Chapter 5

Computational Results

5.1 Introduction

This chapter presents the results of the numerical experiments we have conducted using the infeasible-interior-point potential-reduction algorithm for linear programming developed in Chapter 2. We implemented this algorithm in the MATLAB environment [Mat92] using LIPSOL [Zha95b], Zhang's implementation of an infeasible-interior-point path-following algorithm, as a starting point. We tested the algorithm on the NETLIB set of feasible and infeasible problems [Gay85]. Our results are encouraging; while our implementation performs only slightly worse than LIPSOL on feasible problems (we need less than one extra iteration per problem on average), for most infeasible problems we obtain approximate proofs of infeasibility in fewer iterations than LIPSOL despite the fact that LIPSOL just stops when there is a weak (heuristic) indication of infeasibility.

5.2 Implementation

Before we present our numerical results we describe some details of our implementation.

5.2.1 Upper Bounds

In most linear programming problems a certain subset of the variables are subjected to simple lower and upper bounds of the form

$$l_i \leq x_i \leq u_i, \tag{4.1}$$

where l_i and u_i are the lower and upper bounds on the variable x_i. We will assume that all lower bounds are zero. This is not a restrictive assumption since by shifting the variables we can transform nonzero lower bounds into nonnegativity constraints. This is indeed what we do in our implementation. Now, letting $\mathcal{U}$ be the subset of the index set of variables that corresponds to the upper bounded variables, we can state the problem as follows:

$$\begin{array}{rrcl} \min_x & c^T x & & \\ & Ax & = & b \\ & x_i & \leq & u_i, \;\; i \in \mathcal{U} \\ & x & \geq & 0, \end{array} \tag{4.2}$$

where $A \in I\!R^{m \times n}$, $b \in I\!R^m$, $c \in I\!R^n$, and the u_i's are the nonnegative upper bounds. Without loss of generality we will assume that

$$\mathcal{U} \;\; = \;\; \{1, 2, \ldots, n_u\},$$

where $n_u \leq n$ is the number of upper bounded variables. Let x_u be the subvector of x that corresponds to the index set $\mathcal{U}$. Now adding slack variables v_i for $i =$

$1,\ldots,n_u$ we express (4.2) in the standard form:

$$\begin{array}{rrcrcl}\min_{x,v} & c^T x & & & & \\ & Ax & & & = & b \\ & x_u & + & v & = & u \\ & x & , & v & \geq & 0.\end{array} \tag{4.3}$$

The dual of this problem is:

$$\begin{array}{rrcrcrcl}\max_{y,s,w} & b^T y & & & - & u^T w & & \\ & A^T y & + & s & - & E^T w & = & c \\ & & & s & , & w & \geq & 0,\end{array} \tag{4.4}$$

where the $n_u \times n$ dimensional matrix E is defined by $E := [I_{n_u}, 0]$, i.e., E^T is the matrix of an operator that appends $n - n_u$ zeros to the end of an n_u dimensional vector. Note that the equation $x_u + v = u$ in (4.3) can be rewritten as $-Ex - v = -u$ using the matrix E.

Let us now write the above primal-dual pair of problems in the homogeneous self-dual framework described in Chapter 2. Given any $x_0 > 0$, $v_0 > 0$, $s_0 > 0$, $w_0 > 0$, $\kappa_0 > 0$, and y_0, we formulate the following artificial problem:

$$
\begin{array}{ll}
\min & (n+n_u+1)\,\mu_0\,\theta \\
& \begin{bmatrix} 0 & 0 & A & -b & \bar{b} & 0 & 0 & 0 \\ 0 & 0 & -E & u & -\bar{u} & -I & 0 & 0 \\ -A^T & E^T & 0 & c & -\bar{c} & 0 & -I & 0 \\ b^T & -u^T & -c^T & 0 & \bar{z} & 0 & 0 & -1 \\ -\bar{b}^T & \bar{u}^T & \bar{c}^T & -\bar{z} & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} y \\ w \\ x \\ \tau \\ \theta \\ v \\ s \\ \kappa \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ -(n+n_u+1)\mu_0 \end{bmatrix} \\
& y,\,\theta \text{ free},\ w,\ x,\ \tau,\ v,\ s,\ \kappa \ \geq \ 0,
\end{array}
$$

where

$$\bar{b} = b - Ax_0,\ \ \bar{u} = u - Ex_0 - v_0,\ \ \bar{c} = c - A^T y_0 + E^T w_0 - s_0,$$

$$\bar{z} = c^T x_0 + \kappa_0 - b^T y_0 + u^T w_0 \ \text{ and } \mu_0 = \frac{x_0^T s_0 + v_0^T w_0 + \kappa_0}{n + n_u + 1}.$$

With this formulation in mind we extend our algorithm to treat upper bounds on the variables separately. Note that in this case we use the following generalization of our infeasible-point primal-dual potential function:

$$
\begin{aligned}
\psi'(x,v,y,s,w,\kappa) \ := \ & 2(n+n_u+1)\ln\max\{\|Ax-b\|, \|A^T y - E^T w + s - c\|, \\
& \|u - x_u - v\|, c^T x + \kappa - b^T y + u^T w\} \\
& - \textstyle\sum_{i=1}^{n} \ln x_i - \sum_{i=1}^{n} \ln s_i - \sum_{i=1}^{n_u} \ln v_i - \sum_{i=1}^{n_u} \ln w_i - \ln \kappa.
\end{aligned} \tag{4.5}
$$

The properties of the potential function ψ established in Section 2.4 also hold for ψ'. Therefore, the analysis of the algorithm presented in Section 2.5 extend to an algorithm using ψ' and identical computational complexity results hold for such an algorithm, since $n + n_u \leq 2n = \mathcal{O}(n)$.

5.2.2 Search Directions

In Chapter 2 we established that our infeasible-interior-point potential-reduction algorithm has polynomial complexity by showing that the potential function ψ can be reduced by at least $\mathcal{O}(\frac{1}{\sqrt{n}})$ from any infeasible point using one of the Kojima-Mizuno-Yoshise (KMY) or centering directions. We also emphasized that our algorithm has the flexibility to employ any heuristic technique to obtain better reductions in the potential function. In this sense, we consider the KMY and centering directions described in Chapter 2 as "back-up" directions to be used to guarantee reductions (and polynomial convergence) when heuristics fail. We use this approach in our implementation.

Let us now elaborate on the heuristics we use to generate search directions. In every iteration we calculate predictor-corrector-type search directions and try to decrease the potential function along these directions. The reason for our choosing the predictor-corrector directions in our implementation is their documented success in other interior-point implementations, see e.g., [Meh92,LMS92, XHY93,AGMX96,Zha96]. Note that when predictor-corrector directions are used in path-following algorithms the theoretical analysis requires the iterates to stay in a neighborhood of the central path to ensure polynomial convergence. Our potential-reduction algorithm alleviates such restrictions. If we find that the potential function can not be decreased along the search directions generated, we then resort to the KMY or centering directions that guarantee reductions in the potential function as discussed in Chapter 2.

The *predictor* direction that we generate is the unique solution to the following system:

$$
\begin{bmatrix}
0 & 0 & A & -b & \bar{b} & 0 & 0 & 0 \\
0 & 0 & -E & u & -\bar{u} & -I & 0 & 0 \\
-A^T & E^T & 0 & c & -\bar{c} & 0 & -I & 0 \\
b^T & -u^T & -c^T & 0 & \bar{z} & 0 & 0 & -1 \\
-\bar{b}^T & \bar{u}^T & \bar{c}^T & -\bar{z} & 0 & 0 & 0 & 0 \\
0 & V & 0 & 0 & 0 & W & 0 & 0 \\
0 & 0 & S & 0 & 0 & 0 & X & 0 \\
0 & 0 & 0 & \kappa & 0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
\Delta y^p \\ \Delta w^p \\ \Delta x^p \\ \Delta \tau^p \\ \Delta \theta^p \\ \Delta v^p \\ \Delta s^p \\ \Delta \kappa^p
\end{bmatrix}
=
\begin{bmatrix}
0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -VWe \\ -XSe \\ -\kappa
\end{bmatrix}
\qquad (4.6)
$$

where $V = \operatorname{diag}(v)$, etc. This is precisely the affine-scaling direction described in Chapter 2. As discussed in Section 2.3 this direction corresponds to the Newton step for estimating the solution of the nonlinear equality system defining the optimality conditions for $\mathcal{LP}$ (without the nonnegativity constraints). If the full Newton step can be taken without violating the nonnegativity constraints, we take that step and achieve primal and dual optimal solutions or a proof of primal or dual infeasibility for the "original" problem immediately (unless τ and κ both become zero with this full Newton step). If the full Newton step takes the variables out of the nonnegative orthant we combine it with a centering step. This centering direction is often called the *corrector* direction and is obtained by solving the following system:

$$\begin{bmatrix} 0 & 0 & A & -b & 0 & 0 & 0 \\ 0 & 0 & -E & u & -I & 0 & 0 \\ -A^T & E^T & 0 & c & 0 & -I & 0 \\ b^T & -u^T & -c^T & 0 & 0 & 0 & -1 \\ 0 & V & 0 & 0 & W & 0 & 0 \\ 0 & 0 & S & 0 & 0 & X & 0 \\ 0 & 0 & 0 & \kappa & 0 & 0 & \tau \end{bmatrix} \begin{bmatrix} \Delta y^c \\ \Delta w^c \\ \Delta x^c \\ \Delta \tau^c \\ \Delta v^c \\ \Delta s^c \\ \Delta \kappa^c \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \mu e - \Delta V^p \Delta W^p e \\ \mu e - \Delta X^p \Delta S^p e \\ \mu - \Delta \tau^p \Delta^p \kappa \end{bmatrix} \tag{4.7}$$

where $\Delta V^p = \text{diag}(\Delta v^p)$, etc., and μ is an appropriately chosen constant based on the current uncomplementarity and the uncomplementarity that would result from a damped Newton step (see [Meh92]). Note that, in the case of the corrector direction we eliminate the variable θ and therefore do not progress towards feasibility. This step is intended to improve the centrality of our iterates. After solving (4.7) we let $\Delta y = \Delta y^p + \Delta y^c$, etc., and then do the following ratio test:

$$\begin{aligned} \alpha_p &:= \frac{-1}{\min\{\min_i \frac{\Delta x_i}{x_i}, \min_i \frac{\Delta v_i}{v_i}, \Delta\tau, \frac{\Delta\kappa}{\kappa}, -1\}}, \\ \alpha_d &:= \frac{-1}{\min\{\min_i \frac{\Delta s_i}{s_i}, \min_i \frac{\Delta w_i}{w_i}, \Delta\tau, \frac{\Delta\kappa}{\kappa}, -1\}}. \end{aligned}$$

Next we calculate

$$\begin{aligned} \tau_p &:= 1 + \alpha_p \cdot \Delta\tau, \\ \tau_d &:= 1 + \alpha_d \cdot \Delta\tau, \end{aligned}$$

which we use to compute

$$\begin{aligned} \delta x &:= (x + \alpha_p \cdot \Delta x)/\tau_p - x, \\ \delta v &:= (v + \alpha_p \cdot \Delta v)/\tau_p - v, \end{aligned}$$

and

$$\begin{aligned}
\delta y &:= (y + \alpha_d \cdot \Delta y)/\tau_d - y, \\
\delta s &:= (s + \alpha_d \cdot \Delta s)/\tau_d - s, \\
\delta w &:= (w + \alpha_d \cdot \Delta w)/\tau_d - w, \\
\delta \kappa &:= (\kappa + \alpha_d \cdot \Delta \kappa)/\tau_d - \kappa.
\end{aligned}$$

Finally, we perform a simple line search along these directions in order to determine the step size that decreases the potential function ψ' most. To be more precise, we compare the potential function value for seven different values of the step length varying between 0.9995 and 0.5, and we choose the step length that produces the smallest potential. This is similar to the strategy employed by LIPSOL, however LIPSOL chooses the largest step size among the seven possible values that maintains the iterate in a neighborhood of the central path. This simple scheme proved efficient for most problems in the NETLIB set, however we expect to achieve improvements by updating this scheme through further experiments. We should note that if the generated directions happen to be ascent directions for the function but also there is an indication that the next iteration will generate a large potential reduction we allow the potential function to increase in a limited number of iterations on some problems. This option is seldom used and proved to be useful for the test problems. After the line search we update the variables and start the next iteration.

5.2.3 Dense Artificial Columns

The linear systems that we solve to determine the predictor and the corrector directions described in the previous section are larger than the usual linear systems solved in every iteration of interior-point algorithms. Specifically, the linear system for the predictor direction has three additional rows and columns, and the system for the corrector direction has two additional rows and columns when compared with what other interior-point algorithms do. Considering that the size of the matrix is often of the order of thousands or tens-of-thousands, increasing the dimension by two or three may seem insignificant. However, the columns that we add (those corresponding to the artificial variables τ, θ) are very dense. Consequently, if these columns are not treated separately they create excessive fill-in during the Cholesky factorization of the matrices of the linear system that is solved in every iteration of our algorithm, increasing the computation time tremendously.

Fortunately, there is a way of handling these dense columns cheaply. In what follows, we will explain how to calculate the predictor and corrector directions solving systems of linear equations that are exactly the same size as the ones solved by standard interior-point algorithms, however with two different right-hand sides. Computationally the most expensive part of the solution of a linear system is the factorization of the constraint matrix. Once this is done the solution of the system with a second right-hand side requires only one extra back-solve which can be done in $\mathcal{O}(n^2)$ arithmetical operations. The strategy we follow was suggested by Ye, Todd, and Mizuno [YTM94] and simplified by Xu, Hung, and Ye [XHY93].

Let us now explain this strategy. When we solve (4.6) we first determine $\Delta\theta^p$ explicitly without solving the system. The next proposition demonstrates how this can be done:

Proposition 5.2.1 *Let (4.6′) be the linear equality system obtained by eliminating the superscripts on the variables and replacing the right-hand side vector of (4.6)* *with* $\left[0,0,0,0,0,r_{vw}^T,r_{xs}^T,r_{\tau\kappa}\right]^T$ *where* $r_{vw}=\alpha(\gamma\mu e-VWe)$, $r_{xs}=\alpha(\gamma\mu e-XSe)$, $r_{\tau\kappa}=\alpha(\gamma\mu-\kappa)$, *with* $\alpha>0$, $0\le\gamma\le 1$, *and* $\mu=\dfrac{x^Ts+v^Tw+\kappa}{n+n_u+1}$. *Then,* $\Delta\theta$ *that solves (4.6′) is given by the following formula:*

$$\Delta\theta \;=\; -\alpha(1-\gamma). \tag{4.8}$$

Proof:

From the skew-symmetry of the first five set of equations in (4.6′) we have the following identity:

$$\Delta x^T\Delta s+\Delta v^T\Delta w+\Delta\tau\Delta\kappa \;=\; 0. \tag{4.9}$$

Also, as in Proposition 2.2.1(i),

$$\begin{aligned}(x+\Delta x)^T(s+\Delta s)+(v+\Delta v)^T(w+\Delta w)+(1+\Delta\tau)(\kappa+\Delta\kappa)&\\ = \;(1+\Delta\theta)(x^Ts+v^Tw+\tau\kappa).&\end{aligned} \tag{4.10}$$

Simplifying (4.10) and using (4.9) we obtain

$$x^T\Delta s+s^T\Delta x+v^T\Delta w+w^T\Delta v+\Delta\kappa+\kappa\Delta\tau=\Delta\theta(x^Ts+v^Tw+\tau\kappa). \tag{4.11}$$

Multiplying the sixth, seventh, and eighth set of equations in (4.6′) by e^T from the left and adding, we also get

$$x^T\Delta s+s^T\Delta x+v^T\Delta w+w^T\Delta v+\Delta\kappa+\kappa\Delta\tau=\alpha(\gamma-1)(x^Ts+v^Tw+\tau\kappa). \tag{4.12}$$

Finally, comparing (4.11) and (4.12) we obtain the desired result. ◻

Proposition 5.2.1 indicates that the $\Delta\theta^p$ that solves (4.6) is just -1. We presented the proposition in the above form to demonstrate its applicability to other choices of the right-hand side vector. We note that Proposition 5.2.1 is essentially the same as Lemma 2 of [XHY93]. However, our proof is shorter and applies to a more general formulation.

Having calculated $\Delta\theta^p$ we can treat the systems (4.6) and (4.7) in a unified manner. For this purpose let $\Delta\theta^c := 0$. We will omit the superscripts from the variables in this unified discussion. We also omit the fourth block of equations from both (4.6) and (4.7) (they will be used later to determine $\Delta\tau$) and the fifth block of equations from (4.6) (it has served its purpose in determining $\Delta\theta^p$). Also, let us unify the last three blocks of the right-hand side of the equations (4.6) and (4.7) with the notation $\left[r_{vw}^T, r_{xs}^T, r_{\tau\kappa}\right]^T$. Now, we eliminate Δv and Δs from the the second and third block of equations using the equations that correspond to complementarity relations. Then, we decompose the first three blocks of the resulting system as follows:

$$\begin{bmatrix} 0 & 0 & A \\ 0 & W^{-1}V & -E \\ -A^T & E^T & X^{-1}S \end{bmatrix} \begin{bmatrix} \Delta y \\ \Delta w \\ \Delta x \end{bmatrix} = \begin{bmatrix} 0 \\ W^{-1}r_{vw} \\ X^{-1}r_{xs} \end{bmatrix} - \begin{bmatrix} -b \\ u \\ c \end{bmatrix} \Delta\tau - \begin{bmatrix} \bar{b} \\ -\bar{u} \\ -\bar{c} \end{bmatrix} \Delta\theta$$

Recall that $\Delta\theta$ is -1 for the case of predictor direction and 0 for the case of corrector direction. Next, we substitute for Δw from the second set of equations above to get:

$$\begin{bmatrix} 0 & A \\ -A^T & D^{-2} \end{bmatrix} \begin{bmatrix} \Delta y \\ \Delta x \end{bmatrix} = \zeta_1 + \zeta_2 \Delta\tau, \tag{4.13}$$

where

$$\begin{aligned} D^{-2} &= X^{-1}S + E^T V^{-1} W E, \\ \zeta_1 &= \begin{bmatrix} -\bar{b}\Delta\theta \\ X^{-1}r_{xs} + \bar{c}\Delta\theta - E^T V^{-1}(r_{vw} + W\bar{u}\Delta\theta) \end{bmatrix}, \end{aligned}$$

$$\zeta_2 = \begin{bmatrix} b \\ -c + E^T V^{-1} W u \end{bmatrix}.$$

Note that D is a diagonal matrix. Equation (4.13) can be further simplified by eliminating Δx yielding the so-called *Schur complement* equation:

$$(AD^2A^T)\Delta y = \xi_1 + \xi_2 \Delta\tau, \tag{4.14}$$

where

$$\xi_1 = -\bar{b}\Delta\theta - AD^2\bar{c}\Delta\theta - AD^2X^{-1}r_{xs} + AD^2E^TV^{-1}r_{vw} + AD^2E^TV^{-1}W\bar{u}\Delta\theta,$$

and

$$\xi_2 = b + AD^2c - AD^2E^TV^{-1}Wu.$$

The next step is to solve for λ_1 and λ_2 that satisfy

$$\begin{aligned} (AD^2A^T)\lambda_1 &= \xi_1, \\ (AD^2A^T)\lambda_2 &= \xi_2, \end{aligned}$$

and then use λ_1 and λ_2 to express Δy, Δx, Δs, Δv, and Δw in terms of $\Delta\tau$. Next, using the fourth block equation (which consists of one row) we can solve for $\Delta\tau$. Finally, the remaining unknown vectors can be obtained by direct substitution of $\Delta\tau$ into corresponding equations.

So as we argued above, the additional work in calculating the search directions for the homogeneous self-dual formulation consists of computing the vectors ξ_1 and ξ_2, and then solving the Schur complement equation with a second right-hand side. The computational effort involved in the solution of an already factorized system with a second right-hand side is negligible compared to the work involved in factorizing the Schur complement matrix and in our case has the benefit of producing the auxiliary variable κ that we use to check for infeasibilities.

5.2.4 Termination

We terminate the algorithm when we find either an approximate optimal solution or an approximate proof of infeasibility. To be more specific, we terminate if

$$\max\{\frac{\|Ax-b\|}{1+\|x\|}, \frac{\|u-x_u-v\|}{1+\|v\|}, \frac{\|A^Ty-E^Tw+s-c\|}{1+\|s\|}, \frac{\|c^Tx-b^Ty+u^Tw}{1+\|b^Ty-u^Tw\|}\} \leq \varepsilon_1, \tag{4.15}$$

or if

$$b^Ty-u^Tw>0 \text{ and } \frac{\|A^Ty-E^Tw+s\|}{b^Ty-u^Tw} < \varepsilon_2, \tag{4.16}$$

or if

$$c^Tx<0 \text{ and } \frac{\|Ax\|}{-c^Tx} < \varepsilon_2. \tag{4.17}$$

We also use infeasibility tests similar to (4.16) and (4.17) by checking the direction vectors instead of the iterates themselves. These additional tests proved to be extremely effective on certain problems.

We report our results obtained for $\varepsilon_1 = 10^{-8}$, and $\varepsilon_2 = 10^{-8}$. When (4.15) is satisfied we obtain an approximately feasible and optimal primal and dual pair of solutions. We note that LIPSOL's original termination scheme uses a test similar to but slightly different from (4.15). For purposes of comparison, the iteration counts we report for LIPSOL are obtained using the stopping criterion (4.15). As in LIPSOL, when the problem data is badly scaled (i.e., if the ratio of the smallest and the largest absolute value of the nonzero entries in the constraint matrix A is smaller than a threshold value, say, 10^{-4}), we scale the data so that the entries of the matrix A are between -1 and 1.

If the algorithm terminates after satisfying (4.16), we get an approximate proof of primal infeasibility. Recalling Lemma 2.4.2 and the definition of α_x, (4.16) indicates that $\alpha_x > \frac{1}{\varepsilon_2}$ meaning that any primal feasible solution must have large norm. Similarly, (4.17) indicates that any dual feasible solution must have large norm.

In contrast, LIPSOL uses heuristic schemes to detect infeasibility. The main drawback of this approach is that, when LIPSOL stops with a possible indication of infeasibility, it does not have a(n) (approximate) proof of this fact and therefore is not very reliable. Furthermore, there is no guarantee that LIPSOL will detect the infeasibility of infeasible problems.

5.3 The Implementation Environment

The implementation of our algorithm has been done under the MATLAB environment. This choice was motivated by the existence of a successful research code implemented in the MATLAB environment by Y. Zhang [Zha95b] . This code is called LIPSOL and is an implementation of a primal-dual infeasible-interior-point predictor-corrector algorithm for linear programming. Most of LIPSOL is written in the high-level language of MATLAB, but it also uses fast external FORTRAN and C subroutines to perform the computationally most expensive tasks. As a result it is remarkably efficient. Zhang reports his computational experience with LIPSOL in [Zha96].

We should point out that the MATLAB high-level programming language allows matrix and vector manipulation by one-line commands and therefore makes code development, debugging, and modifications significantly easier than in lower-level languages such as C. As a trade-off, it carries a computational overhead making it slower than codes written in such languages. This is why LIPSOL uses non-MATLAB subroutines for the computationally expensive tasks including the minimum degree ordering of the columns of the constraint matrix and the sparse Cholesky factorization of the matrix that appears in the linear systems solved in each iteration.

We used LIPSOL as our starting point which made our implementation task much easier than what it would be otherwise. We changed most of the MATLAB subroutines used by LIPSOL to incorporate the homogeneous self-dual formulation. In addition, LIPSOL uses a path-following scheme and forces the iterates to stay in a relatively large neighborhood of the central path and chooses step lengths with this consideration. In contrast, our step lengths selection scheme tries to maximize the reduction in our potential function. As a result of these modifications our implementation is significantly different from LIPSOL and any shortcomings of our implementation should be attributed to our approach, not to LIPSOL.

5.4 Results

The test problems used in our computational experiments are the NETLIB set of feasible and infeasible linear programming problems. The sizes of the problems in this set range from very small (9 rows, 8 columns, 16 nonzeros) to very large (16676 rows, 15695 columns, 74004 nonzeros). The NETLIB set is distributed electronically [Gay85] and is also available from the web site `http://www.netlib.org/`. We performed our experiments on a Sun Ultra Sparc 1, Model 140 with 133 MHz CPU, 64 megabytes memory and 256k cache.

Our implementation of the infeasible-interior-point potential-reduction algorithm, which we will denote by IPR in the tables, proved to be fast and robust on both the feasible and infeasible sets of problems. Tables 5.1 to 5.5 list the number of iterations taken by IPR in comparison to the iterations taken by LIPSOL (which is abbreviated to LIP in the tables) to solve feasible linear programming problems in the NETLIB set. In addition to the number of rows, columns, and nonzeros in each problem, we also display whether the problems have upper bounds or ranges on some of the variables in the column labeled 'BR'. On 48 of the 95 problems IPR required fewer iterations. On 34 of these problems LIPSOL took fewer iterations and the two implementations needed the same number of iterations on 13 problems. In total, IPR took only 73 more iterations than LIPSOL, or about 0.77 more iterations per problem. The larger problems including `dfl001, greenbea, greenbeb,` and `pilot87` are the main contributors to this difference. For the hard problem `dfl001`, IPR generated an iterate satisfying (4.15) with $\varepsilon_1 = 2 \cdot 10^{-8}$ in iteration 88 and then started to stall because of ill-conditioning. Therefore we report this number for the iteration count of `dfl001`.

In terms of the computational times, LIPSOL was 40 to 75% faster than IPR. As we discussed in Section 5.2.3 IPR needs to perform two extra back-solves in each iteration. However, the most important factor for the discrepency in the computational times is the expensive potential function calculations. Faster or approximate calculation of the potential function would certainly close or narrow the gap between the computational times of the two implementations and we see this as a future area of research. Also we should add that LIPSOL is geared to solve feasible problems to optimality while IPR is also concerned about proving infeasibility of the infeasible problems, and consequently LIPSOL is expected to perform better on feasible problems than IPR, even after IPR is improved to reduce its computational work per iteration.

Tables 5.6 and 5.7 list the number of iterations required by LIPSOL and IPR to detect the infeasibility of infeasible problems in the NETLIB set. Recall that $\varepsilon_2 = 10^{-8}$ is used by the stopping criterions (4.16) and (4.17) of IPR and this choice of ε_2 corresponds to requiring a strong approximate proof of infeasibility to terminate the algorithm. For some of the easier infeasible problems the ratio $\dfrac{\|A^T y - E^T w + s\|}{b^T y - u^T w}$ was smaller than 10^{-16} in the final iteration. This is a convincing proof of infeasibility given that we are using inexact arithmetic with machine epsilon around $2 \cdot 10^{-16}$. For two of the problems, `gran` and `woodinfe`, infeasibility was detected in the preprocessing phase. The words `pre` appearing in Tables 5.6 and 5.7 signifies this fact.

We stress that LIPSOL detects infeasibility using heuristic tests. Namely, LIPSOL suggests infeasibility if it detects a blow-up in the size of the iterates or stalling. However, these are not proofs of infeasibility of the problems and there is no guarantee that these heuristic schemes will work for every infeasible problem. In this sense, the iteration counts for the two algorithms are not really comparable since totally different termination schemes are used for infeasibility detection. Yet, IPR often takes fewer iterations than LIPSOL even though it looks for a strong (but approximate) proof of infeasibility. In some problems LIPSOL has lower iteration counts simply because it cannot progress after a few iterations. For example, LIPSOL stops after 6 to 12 iterations for the infeasible problems `klein1, klein2`, and `klein3` with no progress and unable to suggest the infeasibility of these problems. Our implementation takes a few more iterations for these problems but produces a proof of their infeasibility. The largest one of these three problems, `klein3`, proved to be rather difficult for IPR, too. For the first 18 iterations, the largest lower bound the algorithm can suggest for the norm of primal feasible solutions is only around 300, but then the 19th iteration generates an iterate pulling this lower bound to 10^{14}, a clear indication of infeasibility of the problem. The disproportionately large computational time on this problem is caused by ill-conditioning of the linear systems solved in each iteration.

IPR had difficulty in generating approximate proofs of infeasibility for two of the problems in the set. The algorithm generated inconclusive iterates for the large infeasible problem `ceria3d` for 100 iterations at which point it was stopped. Also, for the infeasible problem `gosh` the stopping criterion (4.16) was satisfied for $\varepsilon_2 = 10^{-4}$ after 31 iterations, however the algorithm made little progress after this point. We attribute these failures to the preliminary nature of our implementation and expect to improve the algorithm to process these problems as a result of future research. The data file of the infeasible problem `greenbeai` we obtained was contaminated and we failed to test either method on this problem.

To summarize, the computational tests on the algorithm developed in Chapter 2 shows that the algorithm may be a viable alternative to others being used in practice. It offers fast and accurate solutions for feasible problems, and reliable detection of infeasibility for infeasible problems.

Table 5.1: Performance on NETLIB feasible problems (25fv47-d2q06c)

Problem Data					Iterations			CPU time
Name	Rows	Cols	Non0s	BR	IPR	LIP	Diff	sec.
25fv47	822	1571	11127		28	25	+3	22.27
80bau3b	2263	9799	29063	B	43	37	+6	137.18
adlittle	57	97	465		11	12	−1	0.77
afiro	28	32	88		7	7	–	0.37
agg	489	163	2541		28	18	+10	9.92
agg2	517	302	4515		15	16	−1	6.94
agg3	517	302	4531		15	16	−1	6.92
bandm	306	472	2659		14	17	−3	2.51
beaconfd	174	262	3476		10	13	−3	1.85
blend	75	83	521		11	12	−1	0.86
bnl1	644	1175	6129		39	26	+13	16.92
bnl2	2325	3489	16124		28	32	−4	49.31
boeing1	351	384	3865	BR	19	21	−2	8.09
boeing2	167	143	1339	BR	14	18	−4	2.54
bore3d	234	315	1525	B	20	17	+3	4.11
brandy	221	249	2150		15	16	−1	2.20
capri	272	353	1786	B	17	20	−3	4.22
cycle	1904	2857	21322	B	29	28	+1	58.09
czprob	930	3523	14173	B	35	31	+4	29.35
d2q06c	2172	5167	35674		37	31	+6	123.87

Table 5.2: Performance on NETLIB feasible problems (d6cube-grow22)

Problem Data					Iterations			CPU time
Name	Rows	Cols	Non0s	BR	IPR	LIP	Diff	sec.
d6cube	416	6184	43888	B	19	27	−8	37.25
degen2	445	534	4449		13	14	−1	4.89
degen3	1504	1818	26230		15	21	−6	43.84
dfl001	6072	12230	41873	B	88†	55	+33	6235.26
e226	224	282	2767		17	22	−5	3.22
etamacro	401	688	2489	B	25	28	−3	6.98
fffff800	525	854	6235		31	25	+6	15.65
finnis	498	614	2714	B	23	27	−4	9.01
fit1d	25	1026	14430	B	25	18	+7	20.99
fit1p	628	1677	10894	B	13	15	−2	46.26
fit2d	26	10500	138018	B	21	22	−1	165.93
fit2p	3001	13525	60784	B	18	20	−2	265.12
forplan	162	421	4916	BR	21	20	+1	5.81
ganges	1310	1681	7021	B	16	18	−2	11.63
gfrd-pnc	617	1092	3467	B	16	19	−3	5.85
greenbea	2393	5405	31499	B	57	45	+12	135.87
greenbeb	2393	5405	31499	B	43	36	+7	94.71
grow15	301	645	5665	B	16	16	—	8.21
grow22	441	946	8318	B	16	17	−1	11.82

† For the problem dfl001 (4.15) is satisfied with $\varepsilon_1 = 2 \cdot 10^{-8}$ after 88 iterations.

Table 5.3: Performance on NETLIB feasible problems (grow7-sc50b)

Problem Data					Iterations			CPU time
Name	Rows	Cols	Non0s	BR	IPR	LIP	Diff	sec.
grow7	141	301	2633	B	15	15	—	3.81
israel	175	142	2358		19	22	−3	6.18
kb2	44	41	291	B	15	14	+1	1.30
lotfi	154	308	1086		21	18	+3	2.41
maros	847	1443	10006	B	31	32	−1	22.12
maros-r7	3137	9408	151120		14	14	—	511.90
modszk1	688	1620	4158		27	24	+3	12.51
nesm	663	2923	13988	BR	30	32	−2	41.25
perold	626	1376	6026	B	33	33	—	20.32
pilot	1442	3652	43220	B	33	34	−1	150.10
pilot.ja	941	1988	14706	B	33	30	+3	39.39
pilot.we	723	2789	9218	B	32	36	−4	24.21
pilot4	411	1000	5145	B	34	28	+6	18.68
pilot87	2031	4883	73804	B	55	38	+17	680.46
pilotnov	976	2172	13129	B	19	20	−1	21.13
recipe	92	180	752	B	8	9	−1	1.04
sc105	106	103	281		11	10	+1	0.77
sc205	206	203	552		12	11	+1	1.18
sc50a	51	48	131		10	9	+1	0.57
sc50b	51	48	119		8	8	—	0.47

Table 5.4: Performance on NETLIB feasible problems (scagr25-ship08l)

Problem Data					Iterations			CPU time
Name	Rows	Cols	Non0s	BR	IPR	LIP	Diff	sec.
scagr25	472	500	2029		17	16	+1	3.27
scagr7	130	140	553		13	12	+1	1.09
scfxm1	331	457	2612		18	18	—	3.65
scfxm2	661	914	5229		21	19	+2	7.60
scfxm3	991	1371	7846		21	20	+1	11.07
scorpion	389	358	1708		13	14	−1	1.97
scrs8	491	1169	4029		25	24	+1	6.63
scsd1	78	760	3148		9	10	−1	1.56
scsd6	148	1350	5666		11	12	−1	3.03
scsd8	398	2750	11334		10	11	−1	5.32
sctap1	301	480	2052		15	17	−2	2.48
sctap2	1091	1880	8124		11	18	−7	6.35
sctap3	1481	2480	10734		13	18	−5	9.76
seba	516	1028	4874	BR	25	20	+5	21.83
share1b	118	225	1182		25	20	+5	2.53
share2b	97	79	730		12	12	—	1.02
shell	537	1775	4900	B	16	19	−3	6.26
ship04l	403	2118	8450		12	12	—	5.26
ship04s	403	1458	5810		12	13	−1	3.60
ship08l	779	4283	17085		16	15	+1	13.48

Table 5.5: Performance on NETLIB feasible problems (ship08s-woodw)

Problem Data					Iterations			CPU time
Name	Rows	Cols	Non0s	BR	IPR	LIP	Diff	sec.
ship08s	779	2387	9501		13	14	−1	5.70
ship12l	1152	5427	21597		28	15	+13	27.74
ship12s	1152	2763	10941		16	16	—	7.26
sierra	1228	2036	9252	B	15	16	−1	19.44
stair	357	467	3857	B	15	15	—	5.31
standata	360	1075	3038	B	14	16	−2	4.53
standgub	362	1184	3147	B	14	16	−2	4.79
standmps	468	1075	3686	B	17	24	−7	6.06
stocfor1	118	111	474		16	17	−1	1.25
stocfor2	2158	2031	9492		32	22	+10	26.29
stocfor3	16676	15695	74004		42	33	+9	268.21
truss	1001	8806	36642		18	18	—	32.87
tuff	334	587	4523	B	17	23	−6	5.73
vtp.base	199	203	914	B	25	25	—	4.38
wood1p	245	2594	70216		16	20	−4	40.80
woodw	1099	8405	37478		24	27	−3	46.14
TOTAL					2007	1934	+73	

Table 5.6: Performance on NETLIB infeasible problems (bgdbg1-gosh)

Problem Data					Iterations		CPU time
Name	Rows	Cols	Non0s	BR	LIP	IPR	sec.
bgdbg1	349	407	1485	B	8	4	1.32
bgetam	401	688	2489	B	8	4	1.83
bgindy	2672	10116	75019		8	10	69.89
bgprtr	21	34	90		11	4	0.33
box1	232	261	912	B	8	17	2.22
ceria3d	3577	824	17604	B	16	†	†
chemcom	289	720	2190	B	8	4	1.65
cplex1	3006	3221	10664	B	9	18	39.50
cplex2	225	221	1059	B	78	73	17.79
ex72a	198	215	682	B	7	16	1.86
ex73a	194	211	668	B	8	11	1.27
forest6	67	95	270	B	13	4	0.57
galenet	9	8	16	B	7	2	0.35
gosh	3793	10733	97257	B	18	31‡	228.00

† IPR failed to generate a proof of infeasibility for `ceria3d` in 100 iterations.

‡ (4.16) is satisfied with $\varepsilon_2 = 10^{-4}$ after 31 iterations.

Table 5.7: Performance on NETLIB infeasible problems (gran-woodinfe)

Problem Data					Iterations		CPU time
Name	Rows	Cols	Non0s	BR	LIP	IPR	sec.
gran	2569	2520	20151	B	pre	pre	0.00
itest2	10	4	17		7	2	0.28
itest6	12	8	23		5	4	0.40
klein1	55	54	696		12	19	2.06
klein2	478	54	4585		9	21	42.44
klein3	995	88	12107		6	19	8769.75
mondou2	313	604	1623	B	5	3	1.21
pang	362	460	2666	B	27	33	10.64
pilot4i	411	1000	5145	B	21	17	10.37
qual	324	464	1714	B	58	49	14.77
reactor	319	637	2995	B	13	17	7.54
refinery	324	464	1694	B	17	12	3.73
vol1	324	464	1714	B	41	34	9.82
woodinfe	36	89	209	B	pre	pre	0.00