A Primal Truncated Newton Algorithm with Application to Large-Scale Nonlinear Network Optimization[1]

Ron Dembo
Yale University

YALEU/DCS/TR304

Presented at the NETFLOW '83 Conference, Pisa, Italy
March 1983

Yale School of Organization and Management Working Paper
Series B #72

(Draft Copy)

[1] This research supported in part by DOT Grant CT-06-0011 and by NSF Grant ECS-8119513.

Abstract

We describe a new, convergent, primal-feasible algorithm for linearly constrained optimization. It is capable of rapid asymptotic behavior and has relatively low storage requirements. Its application to large-scale nonlinear network optimization is discussed and computational results on problems of over 2,000 variables and 1,000 constraints are presented. Indications are that it could prove to be significantly better than known methods for this class of problems.

Keywords: Nonlinear Optimization, Nonlinear Network Optimization, Primal Methods, Truncated Newton Methods.

1. Introduction

The canonical linearly-constrained nonlinear programming problem (LCNLP) has the form:

$$\textbf{LCNLP} \quad \text{Minimize} \quad f(x) \tag{1}$$

$$\text{subject to} \quad Ax = b \tag{2}$$

$$l \leq x \leq u \tag{3}$$

where $f : R^n \rightarrow R$, A is a given matrix of dimension $m \times n$ and $b \in R^m$ and $u, l \in R^n$ are given vectors.

A very rich class of applications that may be modeled as problems of the form LCNLP arises in computing economic equilibria in spatially separated markets [5,6], the study of telecommunications networks, the estimation of input-output matrices [35, 36], water distribution [7] and hydroelectric power management [8, 9, 10], resistive electrical network analysis [11] and the analysis of urban traffic networks [12, 13]. All these models have one thing in common, that is, the constraints $Ax = b$ are flow conservation constraints on some network. We refer to such problems as **nonlinear network optimisation problems.**

In practice, these problems tend to be large with many thousands of arcs (variables) and nodes (equality constraints). They are certainly an order of magnitude larger than problems that can be handled routinely by off-the-shelf, state-of-the-art software for linearly-constrained nonlinear optimization [2, 3]. Furthermore, even in cases where off-the-shelf software may be used, the efficiencies that can be gained by working with algorithms and codes that are specialized to such problems may make the difference between software that is practical for use by managers/engineers and software that is useful for research purposes only. Typically, one can expect something between one and two orders-of-magnitude improvement when using the network specializations of algorithms described in this paper. Furthermore, for a given amount of storage, specialized codes are capable of handling much larger problems.

Much of the motivation for this work comes from the exceptional success story of linear network optimization where specialized algorithms and software permit the practical solution of extremely large problems (millions of variables, see [1] for example). It is also not unusual to obtain solution times that are one hundredth of the time general-purpose LP software would take to solve the same problem. There is, however, a significant difference when attempting to emulate the linear network success story. Whereas large-scale LP software was commercially

available and was well studied and understood at the time specialized linear network codes were first developed (starting in the early 1970's), this is not the case for nonlinear programming (NLP). Both the theory and practical implementations are rare for large-scale NLP. It is not even the case that one knows which algorithm to specialize. Unlike in LP there are many possibilities, each of which appears to suit a particular combination of problem attributes.

At the time this research first began, in the late 1970's, many theoretical issues pertaining to the solution of LCNLP remained largely unanswered. What should be done in the presence of degeneracy (a perennial problem in networks)? What could be done to achieve rapid convergence when storage was limited? Would traditional active-set strategies prove to be effective for treating very large problems? What data structures were required for efficient implementations? Could practical implementations also be shown to fit within a sound theoretical framework?

The study of algorithms for large-scale nonlinear network optimization has proved to be a convenient vehicle for generating new, appropriate theoretical results for devising practical algorithms for LCNLP and for analyzing their convergence properties. This is primarily because of the ability to experiment with a limited budget on very large nonlinear optimization problems. A number of important breakthroughs have resulted:

(a) the **maximal basis** approach to handling degeneracy (in Dembo and Klincewicz [14, 16]);

(b) the Inexact Newton rate-of-convergence characterization (in Dembo, Eisenstat and Steihaug [15]);

(c) the Truncated-Newton algorithm (in Dembo and Steihaug [17]) which describes a cost- and storage-effective way to achieve gradient-related descent directions with rapid convergence characteristics;

(d) the global convergence framework and analysis (Dembo and Sahi [18]), in which a practical convergent framework for large-scale optimization is developed.

In all of the above results there is a common theme, which is an outcome of the discipline imposed by having to deal with large problems. They all present theoretical results that are **expressed in terms of measurable quantities and assumptions that can be verified.** It is my firm belief that this should be a major goal to strive for in computational mathematics. All too often one sees computational theory that purports to be useful, yet is expressed in terms of quantities that are not measurable or based on assumptions that cannot be verified.

This paper synthesizes the ideas presented in (a), (b), (c) and (d) above into a particular primal feasible algorithm for large problems of the form LCNLP. The PTN algorithm described in Section 2 is a theoretically sound, (convergent) algorithm that is able to trade off overhead per

iteration with asymptotic convergence rate. It has mechanisms for rapidly identifying an optimal set of active constraints, which is essential for large problems. It also can cope efficiently with the massive degeneracy characteristic of network optimization problems.

Section 2 begins with a presentation of the PTN algorithm. To place it in context and to highlight possible modifications to the algorithm that will preserve its theoretical properties, the algorithm is described in terms of the Dembo-Sahi framework [18]. Section 3 describes the data structures required to implement PTN efficiently and Section 4 outlines computational experiments with the NLPNET code [4], an implementation of PTN for nonlinear network optimization.

In some sense the order of presentation in this paper is counter-chronological. The PTN algorithm was coded and operating in the NLPNET system long before the Dembo-Sahi framework and global convergence analysis was completed. In fact, it was experiments with NLPNET that motivated the analysis in [18] and not *vice versa*.

2. Overview of the Primal Truncated-Newton (PTN) Algorithm

The PTN algorithm is a feasible direction method for LCNLP that fits into the convergent framework as outlined in Dembo and Sahi [18]. Their framework involves two types of feasible directions:

1. **restricted** directions, which are defined as feasible descent directions restricted to lie in a subspace containing the current active set (the set of constraints that currently hold as equalities); and
2. **relaxing** directions, which are defined as feasible descent directions along which one or more constraints may be relaxed.

For the restricted directions, PTN uses a primal feasible truncated-Newton direction from whence the name "Primal Truncated Newton" method is derived. Details are given in subsection 2.2 .

For relaxing directions a (scaled) projected gradient direction is used. It is not practical to compute the projection of the gradient onto a general polyhedral set. However, it is relatively easy if projection is done on a particular **restriction** of the problem. Details are given in subsection 2.2 .

In order to present the Dembo-Sahi framework we need to introduce some terminology.

Definition 2.1 (Acceptable points)

Let p be a descent direction computed at a feasible point, x. We refer to a point $x^+ = x + \alpha p$ as **acceptable** *if x^+ is feasible and the step αp satisfies* either

(a) both Goldstein-Armijo conditions

$$\textbf{(GA1)}: \quad f(x^+) \leq f(x) + \gamma \alpha g(x)^\top p \; ; \quad \gamma \in (0, 1)$$

$$\textbf{(GA2)}: \quad g(x^+)^\top p(\alpha) \geq \beta \alpha g(x)^\top p \; ; \quad \beta \in (\gamma, 1)$$

or

(b) $\alpha = \bar{\alpha}$ and x^+ satisfies GA1 only; where $\bar{\alpha}$ is the maximum feasible steplength along p.

Remark 2.1

For the purposes of proving convergence, GA1 and GA2 may be replaced by a backtracking Armijo-type linesearch procedure [19] or alternatively GA2 may be replaced by any of the standard conditions (see Fletcher [20], for example) that bound the stepsize away from zero at points that are not optimal.

The Dembo-Sahi framework is then:

Algorithmic Framework

START with x_0 feasible

(Major Iteration; Index = k)

IF optimal at x_k **THEN** exit.

(Minor Iteration)

ELSE compute a **relaxing** direction, p_k

and an **acceptable point** $x_k^+ = x_k + \alpha_k p_k$,

set $y \leftarrow x_k^+$;

WHILE a **constraint relaxation condition** is not satisfied at y,

compute a **restricted** direction p

and an **acceptable point** $y^+ = y + \alpha p$,

set $y = y^+$ and **repeat.**

ELSE $k \leftarrow k + 1$

$x_k = y$

start a new major iteration.

In order to completely specify an algorithm within this framework it is necessary to describe precisely

1. how relaxing and restricted directions are to be computed, and
2. the rule for terminating a minor iteration (i.e., the constraint relaxation condition).

This is done in subsections 2.2, 2.3 and 2.4 below.

Convergence may then be established simply by verifying that the search directions, steplengths and constraint relaxation conditions satisfy certain properties [18], which is done in subsection 2.5.

2.1 Computing feasible descent directions for LCNLP

In the discussion below it is assumed that a feasible point, $\tilde{x}$, is available or has been calculated by solving a linear programming problem (Phase I). Thus, given $\tilde{x}$, computing a feasible descent direction, p, amounts to finding a vector p satisfying:

$$g(\tilde{x})^T p < 0 \tag{4}$$

$$Ap = 0 \tag{5}$$

$$l - \tilde{x} \leq p \leq u - \tilde{x} \tag{6}$$

Reduced gradient methods (sometimes referred to as variable-reduction methods [21]) provide a convenient method for generating feasible descent directions for LCNLP. The principal idea behind such methods is that by considering only directions **restricted** to the tangent plane $Ap = 0$, the general linearly-constrained problem becomes **locally equivalent**[1] to one with box constraints.

A convenient way of doing this[2] is to eliminate certain (basic) variables by expressing them in terms of other (nonbasic) variables using the constraints $Ax = b$.

---

[1]This terminology will be made precise later on in the presentation

[2]For other approaches see Chapter 2 of Gill and Murray [22].

Assume A is of full row rank and partition it as follows: $A = [\mathrm{B}\ \bar{\mathrm{B}}]$, where B is a square nonsingular basis matrix. Similarly, partition x, p, u and l. Then

$$x_{\mathrm{B}} = \mathrm{B}^{-1}(b - \bar{\mathrm{B}}\, x_{\bar{\mathrm{B}}}) \equiv \Phi(x_{\bar{\mathrm{B}}}) \tag{7}$$

where the x_{B} and $x_{\bar{\mathrm{B}}}$ are referred to as basic and out-of-basis variables respectively. The problem LCNLP is therefore equivalent to the reduced n - m dimensional problem:

$$\textbf{RCNLP} \quad \text{Minimize} \quad f^R(x_{\bar{\mathrm{B}}}) \equiv f(\Phi(x_{\bar{\mathrm{B}}}), x_{\bar{\mathrm{B}}}) \tag{8}$$

$$\text{subject to} \quad l_{\mathrm{B}} \leq \Phi(x_{\bar{\mathrm{B}}}) \leq u_{\mathrm{B}} \tag{9}$$

$$l_{\bar{\mathrm{B}}} \leq x_{\bar{\mathrm{B}}} \leq u_{\bar{\mathrm{B}}} \tag{10}$$

The function f^R is referred to as the reduced objective function, and its gradient $g^R(x_{\bar{\mathrm{B}}}) = -(\mathrm{B}^{-1}\,\bar{\mathrm{B}})^{\mathsf{T}} g_{\mathrm{B}} + g_{\bar{\mathrm{B}}}$ is called the **reduced gradient.**

At any given feasible point, if the problem is **nondegenerate**, there exists a basis such that the linear inequality constraints (9) introduced by this transformation hold with strict inequality, in which case the problem is **locally equivalent** to the box-constrained problem

$$\textbf{BNLP} \quad \text{Minimize} \quad f^R(x_{\bar{\mathrm{B}}}) \tag{11}$$

$$\text{subject to} \quad l_{\bar{\mathrm{B}}} \leq x_{\bar{\mathrm{B}}} \leq u_{\bar{\mathrm{B}}} \tag{12}$$

that is, RCNLP with constraints (9) ignored.

By **locally equivalent** we mean that **any direction computed at a feasible point that is feasible in BNLP will also be feasible for the original problem LCNLP.** Thus the simplified problem BNLP may be used to generate feasible descent directions for LCNLP.

Difficulties arise when degeneracy is present, which is typically the case in network optimization. In degenerate problems, some of the constraints (9) may be active in every possible basis and then there is no assurance that a direction that is feasible for **BNLP** will also be feasible in **LCNLP** (i.e., the two problems are no longer locally equivalent).

This difficulty may be circumvented by working with a particular **restriction of LCNLP.** The key idea is to affect the reduction of LCNLP to RCNLP using a **maximal basis**[3] in the

---

[3]A maximal basis is one that contains as many columns as possible corresponding to free variables. The concept of a maximal basis was first introduced by Dembo and Klincewicz [14, 16]. For details on how to compute and maintain a maximal basis we refer readers to [14, 16]. It suffices to say that such a basis is easy to compute and update.

transformation in (7). It can then be shown that a particular **restriction** of BNLP is **locally equivalent** to the corresponding restricted version of LCNLP. Some new terminology is needed to facilitate the description of the locally-equivalent problem.

Definition (Blocked Variables)

A variable that is **not basic** *and is currently at one of its bounds is said to be* **blocked** *if an arbitrarily small move off its bound will induce infeasibilities in some basic variables.*

Remark 2.2

In nondegenerate problems a maximal basis ensures that no variables are blocked.

Remark 2.3

To determine whether or not a variable is blocked, one simply generates a column of $B^{-1}(\bar{B})$ and then examines the signs of the nonzero elements. The basic variables affected by a change in a variable out of the basis are said to form a **circuit** with this variable. If the constraints $Ax = b$ are the conservation of flow constraints in a network, then the nonzero elements of $B^{-1}(\bar{B})$ are the cycles that nonbasic arcs form with arcs in the basis tree. The **locally equivalent** (restricted) reduced problem is then

$$\textbf{BRNLP} \quad \text{Minimize} \quad f^R(x_{\bar{B}}) \tag{13}$$

$$\text{subject to} \quad l_{\bar{B}} \leq x_{\bar{B}} \leq u_{\bar{B}} \tag{14}$$

$$(x_{\bar{B}})_j = (u_{\bar{B}})_j \text{ or } (l_{\bar{B}})_j \text{ if } (x_{\bar{B}})_j \text{ is blocked.} \tag{15}$$

To see that this is **locally equivalent** to the restriction of LCNLP given by (15), note that:

1. any free nonbasics will only induce changes in free basics (a property of the maximal basis, see Dembo and Klincewicz [14, 16]), and
2. the only nonbasics at a bound that are allowed to move are those that induce feasible changes in basic variables (by construction of BRNLP).

Thus any feasible direction computed for BRNLP will also be feasible for the restriction of LCNLP with the blocked variables held fixed. We are now in a position to describe the restricted and relaxing steps in PTN.

2.2 Computing a search direction for the relaxing step

The relaxing step plays an extremely important role in the PTN algorithm. Its primary function is to facilitate rapid identification of the active set. This is an essential property of any algorithm for large-scale optimization, since even if an algorithm has fast asymptotic behavior on the optimal active set, its overall performance will depend heavily on how quickly it is able to identify an optimal set of active constraints.

Thus for large problems it is crucial for the relaxing step to be capable of making radical changes to the active set. One way to do so economically is to move in the direction of a negative **projected gradient** $[-g^R]^+$ for the box constrained (reduced) problem, where the components of $[-g^R]^+$ are given by:

$$[-g_j^R]^+ \equiv \begin{cases} (-g^R)_j & \text{if } (l_{\bar{B}})_j < (x_{\bar{B}})_j < (u_{\bar{B}})_j \\ \max\{0, -g_j^R\} & \text{if } (x_{\bar{B}})_j = (l_{\bar{B}})_j \\ \min\{0, -g_j^R\} & \text{if } (x_{\bar{B}})_j = (u_{\bar{B}})_j \ . \end{cases} \tag{16}$$

Remark 2.4

$\|[g^R]^+\| = 0$ is a first-order necessary condition for optimality at a feasible point.

In degenerate problems $[-g^R]^+$ might not yield a feasible direction for LCNLP. However, a restriction of $[-g^R]^+$ to the space of blocked nonbasics will generate feasible directions. This is precisely the projection of the reduced gradient, $-g^R$, onto the constraint set of **BRNLP** and is given by:

$$(p_{\bar{B}})_j = \begin{cases} [-g_j^R]^+ & \text{if } (x_{\bar{B}})_j \text{ is not blocked} \\ 0 & \text{otherwise} \ . \end{cases} \tag{17}$$

The relaxing direction used by **PTN** is[4]

$$p = \begin{bmatrix} p_B \\ p_{\bar{B}} \end{bmatrix} = \begin{bmatrix} -B^{-1}(\bar{B}p_{\bar{B}}) \\ p_{\bar{B}} \end{bmatrix} \tag{18}$$

which satisfies (4), (5), and (6) (i.e., is primal feasible).

Since $\|p\| = 0$ in (18) is a first-order optimality condition only if no blocked variables are present, the following special case arises.

Consider the situation where $\|p_{\bar{B}}\| = 0$ and there are blocked variables with corresponding projected gradient elements that are nonzero (i.e., the current point is not optimal). This then resembles the situation in linear programming when a degenerate pivot is required. Just as in linear programming, using a pivoting rule such as the one described by Bland [24], after a finite number of pivots one will either prove optimality or $p_{\bar{B}}$ will have a nonzero component and a relaxing step may be taken.

Thus, if the current point is not optimal, the above procedure always produces a nonzero relaxing direction which is gradient-related (see [18]).

Alternatives to the relaxing direction used in PTN

The relaxing direction we use is cheap to compute and has the added benefit that it may result in a radical change in the active set. One could always use relaxing directions such as those used in MINOS [3], GRG2 [2], or CONOPT [24], that drop one constraint at a time.[5] We have ruled these out for large problems since the overhead per iteration is such that it might take an inordinate amount of time to identify an optimal active set.

The chief objection to using the negative projected gradient (or a scaled projected gradient for that matter) is that it would lead to slow convergence since it is a direction of steepest descent. This reasoning, however, overlooks the fact that relaxing steps are taken relatively rarely (see Tables 4.2 and 4.12) and (for our purposes) are used only to help identify an optimal active set quickly. The rate of convergence then depends primarily on the manner in which a restricted direction is computed.

---

[4]In the NLPNET implementation [4], the user has an option of using either $[-g^{FR}]^+$ in (17) or $[-Dg^{FR}]^+$ where D is a diagonal scaling matrix with elements equal to the diagonal of the reduced Hessian.

[5]MINOS does have a "multiple price option" which allows for more than one variable to move off a bound in a relaxing step. However, the design of the code explicitly attempts to keep the number of superbasics small and to move no more than a few variables off their bounds in a relaxing step (see [3], page 69).

There is an entire spectrum of possible algorithms that are theoretically convergent (see Dembo and Sahi [18]) and that range in the use of relaxing steps from using them exclusively (such as in Bertsekas [19]) to relatively infrequently as in PTN, depending on the amount of work that is done on restricted subproblems. Naturally then, the direction choice in a relaxing step will depend on how often (relative to restricting steps) one expects to relax constraints. In our case a relaxing step is relatively infrequent and so it probably does not make much difference as to what is used. So, one might as well choose the projected gradient direction (or a scaled projected gradient) as we have done. In cases where the relaxing step is relatively more frequent it is probably better to use the projected reduced-Newton direction described in Dembo and Tulowitzki [25], Bertsekas [26] or in Murtagh and Saunders [3].

2.3 Computing a search direction when taking restricted steps

In a restricted step, **all variables currently at their bounds must remain there** (their corresponding components in the search direction vector are zero). Murtagh and Saunders [3] have provided a framework and new terminology for describing this process in the context of reduced gradient methods.

Following [3], let $\bar{B}$ be partitioned into $\bar{B} = [S\ N]$ where the columns in S correspond to **superbasic variables**, x_S, and the columns in N correspond to **nonbasic variables**, x_N. The idea is that one computes a descent direction for the reduced problem in which $p_N = 0$, that is, the nonbasic variables are held fixed. This direction then induces a direction in the basic variables given by $p_B = -B^{-1}Sp_S$.

Murtagh and Saunders' framework is perfectly adequate for both degenerate and nondegenerate problems provided the partition is chosen as follows[6]:

1. the basis is **maximal** and

2. superbasics are strictly between bounds (free).

This follows since free superbasics affect only free basics and so, implicitly, all basic variables that are currently at their bound remain so. In this way one knows *a priori* that a descent direction, $p_{\bar{B}}^T = (p_S^T, 0)$, in the reduced problem **RCNLP** will induce a feasible restricted

---

[6]Murtagh and Saunders do not use this partition in their code MINOS [3] and hence, strictly speaking, they are never sure at the start of a minor iteration whether or not the direction given by $p_B = -B^{-1}Sp_S$, $p_N = 0$ lies on the correct manifold, since some components of p_B might be required to be zero in a reduced step.

direction, p in LCNLP, with all components of p corresponding to variables currently at a bound equal to zero. Thus, in the terminology introduced at the start of this section, **the restricted problems LCNLP and RCNLP are locally equivalent** provided the basis is chosen appropriately.

Define Z by

$$Z = \begin{bmatrix} -B^{-1}S \\ I \\ 0 \end{bmatrix} \quad \text{such that } AZ = 0 \tag{19}$$

where the columns of Z span the nullspace of A. The restricted search direction is then given by $p = Zp_S$ and the (reduced) gradient and Hessian of the restricted problem are then $(Z^T g)$ and $Z^T HZ$ respectively.

The reduced problem is **(locally) unconstrained** and therefore **any gradient-related descent direction on this unconstrained problem would suffice for convergence.** In PTN we have chosen to use the truncated-Newton search direction (see Dembo and Steihaug [17]) which gives us the name **Primal Truncated Newton (PTN)** algorithm. The computation of p_S is exactly the same as in the unconstrained case in which $(Z^T HZ)$ and $(Z^T g)$ play the roles of the Hessian and the gradient, respectively.

For PTN, p_S is a descent direction satisfying

$$(Z^T HZ)p_S = -(Z^T g) + r \tag{20}$$

$$\text{with} \quad \frac{\|r\|}{\|Z^T g\|} \leq \eta \tag{21}$$

where $\eta \equiv \max \{\epsilon_1, \min (\epsilon_2, \|Z^T g\|^q)\}$.

In NLPNET [4], $\epsilon_1 = \sqrt{\text{machine precision}}$ and ϵ_2, which must be less than 1 [17], has a default value of $\epsilon_2 = 0.01$. The value of $q \in (0, 1)$, which determines the asymptotic convergence rate, has a default value of 1 which corresponds to a quadratic rate [15], if the active set settles down.

The solution p_S to the above system is computed either using a conjugate gradient or conjugate residual method (for details see Dembo and Steihaug [17], page 194).

There are some drawbacks to the use of a conjugate gradient or conjugate residual method for

solving the Newton equations. For one thing, if $Z^\top HZ$ is very poorly conditioned[7], the convergence of a conjugate gradient method might be inordinately slow (see Table 4.12 for an example). We have found that the use of diagonal scaling, coupled with a heuristic for choosing a "well scaled basis" (see [14]) improves the performance of the conjugate gradient method. In NLPNET [4] the user has an option to use a preconditioned conjugate gradient method with diag($Z^\top HZ$) as the preconditioner.

Essentially, any other iterative method could be used to solve the (reduced) Newton equations provided it met the criteria necessary for global convergence [17, 18]. However, there are many advantages to the use of a conjugate gradient inner iteration. Firstly, the above method does not require one to form the product $Z^\top HZ$ (which may be dense and exacerbates the problem of round-off errors). Secondly, the above PTN method generates gradient-related descent directions **whether or not $Z^\top HZ$ is positive definite or singular and is thus appropriate for nonconvex problems** [17]. It is also able to take advantage of the sparsity of Z and H and can, in cases where the Hessian is not available, dispense with $Z^\top HZ$ altogether by computing the product $(Z^\top HZ)d$ using sparse finite differences [17] along the reduced gradient $Z^\top g$. This requires one extra (reduced) gradient evaluation per CG iteration.

Without the use of a maximal basis and free superbasics, a situation which exists in MINOS [3], GRG [2] and CONOPT [24], it is likely, particularly in degenerate problems, that a direction computed using (20) and (21) will have to be discarded when $p_B = -B^{-1}Sp_S$ proves to be infeasible. This could make a truncated-Newton direction a very costly one for these codes (some experiments using a truncated-Newton direction in MINOS as reported in Section 4 seem to confirm this).

2.4 Terminating a minor iteration

The decision to permit constraints to be relaxed (*i.e.*, to start a new major iteration) is a crucial one since if affects the convergence properties of the algorithm. If done too soon, zigzagging between different constraints sets might result, thereby slowing the convergence rate or, more seriously, resulting in convergence to points that do not satisfy the Kuhn-Tucker conditions. (For a more detailed discussion see Dembo and Sahi [18].)

There are a number of practical rules for deciding when to relax constraints, some of which are

---

[7]The condition number of $Z^\top HZ$ is the ratio of the largest to the smallest eigenvalue.

known to be convergent [18].[8] We have found the following "forcing sequence strategy" to be easy and cheap to implement and effective in practice.

The forcing sequence strategy for terminating a minor iteration:

"Relax constraints when $\|g^R(x)\| \leq \eta_k \|g(x_0)\|$"

where $\eta_k \to 0$ is some preassigned sequence and x is the current value of the minor iterate.

In NLPNET [4] for example, we choose η_k such that, after NDECR major iterations, the tolerance required on a subspace is equal to the final optimality tolerance required of the reduced gradient. The parameter NDECR has a default value of 5 and can also be specified upon input.

2.5 Convergence of the PTN algorithm

Since the PTN algorithm fits into the Dembo-Sahi framework, to prove convergence one has to simply show that [18]:

(a) the restricted and relaxing directions are **gradient related**;[9]

(b) the constraint relaxation test is acceptable;[10] and

(c) the relaxing steps are "sufficiently-long".[11]

The relaxing direction is **gradient related** since it is a projection of the reduced gradient onto

---

[8] It is interesting to note that other constraint relaxation rules that have been found to work well in practice and are used in state-of-the-art codes such as MINOS [3] and GRG2 [2] are not known to produce convergent algorithms. They use the rule $\|g^R(x)\| \leq \eta \|g^R(x_k)\|$ with $\eta \in (0, 1)$.

[9] A sequence of search directions $\{p_k\}$ is said to be **gradient related** at points $\{x_k\}$ if along any convergent subsequence $\{x_{k_i}\} \to \bar{x}$, with $\bar{x}$ not a first-order optimum of LCNLP, we have $\lim_{\{x_{k_i}\} \to \bar{x}} g(x_{k_i})^T p_{k_i} < 0$.

[10] A constraint relaxation test is said to be **acceptable** if the limit point of every convergent subsequence on some active set is a first order minimum for LCNLP restricted to the active set [18].

[11] A relaxing step is said to be "sufficiently long" if for every subsequence $\{x_{k_i}\}$ converging to a nonoptimal critical point, $\bar{x}$, of some restricted problem $\lim_{\{x_{k_i}\} \to \bar{x}} \bar{\alpha}_{k_i} \neq 0$.

where $\bar{\alpha}$ is defined as in Definition 2.1 (for further details see Dembo and Sahi [18]).

the "box" constraints (12) with the property that $p_k = 0$ if and only if x_k is a first order optimum. The restricted directions are shown to be (reduced) gradient-related in Dembo and Steihaug [17].

Dropping constraints is done according to the "forcing sequence strategy" and thus the constraint relaxation test is acceptable.

The most difficult aspect of the convergence proof lies in showing that the relaxing steps are "sufficiently long". This would be automatically true if PTN were to be applied to nondegenerate problems [18]. However, assuming nondegeneracy is unreasonable in large-scale programming, particularly in network optimization.

Fortunately, there is a simple and practical way of ensuring sufficiently-long relaxing steps. The Dembo-Sahi **multiplier dropping rule** states that if one drops only inequality constraints "whose multiplier estimates are within a fixed fraction of the most negative multiplier estimate" then relaxing steps will be sufficiently long. It is not obvious how this rule may be applied in practice in PTN. It is first necessary to specify what the multiplier estimates are. Then, in order for the rule to be practical for large problems, it must be capable of being implemented **without identifying the most negative multiplier**. This appears to be a contradiction in terms since the dropping rule is expressed in terms of the "most negative multiplier".

Multiplier estimates,[12] evaluated at some known point $\tilde{x}$, are some approximation to the solution of the Kuhn-Tucker system[13]

$$g(\tilde{x}) + A^T\lambda + \bar{\mu} - \underline{\mu} = 0 \tag{22}$$

$$\bar{\mu} \geq 0 \ , \ \underline{\mu} \geq 0 \tag{23}$$

$$\bar{\mu}_j(x_j - u_j) = 0 \qquad \text{all } j \tag{24}$$

$$\underline{\mu}_j(x_j - l_j) = 0 \qquad \text{all } j \tag{25}$$

where λ represents the multiplier estimates corresponding to the equality constraints $Ax = b$, and $\bar{\mu}$ and $\underline{\mu}$ are multiplier estimates for the upper and lower bounding constraints respectively.

---

[12]For a detailed discussion of the properties of various multiplier estimates see Gill and Murray [27].

[13]There exist $\bar{\mu}$, $\underline{\mu}$ and λ satisfying (22) to (25) if A is of full rank and $\tilde{x}$ is a first order optimum. The system is inconsistent for all points x that are not critical.

Assume now that we restrict ourselves to multiplier estimates that always satisfy (24) and (25) exactly and deviations from satisfying the Kuhn-Tucker conditions above occur only in (22) and (23). Now consider partitioning (22) as follows:

$$g_B(\tilde{x}) + B^T\lambda + \bar{\mu}_B - \underline{\mu}_B = 0 \tag{26}$$

$$g_{\bar{B}}(\tilde{x}) + \bar{B}^T\lambda + \bar{\mu}_{\bar{B}} - \underline{\mu}_{\bar{B}} = 0 \ . \tag{27}$$

One convenient and cheap multiplier estimate is the following:

$$\lambda = -B^{-T} g_B \tag{28}$$

$$\bar{\mu}_B = \underline{\mu}_B = 0$$

which solves (26) exactly.

If, in addition, we let

$$(\bar{\mu}_{\bar{B}})_j = \begin{cases} -g_j^R(\tilde{x}) & \text{if } (\tilde{x}_{\bar{B}})_j = (u_{\bar{B}})_j \\ 0 & \text{otherwise} \end{cases} \tag{29}$$

and

$$(\underline{\mu}_{\bar{B}})_j = \begin{cases} g_j^R(\tilde{x}) & \text{if } (\tilde{x}_{\bar{B}})_j = (l_{\bar{B}})_j \\ 0 & \text{otherwise} \end{cases} \tag{30}$$

then there are only two possible sources of error in the Kuhn-Tucker conditions, namely

(a) $\|g_B^R(\tilde{x}) + (\underline{\mu}_{\bar{B}})_j - (\bar{\mu}_{\bar{B}})_j\| \neq 0$

(*i.e.*, (27) is not satisfied) or

(b) $\bar{\mu}_j$ or $\underline{\mu}_j < 0$ for some j.

If all the components of $\bar{\mu}$ and $\underline{\mu}$ computed using (29) and (30) are nonnegative and (27) is satisfied, then $\tilde{x}$ is optimal (to first order). Otherwise the multiplier dropping rule requires identification of the most negative component of $\bar{\mu}$ and $\underline{\mu}$. This might be expensive, particularly if candidate list strategies are used wherein only a subset of the negative multipliers is computed.

An easy and eminently practical way to obviate the need for the most negative multiplier is to **ignore all elements of the reduced gradient that are less than some fixed small tolerance, $\epsilon > 0$.** This amounts to considering (23) to be satisfied if $\bar{\mu}_j \geq -\epsilon$ or $\underline{\mu}_j \geq -\epsilon$ for all j and (22) to be satisfied if $\|[g^R]^+\|_\infty \leq \epsilon$. In any computer implementation, such an ϵ must be supplied, since it is only possible to satisfy the Kuhn-Tucker conditions to within some specified tolerance. In NLPNET [4] this ϵ is fixed *a priori* at $\sqrt{\text{machine precision}}$.

This requires a slight modification to the relaxing direction in (17) in which $p_{\bar{B}}$ is replaced by:

$$(p_{\bar{B}})_j^+ = \begin{cases} [-g_j^R]^+ & \text{if } (x_{\bar{B}})_j \text{ is not blocked} \\ 0 & \text{if } (x_{\bar{B}})_j \text{ is blocked or } |[-g_j^R]^+| \leq \epsilon . \end{cases} \tag{31}$$

Notice that this choice of $p_{\bar{B}}$ will result in dropping only those constraints whose multiplier estimates are within some fixed (implicit) fraction of the most negative multiplier. This is precisely what is implemented in NLPNET [4]. Similar devices are used in all reduced gradient and simplex codes, which seems to indicate that in practice it is not difficult to satisfy the "sufficiently long step" requirement in the Dembo-Sahi framework.

Definition 2.2 (**ϵ-optimality**)

Given some **fixed** *scalar $\epsilon > 0$, let $x^*(\epsilon)$, $\lambda^*(\epsilon)$, $\bar{\mu}^*(\epsilon)$, $\underline{\mu}^*(\epsilon)$ be a point with the following properties:*

(a) $x^*(\epsilon)$ is feasible [satisfies (2) and (3)] ;

(b) the multiplier estimates $\lambda^*(\epsilon)$, $\bar{\mu}^*(\epsilon)$, $\underline{\mu}^*(\epsilon)$ satisfy (24), (25) and (26) ;

(c) $\bar{\mu}^*(\epsilon)$, $\underline{\mu}^*(\epsilon)$ satisfy a modification of (23) given by

$$(\bar{\mu}^*(\epsilon))_j \geq -\epsilon \qquad \text{all } j \tag{32}$$

$$(\underline{\mu}^*(\epsilon))_j \geq -\epsilon \qquad \text{all } j . \tag{33}$$

and a modification of (27) given by:

$$\|g_{\bar{B}}^R(\tilde{x}) + (\underline{\mu}_{\bar{B}}) - (\underline{\mu}_{\bar{B}})\|_\infty \leq \epsilon . \tag{34}$$

Then $x^*(\epsilon)$ is said to be an **ϵ-optimal Kuhn-Tucker point.**

Proposition 2.1

Suppose $f(x)$ is continuously differentiable and bounded below and that $g(x)$ is Lipschitz continuous on the feasible region of LCNLP. Further suppose that in the PTN algorithm

(a) a modified direction with fixed parameter $\epsilon > 0$ in (31) is used on relaxing steps;

(b) a truncated-Newton direction is used on restricted steps;

(c) a "forcing sequence strategy" is used to terminate the minor iteration; and

(d) the steplength is always acceptable.

Then, starting at an arbitrary feasible point, any limit point $x^*(\epsilon)$, of sequence of major iterates $\{x_k\}$ generated by the PTN algorithm is an ϵ-optimal point.

The proof of this proposition follows directly from the global convergence theorem in Dembo and Sahi [18] due to the fact that the restricted and relaxing directions are gradient related, the steps are acceptable and a multiplier dropping rule is used.

Remark 2.5

An alternative to the multiplier dropping rule for guaranteeing sufficiently-long relaxing steps is the use of ϵ-active constraint sets [26, 30]. However in practice, (except perhaps for problems with box constraints [26]), computing such an ϵ might be expensive. The advantage of the modified multiplier dropping rule used above is that ϵ may be fixed *a priori* (it is a final tolerance on the reduced gradient at optimality which is standard input to any reduced gradient software).

3. Data Structures for Implementing PTN

3.1 Use of Linear Programming Data Structures

There are many features common to both the primal simplex algorithm and reduced-gradient methods. This is because they both operate on the reduced problem RCNLP. It therefore seems natural to capitalize on the wealth of experience accumulated by designers of LP software. Good examples of this are the reduced-gradient codes MINOS [3] and CONOPT [24] which use large-scale LP data structures to create, store, maintain and update the basis.

On the other hand there are many significant differences between the primal simplex method for LP and a reduced-gradient method such as PTN for NLP. For one thing, basis changes are far less frequent than in PTN. Also, there is no need to compute dual variables in minor iterations. These two aspects alone make all the intricate structures that enable one to pivot, retriagularize the basis and update the dual variables seem unwarranted for reduced-gradient codes. The case for different data structures becomes even stronger when examining the relative amount of work per iteration spent on calculating the search direction, linesearch, etc.

In most reduced gradient codes for LCNLP it is likely that the initial feasible guess will be calculated using LP, in which case LP data structures will be available for use in a reduced-gradient code. This is almost surely going to be the case for network optimization software because of the efficiencies of specialized primal simplex network codes.[14] We will therefore concentrate only on the additional data structures that are required to execute PTN efficiently.

3.2 Data Structures for the Efficient Computation of Search Directions

The single most expensive part of a PTN algorithm is likely to be the computation of restricted search directions. In solving for a truncated-Newton direction one has to either form the reduced Hessian $Z^\top HZ$ and use a direct solver to solve (20) or if an iterative method is used on (20), many products of the form $(Z^\top HZ)d$ will be needed per restricted direction calculation.

Forming $Z^\top HZ$ is undesirable for two reasons: it is likely to be dense even if Z and H are sparse, and it involves a loss of precision. Calculating products of the form $(Z^\top HZ)d$ require an efficient means of multiplying by $B^{-1}S$ and $(B^{-1}S)^\top$. There are two distinct cases to consider:

1. $B^{-1}S$ is dense, as is likely to be the case for general systems of linear constraints; and

2. $B^{-1}S$ is sparse which is the case when the constraints have special structure as in networks.

If $(B^{-1}S)$ is dense then it might make sense to actually compute and store $Z^\top HZ$, which could require significantly less storage than Z. An alternative is to approximate $Z^\top HZ$ using a Quasi-Newton update (this will not save on storage) or, as in MINOS [3], by using a diagonal

---

[14] In the NLPNET implementation of PTN we use the specialized primal simplex data structures thread, reverse thread, predecessor and depth arrays described by Bradley, Brown and Graves [28]

approximation to Z^THZ when storage runs out. A third possibility is to use a nonlinear conjugate gradient method on the reduced problem (another option in MINOS).

Limited computational experience on large unconstrained problems in Dembo and Steihaug [17] appears to suggest that a truncated-Newton search direction is preferable to the MINOS strategy, even when $(Z^THZ)d$ has to be approximated by finite differences Naturally, such evidence is not conclusive, or possibly not even indicative of the behavior in a constrained setting since the **relative** cost of the matrix vector multiplication $(Z^THZ)d$ might be much higher. If $B^{-1}S$ is sparse, as it is in network optimization, then it is possible to store it as a sparse matrix in such a way as to make products of the form $(B^{-1}S)u$ and $(B^{-1}S)^Ty$ cheap to compute.

In **networks** the columns of $B^{-1}S$ are cycles formed by joining a superbasic arc to the basis tree. The elements of $B^{-1}S$ are +1, -1 or 0 and thus storage of $B^{-1}S$ can be accomplished by storing two integer arrays, one of length equal to the number of nonzeros in $B^{-1}S$ and one of length equal to the number of superbasics. This is done in NLPNET [4].

To place this in perspective, in a nonlinear network problem obtained from a matrix balancing application, the number of nodes (constraints) was approximately 1200 and the number of superbasics was approximately 900 (a large linearly constrained NLP by today's standards). Thus $B^{-1}S$ was 1200 x 900 but, on average, there were only seven nonzero elements per column in $B^{-1}S$ and therefore $B^{-1}S$ was stored explicitly columnwise using two integer arrays, one of length 5600 and another of length 900.

There are some interesting new issues that crop up when $B^{-1}S$ is stored explicitly. For one, when a basic variable hits its bound and is pivoted out[15], one would like to be able to update $B^{-1}S$ cheaply without recomputing all columns. To do this, one first has to identify all the nonzero intersections in a row of $B^{-1}S$ (all the superbasics whose cycles/circuits contain the basic variable that hit its bound). This can be done by searching through a row of $B^{-1}S$ or by solving

$$B^Tq = e_i$$

where i is the index of the basic variable that hit a bound and e_i is a unit vector. The nonzero elements of S^Tq correspond to the columns of $B^{-1}S$ that change when the i^{th} basic variable is replaced.

In network optimization this may be done very efficiently using the THREAD and DEPTH

---

[15]To maintain a **maximal basis**, a basic variable that hits a bound is pivoted-out and is replaced by a free superbasic, if possible. This resembles a "dual simplex" pivot where one knows which basic variable will leave and searches for a superbasic to replace it with.

data structures [28, 29]. Let i be the index of the basic variable that hits a bound. Removing arc i from the basis tree results in the formation of two subtrees. To test whether a given superbasic is in a cycle with arc i, one simply tests whether the ends of the superbasic arc are in different subtrees.

Instead of updating $B^{-1}S$ each time a basic hits a bound, one could simply rearrange the pointers in the sparse matrix representation of $B^{-1}S$ (effectively ignoring the columns that were to be removed) and then add the new cycles that the incoming basic arc makes with the superbasic set. Periodically, if storage becomes a problem, redundant columns could be flushed out of $B^{-1}S$ and the sparse representation readjusted accordingly.

Thus, when it is possible to store $B^{-1}S$ (and provided it does not require more storage than $(Z^\top HZ)$ the truncated-Newton direction can be computed efficiently. Otherwise, it does require many operations involving B^{-1} and $B^{-\top}$ which may prove to be costly. Still, there is no conclusive evidence to indicate whether or not it is worthwhile.

In the matrix balancing problem cited earlier, a total of approximately 20,000 products of the form $(Z^\top HZ)d$ were required to compute a solution. Thus without storing $B^{-1}S$ approximately 40,000 multiplications of the form $B^{-1}Sy$ or $(B^{-1}S)^\top u$ were required. Now since $|S| \approx 900$ this means that 36,000,000 cycle traces would have been required. Storing $B^{-1}S$ reduced this to approximately 90,000 cycles that had to be traced at the meager cost of storing approximately 6,000 integer words.

3.3 Special features of the reduced Hessian $(Z^\top HZ)$ for future research

For a network[16], the reduced Hessian matrix $Z^\top HZ$ has nonzero elements whenever the cycles formed by two superbasic arcs intersect (*i.e.*, they have at least one basic arc in common). This leads to some intriguing possibilities. For example, if H is diagonal (as it almost always is in the network applications we have encountered) and the superbasic arcs are chosen so that they do not intersect (*i.e.*, the columns of $B^{-1}S$ are mutually orthogonal) then $Z^\top HZ$ is diagonal! Therefore, for particular choices of the superbasic set the (reduced) Newton direction is no more expensive to compute than a steepest descent (reduced gradient) direction.

In many applications (reservoir systems management is a good example) the physical structure of the network makes it easy to identify non-intersecting cycles. Rosenthal [8] gives a heuristic for identifying such cycles which he used in a reduced gradient code. The advantage for him was

---

[16]Although the discussion here is in terms of networks it extends in a very natural way to general linear systems with "circuit" replacing "cycle" and "variable" replacing "arc".

that his reduced gradient code could capitalize on the advantages of a convex simplex method [29].

The choice of nonintersecting cycles, while appealing, might result in operating in too restrictive a subspace. However, an extension of the idea allows one to work in as large a subspace as desired.

Let the columns of S be partitioned into $(S_1, S_2, \ldots S_k)$ with the property that the columns of $B^{-1}S_k$ are **mutually orthogonal** to $B^{-1}S_j$ for all $j \neq i$. Finding a partition in which the cardinality of the S_i are approximately equal and k is maximal is difficult but is analogous to sparse finite difference approximation techniques for approximating Jacobian matrices. These techniques have been the subject of much research recently and excellent software for sparse finite differencing is now available [31]. The importance in optimization is that when H is diagonal (or, in some cases, block diagonal) the reduced Hessian is **block diagonal** with blocks corresponding to the S_i . To our knowledge this fact has not yet been exploited in nonlinear network optimization. It could prove to be very important in finding ways to solve for truncated-Newton directions efficiently. For example, even if H is not diagonal, a block diagonal reduced Hessian could be used as a **preconditioner** in a truncated-Newton direction calculation.

4. Numerical Experiments with the PTN Algorithm

The PTN algorithm described in Sections 2 and 3 has been implemented in a specialized nonlinear optimization system called NLPNET [4]. It uses the depth, predecessor, thread and reverse-thread data structures described in [28, 29] and in addition stores the cycles $B^{-1}S$ generated by a superbasic set S explicitly in sparse matrix format. All test runs reported here were conducted on a DEC 20/60 computer under the TOPS 20 operating system. The code is written in FORTRAN and all test runs were conducted in double precision in a work space of 50,000 double precision words or less.

4.1 Methodology

Since there is little in the way of data on the performance, on the above computer, of other algorithms for nonlinear network problems or for large-scale NLP for that matter, I have chosen to present the results relative to a number of **benchmarks**. This permits one to measure efficiencies in a way that gives some (limited) measure of the performance of the PTN algorithm relative to other algorithms for the same problem structure.

As benchmarks I have chosen to measure performance relative to:

1. MINOS [3], an excellent state-of-the-art code for large-scale NLP;
2. the **reduced-Newton** option in NLPNET;
3. the specialized Convex Simplex [29] and Scaled-Reduced Gradient [16] methods;
4. the time taken to solve a piecewise linear approximation to the problem;
5. the time taken to solve a linear program obtained by linearizing the objective function; and
6. the work required to solve a single reduced-Newton system of equations to high accuracy.

These benchmarks give some idea of the efficiencies one can expect in a specialization of an NLP algorithm that takes advantage of network structure (see 1. and 3. above).

This methodology also permits one **to compare algorithms** (albeit on a limited number of test problems). The PTN **algorithm** is compared with the **default algorithm** used in MINOS and an indication is given as to how PTN might compare with MINOS on linearly constrained optimization problems that do not necessarily arise in networks.

Examining the behavior of PTN versus a reduced-Newton algorithm highlights the advantages of the rate-of-convergence theory developed in [15]. These are two algorithms that possess the same theoretical rates-of-convergence yet have vastly different overhead.

Finally, a comparison with solving an LP of the same structure and size using a specialized primal simplex method, gives some indication of the degree of efficiency of the NLPNET software and PTN algorithm. I feel that any NLP code that can, on average, achieve one order of magnitude CPU time difference when solving a comparable LP, is probably close to the limit of efficiency. For separable problems, a comparison with the time taken to solve one piecewise linearization of the problem gives some idea of the potential for a successive piecewise linearization algorithm, such as the one described in [32], to be competitive with PTN.

Naturally, no absolute statements can be made, however indications of the potential of an

algorithm *vis á vis* another are possible to detect. In a large-scale setting, it is not going to be possible to require researchers to conduct massive batch testing such as in [33]. Furthermore, I am not convinced that such batch testing yields any more information than carefully thought out experiments on a few test problems. The purpose of the tests conducted for this paper was to examine various aspects of the behavior of the PTN algorithm and secondarily, to test the efficiency of NLPNET.

Test problem characteristics and NLPNET's performance

There were five test problems used in this study all of which were derived from real-world nonlinear network models. Their characteristics are summarized in Table 4.1 . The first three, W30, W150 and W666, are derived from real data from the water distribution system of Dallas, Texas. They have been used in previous tests in the literature (see [16] for example) and provide a benchmark for comparison with other reported computational results.

The last two test problems, MB64 and MB1116, were matrix balancing problems derived from the input-output matrices for Thailand, as developed by the World Bank.

Problems W666 and MB1116 are not only very large by current NLP standards but also pose numerical difficulties because of the eigenvalue structure of the reduced Hessian $Z^{\top}HZ$. There are, however, biases in this test-problem set. Almost all out-of-basis variables are free at the solution in every problem and the problems are all separable and convex with very flat objective functions. The PTN algorithm does not require such characteristics nor does it take particular advantage of them. Still, it could mean that the behavior of PTN might be quite different on problem sets with different characteristics (*e.g.*, solution almost at a vertex, nonconvex nonseparable problems, etc.). Some experiments with variations of MB64 were run in which the out-of-basis variables ranged from almost totally constrained to almost totally free. In all cases, the additional constraints led to faster solution times.

A summary of NLPNET's performance on these problems is given in Table 4.2.

For all problems except the largest, time taken to solve a linearized problem (Phase I) was about one order of magnitude faster than the time taken to solve the nonlinear problem–almost ideal performance. For the largest problem (MB1116) this time was about three orders of magnitude slower. This can be explained by the fact that the PTN algorithm using a conjugate gradient solver will be sensitive to conditioning of the problem. This large problem was very poorly conditioned, as evidenced by the following benchmarks.

Table 4.1 Characteristics of the Test Problems

Problem Name	Description	No. of Nodes (Equality Constraints)	No. of Arcs (Variables)	Lower Bound on Cond. No. of Z^THZ at Solution	Dimension of Reduced Prob. at Optimality (# of Superbasics)
W30	Water Distribution Models Derived from Dallas, TX	30	46	10^4	15
W150		150	196	10^4	44
W666	Water Distribution System	666	906	10^6	240
MB64	Matrix Balancing Problems Derived from an Input-Output Matrix of Thailand	64	117	10^4	54
MB1116		1116	2230	10^8	946

Table 4.2 Performance of NLPNET on the Test Problems

Phase I (Finding a Feasible Point)

W30	49	45	.32
W150	206	201	1.32
W666	925	911	6.8
MB64	65	64	.44
MB1116	NA	NA	8.0

Phase II (Computing an Optimal Solution)

Problem	$\|\|RG\|\|_\infty$	Minor Iters.	CG Iters.	Funct. Evals.	CPU TIME, SECS. Line-Search	Search Dir.	Total	PhaseI/PhaseII TIME RATIO
W30	.04	12	62	24	1	.34	1.68	5
W150	.07	14	207	15	2.9	3.9	7.8	6
	.00009	15	485	17	3.1	8.7	12.8	9
W666	.02	19	580	26	23	66	99	14
	.00004	25	965	28	25	109	146	20
MB64	.02	16	125	37	1.3	3.0	5.1	12
MB1116	.05	48	13,997	51	55	7000 (approx.)	7560	940

To solve **one reduced-Newton system** to within a relative error of 10^{-6} required over 10,000 scaled-CG iterations for the MB1116 problem, which is of the same order as the total cumulative CG iterations required to solve the original problem. This incidentally, was true throughout. **The observed number of CG iterations for all the problems was of the same order as the number of CG iterations taken to solve one reduced-Newton system accurately.** This was also observed by Dembo and Steihaug [17] for unconstrained problems and is an indication of the power of the truncated-Newton direction.

Most of the time for solving MB1116 was therefore spent on computing search directions. The overall time for solving this problem would decrease by orders of magnitude if sufficient storage were available to solve the reduced-Newton system directly or if a potentially good preconditioner, such as the one discussed in Section 3.3, were available to speed up the CG iterations. This is an important area for future research.

A remarkable aspect of the PTN algorithm is the number of function gradient and Hessian evaluations[17] required to solve these test problems. It is amazing to think that one can solve a 2230 variable constrained optimization problem (MB1116) from a "cold start" **using only 51 function evaluations!** As is shown in Table 4.2, the number of function, gradient and Hessian evaluations is very small in all cases.

All the objective functions were extremely flat (see Table 4.3). For the 666 node problem the relative error in the objective function in 10^{-9} while the ||reduced gradient|| is approximately equal to 30. Thus, optimality tolerances of the order of 10^{-2} to 10^{-5} on the reduced gradient are very accurate stopping criteria. A characteristic of PTN algorithms is the ability to achieve a high degree of accuracy.

The main purpose for introducing Table 4.3, however, was to provide a benchmark with piecewise linear approximation algorithms such as [32, 34]. For these algorithms the stopping criterion that is most often used is the relative error in the objective function. The stopping criteria in in Table 4.2 imply an equivalent relative error criterion of less than 10^{-15}.

---

[17]For all the test problems the cost of obtaining function gradient and Hessian was the same as the cost of obtaining the function alone.

Table 4.3 Characteristics of the 660 Node Problem[1]: A Very Flat Objective Function

$\|RG\|_\infty$	$\|RG\|_2$	Relative Error $\frac{\|OBJ^* - OBJ\|}{\|OBJ^*\|}$	OBJective Function Value (OBJ)
.00004	.0005	0	-.20610749706 (=OBJ*)
.02	.03	0	-.20610749706 (=OBJ*)
.12	.20	8.7D-14	-.20610747906
1.9	4.1	5.3D-12	-.20610641306
27	111	6.4D-9	-.20479164906
275	754	9.3D-8	-.18700506806

Benchmarks:

$\|RG\|_\infty$ after piecewise-linear LP approximation with 1 segment = 783,000

$\|RG\|_\infty$ " " " " " " 4 segments = 460

(1) All the test problems we used had objective functions that were similarly flat.

4.3 Comparison of PTN and MINOS

Although MINOS is a general purpose code and NLPNET is a code designed to exploit network structure it is still possible and instructive to compare the behavior of the underlying algorithms.

For this purpose we implemented a truncated-Newton option in MINOS. Unfortunately, due to the larger core requirements of MINOS, the largest problem that we could solve was W150.

The behavior of MINOS on W150 is shown in Tables 4.4 and 4.5. It is interesting to note that the truncated-Newton option does worse than the variable metric option (which requires far more storage and terminates at a less accurate solution) and is superior to the Polak-Ribiere conjugate gradient option. Polak-Ribiere failed to reduce the reduced gradient below 2.54 . This might cause one to conclude that such an option was not worthwhile. However, this would be dangerous since many other aspects of the algorithms differ.

From the limited computational tests I have done, it appears as if the most significant difference between PTN and the algorithm used in MINOS is in the manner in which the optimal active set is identified and in the use of a maximal basis. If the results from the tests conducted with NLPNET are at all indicative, it shown that PTN is far superior to MINOS in this regard.[18] Consider the comparison of PTN with MINOS with a truncated-Newton direction shown in Table 4.5 . The PTN algorithm requires far fewer function evaluations (15 v.s 490 for MINOS on W150) to achieve the same solution accuracy. However, the best measure of the overall work involved is in the total number of CG iterations which is about five times greater for MINOS.

When evaluating the above experiment it is important to note that both algorithms MINOS-TN and PTN use truncated-Newton directions. What differentiates them is the manner in which relaxing and restricted steps are taken. It appears as if PTN strategy merits attention and further experimentation in a general purpose setting. A general-purpose code using the PTN algorithm and MINOS modules is now under development.

In all cases, NLPNET was more than one order of magnitude faster than the best MINOS run. This is reassuring but not surprising since NLPNET is a specialized code. However, the above evidence strongly suggests that the gain in speed is not due solely to the specialization to networks.

---

[18] In all tests the default pricing strategy was used. In newer versions of MINOS with multiple pricing, we expect an improved performance in identifying the active set.

Table 4.4 Behavior of MINOS[1] on the W150 Problem

	Variable Metric	Truncated Newton[2]	Polak Ribiere[3]
Total Number of Minor Iterations	328	278	530
Total Number of Function Evaluations	728	1536	1504
$\|\|$Reduced Gradient$\|\|_\infty$ at Termination	.03	.005	2.54
Total CPU Time (Secs)	109	177	163

(1) Default settings.

(2) $(Z^THZ)d$ was obtained by finite differencing since the Hessian was assumed to be unavailable. Cumulative CG iterations = 1046 which means this number of function evaluations could have been saved if the Hessian were readily available.

(3) The Fletcher-Reeves option performed significantly worse.

Table 4.5 PTN vs. MINOS with a Truncated-Newton Restricted Direction

	W30		W150	
	PTN	MINOS-TN	PTN	MINOS-TN
Total Minor Iterations	12	87	15	278
Total CG Iterations	62	280	247	1046
Total Function Evaluations with Known Hessian (with Finite Differencing)	24 (86)	188 (468)	15 (262)	490 (1536)
$\|\|RG\|\|_\infty$	.04	.002	.005	.005
Total CPU Time (Secs) --Cold Start				

4.4 Comparison of PTN and Specialized Scale-Reduced, Convex Simplex and Reduced-Newton Algorithms

Tables 4.6, 4.7 and 4.8 show how PTN behaves relative to the best (optimized) runtimes reported in Dembo and Klincewicz [16] for specialized reduced gradient and convex simplex codes. Also shown is the performance of PTN vs. a Reduced-Newton option in NLPNET [4].

It is clear that PTN requires far fewer function evaluations than either the SRG or CS algorithms (compare 21 for PTN vs. 870 for SRG and over 10,000 for CS on the W666 problem). Overall CPU time and final accuracy are also significantly better.

The reduced-Newton option in NLPNET is simply one FORTRAN statement that determines when to terminate the CG iteration on the Newton equations. It allows one to compare Newton's Method to PTN. Both algorithms exhibit a quadratic convergence rate on the reduced problem but at significantly different overhead (see Tables 4.7 and 4.8). For the large water distribution problem (W666) the total number of CG iterations required to reduce the norm of the final reduced gradient to 0.004 was an order of magnitude less for PTN than for reduced-Newton; PTN was also almost eight times faster.

These encouraging results are similar to the observed behavior on analogous tests executed by Dembo and Steihaug on unconstrained problems [17].

Table 4.6 PTN vs. Scaled Reduced Gradient (SRG) and Convex Simplex (CS) Methods

	W150			W660		
	PTN	SRG(1)	CS	PTN	SRG(1)	CS
Total Minor Iterations (CG)	18	124	1735	20 (591)	728	10,087
Total Function Evaluations	20	128	> 1735	21	870	> 10,087
$\|\|RG\|\|_\infty$	.03	.1	.1	.02	.1	.1
Total CPU Time (Secs)	8.9	12	119	113	390	1219

(1) The runs for SRG are the best reported in Dembo and Klincewicz [16] using a tuned conditioning heuristic.

Table 4.7 PTN vs. Reduced Newton (RN) on the W666 Problem

	PTN(1)	RN(2)
Total Minor Iterations	21	17
Total CG Iterations (With Preconditioning	667	6479
Total Function Evaluations	22	18
$\|\|RG\|\|_\infty$	.004	.006
CPU Time (Secs) (Feasible Starting Point)	111	794

Terminate superbasic restricted-direction (p_S) calculation when:

(1) [PTN] $\|Z^THZp_S + Z^Tg\| / \|Z^Tg\| \leq \min \{.01, \|Z^Tg\|\}$

(2) [RN] $\|Z^THZp_S + Z^Tg\| / \|Z^Tg\| \leq$ 1.0D - 10

Table 4.8 PTN vs. Reduced Newton (RN) on the W150 Problem

	No Preconditioning in CG		Preconditioned CG	
	PTN	RN	PTN	RN
Total Minor Iterations	16	16	18	15
Total CG Iterations	435	1293	265	543
Total Function Evaluations	19	18	18	17
$\|RG\|_\infty$	.0006	.006	.002	.0003
CPU Time (Secs) (Feasible Start)	12.9	26.5	8.8	13.6

Terminate superbasic restricted-direction calculation (p_S) when

(1) [PTN] $\|Z^THZp_S + Z^Tg\| / \|Z^Tg\| \leq \min\{0.01, \|Z^Tg\|\}$

(2) [RN] $\|Z^THZp_S + Z^Tg\| / \|Z^Tg\| \leq 1.D - 10$

4.5 PTN Benchmarked Against a Single Piecewise Linearization

Since all the test problems are separable it is natural to test whether a piecewise linear approximation algorithm could outperform PTN. To examine this, a number of experiments were run in all cases solving a single piecewise linear approximation. This was then compared with results using PTN on the same problem. Some interesting conclusions can be drawn for this limited experiment.

1. Use of a **single** piecewise linearization for Phase I, with a limited number of segments per variable, can sometimes yield a better starting point for Phase II and lower overall solution times (see Table 4.10 for results on the W666 problem and Table 4.11 for results on MB64).
2. To obtain and solve a piecewise linearization approximation that even approaches the accuracy of PTN may cost more than one order of magnitude more work (see Table 4.9).
3. The cost of solving one piecewise linearized problem with two segments is only one-sixth the cost of solving the problem exactly using PTN. Thus to be competitive, piecewise linearization algorithms would have to converge very rapidly. This is unlikely, even for ingenious algorithms such as Meyer's [32].

Details of PTN's Behavior on MB1116

Table 4.12 shows details of an NLPNET run on the 2230 variable, 1116 equality constrained matrix balancing problem. The results in this table are instructive since they highlight the power of the various ideas embodied in PTN.

Starting at a vertex (a poor starting point for this problem given that almost all out-of-basis variables are free at an optimum) **the first relaxing direction frees 712 constraints!** Twenty restricted steps are taken in the first major iteration and 19 constraints are added to the active set.

The second major iteration begins with a relaxing step in which **106 constraints are dropped** from the active set. Two restricted steps are then taken and one constraint is added to the active set.

The third iteration starts with a restricted-projection relaxing step in which **25 constraints are dropped**. Eleven restricted steps then result in 10 more constraints being added to the active set.

Table 4.9 PTN vs. Piecewise Linearization (PL) on W31 (a Separable Problem)

Number of Sections Per Arc	Number of Pivots	Number of Minor Iterations	Objective Value	$\|RG\|_\infty$	CPU Time (Secs)
30	3316	3819	-.323D5	.14	22.5
20	1620	1854	-.321D5	6.6	11.0
10	583	770	-.314D5	1.2	4.6
5	211	273	.102D6	65	1.6
1	45	49	.130D8	201	.31
Benchmark (PTN on W31):			-.324D5	.08	2.0

Table 4.10 Use of a Piecewise Linear Approximation for Phase I (Computing an Initial Feasible Point) (Runs on W660)

Number of Approximating Segments per Arc	1	2	4
Total Phase I Iterations (Pivots)	925 (911)	2487 (2412)	5270 (4951)
$\|\|RG\|\|_\infty$ at the End of Phase I	.77D6	.86D3	.86D3
Total Phase II Minor Iterations	28	12	12
Total CG Iterations	721	255	178
Total Function Evaluations	46	22 + 2	19 + 4
Phase I CPU	6.6	16.1	37.5
Phase II CPU	140	63.2	48.9
Total CPU	146.6	79.3	86.4

Table 4.11 Use of a Piecewise Linear Approximation[1] for Phase I (Computing an Initial Feasible Point) (Tests on MB64)

Number of Approximating Segments per Arc	1	2	4	8	16
Total Phase I Iterations (Pivots)	65 (64)	86 (85)	127 (126)	201 (200)	382 (381)
$\|\|RG\|\|_\infty$ at the End of Phase I	.19D3	.11D3	.68D2	.43D2	.25D2
Total CG Iterations	125	115	118	135	105
Total Function Evaluation	37	32 + 2	36 + 4	35 + 8	31 + 16
$\|\|RG\|\|_\infty$ at Solution			< .1		
Phase I CPU Time (Secs)	.44	.63	.84	1.5	2.8
Phase II CPU Time (Secs)	5.11	3.87	4.40	5.31	3.95
Total CPU Time (Secs)	5.55	4.50	5.24	6.81	6.75

[1] The segments were spaced evenly. Improvements in Phase I time may be possible with spacing that reflects the changes in curvature more closely. However, this is at the cost of additional overhead in computing the piecewise linear approximation.

Table 4.12 Behavior of PTN on MB1116

(1116 Equality Constraints, 2230 Variables)

Major Iteration	Cumulative Minor Iterations (PCG Iters.)	Cumulative Function Evaluations	No. of Superbasics Initial	No. of Superbasics Final	For Final Restricted Direction $\|RG\|_\infty$	For Final Restricted Direction $\|Z^THZp_S+Z^Tg\|$
Feasible Start	0	--	0	0	--	--
1	20 (141)	20	712	693	959	96
2	22 (400)	22	799	798	286	27
3	33 (4,121)	33	823	813	50.9	6.5
4	41 (11,673)	43	940	936	1.72	.49
5	46 (13,987)	49	943	943	.031	.024
6	48 (13,997)	51	946	946	.055	.138

CPU Time: Feasible Flow 8.0 Secs.

Linesearch 55 Secs.

Total 7560 Secs (126 Minutes)

CG Iterations to solve Newton equations once with

$\|Z^THZp_S + Z^Tg\| / \|Z^Tg\| \leq 1.D - 8$ > 10,000

In this manner PTN is able to rapidly identify the optimal active set with very little overhead (a total of 48 minor iterations). In particular it is worth noting how few major iterations (and hence how few relaxing steps) were required. This reinforces the notion that using a cheap gradient projection relaxing step will probably not affect the convergence rate for PTN algorithms.

The forcing sequence strategy works in tandem with the truncated-Newton termination condition. When the reduced gradient is large, not much work is done before releasing constraints. Little work is also done in solving the reduced-Newton equations since the termination criterion depends on the norm of the reduced gradient. In this way, major and minor iterates are synchronized so that only when the reduced gradient is small (*i.e.*, when one is already confident that the correct active set has been identified by the forcing sequence) are the reduced-Newton equations solved accurately (see the last two columns of Table 4.12). By only solving for search directions and on various subspaces as accurately as is needed, overall effort is kept to a minimum thereby permitting one to compute a solution to the problem with the effort it takes CG to solve one reduced-Newton system exactly.

References

1. Barr, R. S., and J. S. Turner, "Microdata File Merging Through Large-Scale Network Technology", Mathematical Programming Study, Vol. 15 (1981) pp. 1-22.
2. Lasdon, L. S., A. D. Waren, A. Jain and M. W. Ratner, "Design and Testing of a Generalized Reduced Gradient Code for Nonlinear Programming", ACM Transactions on Mathematical Software, Vol. 4 (1978) pp. 34-50.
3. Murtagh, B., and M. Saunders, "Large-Scale Linearly Constrained Optimization", Mathematical Programming, Vol. 14 (1978) pp. 41-72.
4. Dembo, R. S., "NLPNET - User's Guide and System Documentation", School of Organization and Management Working Paper Series B #70, Yale University (1983).
5. Glassey, C. R., "A Quadratic Network Optimization Model for Equilibrium Single-Commodity Trade Flows", Mathematical Programming, Vol. 14 (1978) pp. 98-107.
6. Samuelson, P., "Spatial Price Equilibrium and Linear Programming", American Economic Review, Vol. 42 (1952) pp. 283-303.
7. Collins, M., L. Cooper, R. V. Helgason, J. L. Kennington, and L. J. LeBlanc, "Solving the Pipe Network Analysis Problem Using Optimization Techniques", Management Science, Vol. 24 (1978) pp. 747-760.
8. Rosenthal, R. E., "A Nonlinear Network Flow Algorithm for Maximizing the Benefits in a Hydroelectric Power System", Operations Research, Vol. 29 (1981) pp. 763-786.
9. Jacoby, S. L. S., and J. S. Kowalik, Mathematical Modeling with Computers, Prentice Hall, New York (1980).
10. Hanscom, M., L. Lafond, L. S. Lasdon and G. Pronovost, "Modeling and Resolution of the Deterministic Mid-Term Energy Production Problem for the Hydro-Quebec System", Management Science, Vol. 26 (1980) pp. 659-668.
11. Cooper, L. and J. Kennington, "Steady-State Analysis of Nonlinear Resistive Networks Using Optimization Techniques", Technical Report IEOR 77012, Southern Methodist University, Dallas (1977).
12. Florian, M. and S. Nguyen, "An Application and Validation of Equilibrium Trip Assignment Methods", Transportation Science, Vol. 10 (1976) pp. 374-389.
13. Aashtiani, H. and T. Magnanti, "Equilibria on a Congested Transportation Network", SIAM Journal on Algebraic and Discrete Methods, Vol. 2 (1981) pp. 213-226.
14. Dembo, R. S. and J. G. Klincewicz, "Resolving Degeneracy Using a Maximal Basis", School of Organization and Management Working Paper Series B #63, Yale University (1983).
15. Dembo, R. S., S. C. Eisenstat and T. Steihaug, "Inexact Newton Methods", SIAM Journal of Numerical Analysis, Vol. 19 (1982) pp. 400-409.
16. Dembo, R. S. and J. G. Klincewicz, "A Scaled Reduced Gradient Algorithm for Network Flow Problems with Convex Separable Costs", Mathematical Programming Study, Vol. 15 (1981) pp. 125-147.
17. Dembo, R. S. and T. Steihaug, "Truncated-Newton Algorithms for Large-Scale Unconstrained Optimization", Mathematical Programming, Vol. 26 (1983) pp. 190-212.

18. Dembo, R. S. and S. Sahi, "A Convergent Framework for Constrained Nonlinear Optimization", School of Organization and Management Working Paper Series B #69, Yale University (1983).

19. Bertsekas, D. P., "On the Goldstein-Levitin-Polyak Gradient Projection Method", IEEE Transactions on Automatic Control, Vol. 21 (1976) pp. 174-184.

20. Fletcher, R., Practical Methods of Optimization Volumes I and II, John Wiley and Sons, New York (1981, 1982).

21. McCormick, G. P., "The Variable Reduction Method for Nonlinear Programming", Management Science, Vol. 17 (1970) pp. 146-160.

22. Gill, P. E. and W. Murray, Numerical Methods in Constrained Optimization, Academic Press, New York (1974).

23. Bland, R. G., "New Finite Pivoting Rules for the Simplex Method", Mathematics of Operations Research, Vol. 2 (1977) pp. 103-107.

24. Drud, A., "CONOPT - A GRG Code for Large Sparse Dynamic Nonlinear Optimization Problems", Discussion Paper DRD59, The World Bank, August 1983.

25. Dembo, R. S. and U. Tulowitzki, "On the Minimization of Quadratic Functions Subject to Box Constraints", School of Organization and Management Working Paper Series B #71, Yale University, 1983.

26. Bertsekas, D. P. "Projected Newton Methods for Optimization Problems with Simple Constraints", SIAM Journal of Control and Optimization, Vol. 20 (1980) pp. 221-246.

27. Gill, P. E. and W. Murray, "The Computation of Lagrange Multiplier Estimates for Constrained Minimization", Report NAC 77, National Physical Laboratory, England.

28. Bradley, G. H., G. G. Brown and G. W. Graves, "Design and Implementation of Large-Scale Primal Transshipment Algorithms", Management Science, Vol. 24 (1977) pp. 1-34.

29. Kennington, J. L. and R. V. Helgason, Algorithms for Network Programming, John Wiley and Sons, New York (1980).

30. Zoutendijk, G., "Nonlinear Programming: A Numerical Survey", SIAM Journal on Control, Vol. 1 (1966).

31. Coleman, T. F. and J. J. More', "Estimation of Sparse Hessian Matrices and Graph Coloring Problems", Research Report ANL/MCS-TM-4, Argonne National Laboratory (1982).

32. Meyer, R. R., "Two Segment Separable Programming", Management Science, Vol. 25 (1979) pp. 385-396.

33. Schittkowski, K., Nonlinear Programming Codes--Information, Tests, Performance from the series Lecture Notes in Economics and Mathematical Systems, No. 183, Springer-Verlag, Berlin (1980).

34. Belling-Seib, K., "Mathematische Programmierungsprobleme mit Netzwerkstruktur", Ph.D. Thesis, Free University of Berlin, (1983).

35. Pyatt, G., A.R. Roe, R.M. Lindley and J.I. Round, Social Accounting for Development and Planning , Cambridge University Press (1977).

36. Eriksson, "A Note on the Solution of Large Sparse Maximum Entropy Problems with Linear Equality Constraints", Mathematical Programming, Vol. 18 (1980) pp. 146-154.