Numerical experiments with the LANCELOT package (Release A) for large-scale nonlinear optimization

by A.R. Conn[1], Nick Gould[2], and Ph.L. Toint[3]

Report 92/16 (3rd revision) September 5, 1995

[1] IBM T.J. Watson Research Center, P.O.Box 218, Yorktown Heights, NY 10598, USA
Email : arconn@watson.ibm.com

[2] Computing and Information Systems Department, Rutherford Appleton Laboratory, Chilton, England, EC
Email : nimg@letterbox.rl.ac.uk
Current reports available by anonymous ftp from the directory "pub/reports" on camelot.cc.rl.ac.uk (internet 130.246.8.61)

[3] Department of Mathematics, Facultés Universitaires ND de la Paix, 61, rue de Bruxelles, B-5000 Namur, Belgium, EC
Email : pht@math.fundp.ac.be
Current reports available by anonymous ftp from the directory "pub/reports" on thales.math.fundp.ac.be (internet 138.48.4.14)

Keywords : Large-scale problems, nonlinear optimization, numerical algorithms.

Numerical experiments with the LANCELOT package (Release A) for large-scale nonlinear optimization

A.R. Conn Nick Gould Ph.L. Toint

September 5, 1995

Abstract

In this paper, we describe the algorithmic options of Release A of LANCELOT, a Fortran package for large-scale nonlinear optimization. We then present the results of intensive numerical tests and discuss the relative merits of the options. The experiments described involve both academic and applied problems. Finally, we propose conclusions, both specific to LANCELOT and of more general scope.

1 Introduction

Research in large-scale optimization has been, in recent years, a major subject of interest within the mathematical programming community, as is clear from the programs of the main conferences and symposia on optimization techniques during this period. One such project was initiated by the authors of this paper [12] and has resulted in both theoretical contributions and software for large nonlinear optimization problems. A detailed description of the algorithms developed and implemented in LANCELOT, the resulting Fortran package, is presented in [15]. The purpose of the present paper is to report on the numerical experiments performed with this software on a sizeable collection of test problems, and to draw some first conclusions on the respective merits of the algorithmic options available in the package. A comparison of LANCELOT and MINOS [45] is currently being conducted on a large set of test problems. However, due to the diversity of algorithmic options and complexity of these two packages, a fair and informative comparison is, in itself, a major research effort. It will be reported on separately.

The paper is organized as follows. Section 2 briefly presents the main features and structure of LANCELOT. Section 3 contains a general description of SBMIN, the kernel algorithm for the software that handles simple bounds. AUGLG, the component that handles the extension to general constraints, is then outlined in Section 4. Section 5 discusses the various algorithmic options that are available within the package. Section 6 presents the testing framework and the

[0] This research was supported in part by the Advanced Research Projects Agency of the Department of Defense and was monitored by the Air Force Office of Scientific Research under Contract No F49620-91-C-0079. The United States Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation hereon.

strategy used to analyze the results. These results are then discussed in more detail in Section 7, where the efficiency and robustness of various algorithmic options are compared. Finally, some conclusions and perspectives are drawn in Section 8.

2 General features and structure of the LANCELOT package

2.1 Package presentation

The purpose of the LANCELOT package is to solve the general nonlinear programming problem

$$\min_{x \in \Re^n} f(x) \tag{2.1}$$

subject to the constraints

$$c(x) = 0, \tag{2.2}$$

and to the simple bounds

$$l_i \leq x_i \leq u_i, \quad (i = 1, \ldots, n), \tag{2.3}$$

where f and c are assumed to be smooth functions from $\Re^n$ into $\Re$ and from $\Re^n$ into $\Re^m$, respectively. *The package is specially intended for problems where n and/or m are large.* Indeed, it exploits the *(group) partially separable* structure (see [12]) of most large-scale optimization problems. However, the package can also be applied successfully to small problems. The algorithms are designed to provide convergence of the generated iterates to local minimizers from all starting points.

There is no loss in assuming that all the general constraints are equality constraints, as inequality constraints may easily be transformed to equations by the addition of extra slack or surplus variables (see, for example, [31, Section 5.6]). Indeed, LANCELOT automatically transforms inequality constraints to equations. This technique is extensively used in simplex-like methods for large-scale linear and nonlinear programs.

General features include facilities to compute numerical derivatives, an analytical derivative checker and an automated restart. The software also uses a full reverse communication interface for greater flexibility and adaptability.

The package is written in standard ANSI Fortran77. It has already been ported to CRAY and IBM mainframes, to Digital VAX minicomputers, and to Digital, Hewlett-Packard, IBM, Silicon Graphics and Sun workstations, as well as to DOS-based personal computers. A fully automated installation procedure is supported for all these machines/systems. Single and double precision versions are available. The program's dimensions are also adaptable to fit within machines with different memory sizes.

Full information on the package is available in [15]. Interested parties should contact one of the authors.

2.2 The algorithmic structure of the package

Because the purpose of this paper is to discuss the relative merits of several algorithmic options within the package, it is necessary to provide first a general description of the numerical methods used. The structure of the LANCELOT algorithms is summarized in Figure 1.

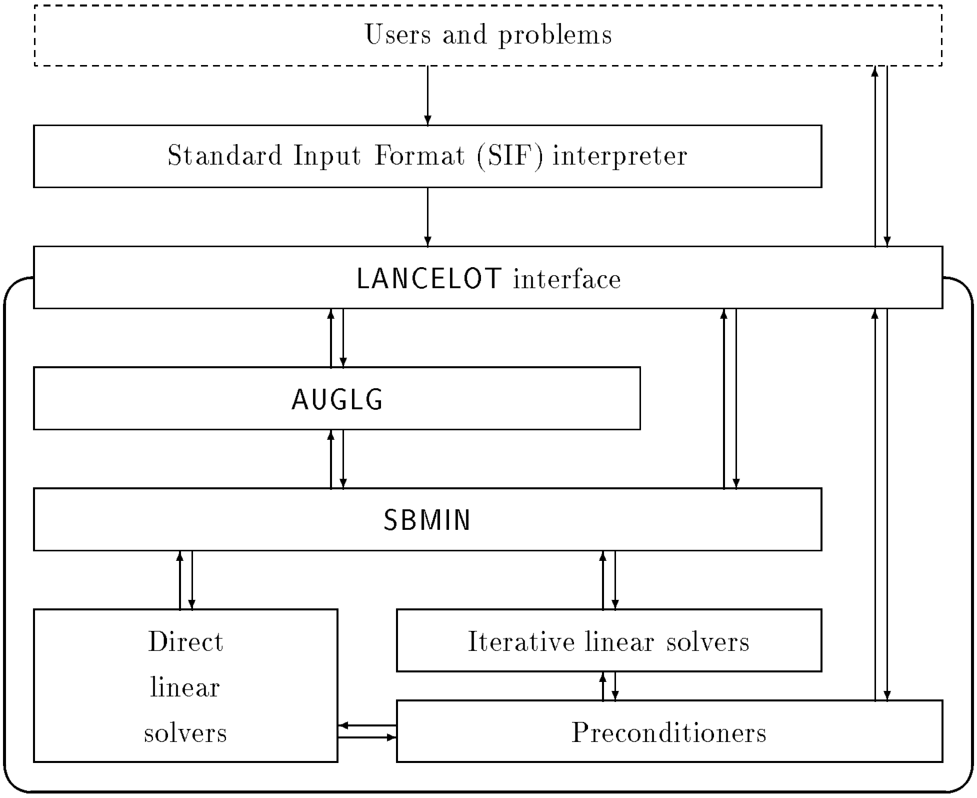

Figure 1: Structure of the LANCELOT package

The package (whose algorithmic components appear in the rounded box) reads the problem as a set of data and Fortran subroutines (for computing function and derivatives values, as well as other problem related tasks). The way in which these subroutines and the associated datafile are produced is not the subject of this paper. It suffices to say that they can be written directly by the user, or obtained as the result of the automated interpretation of the problem expressed in a more friendly Standard Input Format. These techniques are described in detail in [15] and will not be discussed further here. We will rather concentrate on the algorithms used by LANCELOT to solve the problem, once properly specified. As suggested by the picture, LANCELOT either uses an augmented Lagrangian approach (if constraints of the type (2.2) are present), or directly attempts to solve problems whose only constraints are simple bounds, (2.3).

The augmented Lagrangian algorithm AUGLG is outlined in Section 4. Its convergence theory has been analyzed in [13] and [16]. This theory guarantees that, under standard assumptions, the sequence of iterates calculated by the algorithm converges to a local minimizer of the problem. This augmented Lagrangian method proceeds by solving a sequence of suitably defined nonlinear optimization problems with simple bound constraints. We will call these iterations of the augmented Lagrangian algorithm *major iterations*.

If the problem under consideration possesses only simple bounds, a specialized algorithm, SBMIN, can be applied. This algorithm is of trust region type and is presented in Section 3. Its strong convergence properties have been analyzed in [10], [38] and [51]. At the heart of SBMIN, quadratic problems with bound constraints (BQP) are solved repeatedly. In fact, a BQP is approximately solved at every SBMIN iteration. We call these *minor iterations*.

The process of (approximately) solving the BQP involves the (approximate) solution of a linear system of equations. This can be achieved by applying either direct or iterative linear solvers. The latter typically require preconditioning, which in turn might call specialized versions of the direct solvers, as is shown in the figure above. The iterative technique used with the package is preconditioned conjugate gradients. Iterations at this level are simply called *cg-iterations*. Note that some form of preconditioning might require a very problem specific technique; hence there is the possibility to return to the user level for such a calculation.

The three nested iteration levels (major iterations at the augmented Lagrangian level, minor iterations at the SBMIN level, and cg-iterations at the BQP level) are illustrated in Figure 2, where the dashed boxes indicate iteration levels that need not be present for all problems and all choices of algorithmic options.

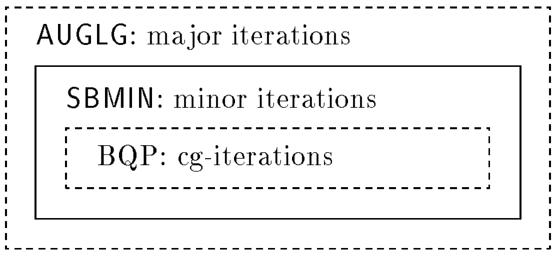

Figure 2: The nested iteration levels within LANCELOT

As the bulk of the computational work is performed in the minor and cg-iterations, we now summarize these parts of the algorithm. The reader is urged to consult Chapter 3 of [15] for further details.

3 An outline of SBMIN

SBMIN is a method for solving the bound-constrained minimization problem defined by (2.1) and the simple bound constraints (2.3). Here, f is assumed to be twice-continuously differentiable and any of the bounds in (2.3) may be infinite. We will denote the vector of first partial derivatives, $\nabla_x f(x)$, by $g(x)$ and the Hessian matrix, $\nabla_{xx} f(x)$, will be denoted by $H(x)$. We shall refer to the set of points which satisfy (2.3) as the *feasible box* and any point lying in the feasible box is said to be *feasible*.

SBMIN is an iterative method. At the end of the k-th iteration, an estimate of the solution, $x^{(k)}$, satisfying the simple bounds (2.3), is given. The purpose of the $(k+1)$-st iteration is to find a feasible iterate $x^{(k+1)}$ which is a significant improvement on $x^{(k)}$.

In the $(k+1)$-st iteration, we build a quadratic model of our (possibly) nonlinear objective function, $f(x)$. This model takes the form

$$m^{(k)}(x) = f(x^{(k)}) + g(x^{(k)})^T(x - x^{(k)}) + \frac{1}{2}(x - x^{(k)})^T B^{(k)}(x - x^{(k)}), \tag{3.1}$$

where $B^{(k)}$ is a symmetric approximation to the Hessian matrix $H(x^{(k)})$. We also define a scalar $\Delta^{(k)}$, the *trust-region radius*, which defines the *trust region*,

$$\|x - x^{(k)}\| \le \Delta^{(k)}, \tag{3.2}$$

within which we trust that the values of $m^{(k)}(x)$ and $f(x)$ will generally agree sufficiently. An appropriate range of values for the trust-region radius is accumulated as the minimization proceeds.

The $(k+1)$-st iteration proceeds in a number of stages. These may be summarized, in order, as:

1. Test for convergence. The calculation is stopped when the *projected gradient* is small enough, that is when
$$\|x^{(k)} - P(x^{(k)} - g(x^{(k)}), l, u)\|_\infty \le \epsilon_g \tag{3.3}$$
holds for some appropriate small convergence tolerance ϵ_g, where
$$P(x, l, u)_i = \min(\max(l_i, x_i), u_i). \tag{3.4}$$

2. Find the generalized Cauchy point of the quadratic model (see Section 3.1).

3. Obtain a new point which further reduces the quadratic model within the intersection of the feasible box and the trust region (see Section 3.2).

4. Test whether there is a general agreement between the values of the model and true objective function at the new point. If so, accept the new point as the next iterate (the iteration is then said to be *successful*). Otherwise, retain the existing iterate as the next iterate (the iteration is *unsuccessful*). In either case, adjust the trust region radius as appropriate (see Section 3.2.4 of [15]).

3.1 The generalized Cauchy point

The approximate minimization of the quadratic model (3.1) within the intersection of the feasible box and the trust region at the $(k+1)$-st iteration is accomplished in two stages. In the first, we obtain the so called generalized *Cauchy point* (GCP), which is the result of this minimization carried out only on the path defined by the projection of the model's negative gradient onto this intersection. This point is important mostly because convergence of the algorithm to a point at which the projected gradient is zero can be guaranteed provided the value of the quadratic model at the end of each minor iteration is no larger than that at the generalized Cauchy point (see [10]).

An efficient algorithm for this calculation, when the trust region is defined in the infinity-norm (the LANCELOT default), is given in [11]. However, it is not necessary that the generalized Cauchy point be calculated exactly. Indeed, a number of authors have considered approximations which are sufficient to guarantee convergence (see [6], [7], [8], [40], [51]). Consequently we provide the option of using the approximation suggested by Moré in [40]. Since in our experience this option has proved to be less reliable and less efficient than the exact calculation, we will not discuss it further. Interested readers are referred to [15].

3.2 Beyond the generalized Cauchy point

We have ensured that SBMIN will converge by determining the generalized Cauchy point. Convergence at a reasonable rate is achieved by, if necessary, further reducing the quadratic model.

Those variables which lie on their bounds at the generalized Cauchy point are fixed. Attempts are then made to reduce the quadratic model by changing the values of the remaining free variables. Let $x^{(k,1)}$ be the obtained generalized Cauchy point and let $x^{(k,j)}, j = 2, 3, \ldots$ be distinct points such that:

- $x^{(k,j)}$ lies within the intersection of the feasible box and the trust region;
- those variables which lie on a bound at $x^{(k,1)}$ lie on the same bound at $x^{(k,j)}$;
- $x^{(k,j+1)}$ is constructed from $x^{(k,j)}$ by
  1. determining a nonzero search direction $p^{(k,j)}$ for which
  $$\nabla_x m^{(k)}(x^{(k,j)})^T p^{(k,j)} < 0; \tag{3.5}$$
  2. finding a steplength $\alpha^{(k,j)} > 0$ which minimizes $m^{(k)}(x^{(k,j)} + \alpha p^{(k,j)})$ within the intersection of the feasible box and the trust region; and
  3. setting
  $$x^{(k,j+1)} = x^{(k,j)} + \alpha^{(k,j)} p^{(k,j)}. \tag{3.6}$$

This process is stopped when the norm of the free gradient of the model at $x^{(k,j)}$ is sufficiently small. The *free gradient of the model* is

$$Q(\nabla_x m^{(k)}(x^{(k,j)}), x^{(k,j)}, l, u), \tag{3.7}$$

where the operator

$$Q(y, x, l, u)_i = \begin{cases} y_i & \text{if } l_i < x_i < u_i, \\ 0 & \text{otherwise,} \end{cases} \tag{3.8}$$

zeros components of the gradient corresponding to variables which lie on their bounds. In LANCELOT, we stop when

$$\|Q(\nabla_x m^{(k)}(x^{(k,j)}), x^{(k,j)}, l, u)\| \leq \|Q(\nabla_x m^{(k)}(x^{(k)}), x^{(k,1)}, l, u)\|^{1.5}, \tag{3.9}$$

which is known (see [38]) to guarantee that the convergence rate of the method is asymptotically superlinear.

There is much flexibility in obtaining a search direction which satisfies (3.5). We determine such a direction by finding an approximation to the minimizer of the quadratic subproblem (3.1), where certain of the variables are fixed on their bounds but the constraints on the remaining variables are ignored. Specifically, let $\mathcal{I}^{(k,j)}$ be a set of indices of the variables which are to be fixed, let e_i be the i-th column of the n by n identity matrix I and let $\bar{I}^{(k,j)}$ be the matrix made up of columns e_i, $i \notin \mathcal{I}^{(k,j)}$. Now define

$$\bar{g}^{(k,j)} \equiv \bar{I}^{(k,j)T} g^{(k,j)} \text{ and } \bar{B}^{(k,j)} \equiv \bar{I}^{(k,j)T} B^{(k,j)} \bar{I}^{(k,j)}. \tag{3.10}$$

Then the quadratic model (3.1) at $x^{(k,j)} + p$, considered as a function of the free variables $\bar{p} \equiv \bar{I}^{(k,j)T} p$, is

$$\bar{m}^{(k,j)}(\bar{p}) = m^{(k)}(x^{(k,j)}) + \bar{g}^{(k,j)T} \bar{p} + \frac{1}{2} \bar{p}^T \bar{B}^{(k,j)} \bar{p}. \tag{3.11}$$

We may attempt to minimize (3.11) using either a direct or iterative method.

In a direct minimization of (3.11), one factorizes the coefficient matrix $\bar{B}^{(k,j)}$. If the factors indicate that the matrix is positive definite, the Newton equations

$$\bar{B}^{(k,j)} \bar{p}^{(k,j)} = -\bar{g}^{(k,j)} \tag{3.12}$$

may be solved and the required search direction $p^{(k,j)} = \bar{I}^{(k,j)} \bar{p}^{(k,j)}$ recovered. If, on the other hand, the matrix is merely positive semi-definite, a direction of linear infinite descent or a weak solution to the Newton equations can be determined. Finally, if the matrix is truly indefinite, a direction of negative curvature may be obtained.

In an iterative minimization of (3.11), the index set $\mathcal{I}^{(k,j)}$ may stay constant over a number of iterations, while at each iteration the search direction may be calculated from the current model gradient and Hessian $\bar{B}^{(k,j)}$ and previous search directions. The iterative method used in LANCELOT is the method of conjugate gradients. The convergence of such a method may be accelerated by preconditioning (see below) . In fact the boundary between a good preconditioned iterative method and a direct method is quite blurred.

4 An outline of AUGLG

AUGLG is a method for solving the generally-constrained minimization problem defined by (2.1)–(2.3). As above, f and the c_j are all assumed to be twice-continuously differentiable and any of the bounds in (2.3) may be infinite.

The objective function and general constraints are combined into the *augmented Lagrangian*

$$\Phi(x, \lambda, S, \mu) = f(x) + \sum_{i=1}^{m} \lambda_i c_i(x) + \frac{1}{2\mu} \sum_{i=1}^{m} s_{ii} c_i(x)^2, \tag{4.1}$$

where the components λ_i of the vector λ are known as *Lagrange multiplier estimates*, the entries s_{ii} of the diagonal matrix S are positive scaling factors, and μ is known as the *penalty parameter*.

The constrained minimization problem (2.1)–(2.3) is now solved by finding approximate minimizers of Φ subject to the simple bounds (2.3), for a carefully constructed sequence of Lagrange multiplier estimates, constraint scaling factors and penalty parameters.

The $(k+1)$-st major iteration of AUGLG is made up of three steps. At the start of the iteration, Lagrange multiplier estimates, $\lambda^{(k)}$, constraint scaling factors, $S^{(k)}$, and a penalty parameter $\mu^{(k)}$ are given. The steps performed may be summarized, in order, as follows:

1. Test for convergence. The calculation is stopped when the projected Lagrangian gradient and the constraint violation are both small enough, that is when

$$\|x^{(k)} - P(x^{(k)} - \nabla_x L(x^{(k)}, \lambda^{(k)}), l, u)\|_\infty \leq \epsilon_l \text{ and } \|c(x^{(k)})\|_\infty \leq \epsilon_c, \tag{4.2}$$

hold for some appropriate small convergence tolerances ϵ_l and ϵ_c.

2. Use SBMIN to find an approximate minimizer, $x^{(k+1)}$, of the augmented Lagrangian function $\Phi(x, \lambda^{(k)}, S^{(k)}, \mu^{(k)})$ in the feasible box, (2.3). This approximate minimization is terminated when

$$\|x^{(k+1)} - P(x^{(k+1)} - \nabla_x \Phi(x^{(k+1)}, \lambda^{(k)}, S^{(k)}, \mu^{(k)}), l, u)\| \leq \omega^{(k)} \tag{4.3}$$

is satisfied for some tolerance $\omega^{(k)}$.

3. Update the Lagrange multiplier estimates or the penalty parameter, depending on the value of $\|c(x^{(k+1)})\|$, in addition to convergence and feasibility tolerances and constraint scaling factors (see Section 3.4.3 of [15]).

5 Algorithmic options within LANCELOT

We now discuss the most successful algorithmic options available in LANCELOT. We refer the reader to [15] for a comprehensive description of all options, and to [18] for exhaustive numerical results.

5.1 Constraint and variable scaling

LANCELOT allows the user to specify variable and constraint scalings as input parameters and the scalings are then used implicitly by the algorithms. It is also possible to construct automatic scalings independent of the minimization routines bt applying the matrix equilibration algorithm of Curtis and Reid [20] to the matrix formed by augmeting the constraint Jacobian with the objective function gradient. The resulting scale factors may then be used as scalings for the nonlinear problem (see Section 3.5 of [15]). Specifically, LANCELOT uses the implementation given by MC29 in the Harwell Subroutine Library. This automatic scaling procedure is available as an option within LANCELOT and will be referred to as the "scaling" option. Note that the stopping criteria (3.3) and (4.2) are suitably adapted to reflect scaling when this option is invoked.

5.2 Linear solvers

Most of the LANCELOT algorithmic options are related to the way in which an (approximate) minimizer of (3.11) is computed. This is hardly surprising since one expects the burden of the numerical calculation to be at this level.

5.2.1 Direct methods

Once the set $\mathcal{I}^{(k,j)}$ is determined, the nature of the quadratic model restricted to the subset of free variables is characterized by the inertia of the matrix $\bar{B}^{(k,j)}$. If all the eigenvalues of $\bar{B}^{(k,j)}$ are strictly positive, the unique minimizer of (3.11) is given as the solution to the Newton equations (3.12). In all other cases, the model (3.11) is either singular or unbounded below.

The use of a sparse multifrontal direct method to solve large-scale optimization problems has been advocated in [9]. Briefly, the matrix $\bar{B}^{(k,j)}$ is factorized using the Harwell Subroutine Library code MA27 [26], [27] as

$$\bar{B}^{(k,j)} = \bar{\Pi}^{(k,j)}\bar{L}^{(k,j)}\bar{D}^{(k,j)}\bar{L}^{(k,j)T}\bar{\Pi}^{(k,j)T}, \tag{5.1}$$

where $\bar{\Pi}^{(k,j)}$ is a permutation matrix, $\bar{L}^{(k,j)}$ is unit lower triangular and $\bar{D}^{(k,j)}$ is block diagonal with 1 by 1 and 2 by 2 diagonal blocks. The inertia of $\bar{B}^{(k,j)}$ and $\bar{D}^{(k,j)}$ are identical.

An option within LANCELOT, denoted by the "semltf" symbol, has the key property that the Newton direction is always chosen if $\bar{B}^{(k,j)}$ is positive definite and is based on the modified Cholesky methods of Schnabel and Eskow [49]. Here, we form a factorization

$$\bar{B}^{(k,j)} + \bar{E}^{(k,j)} = \bar{L}^{(k,j)}\bar{D}^{(k,j)}\bar{L}^{(k,j)T}, \tag{5.2}$$

where $\bar{L}^{(k,j)}$ is unit lower triangular, $\bar{D}^{(k,j)}$ is positive definite and diagonal, and $\bar{E}^{(k,j)}$ is positive semi-definite, diagonal and nonzero only when $\bar{B}^{(k,j)}$ is not (sufficiently) positive definite. It is straightforward to modify the Harwell subroutine MA27 to achieve this factorization. Now, the modified Newton equations

$$(\bar{B}^{(k,j)} + \bar{E}^{(k,j)})\bar{p}^{(k,j)} = -\bar{g}^{(k,j)} \tag{5.3}$$

are solved to obtain a suitable search direction. More than one cycle of improvement beyond the Cauchy point is allowed with this option.

We stress that an advantage of this technique is that $B^{(k)}$ will typically not be modified as we approach the solution to the problem. Moreover, provided the trust-region radius is sufficiently large that the Newton step (3.12) may be taken, we would also expect to take very few inner-iterations (indeed, in the nondegenerate case, one) before (3.9) is satisfied.

Another option of the package, based on factorizing $\bar{B}^{(k,j)}$ instead of $\bar{B}^{(k,j)}+\bar{E}^{(k,j)}$, is discussed in [9]. Its performance is generally inferior to that of semltf.

5.2.2 Iterative methods

In LANCELOT, the iterative method of choice is the method of conjugate gradients (see, for example, [31, Section 4.8.3], or [32, Sections 10.2 and 10.3]). Such a method attempts to find a stationary point of a quadratic function, in our case (3.11), by generating a sequence of (conjugate) search directions, $\bar{p}^{(k,j)}$. If $\bar{B}^{(k,j)}$ is not positive definite, the sequence of conjugate gradients may terminate with a direction along which the model (3.11) is either constant or unbounded below.

The convergence of the conjugate gradient method may be enhanced by preconditioning the coefficient matrix $\bar{B}^{(k,j)}$. A preconditioner is a symmetric, positive definite matrix $\bar{P}^{(k,j)}$ which is chosen to make the eigenvalues of the product $\bar{P}^{(k,j)-1}\bar{B}^{(k,j)}$ cluster around as few distinct values as possible. We have tried to supply a representative cross-section of widely used preconditioners. We recognize that users may have a better idea of a good preconditioner for their problem by allowing them to provide their own.

Band Preconditioners Many application areas give rise to problems whose Hessian matrices are banded. A band matrix is a matrix B for which $b_{ij} = 0$ for all $|i-j| > m_b$. The smallest integer m_b for which this is so is known as the semi-bandwidth of the matrix. The significant property as far as we are concerned is that, if B is positive definite, the Cholesky factors fit within the band. Moreover, clever storage schemes have been constructed to make the factorization and subsequent solutions extremely efficient (see, for example, [25, Section 10.2]) and [29, Chapter 4]). We offer a band preconditioner within LANCELOT. This works in two stages. The desired semi-bandwidth, m_b, is assumed to have been specified. The band matrix $\bar{M}^{(k,j)}$, with semi-bandwidth m_b, is chosen so that

$$\bar{M}_{il}^{(k,j)} = \bar{B}_{il}^{(k,j)} \text{ for all } |i-l| \le m_b. \tag{5.4}$$

Then, we obtain a modified Cholesky factorization of $\bar{M}_{il}^{(k,j)}$, as described in Section 5.2.1.

When $\bar{B}^{(k,j)}$ is positive definite and m_b is chosen large enough, the preconditioned conjugate gradient method will converge in a single iteration. The effect of the preconditioner in other cases has not been formally analyzed. Band preconditioners are denoted below by "band(m_b)".

Incomplete Factorization Preconditioners It is sometimes possible to construct good preconditioners for specially structured problems by either rejecting all fill-in during the factorization or by tolerating a modest amount. Such incomplete factorization preconditioners are very popular with researchers in partial differential equations and it is possible to get off-the-shelf software to form them. We include in LANCELOT the example MA31, due to Munksgaard [44], from the Harwell Subroutine Library. We denote this option by "munksg".

Full-Matrix Preconditioners Finally, as we alluded to in Section 5.2.2, if space permits and $\bar{B}^{(k,j)}$ is positive definite, one can always use a complete factorization of $\bar{B}^{(k,j)}$ as a preconditioner. However, if $\bar{B}^{(k,j)}$ is not positive definite, it is possible to use the modification (5.2) suggested in Section 5.2.1 to determine a preconditioner.

We consider two possible ways to obtain the perturbation matrix $\bar{E}^{(k,j)}$ in (5.2). The first is, as above, the modified factorization algorithm proposed by Schnabel and Eskow in [49]. We will use "seprc" to denote this strategy.

It is worthwhile noting the parallel between seprc and semltf. They both use the direct modified factorization of $\bar{B}^{(k,j)}$ to compute the Newton direction in the subspace of free variables.

They differ in that this process is stopped in seprc as soon as the only bounds encountered are trust region bounds, while the minimization may be pursued, in semltf, along the trust region boundaries.

The second is another modification of MA27 advocated by Gill, Murray, Ponceléon and Saunders in [30]. Here, the factorization (5.1) is not modified as it is formed, but it is instead computed and *then* modified. The resulting algorithmic option is denoted below by "gmpsprc".

Expanding Band Preconditioners One further possibility is to use an expanding band preconditioner. Consider the band matrix $\bar{M}^{(k,j)}$ given by (5.4), where the semi-bandwidth m_b is given by

$$m_b^{(k)} = \begin{cases} n & \text{if } \|x^{(k)} - P(x^{(k)} - g(x^{(k)}), l, u)\| \leq 10^{-2}, \\ n/2 & \text{if } 10^{-2} < \|x^{(k)} - P(x^{(k)} - g(x^{(k)}), l, u)\| \leq 10^{-1}, \\ n/5 & \text{otherwise.} \end{cases} \quad . \tag{5.5}$$

The idea is to select the semi-bandwidth $m_b^{(k)}$ at each iteration to reflect the speed and accuracy which one wants from the preconditioned conjugate gradient method. In particular, if low accuracy is required, a preconditioner with a small semi-bandwidth (such as a diagonal preconditioner) is often very effective. But if high accuracy is desired, it may be better to pick a preconditioner which is a better approximation to $\bar{B}^{(k,j)}$.

Having obtained the preconditioner, we obtain a modified Cholesky factorization of $\bar{M}_{il}^{(k,j)}$, as described in Section 5.2.1. However, unlike the band preconditioners described above, the matrix and its factorization are stored as a general sparse, rather than band, matrix.

We realize that further sophistication may be desirable but have found that this simple scheme is effective in practice. This preconditioning option will be denoted by "expband".

5.3 Derivative approximations

Further algorithmic options in LANCELOT are related to the various ways in which derivatives or their approximations are computed. However, the structure of these derivatives crucially depends on the structure of the nonlinear functions themselves. In order to derive an efficient algorithm for large-scale calculations, we first need to know a way to handle the structure typically inherent in functions of many variables.

A function $f(x)$ is said to be *group partially separable* if:

1. the function can be expressed in the form

$$f(x) = \sum_{i=1}^{n_g} g_i(\alpha_i(x)) \quad \text{where} \quad \alpha_i(x) = \sum_{j \in \mathcal{J}_i} w_{i,j} f_j(x^{[j]}) + a_i^T x - b_i \tag{5.6}$$

($\alpha_i(x)$ is known as the i-th *group*);

2. each of the *group functions* $g_i(\alpha)$ is a twice continuously differentiable function of the single variable α;

3. each of the index sets $\mathcal{J}_i$ is a subset of $\{1, \ldots, n_e\}$, where n_e is the number of nonlinear element functions;

4. each of the *nonlinear element functions* f_j is a twice continuously differentiable function of a subset $x^{[j]}$ of the variables x. Each function is assumed to have a large invariant subspace. Usually, this is manifested by $x^{[j]}$ comprising a small fraction of the variables x.

This structure is extremely general. Indeed, any function with a continuous, sparse Hessian matrix may be written in this form (see [34]). A more thorough introduction to group partial separability is given in [12]. LANCELOT assumes that the objective function $f(x)$ is of this form. When equality constraints are present, they are handled via the augmented Lagrangian and thus become part of the objective function for the subproblem given to SBMIN. Each such constraint then gives rise to the group function $\alpha^2/2\mu$, which imposes the restriction that each equality constraint has only a single group.

One of the main advantages of the group partially separable structure is that it considerably simplifies the calculation of derivatives of $f(x)$. If we consider (5.6), we see that we merely need to supply derivatives of the nonlinear element and group functions. LANCELOT then assembles the required gradient and, possibly, Hessian matrix of f from this information.

The gradient of (5.6) is given by

$$\nabla_x f(x) = \sum_{i=1}^{n_g} g_i'(\alpha_i(x)) \nabla_x \alpha_i(x), \quad \text{where} \quad \nabla_x \alpha_i(x) = \sum_{j \in \mathcal{J}_i} w_{i,j} \nabla_x f_j(x^{[j]}) + a_i. \tag{5.7}$$

Similarly, the Hessian matrix of the same function is given by

$$\nabla_{xx} f(x) = \sum_{i=1}^{n_g} g_i''(\alpha_i(x)) \nabla_x \alpha_i(x) (\nabla_x \alpha_i(x))^T + \sum_{i=1}^{n_g} g_i'(\alpha_i(x)) \nabla_{xx} \alpha_i(x), \tag{5.8}$$

where the Hessian matrix of the i-th group is

$$\nabla_{xx} \alpha_i(x) = \sum_{j \in \mathcal{J}_i} w_{i,j} \nabla_{xx} f_j(x^{[j]}). \tag{5.9}$$

Notice that the Hessian matrix is the sum of two different types of terms. The first is a sum of rank-one terms involving only first derivatives of the nonlinear element functions. The second involves second derivatives of the nonlinear elements. LANCELOT assumes that the first and second derivatives of the group functions are available. This is frequently the case in practice.

The quadratic model (3.1) uses the gradient of f by default. However, LANCELOT provides an option (which we will denote by "fdg") with which this gradient is evaluated by finite differences (see Section 3.3.2.3 of [15]). LANCELOT also offers two choices for the Hessian matrix of (3.1).

- We can calculate the true first and second derivatives of each nonlinear element and group function and use the exact Hessian $B^{(k)} = \nabla_{xx} f(x^{(k)})$.

- We can calculate the true first and second derivatives of each group function, calculate the first derivatives of the nonlinear elements but use approximations, $B_i^{[j](k)}$, to their second derivatives. We then use the approximation

$$B^{(k)} = \sum_{i=1}^{n_g} g_i''(\alpha_i(x^{(k)}))\nabla_x\alpha_i(x^{(k)})(\nabla_x\alpha_i(x^{(k)}))^T + \sum_{i=1}^{n_g} g_i'(\alpha_i(x^{(k)}))B_i^{(k)}, \tag{5.10}$$

where $B_i^{(k)}$ satisfies

$$B_i^{(k)} = \sum_{j\in\mathcal{J}_i} w_{i,j}B^{[j](k)} \tag{5.11}$$

for some suitable matrices $B^{[j](k)}$.

We strongly recommend the use of exact second derivatives whenever they are available. LANCELOT fully exploits this information. In our experience, because of the advantages of using partial separability, exact second derivatives are often available by direct calculation. Alternatively, one may use automatic differentiation tools (see [24] and [33], for instance). Using exact second derivatives is therefore the default option in the package.

However, it may sometimes be useful to approximate the matrices (5.11). LANCELOT presently uses the same type of derivative approximation for all elements. The symmetric-rank-one (SR1), Broyden-Fletcher-Goldfarb-Shanno (BFGS), Powell-symmetric-Broyden (PSB) and Davidon--Fletcher-Powell (DFP) updates are provided. We present here the results of the first two choices, which are referred to as the "sr1" and "bfgs" options respectively, since overall they were the most satisfactory. See [23], [28] and [31] for further details on these updating formulae, and Section 3.3.2.3 of [15] for a more detailed discussion of how these updates are implemented.

5.4 Accurate solution of the BQP

Finally, the last option considered in this paper allows the user to specify that the minimization of the objective function model has to be accurate within the intersection of the feasible region for the bound constraints and the trust region. In Section 3.2, we gave a general framework for obtaining a new iterate that is "better" than the generalized Cauchy point. At each stage, an approximation to the minimizer of the model is sought while some of the variables are held fixed at bounds. This set of fixed variables, $\mathcal{I}^{(k,j)}$, always includes those which were fixed at the generalized Cauchy point. In SBMIN, we also include by default all variables which encounter bounds at $x^{(k,j)}$, for $j > 0$ until the test (3.9) is satisfied. Then, optionally, we may free all variables except those which were fixed at the generalized Cauchy point and perform one or more further cycles. This optional process, denoted by "accbqp", is terminated when releasing variables does not improve the model value. This is detected when (3.9) and

$$Q(\nabla_x m^{(k)}(x^{(k,j)}), x^{(k,j)}, l, u) = Q(\nabla_x m^{(k)}(x^{(k,j)}), x^{(k,1)}, l, u) \tag{5.12}$$

are satisfied. At the start of each cycle, we also compute a new generalized Cauchy point for the model fixing the variables which were on a bound at the original Cauchy point. This recursive use of SBMIN is guaranteed to satisfy (3.9) if a sufficient number of cycles is performed.

6 The numerical tests: framework and procedure

6.1 Basic approach

There are many ways to test a complicated, general purpose code like LANCELOT, and even more ways to present the results of these tests. We now briefly discuss the fundamental choices we made when designing our tests and which influence our treatment of the results in this paper.

Our first decision was to test and report on a large number of test cases. In our experience, this is essential for a true assessment of reliability and performance, as smaller test sets are more likely to introduce unwanted bias.

Our second choice was to limit the comparison to reliability and efficiency aspects. Other potential criteria, such as ease of use, accuracy of solutions and availability, did not seem to be as significant when testing a single package.

Our final decision was to present both aggregate and relatively disaggregate performance measures. Specifically, we chose the average performance as our aggregate measure but also report on the ranking of the algorithmic variants in five performance classes (excellent, good, satisfactory, fair and poor). The performance was averaged across many problems which differ, sometimes substantially, in size, nonlinearity or type of constraints.

Although the choice of the average sometimes obscures the performance of algorithmic variants on the easier problems in comparison with the harder ones, it nevertheless seems to correspond to our intuitive appraisal of the variants after our experience of running extensive tests. This is especially true when one also considers the associated rankings, as we hope is apparent later in this section. Furthermore, there is little agreement within the optimization community on alternative aggregate measures.

The authors of course realize that this scheme is not the only one that can be defended. It is however hoped that it provides a sufficient basis to make the testing discussed in this section of interest.

6.2 The test problems

The numerical tests with LANCELOT that we are about to describe were conducted using the Constrained and Unconstrained Testing Environment (CUTE) collection of nonlinear test problems (see [4]). This collection contains a large number of nonlinear optimization problems of various sizes and difficulty, representing both academic and real world applications. As the title of the collection implies, constrained and unconstrained examples are included. For our tests, we have used 624 instances of unconstrained (or bound constrained) problems and 319 instances of constrained problems. These 943 instances are derived from 398 *different* problems, the additional examples being determined by varying the dimension. It is of course undesirable to describe all these examples in the present paper. It will suffice to say that our test set covers, amongst others,

- the “Argonne test set” [42], the Testpack report [5], the Hock and Schittkowski collection [36], the Dembo network problems (see [21]), the Moré-Toraldo quadratic problems [43], the Toint-Tuyttens network model problems [52],

- most problems from the PSPMIN collection [50][1],
- problems inspired by the orthogonal regression report by Gulliksson [35],
- some problems from the Minpack-2 test problem collection[2] [2], [3] and from the second Schittkowski collection [47],
- a number of original problems from various application areas.

We present some of the problems characteristics in Figures 3 and 4 and in Table 1.

- Figure 3 shows the distribution of the problems' *dimensions*.
- Figure 4 illustrates the distribution of the ratio m/n, where m is the total number of general equality and inequality constraints. The higher this ratio, the more constrained the problem. Only constrained problems ($m > 0$) are considered in this statistic.
- Table 1 reports the number of problems for which a given characteristic lies in one of five possible intervals $[0, 0.2]$, $(0.2, 0.4]$, $(0.4, 0.6]$, $(0.6, 0.8]$ and $(0.8, 0.1]$. Four characteristics are examined. These are
  - the *relative nonlinearity of the objective function*, that is the ratio

$$\nu_{\rm obj} \stackrel{\rm def}{=} \frac{\text{number of nonlinear groups in the objective}}{\text{number of groups in the objective}}, \tag{6.1}$$

    where the groups are defined in (5.6) and where a group is declared nonlinear if it contains at least one nontrivial nonlinear element function or if its associated group function is nonlinear;
  - the *relative nonlinearity of the constraints*, i.e.

$$\nu_{\rm cons} \stackrel{\rm def}{=} \frac{\text{number of nonlinear constraints}}{\text{number of constraints}}, \tag{6.2}$$

    where the bounds have been excluded from the denominator;
  - the proportion n_b/n of variables subject to *bound constraints*;
  - the *proportion of equality constraints*, that is of the ratio

$$\gamma \stackrel{\rm def}{=} \frac{\text{number of equality constraints}}{m}. \tag{6.3}$$

[1] Some trivial problems were skipped and also problems for which different local minima were known.
[2] The problems that we could reconstruct from the data given in the report.

	$[0, \frac{1}{5}]$	$(\frac{1}{5}, \frac{2}{5}]$	$(\frac{2}{5}, \frac{3}{5}]$	$(\frac{3}{5}, \frac{4}{5}]$	$(\frac{4}{5}, 1]$
ν_{obj}	48	0	13	2	880
ν_{cons}	139	5	20	8	193
n_b/n	573	25	38	14	293
γ	99	5	7	13	241

Table 1: Further problems characteristics

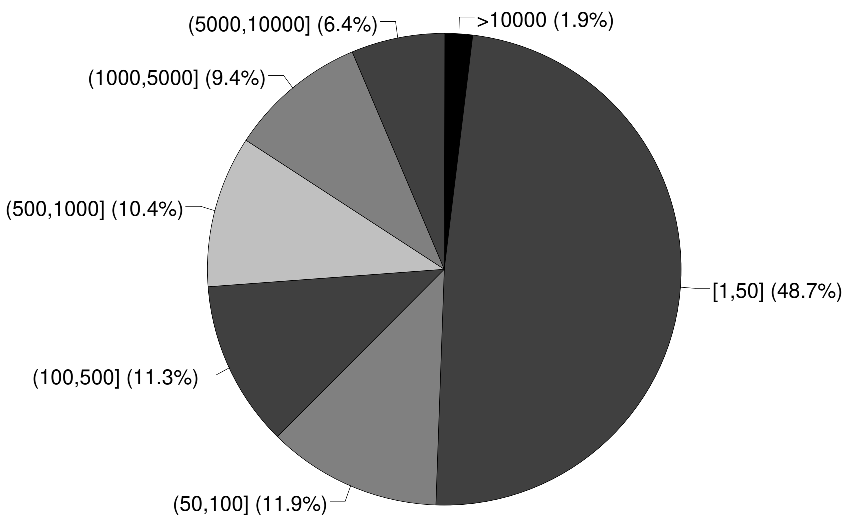

Figure 3: Distribution of problem dimensions

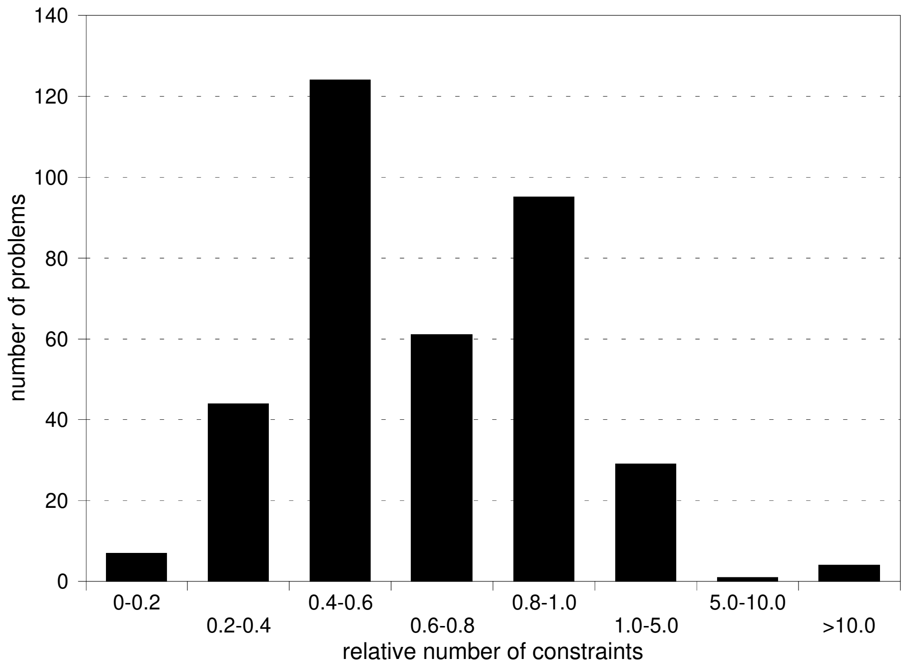

Figure 4: Distribution of the relative number of constraints m/n

We note the following points.

- The majority of the problems are not very large. However, we recall that testing LANCELOT on small problems is meaningful because the package is also intended to solve small-scale problems. Furthermore, the classes of larger problems are far from empty, and we note the presence of examples with more than 15000 variables.

- Most large problems tend to have a somewhat regular structure. As a result, most groups in these problems tend to be structurally similar. This is noticeable in the distribution of the relative nonlinearity of the objective function and constraints, where either most or very few, if any, groups are nonlinear. The same phenomenon is also observed for the proportion of bounded variables which tends to be either very low or close to one.

- There are very few problems involving considerably more general constraints than variables. Many of the problems arise as nonlinear systems of equations, while a fair proportion have approximately half as many constraints as variables. We nevertheless note the presence of problems where the number of constraints is substantially greater than n.

Bearing in mind that one of the LANCELOT's features is its ability to handle large problems, we also selected, amongst the 943 tests problems, all problems in more than 500 variables. This subset contains 268 problems, that is 28.1% of the complete set. The algorithmic conclusions corresponding to the complete problem set and the subset are interesting to compare because only the latter depends more obviously on the way in which the problem structure is handled.

6.3 The testing procedure

Before detailing the testing procedure, we recall the **default** algorithmic choice for LANCELOT:

- no variable/constraint scaling,

- a conjugate gradient linear solver is used with a banded preconditioner of semi-bandwidth 5 (band(5)),

- analytical second derivatives are used, as well as analytical gradients,

- an exact Cauchy point calculation is used,

- the ℓ_∞-norm is used for defining the trust region.

For our tests we also set the maximum number of iterations to 1000, the maximum cpu-time to 18000 seconds, the initial trust region radius to 1.0 and disabled all printing. The accuracy requirements were set to the LANCELOT defaults, that is $\epsilon_l = \epsilon_c = 10^{-5}$. We also turned the derivative checker on but chose to ignore its warning messages. Of course, all derivatives were checked before the actual tests. For the sake of completeness, the default LANCELOT specification file is given in Figure 5.

We next considered basic variants of this default choice, that is a choice of algorithmic options that differs in just one instance from the default. The basic variants are

```
BEGIN
   check-derivatives
   ignore-derivative-bugs
   exact-second-derivatives-used
   bandsolver-preconditioned-cg-solver-used 5
   exact-cauchy-point-required
   trust-region-radius 1.0D+0
   maximum-number-of-iterations 1000
   print-level -1
   start-printing-at-iteration 0
   stop-printing-at-iteration 1000
END
```

Figure 5: The LANCELOT default specification file

noprc: no preconditioner is used within the conjugate gradient solver, i.e., $\bar{P}^{(k,j)} = I$ (see Section 5.2.2),

band(0): a diagonal preconditioner is used for the conjugate gradient solver (see Section 5.2.2),

band(1): a tridiagonal preconditioner is used for the conjugate gradient solver (see Section 5.2.2),

band(10): a 21-diagonals preconditioner is used for the conjugate gradient solver (see Section 5.2.2),

expband: an expanding band preconditioner is used for the conjugate gradient solver (see Section 5.2.2),

munksg: an incomplete factorization preconditioner is used for the conjugate gradient solver (see Section 5.2.2),

seprc: a full matrix preconditioner using the Schnabel-Eskow modified factorization is used for the conjugate gradient solver (see Section 5.2.2),

gmpsprc: a full matrix preconditioner using the Gill-Murray-Ponceléon-Saunders modified factorization is used for the conjugate gradient solver (see Section 5.2.2),

semltf: a modified multifrontal direct linear solver is used (see Section 5.2.1),

sr1: the symmetric-rank-one quasi-Newton formula is used to approximate second derivatives (see Section 5.3),

bfgs: the Broyden-Fletcher-Goldfarb-Shanno quasi-Newton formula is used to approximate second derivatives (see Section 5.3),

scaling: automatic variable/constraint scaling is used, with scalings computed at the starting point (see Section 5.1),

accbqp: an accurate solution to the BQP is sought (see Section 5.4).

To this list we added the **fdg** variant which uses finite difference approximation to gradients and the symmetric-rank-one quasi-Newton formula for approximating second derivatives (see Section 5.3). These variants and the default gives a list of 15 different algorithmic choices.

Note that the variants **scaling**, **semltf**, **expband**, **seprc**, **gmpsprc** and **munksg** depend on code from the Harwell Subroutine Library. Their use is therefore only possible for users with a suitable licence. As a consequence, they could not be selected as defaults for the package.

We then tested all of these 15 choices on the complete problem set, which amounted to running $15 \times 943 = 14145$ test cases. A total of 5658 additional cases were also run to evaluate the less successful options not discussed in this paper. These tests were performed on two Digital DECstations 5000/200 with 48 MBytes of memory, using the Ultrix f77 compiler (version 3.0-2) without optimization[3]. The cpu-times on both machines were checked for consistency.

7 The numerical tests: results and discussion

It is of course impossible to detail the complete set of results obtained on nearly fifteen thousand test cases in a journal article. We will therefore present and discuss summaries and averages extracted from these results. A technical report containing the complete results is however available [18].

7.1 Reliability

7.1.1 General assessment

We first present results on the reliability and failures on the 15 algorithmic variants. Results are given in Table 2, where the occurrences of the LANCELOT exit conditions are reported for all 15 variants in the case of the complete test set and the selected subset. The column headings correspond to the following possible situations.

succ: The minimization was successfully terminated.

stall: The minimization could not progress further, the stepsize being smaller than relative machine precision. Not all runs terminated in this way are unsuccessful from the user's point of view, as it happens in several cases that the algorithm is "stalled" very near the solution.

infs: The package could not find a feasible point for the considered problem.

mem: The workspace required for handling the considered problem is larger than three million double precision and/or three million integer numbers.

[3] An error in the Fortran optimizer of this version prevented its use with the package.

iters: The run was terminated after 1000 iterations without convergence.

cpu: The run was terminated after 18000 cpu seconds (5 hours) without convergence.

error: An arithmetic error occurred in the subprograms evaluating the problem dependent functions and/or derivatives. This typically occurs when the iterates produced by the algorithm "wander off" the part of the feasible region where the values of the objective and constraints are of manageable size.

Note that the algorithmic variants have been ordered, in this table and subsequent figures, to allow for an easy comparison of all preconditioned iterative techniques (themselves ordered by increasing semi-bandwidth, from noprc to gmpsprc) and of these techniques with a direct method (semltf). The default variant has been isolated for easier reference. The two quasi-Newton variants (sr1 and bfgs) are then presented next to each other, followed by the more disparate options (scaling, accbqp and fdg).

Variant	Complete set (943 problems)							Selected subset (265 problems)						
	succ	stall	infs	mem	iters	cpu	error	succ	stall	infs	mem	iters	cpu	error
default	865	11	7	0	31	26	3	231	3	0	0	9	22	0
noprc	850	6	13	0	35	36	3	221	1	1	0	14	28	0
band(0)	844	21	12	0	30	33	3	220	4	0	0	12	29	0
band(1)	862	14	9	0	30	26	2	232	1	0	0	10	22	0
band(10)	864	13	10	0	27	26	3	228	5	0	0	10	22	0
expband	866	7	8	3	25	25	9	225	1	2	2	7	24	4
munksg	851	7	13	2	28	39	3	221	0	1	1	5	37	0
seprc	878	11	7	2	22	21	2	239	1	1	1	3	20	0
gmpsprc	861	9	7	9	26	21	10	222	2	1	8	5	21	6
semltf	812	5	11	2	65	43	5	197	0	4	1	20	42	1
sr1	865	17	9	0	25	24	3	231	5	0	0	8	21	0
bfgs	796	15	12	0	87	23	10	207	5	1	0	25	21	6
scaling	806	46	21	0	29	27	14	206	12	9	0	11	24	3
accbqp	858	14	7	0	15	49	0	221	1	0	0	3	40	0
fdg	787	19	11	0	91	28	7	203	6	0	0	28	25	3

Table 2: Successes and failures per variant

From this table, we can draw the following conclusions.

1. The reliability of the default algorithmic choice is good (91.7% on the complete problem set), nearly identical to that of the expanding band preconditioner variant expband (91.8% on the complete set), and only marginally surpassed by that of the Schnabel-Eskow preconditioner used in conjunction with conjugate gradients (93.1% on the complete set).

The default choice of a semi-bandwidth of 5 also seems to provide excellent reliability amongst the banded preconditioners, both for the complete problem set and the subset.

2. The robustness of the best partitioned quasi-Newton scheme (SR1) appears to be excellent compared with the use of exact second derivatives, even for large problems. This approach therefore confirms its potential amongst quasi-Newton techniques for large-scale applications, at least from the reliability point of view.

3. The scaling variant does not show a globally improved robustness compared with the default. It is the variant most often stalled. This illustrates the difficulty of designing good automatic scaling procedures. It is however worthwhile to note that the scaling variant did solve badly scaled problems where other variants failed. Keeping such an option available therefore seems to be of some value, but it should not be used as a default.

4. It is somewhat surprising that the gmpsprc variant has a significantly lower reliability than the other full matrix preconditioner seprc on the selected test set (and hence also on the complete set).

   One of the reasons is that the Gill-Murray-Ponceléon-Saunders technique seems to generate more arithmetic errors and to run out of memory more often than the Schnabel-Eskow method. On closer analysis, the occurrence of overflow with the Gill-Murray-Ponceléon-Saunders modified factorization seems to be due to numerical difficulties for some singular or nearly singular matrices. The observed problems are probably caused by the low value of the threshold under which eigenvalues are perturbed to ensure positive definiteness of the preconditioning matrix. According to [30], this threshold is set to the machine precision. A posteriori experiments with the threshold raised to (machine precision)$^{3/4}$ (as is used in the Schnabel-Eskow modification) indicate that the overflow problems can be avoided. These observations are consistent with the conclusions of Schlick in [48], where she observes that enforcing a small modification $\bar{E}^{(k,j)}$ in (5.3) might not be beneficial for fast convergence.

   A second reason that gmpsprc more often fails because of excessive memory requirements. This difference between gmpsprc and seprc is due to a possibly larger fill-in in the Gill-Murray-Ponceléon-Saunders technique caused by changes in the pivoting order to preserve stability. As the Schnabel-Eskow modified factorization maintains positive definiteness of the matrix during the factorization, no such changes are necessary.

5. We also note the substantial gain in robustness obtained by using a full matrix factorization as preconditioner. The variant seprc is indeed significantly more reliable than its direct counterpart semltf.

6. The accbqp variant, being more computationally intensive, runs out of time most often. If we assume that some of the truncated computations would effectively terminate successfully, given additional time, this variant probably ranks as the most reliable, but at the expense of substantial additional effort.

7. There does not seem to be a real robustness advantage in using an incomplete factorization preconditioner (munksg) over a banded one for the problems of our test set. One must however notice that discretized continuous problems do not constitute a majority of the tested cases. As incomplete factorizations have earned their good reputation on such problems, one could probably expect a better performance of the munksg variant if the proportion of discretized problems increased.

8. Using finite difference approximations for the first derivatives of the problem's function somewhat reduces the reliability of the package, but fdg still managed to solve 83% of the problems, a quite acceptable score.

We conclude our general reliability analysis by noting that 919 of the 943 problems were solved by at least one variant, while 617 were solved by all of them. This indicates an excellent reliability of the complete package (97.5%) on our large test problem collection, but also the relative lack of robustness for certain algorithmic variants. Amongst the 265 problems of the subset, 254 (95.8%) were solved by at least one variant and 139 (52.5%) by all variants, indicating that the overall good performance does not deteriorate much when only the larger problems are considered.

7.1.2 Further discussion of the unsolved problems

We now comment on the 24 problems in the complete test set that were not solved, within the given iterations and time limits, by any variant. These problems are listed in Table 3, where we also indicate some of their characteristics. These characteristics may provide some insight into why LANCELOT found them difficult.

We first note that 15 of these problems could be solved by the package, but their solution required a number of iterations exceeding 1000 and/or a total cputime over 5 hours. It was also sometimes necessary to reduce the initial value of the penalty parameter below its default value or to combine the features of two of the variants. A further five problems could be "nearly solved" in the sense that a point was found which didn't satisfy the criticality conditions (4.2) within the required tolerance of 0.00001, but was essentially the problem's solution. Amongst these latter problems, one finds constrained cases (HS84, HS99, HS116) where the penalty parameter was reduced by LANCELOT to very small values (below 10^{-7}), which caused subproblem ill-conditioning and slow overall progress. More details are available in Appendix A on the specific options used and timings for the solution of these 20 problems.

Four problems remain that could not be solved by LANCELOT. These are HS99EXP, NGONE and LUBRIF (in 149 and 749 variables). HS99EXP is a variant on the 99th problem in the Hock and Schittkowski collection [36]. NGONE is a two-dimensional geometry problem involving a very large number of inequality constraints. Finally, LUBRIF is the elasto-hydrodynamic lubrification nonlinear complementarity problem described in [37] and [41], which is notoriously difficult to solve by pure nonlinear optimization techniques. It is interesting to note that the difficulty of solving these problems seems to arise not from their size, but rather from their nonlinearity and/or degeneracy.

Problem name	n	m	very nonlinear	degenerate	badly conditioned	solved by LANCELOT
AGG	163	488		$\surd$		nearly
CHEMRCTA	5000	5000	$\surd$		$\surd$	yes
CORKSCRW	4497	3500	$\surd$	$\surd$		yes
CORKSCRW	8997	7000	$\surd$	$\surd$		yes
ERRINBAR	18	9	$\surd$			yes
HS84	5	3			$\surd$	nearly
HS93	6	2	$\surd$			yes
HS99	7	2	$\surd$		$\surd$	nearly
HS103	7	6	$\surd$			nearly
HS116	13	15			$\surd$	nearly
HS99EXP	31	21	$\surd$		$\surd$	no
LEWISPOL	6	9	$\surd$			yes
LUBRIF	149	100	$\surd$			no
LUBRIF	749	500	$\surd$			no
MARATOSB	2	0			$\surd$	yes
NGONE	497	31373		$\surd$	$\surd$	no
NOMSQRT	529	0			$\surd$	yes
NOMSQRT	1024	0			$\surd$	yes
OBSTCLAE	15625	0				yes
OPTMASS	606	505		$\surd$	$\surd$	yes
OPTMASS	1206	1005		$\surd$	$\surd$	yes
OPTMASS	3006	2505		$\surd$	$\surd$	yes
SVANBERG	5000	5000				yes
TENBARS4	18	9	$\surd$			yes

Table 3: 24 difficult problems for LANCELOT

7.1.3 Convergence to different critical points

If we now wish to compare the *relative* efficiency of these variants, the only runs that can really be compared for each variant are those that successfully produce a well specified critical point. We therefore remove from our comparison all runs for which the variant under consideration converged to a critical point whose associated objective function value does not correspond (within 0.001%) to the lowest critical value found for the problem. In total, 617 problems from the complete set and 139 from the subset were successfully solved (according to this criterion) by all variants. *In what follows we confine our attention to these problems.* Figure 6 indicates how many problems per variant gave rise to different local optima.

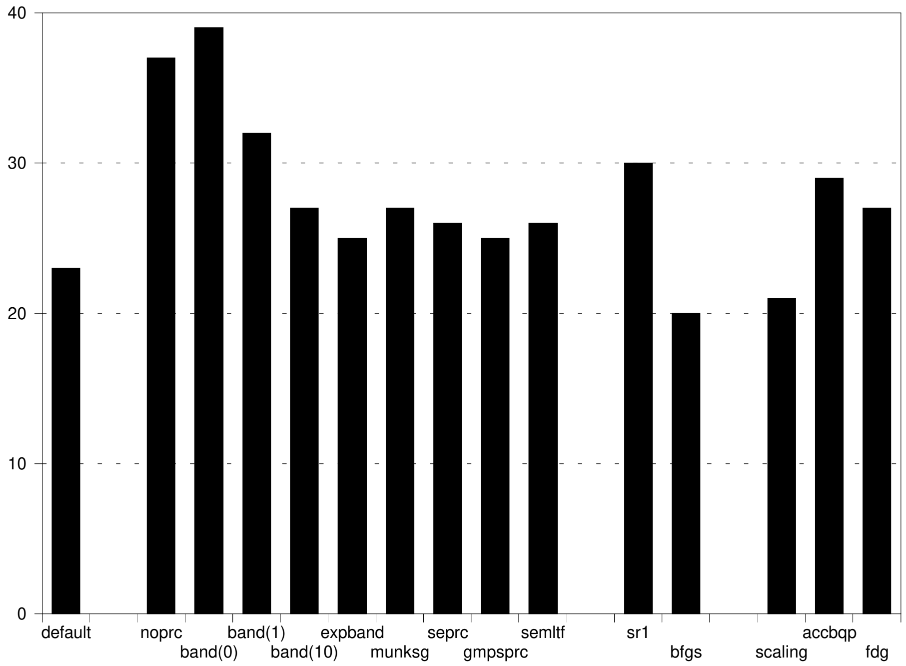

Figure 6: Number of sucessful runs to alternative critical points per variant

7.2 Number of minor iterations

We now start comparing the algorithmic variants for relative efficiency, and first turn our attention to the number of minor iterations required by the variants to find the solution. We recall that the problem's objective function and constraints (if any) are evaluated exactly once per such iteration for all variants except `fdg`, where additional evaluations are required to estimate the first derivatives. We also note that LANCELOT only recomputes the value of the objective function's and constraints' elements whose variables have been modified since the last evaluation: this sometimes implies a substantial reduction in the computational effort required for such an evaluation.

Figure 7 shows the average number of iterations required for solution (on the problems that were successfully solved by all variants). Figure 8 presents an overall view of the relative ranking of the variants based on the number of iterations. All 15 variants were ranked (where best means ranked first and failed means not ranked at all) for each of these 617 problems. We then counted the number of times that a given variant had a given rank. We finally clustered the obtained rankings in classes[4] (excellent: ranks 1 to 3, good: 4 to 6, satisfactory: 7 to 9, fair: 10 to 12, poor: 13 to 15) which are then displayed in a bar chart. For instance, the darker area in the bar corresponding to the seprc variant indicates that this variant is excellent (that is, amongst the three best) for 454 problems, an impressive performance.

Figures 9 and 10 present the corresponding averages and rankings for the 139 successfully solved problems of subset.

We now draw some conclusions from these figures.

1. We immediately note the good results obtained by the semltf variant for the complete problem set. Although less reliable than its preconditioning counterpart seprc, it seems to require fewer iterations to converge when it does so, but the difference is admittedly marginal.

2. The accbqp variant requires amongst the least number of minor iterations. This is not a surprise, since this variant puts more work in an iteration and one therefore expects that less of these more costly iterations are needed.

3. The seprc variant also seems to require fewer iterations on average than the other full factorization preconditioner variant gmpsprc.

4. The default variant appears to be reasonably efficient in terms of minor iterations amongst the tested variants, although not amongst the best. It is however remarkable that it is the variant whose behaviour is least often in the worst ranking variants, as is shown by the size of the "poor" class (in Figure 8). This last characteristic is also displayed by the seprc and accbqp variants on the subset (see Figure 10).

5. Amongst the quasi-Newton variants, the sr1 variant appears to require substantially fewer iterations and function evaluations than its bfgs counterpart.

6. The need to estimate gradients by finite differences also causes the number of iterations to increase, as can be deduced by comparing the performance of the fdg and sr1 variants.

[4] Of course, these classess should be understood as an indication of performance only relative to that of other LANCELOT variants.

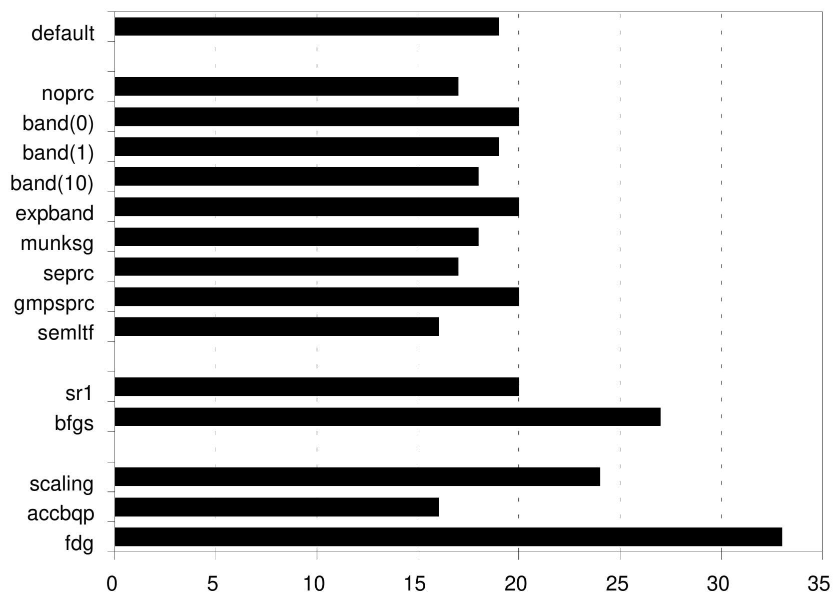

Figure 7: Average number of iterations for 617 problems solved by all variants

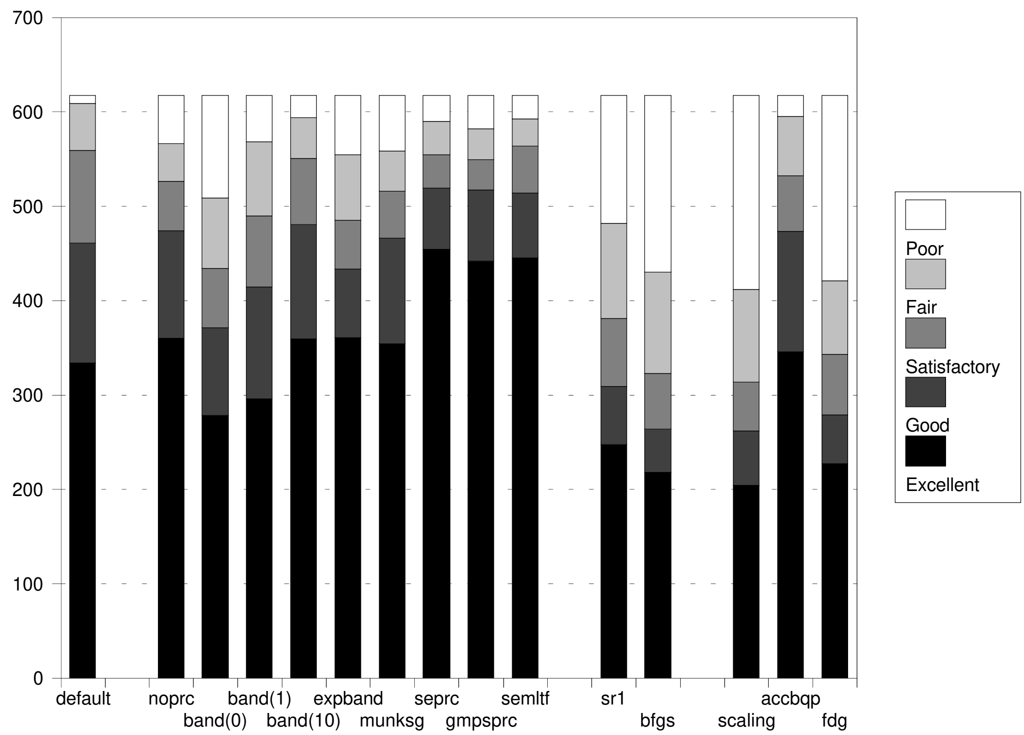

Figure 8: Ranking by iterations for 617 problems solved by all variants

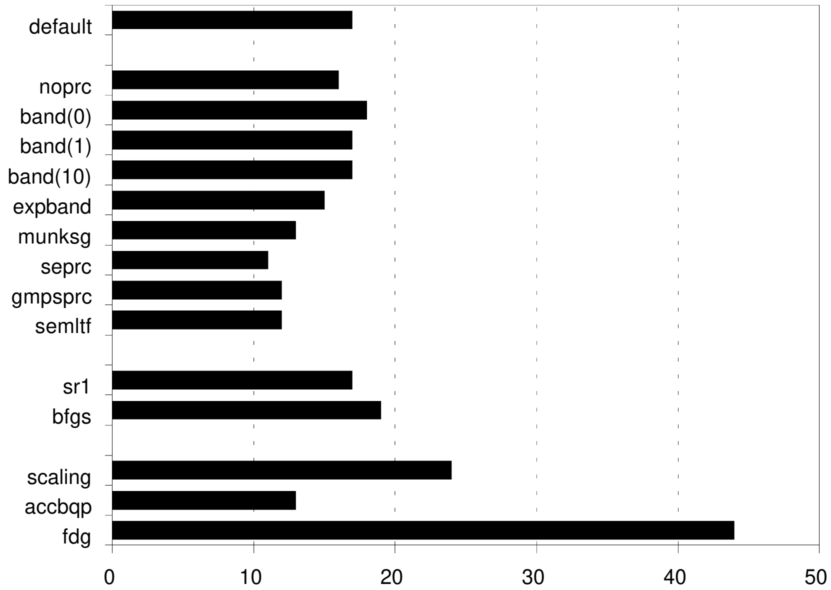

Figure 9: Average number of iterations for 139 problems of the subset solved by all variants

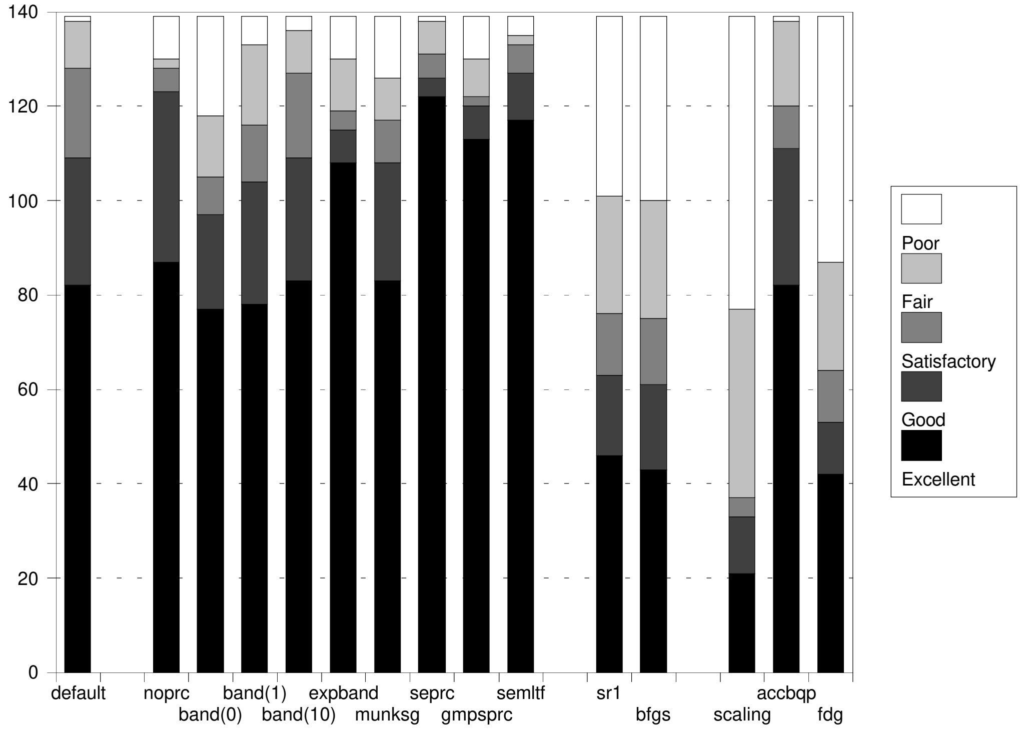

Figure 10: Ranking by iterations for 139 problems of the subset solved by all variants

7.3 Number of cg-iterations

We now examine the total number of conjugate gradient iterations per minor iteration required to solve the test problems by each variant using an iterative linear solver. What is really compared in this section is the overall effect of the various preconditioners and, to some extent, the conditioning of the Hessian matrices generated by the different variants.

Figure 11 shows the average "fraction of cg-iterations" per minor iteration and per problem variable, the average being taken on the 617 problems in the complete set. This fraction indicates how many cg-iterations were performed on average, compared to the problem size. Since conjugate gradients are expected to terminate in at most n cg-iterations on a linear system of size n, the reported measures are all between zero and one. We note that the measure is approximate for two reasons. Firstly, the number of free variables at any given iterations can be lower than the number of problem variables. Secondly, the conjugate gradient iterations may have to be restarted when bounds are encountered. We nevertheless believe that the comparison amongst variants is instructive. Figure 12 presents the same measure taken on the 139 problems of the subset.

1. As anticipated, the full-matrix preconditioners are the clear winners in terms of number of cg-iterations. This behaviour is even more marked on the problem subset. Note that expband can be considered to a full-matrix preconditioner in a vicinity of the problem's solution, because of (5.5).

2. Another expected conclusion is that the quality of the preconditioner seems to increase with the semibandwidth, when a band preconditioner is used. This is clearly apparent when examining the results for noprc, band(0), band(1), default (which is equivalent to band(5)) and band(10). Why the expband variant does not really fit in this framework for the complete problem set is not clear, but may be because of the non-asymptotic behaviour.

3. The incomplete factorization preconditioner munksg shows excellent behaviour. Indeed its performance is nearly comparable to that of the full-matrix variants on the complete problem set.

4. Solving the BQP accurately of course requires more cg-iterations, and we observe this effect when comparing default and accbqp. This is especially noticeable on the problems of the subset, because they are larger.

5. The scaled variant scaling is somewhat less efficient than the default unscaled variant on the complete problem set, which is again an indication that scaling should not be applied blindly to every problem. Its performance is however improved on the larger problems of the subset.

6. The quasi-Newton approximations to the second derivatives do not seem to generate matrices that are, on average, worse conditioned than their analytic counterparts, as is shown by the comparable values for the default, bfgs and sr1 variants. The fact that gradients are

estimated by differences in fdg does not seem to considerably impact the conditioning of the Hessian either, as can be seen by comparing this variant with sr1.

7. The reported measures are typically smaller for the subset than for the complete problem set. This is anticipated as conjugate gradient solvers often require a number of iterations that is more dependent on conditioning and eigenvalue distribution than on system size. Increasing size therefore produce lower measures if one assume that the larger problems have an eigenvalue structure that is, on average, not worse than that of smaller ones.

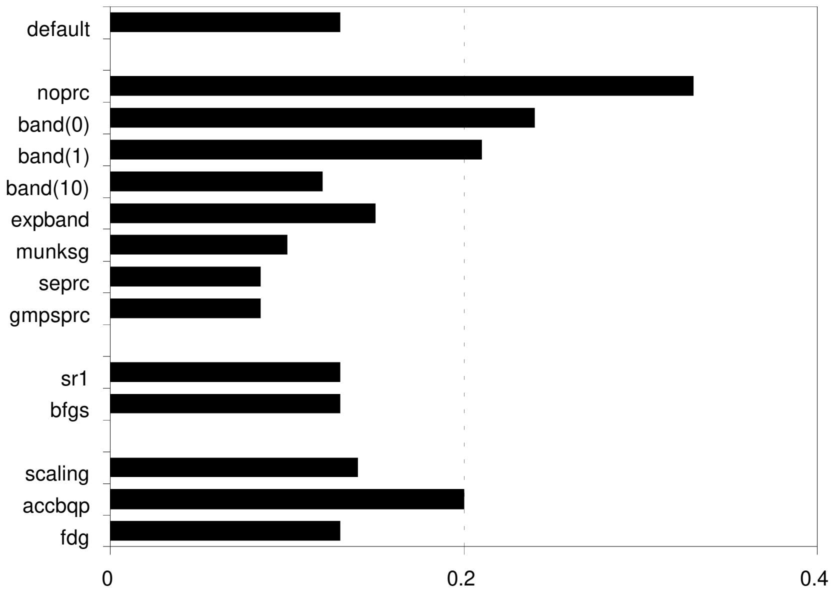

Figure 11: Average fraction of cg-iterations per minor iteration for 617 problems solved by all variants

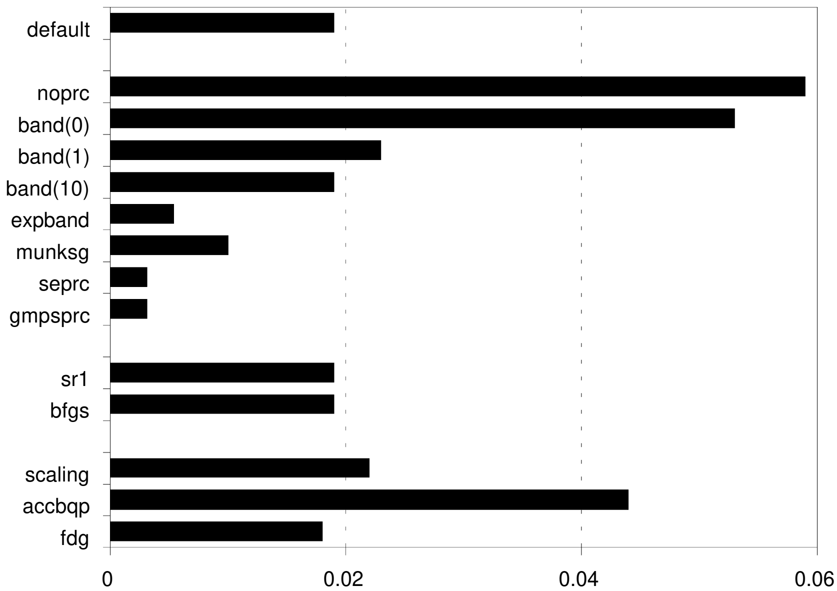

Figure 12: Average fraction of cg-iterations per minor iteration for 139 problems of the subset solved by all variants

7.4 Computational effort

We next compare our 15 algorithmic variants on the basis of their requirements in cpu-time. Figure 13 shows the average cpu-time (in seconds) required for solution, the average being taken on the 617 problems in the complete set that were successfully solved by all methods. Figure 14 presents a overall view of the relative ranking of the variants based on cpu-time. This figure was constructed in the same way as Figure 8. Figure 15 and 16 present the corresponding average and ranking results for the selected subset of test problems.

Some interesting conclusions can be drawn from these figures.

1. The results obtained by the semltf variant are noteworthy. Although its ranking compared with the other variants is amongst the best, its average performance is the poorest. This is caused by the poor behaviour of the variant on a few large unconstrained problems where the Hessian matrix is indefinite in the early iterations. In these cases, the strategy to move along a direction of negative curvature, as in the iterative variants, seems more appropriate than repeatedly calculating a modified Newton direction in smaller and smaller subspaces (corresponding to faces of the trust region), each time recomputing a suitably modified factorization. It should however be noted that, despite its strong effect on average scores, this behaviour occurs rarely, as can be seen from the comparative ranking of the variant.

2. The default and band(1) variants appear to be the fastest on average. Their ranking confirms this excellent behaviour, both for the complete problem set and the subset.

3. The full-matrix preconditioned variant seprc appears to be quite efficient on average, compared to other preconditioners.

4. The scaling variant seems to be somewhat handicapped by the additional work required to compute and handle the variable and constraints "typical" values. Its average performance is indeed somewhat worse than that of the default variant. Its ranking is comparable that that of default on the complete set, but worse on the subset.

5. The relatively acceptable performance of the noprc variant seems to indicate that most of the test problems are reasonably well scaled.

6. The behaviour of banded preconditioners with varying semi-bandwidth is worth a comment. We already noted the good performance of the tridiagonal preconditioner (band(1)) and the default (band(5)), both on the complete problem set and on the subset. The band(10) variant uses more cpu-time as the advantage of improved preconditioning is offset by the higher price of the preconditioner. The good performance of the expanding band variant expband, compared with band(10), seems to be due to the general sparse storage scheme used, which is preferable to the band storage for matrices with higher bandwidth.

7. The more costly iterations of accbqp clearly cause the relatively large average cpu-time of this variant on the complete problem set. However, as the expense of cpu-time is mostly

confined to large problems, and as there are comparatively few such problems in the complete test set, the method ranks reasonably highly. This observation is strengthened by the relatively poor ranking of this variant for the large problems of the subset.

8. Amongst the quasi-Newton variants, **sr1** appears to be the most efficient. Its ranking is also consistently better than that of **bfgs**.

9. The work involved in approximating the gradients by differences causes **fdg** to be slower than **sr1** on average. This effect is enough to cause the relative ranking of **fdg** to fall substantially behind that of **sr1**.

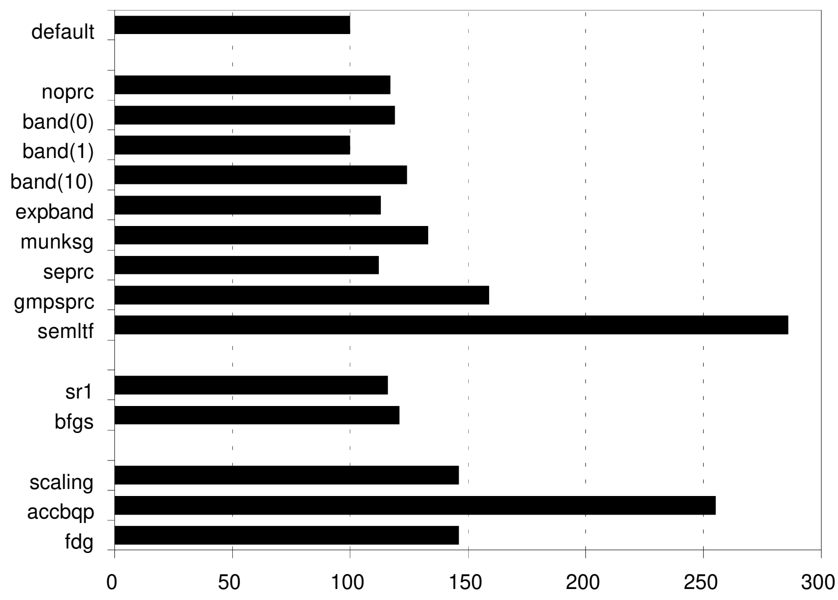

Figure 13: Average cpu-time for 617 problems solved by all variants

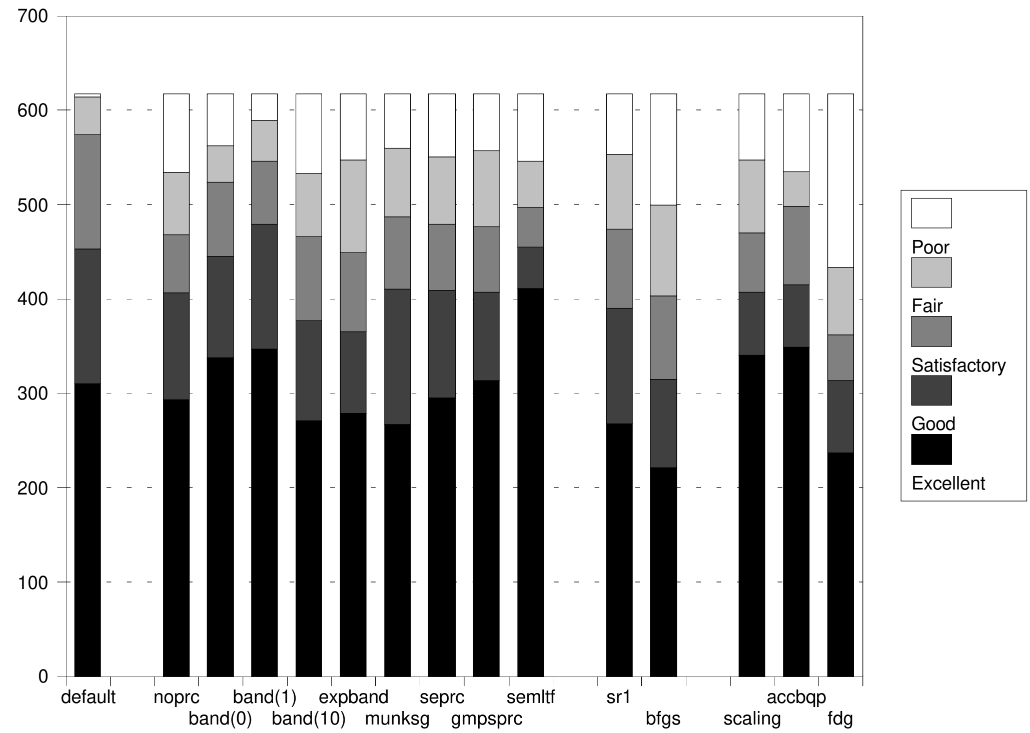

Figure 14: Ranking by cpu-time for 617 problems solved by all variants

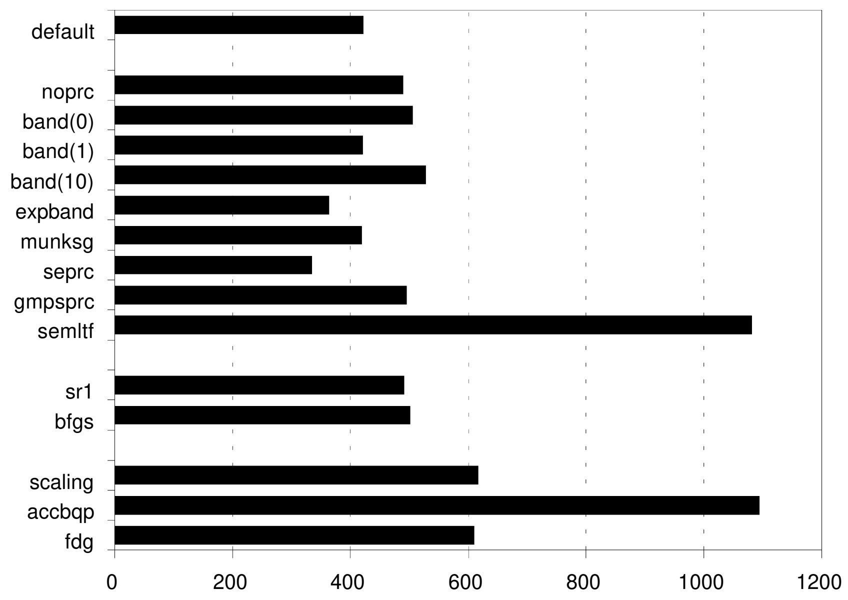

Figure 15: Average cpu-time for 139 problems of the subset solved by all variants

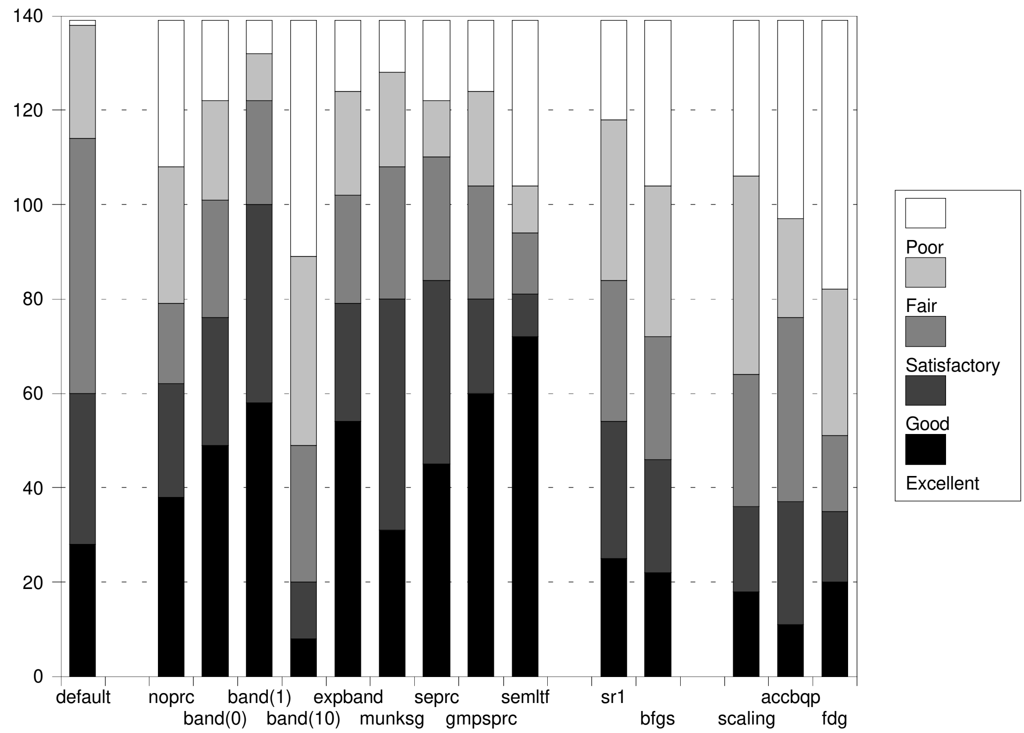

Figure 16: Ranking by cpu-time for 139 problems of the subset solved by all variants

7.5 Additional comments

We did not discuss above the relative number of unsuccessful iterations for each variant. This number is *on average* below one per problem for each variant. It seems to indicate that the trust region management used in LANCELOT is adequate for handling a large class of nonlinear problems.

Besides its algorithmic choices, LANCELOT allows the user to select a number of non-algorithmic options, such as element and group derivative checking, level of printout and frequency at which intermediate data is saved for a possible subsequent restart. None of these options has a significant impact on the overall execution time of the package. The only observable increase in cpu-time occurs when a very detailed printout is required at every iteration of a large scale problem. As one would expect, this effect is slightly more marked for constrained cases, where the details of the major iterations have to be printed as well.

We finally indicate some weak points of LANCELOT (Release A) that we have observed in examining the detailed runs, but that cannot be inferred directly from the summaries presented above.

1. When the number of inequality constraints is large compared with the number of variables, the package currently adds slack variables to transform all inequalities into equalities, which results in a substantial increase in the effective problem size. Although convergence is usually obtained, the computational effort can be relatively large compared with method that use inequality constraints directly (see [14], for instance). The authors are well aware of this aspect of their implementation, and have recently given in [19] a method to overcome this difficulty, although it has not been incorporated in the software.

2. No special provision is made in the present code for linear network constraints, or even for linear constraints. Again, LANCELOT seems to be robust in that convergence is obtained for problems with this kind of structure, but special purpose algorithms are often much more efficient (see [1], [22], [46], [52] and [53], for instance).

3. The ability of the generalized Cauchy point to determine the correct active set is disappointing in practice. In many examples, the correct active set is actually found in the conjugate gradient or direct matrix improvement beyond the GCP, at considerable cost. Although the GCP asymptotically identifies the correct active set as predicted by the theory (see [10], for instance), this is often at the end of a long calculation. A strategy treating the bounds through barrier functions (as proposed in [14]) might therefore be a useful alternative.

8 Conclusions

We first described the algorithms contained in Release A of the LANCELOT package for large-scale nonlinear optimization. We also analyzed the user-selectable variants. We finally presented and discussed the results of extensive numerical tests with these variants.

The main conclusions of these tests, as far as the package is concerned, are as follows.

1. The package is capable of solving a wide class of nonlinear optimization problems, including many large-scale examples.

2. The package is relatively more efficient for unconstrained and bound constrained problems and for generally constrained problems for which the number of constraints does not substantially exceed the problem dimension.

3. The default algorithmic choice in the package appears to be both reliable and acceptably efficient, compared to other variants.

4. Some algorithmic choices (automatic scaling, accurate solution of the inner BQP) should not be used automatically, but may provide excellent behaviour on some harder problems.

Beyond these conclusions relative to the LANCELOT package, our tests also reveal the following points of more general interest.

1. The difficulty of solving a problem is more often linked to its degree of nonlinearity than to its size.

2. The use of direct factorization appears to be most robust when used as preconditioners for a conjugate gradient linear solver.

3. The use of exact second derivatives is recommended whenever available. However, the partitioned symmetric-rank-one technique, as embedded in the package, gives very satisfactory reliability and efficiency (compared to other variants) when analytic second derivatives are not available.

4. When analytical first derivatives are not available, finite difference approximations to the gradients coupled with SR1 quasi-Newton Hessian updating is an acceptably robust technique, even for large problems.

5. The use of full factorizations appears to be reliable for the class of problems analyzed in this paper. It is however expected that this technique would appear less promising if even larger problems were considered. In contrast, banded preconditioners would probably extend well to larger problems.

Of course, only continued experience with LANCELOT will really show its strengths and weaknesses. The authors very much hope to be informed by the users of the package of the (undoubtedly many) aspects where improvements are possible. Progress is expected to come both from the point of view of the algorithms (see [14] and [17], for example) and from that of the implementation details themselves. It is also clear that further comparisons with other packages are desirable, in particular to better assess the efficiency of LANCELOT in a wider context. The ongoing comparison with MINOS should thus provide useful additional conclusions, when completed. At this stage, the results discussed above certainly offer the hope that the software will prove useful in the increasingly important arena of large-scale nonlinear optimization.

9 Acknowledgements

The authors are indebted to Michel Bierlaire and Didier Burton, whose help was invaluable in producing the graphics of this paper and in maintaining the workstation network during the 8 months of nearly uninterrupted computation required for obtaining the results presented here. Nick Gould is grateful to CERFACS, Toulouse, France, for the facilities which made some of this research possible. Thanks are also due to Ingrid Bongartz, Marc Breitfeld and Peihuang Chen for detecting and correcting mistakes in our test problems. Ingrid Bongartz furthermore provided valuable comments on a draft of the manuscript. The authors are grateful for the support provided for their travels across the Atlantic by NATO grant 890867. Finally, the authors acknowledge the contribution of the editor, associate editor and of a referee, whose comments and reports have been very stimulating.

References

[1] D. P. Ahlfeld, R. S. Dembo, J. M. Mulvey, and S. A. Zenios, "Nonlinear programming on generalized networks", *ACM Transactions on Mathematical Software* 13(4) (1987) 350–367.

[2] B. M. Averick, R. G. Carter, and J. J. Moré, "The Minpack-2 test problem collection (preliminary version)", Technical Report ANL/MCS-TM-150, Argonne National Laboratory (Argonne, USA, 1991).

[3] B. M. Averick and J. J. Moré, "The Minpack-2 test problem collection", Technical Report ANL/MCS-TM-157, Argonne National Laboratory (Argonne, USA, 1991).

[4] I. Bongartz, A. R. Conn, N. Gould, and Ph. L. Toint. "CUTE: Constrained and Unconstrained Testing Environment", *ACM Transactions on Mathematical Software* (to appear) (1994).

[5] A. G. Buckley, "Test functions for unconstrained minimization", Technical Report CS-3, Computing Science Division, Dalhousie University (Dalhousie, Canada, 1989).

[6] J. V. Burke and J. J. Moré, "On the identification of active constraints", *SIAM Journal on Numerical Analysis* 25(5) (1988) 1197–1211.

[7] J. V. Burke, J. J. Moré, and G. Toraldo, "Convergence properties of trust region methods for linear and convex constraints", *Mathematical Programming, Series A* 47(3) (1990) 305–336.

[8] P. H. Calamai and J. J. Moré, "Projected gradient methods for linearly constrained problems", *Mathematical Programming* 39 (1987) 93–116.

[9] A. R. Conn, N. Gould, M. Lescrenier, and Ph. L. Toint, "Performance of a multifrontal scheme for partially separable optimization" In *Advances in Optimization and Numerical Analysis, Proceedings of the Sixth Workshop on Optimization and Numerical Analysis, Oaxaca, Mexico* (Kluwer Academic Publishers, Dordrecht, NL, 1994) 79–96.

[10] A. R. Conn, N. Gould, and Ph. L. Toint, "Global convergence of a class of trust region algorithms for optimization with simple bounds", *SIAM Journal on Numerical Analysis* 25 (1988)433–460, 1988. [See also same journal 26 (1989) 764–767.]

[11] A. R. Conn, N. Gould, and Ph. L. Toint, "Testing a class of methods for solving minimization problems with simple bounds on the variables" *Mathematics of Computation* 50 (1988) 399–430.

[12] A. R. Conn, N. Gould, and Ph. L. Toint, "An introduction to the structure of large scale nonlinear optimization problems and the LANCELOT project", In R. Glowinski and A. Lichnewsky, editors, *Computing Methods in Applied Sciences and Engineering* (SIAM, Philadelphia, USA, 1990) 42–51.

[13] A. R. Conn, N. Gould, and Ph. L. Toint, "A globally convergent augmented Lagrangian algorithm for optimization with general constraints and simple bounds", *SIAM Journal on Numerical Analysis* 28(2) (1991) 545–572.

[14] A. R. Conn, N. Gould, and Ph. L. Toint, "A globally convergent Lagrangian barrier algorithm for optimization with general inequality constraints and simple bounds", Technical Report 92/07, Department of Mathematics, FUNDP (Namur, Belgium, 1992).

[15] A. R. Conn, N. Gould, and Ph. L. Toint, "LANCELOT: *a Fortran package for large-scale nonlinear optimization (Release A)*", Springer Series in Computational Mathematics, volume 17 (Springer Verlag, Berlin, 1992).

[16] A. R. Conn, N. Gould, and Ph. L. Toint, "On the number of inner iterations per outer iteration of a globally convergent algorithm for optimization with general nonlinear equality constraints and simple bounds", In D.F Griffiths and G.A. Watson, editors, *Proceedings of the 14th Biennal Numerical Analysis Conference Dundee 1991* (Longmans, 1992) 49–68.

[17] A. R. Conn, N. Gould, and Ph. L. Toint, "Convergence properties of minimization algorithms for convex constraints using a structured trust region", Technical Report 92/11, Department of Mathematics, FUNDP (Namur, Belgium, 1992).

[18] A. R. Conn, N. Gould, and Ph. L. Toint, "Intensive numerical tests with LANCELOT (Release A): the complete results", Technical Report 92/15, Department of Mathematics, FUNDP (Namur, Belgium, 1992).

[19] A. R. Conn, N. Gould, and Ph. L. Toint, "A note on exploiting structure when using slack variables", *Mathematical Programming, Series A* 67(1) (1994) 89–97.

[20] A. R. Curtis and J. K. Reid, "On the automatic scaling of matrices for Gaussian elimination", *Journal of the Institute of Mathematics and its Applications* 10 (1972) 118–124.

[21] R. S. Dembo, "A primal truncated-newton algorithm with application to large-scale nonlinear network optimization", Technical Report 72, Yale School of Management (Yale University, New Haven, USA, 1984).

[22] R. S. Dembo, "The performance of NLPNET, a large scale nonlinear network optimizer", *Mathematical Programming, Series B* 26 (1986) 245–249.

[23] J. E. Dennis and R. B. Schnabel, *Numerical methods for unconstrained optimization and nonlinear equations* (Prentice-Hall, Englewood Cliffs, USA, 1983).

[24] L. C. W. Dixon, "On automatic differentiation and continuous optimization", In E. Spedicato, editor, *Algorithms for Continuous Optimization: the State of the Art* (Kluwer Academic Publishers, Dordrecht, NL, 1994) 501–513.

[25] I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct methods for sparse matrices* (Clarendon Press, Oxford, UK, 1986).

[26] I. S. Duff and J. K. Reid, "MA27: A set of Fortran subroutines for solving sparse symmetric sets of linear equations", Report R-10533, AERE Harwell Laboratory (Harwell, UK, 1982).

[27] I. S. Duff and J. K. Reid, "The multifrontal solution of indefinite sparse symmetric linear equations", *ACM Transactions on Mathematical Software* 9(3) (1983) 302–325.

[28] R. Fletcher, *Practical Methods of Optimization* (J. Wiley and Sons, Chichester, second edition, 1987).

[29] A. George and J. W.-H. Liu, *Computer solution of large sparse positive definite systems* (Prentice-Hall, Englewood Cliffs, USA, 1981).

[30] P. E. Gill, W. Murray, D. B. Ponceléon, and M. A. Saunders, "Preconditioners for indefinite systems arising in optimization", *SIAM Journal on Matrix Analysis and Applications* 13 (1992) 292–311.

[31] P. E. Gill, W. Murray, and M. H. Wright, *Practical Optimization* (Academic Press, London and New York, 1981).

[32] G. H. Golub and C. F. Van Loan, *Matrix Computations* (Johns Hopkins University Press, Baltimore, second edition, 1989).

[33] A. Griewank, "Computational differentiation and optimization", In J. R. Birge and K. G. Murty, editors, *Mathematical Programming: State of the Art 1994* (The University of Michigan, Ann Arbour, USA, 1994) 102–131.

[34] A. Griewank and Ph. L. Toint, "On the unconstrained optimization of partially separable functions", In M. J. D. Powell, editor, *Nonlinear Optimization 1981* (Academic Press, London and New York, 1982) 301–312.

[35] M. Gulliksson, *Algorithms for Nonlinear Least Squares with Applications to Orthogonal Regression*, PhD thesis, Institute of Information Processing, University of Umeå (S-901 87 Umeå, Sweden, 1990).

[36] W. Hock and K. Schittkowski, *Test Examples for Nonlinear Programming Codes*, Lectures Notes in Economics and Mathematical Systems 187 (Springer Verlag, Berlin, 1981).

[37] M. M. Kostreva, "Elasto-hydrodynamic lubrification: a non-linear complementarity problem", *International Journal for Numerical Methods in Fluids* 4 (1984) 377–397.

[38] M. Lescrenier, "Convergence of trust region algorithms for optimization with bounds when strict complementarity does not hold", *SIAM Journal on Numerical Analysis* 28(2) (1991) 476–495.

[39] A. Lewis, private communication, 1990.

[40] J. J. Moré, "Trust regions and projected gradients", In M. Iri and K. Yajima, editors, *System Modelling and Optimization*, Lecture Notes in Control and Information Sciences 113 (Springer Verlag, Berlin, 1988) 1–13.

[41] J. J. Moré, "A collection of nonlinear model problems", Technical Report ANL/MCS-P60-0289, Argonne National Laboratory (Argonne, USA, 1989).

[42] J. J. Moré, B. S. Garbow, and K. E. Hillstrom, "Testing unconstrained optimization software", *ACM Transactions on Mathematical Software* 7(1) (1981) 17–41.

[43] J. J. Moré and G. Toraldo, "On the solution of large scale quadratic programming problems with bound constraints", *SIAM Journal on Optimization* 1(1) (1991) 93–113.

[44] N. Munksgaard, "Solving sparse symmetric systems of linear equations by preconditioned conjugate gradients", *ACM Transactions on Mathematical Software* 6 (1980) 206–219.

[45] B. A. Murtagh and M. A. Saunders, "MINOS 5.1 USER'S GUIDE", Technical Report SOL83-20R, Department of Operations Research, Stanford University (Stanford, USA, 1987).

[46] A. Sartenaer, "A class of trust region methods for nonlinear network optimization problems", *SIAM Journal on Optimization* (to appear) (1994).

[47] K. Schittkowski, "*More Test Examples for Nonlinear Programming Codes*", Lecture notes in economics and mathematical systems 282 (Springer Verlag, Berlin, 1987).

[48] T. Schlick, "Modified Cholesky factorizations for sparse preconditioners", *SIAM Journal on Scientific and Statistical Computing* 14(2) (1993) 424–445.

[49] R. B. Schnabel and E. Eskow, "A new modified Cholesky factorization", *SIAM Journal on Scientific and Statistical Computing* 11 (1991) 1136–1158.

[50] Ph. L. Toint, "Test problems for partially separable optimization and results for the routine PSPMIN", Technical Report 83/4, Department of Mathematics, FUNDP (Namur, Belgium, 1983).

[51] Ph. L. Toint, “Global convergence of a class of trust region methods for nonconvex minimization in Hilbert space”, *IMA Journal of Numerical Analysis* 8 (1988) 231–252.

[52] Ph. L. Toint and D. Tuyttens, “On large scale nonlinear network optimization”, *Mathematical Programming, Series B* 48(1) (1990) 125–159.

[53] Ph. L. Toint and D. Tuyttens, “LSNNO: a Fortran subroutine for solving large scale nonlinear network optimization problems”, *ACM Transactions on Mathematical Software* 18(3) (1992) 308–328.

Appendix A

The purpose of this section is to report the details of what algorithmic options were used to solve the problem discussed in Section 7.1.2, when this proved possible, as well as the corresponding number of minor iterations, cg-iterations and cpu-time. These details are given in Table 4, where the two columns serve to identify the problem, the third indicating the basic variant used and the fourth what further modification of this variant have been specified, if any. When the name of another variant is mentioned in this column, this means that the features of both the variants of columns three and four are used. For instance, CHEMRCTA was solved by using the seprc variant (Schnabel-Eskow preconditioning) with constraints and variables scaling, as specified in the variant scaling. Columns five, six and seven indicate the associated number of minor iterations, cg-iterations and the cpu-time requested for LANCELOT to terminate. The last column refers to the following comments.

(1) The standard seprc variant fails because the step generated by the algorithm is insignificantly small at a point where the projected gradient is not small, but this seems to be due to problem scaling and degeneracy. The optimal objective function produced by MINOS (−35991767.29) is slightly worse than that produced by LANCELOT (−35991766.35) but its solution is closer to feasibility (constraint violation of the order of 10^{-23} for MINOS and of the order of 10^{-7} for LANCELOT).

(2) LANCELOT fails because the step becomes too short. This exit corresponds to the “stall” situation described in Section 7.1.1. However, the optimal solution appears to be found.

We do not report here on the four problems, HS99EXP, NGONE and the two cases of LUBRIF, that we have not managed so far to solve using LANCELOT.

These results of Table 4 show that in most cases (AGG, CHEMRCTA, ERRINBAR, HS84, HS93, HS99, HS103, HS116 LEWISPOL, MARATOSB, NONMSQRT (n = 529), OPTMASS (n = 606 and 1206) and TENBARS4) a satisfactory solution could be computed in reasonable time. Only the six remaining problems requested substantially more computational effort for their solution.

Problem name	n	basic variant	additional specification	minor iterations	cg iterations	cpu-time (secs)	note
AGG	163	seprc		266	76817	6013	(1)
CHEMRCTA	5000	seprc	scaling	35	64	760	
CORKSCRW	4497	seprc		122	60078	40033	
CORKSCRW	8997	seprc		126	215502	264573	
ERRINBAR	18	default		4053	30881	570	
HS84	5	default		74	403	7	(2)
HS93	6	default	$\mu_0 = 10^{-2}$	44	75	4	
HS99	7	default	$\mu_0 = 10^{-5}$	32	28	3	(2)
HS103	7	default	accbqp and $\mu_0 = 10$	2518	34780	460	(2)
HS116	13	seprc	$\mu_0 = 10^{-5}$	4108	15932	458	(2)
LEWISPOL	6	default	$\mu_0 = 10^{-10}$	18	22	2	
MARATOSB	2	default		1715	1286	144	
NOMSQRT	529	seprc	accbqp	225	4421	5239	
NOMSQRT	1024	seprc	accbqp	510	274355	559376	
OBSTCLAE	15625	default		5	7450	39588	
OPTMASS	606	seprc	$\mu_0 = 10^{-4}$	3744	6268	2731	
OPTMASS	1206	seprc	$\mu_0 = 10^{-4}$	13320	25033	18644	
OPTMASS	3006	seprc	$\mu_0 = 10^{-4}$	44309	76825	149512	
SVANBERG	5000	default		100	12985	46730	
TENBARS4	18	default		2690	21031	372	

Table 4: Detailed algorithmic choices and performance for the problems solved using nonstandard variants