/galaxy discrimination with neural networks. *Astronomical Journal*, 103(1):318–331, 1992.

[29] M. C. Pinar. Piecewise-linear pathways to optimal solution set in linear programming. *Journal of Optimization Theory and Applications*, 93:619–634, 1997.

[30] P. W. Smith and H. Wolkowicz. A nonlinear equation for linear programming. *Mathematical Programming*, 34:235–238, 1986.

[31] A. Smola and B. Schölkopf. *Learning with Kernels*. MIT Press, Cambridge, MA, 2002.

[32] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, second edition, 2000.