Benchmarking of bound-constrained optimization software

Anke Tröltzsch (CERFACS)
joint work with S. Gratton (CERFACS) and Ph.L. Toint (University of Namur)

March 01, 2007

CERFACS working note: WN/PA/07/143

Abstract
In this report we describe a comparison of different algorithms for solving nonlinear optimization problems with simple bounds on the variables. Moreover, we would like to come out with an assessment of the optimization library DOT used in the optimization suite OPTALIA at Airbus for this kind of problems.

Keywords
bound constrained optimization software, nonlinear optimization, line search method, trust region method, interior point method

Introduction

0.1 Aim of the study

Airbus is using an optimization library called Design Optimization Tools (DOT) to perform the minimizations occurring in aircraft design. DOT is mainly based on line search techniques and the purpose of this study is to compare it with more recent line search solvers and with other solvers based on different approaches such as trust region and interior point method. In this work, we focus on the case where function and gradient are available, but where the Hessian of the problem, when needed, has to be approximated using e.g. quasi-Newton updates.

0.2 Short outline

In Chapter 1 can be found a description of the mathematical background of the considered bound-constrained optimization problem. In Chapter 2 we describe the algorithms implemented in the considered solvers together with their main properties. The methodology we followed in this comparison is described in Chapter 3. Particular attention is paid on the choice of a relevant set of test cases and on the use of unified stopping criteria. In Chapter 4, the performance profiles associated with the numerical experiments are presented, and some conclusions on the behavior of the different codes are drawn.

In these experiments, we have set up an testing environment which is very easy to adapt to solve generally constrained optimization problems when the solver is already implemented. Thus, we made some additional experiments with two of the solvers that are also able to solve optimization problems with general constraints. The results of this comparison are given in Appendix A.

1 Mathematical Background

In this report, we describe a comparison of different well-known algorithms for solving nonlinear optimization problems with simple bounds on the variables.

1.1 Bound-constrained optimization problem

Problem statement

The bound-constrained optimization problem can be written as

$$\begin{array}{c} \min_{x \in R^n} f(x_i) \\ \text{subject to } l_i \leq x_i \leq u_i \end{array} \tag{1}$$

where f is a nonlinear function with n variables, l_i and u_i are representing vectors of lower and upper bounds on the variables x_i.

Feasible set and active set

The bound-constrained optimization problem (1) is a special case of the constrained optimization problem considered e.g. in (Nocedal and Wright, 1999), (Gill et al., 1981) and (Fletcher, 1987). Here the constraint set, or feasible set, is

$$\mathcal{C} = \{x_i \in R^n | x_i - l_i \geq 0, u_i - x_i \geq 0\}. \tag{2}$$

$x \in R^n$ is called a feasible point for the optimization problem if $x_i \geq l_i$ and $x_i \leq u_i$.

An index i is called active if $x_i - l_i = 0$ or $u_i - x_i = 0$, what means that the i-th component of x lays on its lower or upper bound, respectively. The index i is called inactive if the strict inequalities $x_i - l_i > 0$ and $u_i - x_i > 0$ are satisfied. The active set at x, which is denoted by $\mathcal{A}(x)$, consists of the active indices,

$$\mathcal{A}(x) = \{i | x_i - l_i = 0 \ \text{ or } \ u_i - x_i = 0\}. \tag{3}$$

A component x_i for which $l_i = x_i = u_i$ satisfies both equations and is called a fixed variable. The set of inactive indices is denoted by $\mathcal{I}(x)$. Components x_i for which $i \in \mathcal{I}(x)$ are called free variables.

In some methods the reduced gradient and the reduced Hessian matrix are used, which contains, respectively, first and second order derivative information of f with respect to the free variables. The reduced gradient $\nabla_R f(x)$ has components

$$[\nabla_R f(x)]_i = \begin{cases} 0, & \text{if } i \in \mathcal{A}(f) \\ [\nabla f(x)]_i, & \text{otherwise.} \end{cases} \tag{4}$$

The reduced Hessian $\nabla^2_R f(x)$ has entries

$$[\nabla^2_R f(x)]_{ij} = \begin{cases} 0, & \text{if } i \in \mathcal{A}(f) \ \text{ or } \ j \in \mathcal{A}(f) \\ [\nabla^2 f(x)]_{ij}, & \text{otherwise.} \end{cases} \tag{5}$$

Criticality measure

In the unconstrained case, the necessary condition for optimality at x^* is

$$\nabla f(x^*) = 0 \tag{6}$$

and the Hessian $\nabla^2 f(x^*)$ is positive semidefinite if f is twice continuously differentiable. Condition (6) is called first order necessary condition and a point satisfying that condition is called a critical point.

In the bound-constrained case there must be used a different measure for criticality which has to incorporate the bounds. We then get following extension of (6)

$$\nabla f(x^*) - \sum_{i=1}^{n} (\lambda_i^* \nabla_x (x_i^* - l_i) + \mu_i^* \nabla_x (u_i - x_i^*)) = 0 \tag{7}$$

with

$$\begin{aligned} \lambda_i^* \geq 0 \quad &\text{and} \quad \mu_i^* \geq 0, \\ (x_i^* - l_i) \geq 0 \quad &\text{and} \quad (u_i - x_i^*) \geq 0, \\ \lambda_i^* (x_i^* - l_i) = 0 \quad &\text{and} \quad \mu_i^* (u_i - x_i^*) = 0. \end{aligned} \tag{8}$$

Equations (7) and (8) together are known as the Karush-Kuhn-Tucker (KKT) conditions and λ, $\mu \in R^n$ are called the Lagrange multipliers of the problem. These conditions can be expressed in dependency on the value of x_i. In fact, if $l_i \leq x_i \leq u_i$ it follows that $\lambda_i = 0 = \mu_i$ and therefore it has to be $[\nabla f(x)]_i = 0$. If $x_i = l_i < u_i$ it follows that $\mu_i = 0$ and therefore $[\nabla f(x)]_i = \lambda_i \geq 0$. Finally from $x_i = u_i > l_i$ follows $\lambda_i = 0$ and it must be $[\nabla f(x)]_i = -\mu_i \leq 0$. Gathered these three cases, one can see that the KKT conditions from (7) and (8) are equivalent to

$$\begin{aligned} &[\nabla f(x^*)]_i = 0, \quad \text{when} \quad l_i < x_i^* < u_i \\ &[\nabla f(x^*)]_i \geq 0, \quad \text{when} \quad x_i^* = l_i \\ &[\nabla f(x^*)]_i \leq 0, \quad \text{when} \quad x_i^* = u_i. \end{aligned} \tag{9}$$

These conditions correspond exactly to the definition of the projected gradient at the solution point x^*. The projected gradient $\nabla_{\mathcal{C}} f(x)$ is the mapping from the feasible set $\mathcal{C}$ into R^n with components

$$[\nabla_{\mathcal{C}} f(x)]_i = \begin{cases} [\nabla f(x)]_i, & \text{if} \quad l_i < x_i < u_i, \\ \min\{0, [\nabla f(x)]_i\}, & \text{if} \quad x_i = l_i \\ \max\{0, [\nabla f(x)]_i\}, & \text{if} \quad x_i = u_i. \end{cases} \tag{10}$$

It is defined that x^* is a critical point for the bound-constrained optimization problem if

$$\nabla_{\mathcal{C}} f(x^*) = 0. \tag{11}$$

If f is twice Lipschitz continuously differentiable and x^* is a solution of the bound-constrained optimization problem, the reduced Hessian $\nabla_R^2 f(x^*)$ in (5) is positive semidefinite. This is called the second order necessary condition for optimality.

The following statement provides an alternative characterization of a critical measure for bound-constrained optimization problems. If f is continuously differentiable, then a point $x^* \in \mathcal{C}$ is a critical point for the bound-constrained optimization problem if and only if

$$x^* = \mathcal{P}_{\mathcal{C}}(x^* - \lambda \nabla f(x^*)) \tag{12}$$

for all $\lambda \geq 0$ where $\mathcal{P}_{\mathcal{C}}(x)$ denotes the projection of x onto $\mathcal{C}$. The ith component of the operator $\mathcal{P}_{\mathcal{C}}(x)$ is

$$[\mathcal{P}_{\mathcal{C}}(x)]_i = \begin{cases} l_i, & \text{if} \quad x_i \leq l_i, \\ x_i, & \text{if} \quad l_i < x_i < u_i, \\ u_i, & \text{if} \quad x_i \geq u_i. \end{cases} \tag{13}$$

2 Brief Description of the methods

All methods in the tests use either a line search or a trust region framework to guarantee global convergence to a local solution. Lancelot B is using trust region method. The other solvers DOT, L-BFGS-B, TN-BC and IPOPT are using line search methods where IPOPT uses a primal dual interior point filter line search method. In this Chapter, the main features of these globalisation approaches are presented and we show how they are combined in the considered solvers.

2.1 Algorithmic components of trust region methods

In the trust region algorithmic strategy the information known about the objective function f is used to construct a model function m_k whose behaviour near the current point x_k is similar to that of the actual f. The search for a minimizer of m_k is restricted to some region around x_k, called the trust region. This is the region where the model m_k can be trusted because the model may not be a good approximation of f for x^* far from x_k.

A step in the bound-constrained case is found by solving the subproblem

$$\begin{array}{c} \min_s m_k(s) \\ \text{subject to } \|s\| \leq \Delta_k \text{ and } l \leq x_k + s \leq u, \end{array} \tag{14}$$

where $\Delta_k > 0$ is a scalar called the trust region radius which is varied as the iteration proceeds. In general $\|\cdot\|$ is defined to be the Euclidean norm, but it may also be elliptical or box-shaped. In the latter case it is defined by the l_∞-norm and this yields the equivalent subproblem

$$\begin{array}{c} \min_s m_k(s) \\ \text{subject to } \max(l - x_k, \Delta_k e) \leq s \leq \min(u - x_k, \Delta_k e), \end{array} \tag{15}$$

where e is the unit vector.

The model m_k is usually defined as a quadratic function of the form

$$m_k(s) = f_k + \nabla f_k^T s + \frac{1}{2} s^T B_k s, \tag{16}$$

where f_k and ∇f_k are the function and gradient values at the current point x_k and $s_k = x_{k+1} - x_k$ denotes the step. The matrix B_k is either the Hessian $\nabla^2 f_k$ or an approximation to it.

If the model is generally reliable, producing good steps and accurately predicting the behaviour of the objective function along these steps, the size of the trust region is steadily increased to allow longer steps to be taken. If the solution of the subproblem does not produce a sufficient decrease in f, it can be concluded that the model is an inadequate representation of the objective function over the current trust region, so it has to be shrinked

and the subproblem has to be solved again. In general, the step direction changes whenever the size of the trust region is altered (Nocedal and Wright, 1999).

Generalized Cauchy point

The minimization of the quadratic model (16) within the feasible box and the trust region is usually done in two stages. In the first stage it has to be obtained the so called generalized Cauchy point. This point is very important in order to satisfy the global convergence theory of bound-constrained trust region methods. Thus, 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 the iteration is no larger than that at the generalized Cauchy point. A detailed convergence analysis can be found in (Conn et al., 1988a). The generalized Cauchy point x_k^C is defined as the first local minimizer of

$$m_k(\mathcal{P}(x_k - tg_k, l, u)) \tag{17}$$

within the trust region. That is, x_k^C is the first local minimizer of the quadratic model along the piecewise linear arc defined by projecting the steepest descent direction onto the feasible region, subject to the trust region constraint.

Somehow surprising, it is not necessary that the generalized Cauchy point be calculated exactly. To guarantee convergence it is sufficient to identify an adequate approximation to such a point. Different strategies can be found in (Burke et al., 1990), (Calamai and Moré, 1987) and (Moré, 1988).

In the second stage of the minimization of (16) it is attempted to further reduce the model. This is done by minimizing the model function within the subspace where those variables which lie on their bounds at the generalized Cauchy point are fixed. Then it can be tried to reduce the quadratic model by changing the values of the remaining free variables while restricting them to the trust region bound and the feasible set $\mathcal{C}$. This may be done using either a direct or an iterative method.

Trust region basic algorithm

The first issue to establish this algorithm is how to choose the trust region radius Δ_k at each iteration. This choice is based on the agreement of the model function m_k and the objective function f at the previous iteration as measured by the ratio

$$\rho_k = \frac{f(x_k) - f(x_k + s_k)}{f(x_k) - m_k(s_k)} \tag{18}$$

where the numerator is called the actual reduction and the denominator is the predicted reduction. So, if there is good agreement between the model

m_k and the function f over this step ($\rho_k \approx 1$), then it is safe to increase Δ_k for the next iteration. If ρ_k is positive but not close to 1, the trust region is not altered. But, if the agreement is poor (ρ_k small or negative), then Δ_k is decreased. In the case, ρ_k is negative, the new objective value $f(x_k + s_k)$ is greater than the current value $f(x_k)$ and the current step s_k can not be accepted. The algorithm can be written as in (Conn et al., 2000)

Initialize $x_0 \in \mathcal{C}, \Delta_0$ and the constants $\eta_1, \eta 2, \gamma_1, \gamma_2$
for $k = 1, ..., maxit$ **do**
  Define a model m_k in $\mathcal{C} \cap \mathcal{B}_k$
  Obtain s_k by sufficiently reducing m_k such that $x_k + s_k \in \mathcal{C} \cap \mathcal{B}_k$
  Evaluate ratio ρ_k
  if $\rho_k \geq \eta_1$ **then**
    $x_{k+1} = x_k + s_k$ {Update solution}
  else
    $x_{k+1} = x_k$
  end if
  if $\rho_k \geq \eta_2$ **then**
    $\Delta_{k+1} \in [\Delta_k, \infty)$ {Update trust region radius}
  else if $\rho_k \in [\eta_1, \eta_2)$ **then**
    $\Delta_{k+1} \in [\gamma_2 \Delta_k, \Delta_k]$
  else
    $\Delta_{k+1} \in [\gamma_1 \Delta_k, \gamma_2 \Delta_k]$
  end if
  $k := k + 1$
end for

Algorithm 1: Trust region method

where $0 < \eta_1 < \eta_2 < 1$ and $0 < \gamma_1 < \gamma_2 < 1$.

Solver: Lancelot B

This solver is written by Nicholas I. M. Gould, Andrew Conn and Philippe L. Toint and it is developed to solve unconstrained, bound-constrained and generally constrained optimization problems. It must be remarked, that it is not free for commercial use.

If generally constraint problems are to be solved, Lancelot B uses a sequential augmented Lagrangian method within a trust region framework. In the bound-constrained case there is no need to build up a Lagrangian and Lancelot B reduces to a trust region truncated Newton method. That is, Algorithm 1 is used to ensure global convergence to a local minimum. For solving the quadratic subproblem, the generalized Cauchy point as described above is computed and a truncated Newton method is used to further reduce the model in the subspace. Truncated Newton methods compute an

approximate solution of the Newton equations by using an iterative technique. Lancelot B uses conjugate gradient method and truncates the inner algorithm before the solution of the subproblem is reached. Three different stopping criteria for the conjugate gradient algorithm are implemented. This is, (i) the norm of the reduced gradient is less than some small value depending on the required accuracy of the overall problem, (ii) one or more of the free variables in the subspace violates one of the bounds or the trust region and (iii) a large number of iterations has been taken.

In Lancelot B, some useful adjustments are made to the theoretical algorithms described above and some interesting options are provided. Some of them are described below.

It is possible to choose between the Euclidean and the infinity norm of the trust region. If the default, a box-shaped trust region, is used there will be build up an intersection of the feasible box and the trust region. That means, the two constraints are replaced by the "box" constraints

$$\max(l, x_k - \Delta_k) \equiv l_k \leq x \leq u_k \equiv \min(u, x_k + \Delta_k) \tag{19}$$

for all components $i = 1, 2, ..., n$ of the vectors l, x and u. This can lead to a better generalized Cauchy point because the default definition forces the line minimization to cease at the first point at which the trust region boundary is encountered. So, in the intersection (19) further progress is possible along the boundary of the trust region.

As default the exact generalized Cauchy point described above is computed. But Lancelot B can also, as an option, compute the approximation suggested by (Moré, 1988). Let $\gamma > 0, 0 < \beta < 1$ and $0 < \sigma < 1$. Then it is to choose the approximation $x(t_i)$, where t_i is of the form $\gamma\beta^{m_s}$ and where m_s is the smallest nonnegative integer for which

$$\begin{array}{c} m_k(x_k(l)) \leq m_k(x_k) + \sigma g(x_k)^T (x_k(l) - x_k) \\ \text{and} \quad \|x_k(t) - x_k\| \leq \beta\Delta_k. \end{array} \tag{20}$$

It has to be noticed that the generalized Cauchy point may not satisfy the first of these both equations. Thus there is a possibility of different behaviour for algorithms using the exact or the approximate generalized Cauchy points.

Lancelot B provides an option to approximate first and second order information. As default, the exact gradient and the exact Hessian are used because it is strongly recommended to use of exact derivative information whenever they are available. Different updating formula can be chosen to approximate the second derivatives. The B.F.G.S. update scheme writes

$$B_{k+1} = B_k + \frac{y_k y_k^T}{y_k^T s_k} - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k}, \tag{21}$$

where $s_k = x_{k+1} - x_k$ is the step, $y_k = g_{k+1} - g_k$ is the change in the gradients and $r_k \equiv y_k - B_k s_k$. The update is only performed if the new approximation can be ensured to be positive definite otherwise they are skipped. Other options are the P.S.B. update

$$B_{k+1} = B_k + \frac{r_k s_k^T + s_k r_k^T}{s_k^T s_k} - \frac{r_k^T s_k s_k s_k^T}{(s_k^T s_k)^2} \tag{22}$$

and the Symmetric Rank-one (SR1) update

$$B_{k+1} = B_k + \frac{r_k r_k^T}{r_k^T s_k}. \tag{23}$$

which are two examples of updates that allow indefinite approximations of the Hessian.

Further information of the methods and the implemented algorithm can be found in (Conn et al., 1993), (Conn et al., 1988b) and (Conn et al., 1988a).

2.2 Algorithmic components of line search methods

At the beginning of this section, some algorithms and components of algorithms mainly used in line search methods are described. Afterwards, we show for each considered line search solver how these components are combined and which modifications were made. Projected Newton method in algorithm 4 is not used in one of the considered solvers but it is mentioned for better understanding the algorithms that follow that one.

In the line search strategy, the algorithm chooses a direction d_k and searches along this direction from the current iterate x_k for a new iterate $x_{k+1} = x_k + \alpha_k d_k$ with a sufficiently lower function value $f(x_{k+1})$. The choice of step length α in the bound-constrained case is similar to the unconstrained case. If $x_k + d_k$ violates one of the bounds, there is to compute the largest $\mu_k \in (0, 1)$ such that $x_k + \mu_k d_k$ is feasible. The ideal choice to give a substantial reduction of f would be the global minimizer of the one-dimensional problem

$$\min_{0<\alpha\leq\mu_k} f(x_k + \alpha d_k). \tag{24}$$

Subspace minimization

There are many ways to obtain a new search direction. In Newton, quasi-Newton and truncated Newton algorithms for bound-constrained optimization, it is usual to obtain the direction d_k as an (approximate) minimizer of

the subproblem (Moré and Wright, 1993)

$$\min \nabla f_k^T d + \frac{1}{2} d^T B_k d \tag{25}$$

subject to

$$d_i = 0, i \in \mathcal{A}_k \tag{26}$$

where $\mathcal{A}_k$ is the active set and B_k is the Hessian matrix of f at x_k or an approximation to it. All variables in the active set are fixed during this iteration. This subproblem can be expressed in terms of the free variables by noting that it is equivalent to the unconstrained problem

$$\min \nabla_R f_k^T w + \frac{1}{2} w^T B_{kR} w \tag{27}$$

subject to

$$w \in R^{m_k} \tag{28}$$

where m_k is the number of free variables, B_{kR} is the reduced version of B_k by taking those rows and columns whose indices correspond to the free variables and $\nabla_R f_k$ is the reduced gradient at x_k.

Inexact line search

In general the one-dimensional problem to obtain the step length α is not solved exactly. Requiring an accurate minimizer is generally wasteful of function and gradient evaluations. More practical strategies perform an inexact line search to identify a step length that achieves adequate reductions in f.

Some line search algorithms try out a sequence of values for α and stop when certain conditions are satisfied. Generally, this is done in two stages: Firstly, a bracketing phase to find an interval that contains feasible and desirable step lengths. Secondly, in a bisection or interpolation phase is computed a good step length within the given interval.

A popular criterion for a suitable $\alpha_k \in (0, \mu_k)$ is to require α_k to satisfy the sufficient decrease condition

$$f(x_k + \alpha_k d_k) \leq f(x_k) + c_1 \alpha_k \nabla f(x_k)^T d_k \tag{29}$$

and the curvature condition

$$|\nabla f(x_k + \alpha d_k)^T d_k| \leq c_2 \, |\nabla f(x_k)^T d_k| \tag{30}$$

where c_1 and c_2 are two constants with $0 < c_1 < c_2 < 1$. The sufficient decrease condition guarantees that $f(x_{k+1}) < f(x_k)$, while the curvature condition requires that α_k is not to far from a minimizer of $\phi(\alpha)$.

The sufficient decrease condition is sometimes called the Armijo condition and the sufficient decrease and curvature conditions together are known

as the Wolfe conditions.

If the line search algorithm chooses its candidate step lengths by using a so-called backtracking approach, it is possible to use just the sufficient decrease condition to terminate the line search procedure. In its basic form, backtracking proceeds as follows

Choose $\bar{\alpha} > 0, \rho$ and $c_1 \in (0, 1)$
Set $\alpha \leftarrow \bar{\alpha}$
repeat
  $\alpha \leftarrow \rho\alpha$
until $f(x_k + \alpha d_k) \leq f(x_k) + c_1\alpha\nabla f(x_k)^T d_k$
Terminate with $\alpha_k = \alpha$.

Algorithm 2: Backtracking line search

The backtracking approach ensures either that the selected step length α_k is some fixed value (the initial choice $\bar{\alpha}$), or else that it is short enough to satisfy the sufficient decrease condition but not too short. Further information can be found in (Nocedal and Wright, 1999).

Active set method

The active set of x is $\mathcal{A}(x) = \{i | x_i = l_i \text{ or } u_i = x_i\}$ as stated before in Chapter 1. Active set methods aim to predict the active set at the solution $\mathcal{A}(x^*)$ using suitably chosen disjoint sets $\mathcal{A}(x) \subseteq \{1, 2, ..., n\}$. Once, $\mathcal{A}$ is given, an active set method will aim to solve the problem

$$\begin{aligned} &\min f(x) \\ &\text{subject to } x_i = l_i \text{ or } x_i = u_i, i \in \mathcal{A}. \end{aligned} \tag{31}$$

In fact, this statement describes an unconstrained optimization problem over the variables (x_i) where $i \notin \mathcal{A}$.

But the prediction $\mathcal{A}$ can be incorrect, of course, and active set methods have to adjust the set as the iteration proceeds either by adding variables that violate one of their bounds or removing those for which further progress is predicted. At each iteration at most one variable is added to or dropped from the active set. This can be a serious disadvantage in large problems if the active set at the starting point is quite different from that at the solution. The worst case is to visit each of the 3^n possible active sets before the optimal one is discovered (Gould et al., 2005, page 311).

Consequently, further investigations have led to algorithms that allow radical changes of the active set at each iteration (Moré and Wright, 1993). A prototypical method for that is gradient projection method described below.

Gradient Projection method

The gradient projection method can be viewed as an extension of the steepest descent algorithm to bound-constrained problems.
Given a current iterate x_k the new iterate is

$$x_{k+1} = \mathcal{P}(x_k - \tau \nabla f(x_k)), \tag{32}$$

where τ is a step length parameter given by some line search scheme. The algorithm of the gradient projection method can be written as in (Vogel, 2002):

Initialization
for $k = 1, ..., maxit$ **do**
  $p_k := -\nabla f(x_k)$ {negative gradient}
  Test for termination
  $\tau_k := \arg\min_{\tau>0} f(\mathcal{P}(x_k + \tau p_k))$ {projected line search}
  $x_{k+1} := \mathcal{P}(x_k + \tau_k p_k)$ {update solution}
  $k := k + 1$
end for

Algorithm 3: Gradient projection method

To obtain step length τ in step 3 of the algorithm, the exact projected line search can be replaced by an inexact projected line search. For bound-constrained problems in (Kelley, 1999) the sufficient decrease condition for line searches is expressed as

$$f(x(\tau)) - f(x) \leq \frac{-\alpha}{\tau} \|x - x(\tau)\|^2, \tag{33}$$

where for $\tau > 0$ is defined

$$x(\tau) = \mathcal{P}(x - \tau \nabla f(x)). \tag{34}$$

The parameter α is usually set to 10^{-4} (Dennis and Schnabel, 1996).

Once the active set has been identified, the gradient projection method behaves like the steepest descent method on the inactive variables. This results in a asymptotically linear convergence rate.

To improve this rate, second order information have to be incorporated as in the following method.

Projected Newton method

In a projected Newton method, the descent direction $p = -\nabla f(x)$ is replaced by the projected Newton direction, $s = -H_R(x)^{-1} \nabla f(x)$ where $H_R(x)$ denotes the reduced Hessian as in (5). The algorithm will be locally quadratically convergent if the active set can be correctly identified. The

algorithm of the projected Newton method can be written as in (Vogel, 2002):

Initialization
for $k = 1, ..., maxit$ **do**
$g_k := \nabla f(x_k)$ {gradient}
Test for termination
Identify active set $\mathcal{A}_k$
$H_R :=$ reduced Hessian at x_k
$s := -H_R^{-1} g_k$ {projected Newton step}
$\tau_k := \arg\min_{\tau>0} f(\mathcal{P}(x_k + \tau s))$ {projected line search}
$x_{k+1} := \mathcal{P}(x_k + \tau_k s)$ {update solution}
$k := k + 1$
end for

Algorithm 4: Projected Newton method

In the same manner, any approximation to the Hessian, or respectively its inverse, can be used instead of the exact Hessian in this method. It would then be called projected quasi-Newton method.

Gradient Projection - Reduced Newton method

Each iteration of this method has two stages. The computations in the first stage are identical to those carried out in a gradient projection iteration (see Algorithm 3 above). The second stage can be viewed as the application of Newton's method restricted to the free variables. The algorithm of the gradient projection-reduced Newton method can be written as in (Vogel, 2002):

Initialization
for $k = 1, ..., maxit$ **do**
— Gradient Projection Stage —
$p^{GP} := -\nabla f(x_k)$ {negative gradient}
Test for termination
$\tau^{GP} := \arg\min_{\tau>0} f(\mathcal{P}(x_k + \tau p^{GP}))$ {projected line search}
$x_k^{GP} := \mathcal{P}(x_k + \tau^{GP} p^{GP})$
— Reduced Newton Stage —
Identify active set $\mathcal{A}(x_k^{GP})$
$g_R :=$ reduced gradient at x_k^{GP}
$H_R :=$ reduced Hessian at x_k^{GP}
$s := -H_R^{-1} g_R$ {subspace min.: reduced Newton step}
$\tau^{RN} := \arg\min_{\tau>0} f(\mathcal{P}(x_k^{GP} + \tau s))$ {projected line search}
$x_{k+1} := \mathcal{P}(x_k^{GP} + \tau^{RN} s)$ {update solution}
$k := k + 1$
end for

Algorithm 5: Gradient Projection-Reduced Newton method

As before in Algorithm 4, any approximation to the Hessian, or respectively its inverse, can be used instead of the exact Hessian. It would then be called Gradient Projection-Reduced quasi-Newton method.

Gradient Projection CG method

If the problems are large scaled, it is sometimes not feasible to solve the linear system $Hs = -g$ in the subspace minimization of Algorithm 5 directly. In this case, an alternative is to apply an iterative method like conjugate gradient (CG) method to this system. In this context, CG can be viewed as a tool with which the subspace minimization problem can be solved iterately instead of as a linear solver. Thus, for each outer iteration, there is an inner iteration loop making use of the conjugate gradient method that computes the new search direction. The algorithm of the gradient projection CG method can be written as

Initialization
for $k = 1, ..., maxit$ **do**
— Gradient Projection Stage —
$p^{GP} := -\nabla f(x_k)$ {negative gradient}
Test for termination
$\tau^{GP} := \arg\min_{\tau>0} f(\mathcal{P}(x_k + \tau p^{GP}))$ {projected line search}
$x_k^{GP} := \mathcal{P}(x_k + \tau^{GP} p^{GP})$
— Conjugate Gradient Stage —
$Q_k(p) = f(x_k^{GP}) + p^T \nabla f(x_k^{GP}) + \frac{1}{2} p^T H_k^{GP} p$ {build quadratic model}
Identify active set $\mathcal{A}(x_k^{GP})$
$\min Q_k(p)$ s.t. $p_i = 0, i \in \mathcal{A}(x_k^{GP})$ {subspace min.: CG method}
for $j = 1, ..., maxcg$ **do**
Compute conjugate gradient step
Test for insufficient decrease in Q_k
if Test is true **then**
$\tau^{CG} := \arg\min_{\tau>0} f(\mathcal{P}(x_k^{GP} + \tau p_j))$ {projected line search}
$x := \mathcal{P}(x_k^{GP} + \tau^{CG} p_j)$ {interim solution}
if $\mathcal{A}(x) = \mathcal{A}(x_k^{GP})$ **then**
Resume CG iterations
else
Terminate CG iterations
end if
end if
end for
$x_{k+1} := x$ {update solution}
$k := k + 1$
end for

Algorithm 6: Gradient Projection CG method

Several modifications can be made to the algorithm to increase its efficiency. One possibility is to take more than one iteration in the gradient projection stage. This is cost effective if either the gradient projection steps significantly decrease the functional f or the active set changes rapidly. To quantify this, let $x_{k,j}^{GP}$ denote the jth iterate in the projected gradient stage. One stops the iteration if either

$$f(x_{k,j-1}^{GP}) - f(x_{k,j}^{GP}) \leq \gamma \max_{i<j} f(x_{k,i-1}^{GP}) - f(x_{k,i}^{GP}) \tag{35}$$

with $\gamma > 0$ is fixed, or

$$\mathcal{A}(x_{k,j}^{GP}) = \mathcal{A}(x_{k,j-1}^{GP}). \tag{36}$$

The first condition is called insufficient decrease condition and is also used in the CG stage of the algorithm (Vogel, 2002).

Solver: DOT

This software is developed by Vanderplaats Research & Development, Inc. It can be used to solve unconstrained, bound-constrained and generally constrained optimization problems. It contains two methods DOT-BFGS and DOT-FR that solve problems with simple bounds on the variables. Both methods are based on a line search and differ only in the way the search direction is found.

DOT-BFGS uses an unconstrained quasi-Newton method to obtain the search direction. An approximation H to the inverse of the Hessian matrix is created. In this case, the matrix is updated by using a BFGS formula. The new search direction s is obtained by computing the matrix-vector-product

$$s_{k+1} = -H_{k+1}\nabla f(x_k). \tag{37}$$

DOT-FR uses an unconstrained Fletcher-Reeves conjugate gradient method to obtain the search direction. This method is very simple and requires only very little computer storage because it needs neither the Hessian nor an approximation to it. The search direction is computed as

$$s_{k+1} = -\nabla f(x_k) + \beta s_k \tag{38}$$

with

$$\beta = \frac{|\nabla f(x_k)|^2}{|\nabla f(x_{k-1})|^2}. \tag{39}$$

After computing the unconstrained search direction s it is to check whether some components of s violate the bounds. A component s_i is set to zero if its corresponding variable x_i is at one of their bounds and the component s_i points outside the feasible box. The line search along the corrected direction is done by quadratic or cubic interpolation depending on the amount of information available. More information is given in some detail in DOT User's Manual (Vanderplaats, 1995) and in (Vanderplaats, 1984).

Solver: L-BFGS-B

This software is written by Richard H. Byrd, Peihuang Lu, Jorge Nocedal, Ciyou Zhu and it was especially developed to solve bound-constrained optimization problems.

L-BFGS-B is a limited memory quasi-Newton method that uses the Gradient Projection-Reduced quasi-Newton method stated in Algorithm 5 to obtain a new search direction. To approximate second order information, a limited memory BFGS matrice represented in the compact form described by (Byrd et al., 1994) is used. The limited memory BFGS matrices require only a small amount of computer storage because at every iterate x_k the algorithm stores only a small number of correction pairs $\{s_i, y_i\}, i = k-1, ...k-m$ where

$$s_k = x_{k+1} - x_k, y_k = g_{k+1} - g_k. \tag{40}$$

These correction pairs contain information about the curvature of the function, and in conjunction with the BFGS formula, define the limited memory iteration matrix H_k.

Once obtained the matrix, the linear system in the subspace minimization step of the algorithm is solved by a direct primal method based on the Sherman-Morisson-Woodbury formula (Golub and Loan, 1989). The obtained path will be truncated if violating one or more bounds. The line search is performed by means of the routine by (Moré and Thuente, 1994) which tries to enforce the Wolfe conditions by a sequence of polynomial interpolations. Since step lengths greater than one may be tried, the maximum step length is defined as the step to the closest bound along the current search direction. More details about method and implementation of L-BFGS-B can be found in (Byrd et al., 1995) and (Zhu et al., 1997).

Solver: TN-BC

This software is written by Stephen G. Nash and it is especially developed to solve bound-constrained optimization problems.

TN-BC is based on a line search with a classical active set strategy for treating the bounds. TN-BC is a truncated-Newton method that uses the Gradient Projection CG method stated in Algorithm 6 to obtain a new search direction. But it computes only an approximation to the Newton direction because it is truncated before the solution to the subspace minimization problem is obtained. More about truncated Newton methods in general can be found in (Nash, 2000).

The conjugate gradient inner algorithm is preconditioned by a scaled two-step limited memory BFGS method with Powell's restarting strategy used to reset the preconditioner periodically. A detailed description of the preconditioner may be found in (Nash, 1985).

Since the Hessian matrix with exact second derivative information is oftentimes not given, the Hessian vector product $H_k v$ for a given v required by the inner conjugate gradient algorithm is obtained by finite differencing:

$$\nabla^2 f(x_k)v \approx \frac{\nabla f(x_k + hv) - \nabla f(x_k)}{h} \tag{41}$$

where $h = (1 + \|x_k\|_2)\sqrt{\epsilon}$, and ϵ is the relative machine precision. Each matrix vector product requires one gradient evaluation, since $\nabla f(x_k)$ is already available as the right-hand side of the linear system.

The line search is performed using the iteration described by (Gill and Murray, 1979). It is based on cubic interpolation and is terminated when the strong Wolfe conditions are satisfied.

More detailed information of the method and the algorithm can be found in (Nash, 1984a) and (Nash, 1984b).

2.3 Algorithmic components of primal-dual interior point methods

The standard formulation of a linear programming problem is often called the primal and it is of the form

$$\min c^T x \quad \text{s.t.} \quad Ax = b, x \geq 0, \tag{42}$$

where c and x are vectors in R^n, b is a vector in R^m and A is an $m \times n$ matrix. The dual of the standard form linear programming problem can be written as

$$\max b^T y \quad \text{s.t.} \quad s = c - A^T y \geq 0. \tag{43}$$

The optimality conditions for (x, y, s) to be a primal dual solution are again the Karush-Kuhn-Tucker conditions

$$\begin{aligned} A^T y + s - c &= 0 \\ Ax - b &= 0 \\ XSe &= 0 \\ (x, s) &\geq 0, \end{aligned} \tag{44}$$

where $X = diag(x_1, x_2, ..., x_n)$, $S = diag(s_1, s_2, ..., s_n)$ and e is the unit vector. In the terminology of generally constrained optimization, the vectors y and s are Lagrange multipliers for the constraints $Ax = b$ and the bounds $x \geq 0$. A vector (x^*, y^*, s^*) solves this KKT-system if and only if x^* solves the primal problem and (y^*, s^*) solves the dual problem. The vector (x^*, y^*, s^*) is then called a primal-dual solution.

Primal-dual interior point methods can be thought of as a variant of Newton's method applied to the system of equations formed by the first three optimality conditions. Further details can be found in (Wright, 1997) and (Moré and Wright, 1993).

Solver: IPOPT

This software is written by Andreas Wächter and he developed a primal-dual interior point algorithm with a filter line search. He considers a primal-dual barrier method to solve nonlinear unconstrained, bound-constrained and generally constrained optimization problems. As a barrier method the proposed algorithm computes approximate solutions for a sequence of barrier problems

$$\begin{aligned} &\min_{x \in R^n} \phi_\mu(x) = f(x) - \mu \sum_{i \in I_L} ln(x_i - l_i) - \mu \sum_{i \in I_U} ln(u_i - x_i) \\ &\text{subject to } c(x) = 0, \\ &\text{where } I_L = \{i : l_i \neq -\inf\} \text{ and } I_U = \{i : u_i \neq \inf\} \end{aligned} \tag{45}$$

for a decreasing sequence of barrier parameters μ converging to zero. Equivalently, this can be interpreted as applying a homotopy method to the primal-dual equations,

$$\begin{aligned} \nabla f(x) + \nabla c(x) y - s &= 0 \\ c(x) &= 0 \\ XSe - \mu e &= 0, \end{aligned} \tag{46}$$

with the homotopy parameter μ, which is driven to zero. Further information e.g. in (Byrd et al., 1998) and (Gould et al., 2000).
If $l_i = u_i$ for a variable, this component of x is fixed to this value for all function evaluations and removed from the problem statement. Of course, in the case with only simple bounds on the variables, the constraint condition $c(x) = 0$ is already satisfied.

The method used by IPOPT computes an approximate solution to the barrier problem stated above for a fixed value of μ. Then the barrier parameter is decreased and it is continued to solve the next barrier problem from the approximate solution of the previous one. A short outline of the algorithm is stated below:

Initialization
for $k = 1, ..., maxit$ **do**
  Check for convergence
  Compute the search direction
  Backtracking line search
  Accept the trial point
  Augment the filter if necessary
  Increase k
end for

Algorithm 7: Outline of IPOPT algorithm

Further details of the algorithm and the method can be found in (Wächter and Biegler, 2006).

IPOPT needs for running the BLAS and LAPACK routines and at least two subroutines from the Harwell library. In the forthcoming tests it is run with the freely available Harwell subroutines MA27 and MC19.

3 Methodology

Optimization methods are in general difficult to compare "on the paper", that means, one can't find the best solver only by reading the paper of the algorithm. This leads directly to the consequence that tests had to be made to get a fair comparison of the different solvers.

An important fact for a fair comparison is to use the same information of the function for all solvers. To consider a special type of problem where no exact Hessian matrix is given, it is only taken use of the function value and the exact first derivatives. The solvers either approximate second derivative information or don't need them at all.

A further important question for the test was the choice of the parameters. For that, default values for all methods are used because one would suppose that they are adjusted by the developer to work best with most of the test cases.

Yet another important point was the use of the same stopping criteria for all solvers. This will be discussed in more detail in the next section.

3.1 Used stopping criteria

In this experiments, for solving the problem successfully, the condition

$$\|\mathcal{P}(x^* - \nabla f(x^*)) - x^*\|_\infty \leq \epsilon, \tag{47}$$

where ϵ is the required accuracy, had to be satisfied.

A problem was not successfully solved if the projected gradient did not reach the demanded accuracy for one of the following reasons. Firstly, the solver itself terminates the run and reports an error during solving the problem. In this case, no solution was found or the presented solution was completely wrong. The second case is also reported by the solver that the run was terminated because the next iterate generated by the algorithm was too close to the current iterate to be recognized as distinct. This occurs, if the solver is quite near the solution but is unable to meet the demanded accuracy of the projected gradient. It is sometimes said, the solver is "stuck" because it cannot make further progress towards the solution.

Additionally, two termination criteria - limits for iteration number and time - were set. They are stopping the optimization process from outside, are defined by the user. In these experiments, the codes were run with a maximum iteration limit of 100000 and a CPU time limit of 1800s. If these values are exceeded, the problem will be considered as not successfully solved.

The projected gradient as stopping criterion was already used in L-BFGS-B and Lancelot B. In the solvers DOT, TN-BC and IPOPT the unscaled projected gradient was to implement.

3.2 Testing environment

In the experiments the CUTEr test environment was used to compare the considered optimization software. CUTEr is a testing environment for optimization and linear algebra solvers and it contains a large collection of test

problems in SIF (standard input format). It provides ready-to-use interfaces to existing solvers (e.g. L-BFGS-B and IPOPT) and it is possible to create new interfaces. This was done for DOT and TN-BC. In all cases the algorithms of the solvers were not included in the test environment and had to be implemented.

Lancelot B provides already an own interface to decode and run test problems written in SIF.

3.3 Test case statistics

CUTEr provides 128 bound constrained problems and 76 out of them were used for the experiments. Two test cases were removed from the test set because they could not be solved by all considered solvers. 50 test cases were excluded because two ore more of the codes solved the problem in 0.00s so that no comparison of them was possible. We assume, these problems are too small or too easy to solve.

The number of variables of the remaining test cases varied from 3 to 15625. The type of the objective function was quadratic in 32 cases, a sum of squares in 19 cases and the remaining 25 cases were of other type.

The origin of the test cases was academic in 37 cases, so they have been constructed specifically by researchers to test one or more algorithms. In 26 cases, the problem is part of a modelling exercise where the actual value of the solution is not used in a practical application. In 13 cases the origin is a real application, so the solution of the problem is or has been used in a real application for purposes other than testing algorithms.

3.4 Performance profiles

For the comparisons in the next chapters we made use of the Dolan-Moré performance profiles (Dolan and Moré, 2001). Given a test set $\mathcal{P}$ containing n_p problems and n_s solvers, these profiles provide a way to graphically present the comparison of quantities $t_{p,s}$ (such as required computing time or number of function evaluations to solve problem p by solver s) obtained for each problem and each solver. For this, the performance ratio for a problem p and a solver s is defined as

$$r_{p,s} := \frac{t_{p,s}}{\min\{t_{p,s} : 1 \leq s \leq n_s\}}. \tag{48}$$

If solver s for problem p leads to a failure $r_{p,s} := 2 * \max\{t_{p,s} : 1 \leq s \leq n_s\}$ is defined. Then,

$$\rho_s(\tau) := \frac{1}{n_p}\text{size}\{p \in \mathcal{P} : r_{p,s} \leq \tau\} \tag{49}$$

is the fraction of the test problems which were solved by solver s within a factor $\tau \geq 1$ of the performance obtained by the best solver. The function ρ_s is the (cumulative) distribution function for the performance ratio. The performance plots present ρ_s for each solver s as a function of τ. In this work, a logarithmic scale is used for the τ-axis. That means,

$$\rho_s(\tau) := \frac{1}{n_p} \text{size}\{p \in \mathcal{P} : \log_2(r_{p,s}) \leq \tau\} \tag{50}$$

is plotted in the performance profiles in the following chapters.

4 Results and conclusions

Two runs with different required accuracies as stopping criterion were done. Firstly, tests requiring moderate accuracy were made. Secondly, the same test problems were run again but a higher accuracy was required. Thirdly, another run with the same test problems but with a completely different stopping criterion has been considered. In this case, the iterations were terminated after a certain number of function plus gradient evaluations in order to imply the fact that the evaluations of function and gradient could be very expensive. The aim of this experiment was to see how much each solver was able to reduce the function value after a maximum cost of 70 evaluations. Finally, a last experiment was done to compare the different solvers on a model function of a real Airbus problem.

4.1 Experiment with low accuracy

In this experiment, the infinity norm of the projected gradient of the objective function must be reduced below 10^{-3}. The results of this testing can be seen in Table 1.

Solver	**CPU time**	**Nbr.funct.+grad.evaluations**
DOT-BFGS	0	8
DOT-FR	0	7
L-BFGS-B	37	17
TN-BC	19	3
Lancelot B SR1	14	49
Lancelot B BFGS	5	39
Lancelot B PSB	13	53
IPOPT	0	0

Table 1: Results with required low accuracy

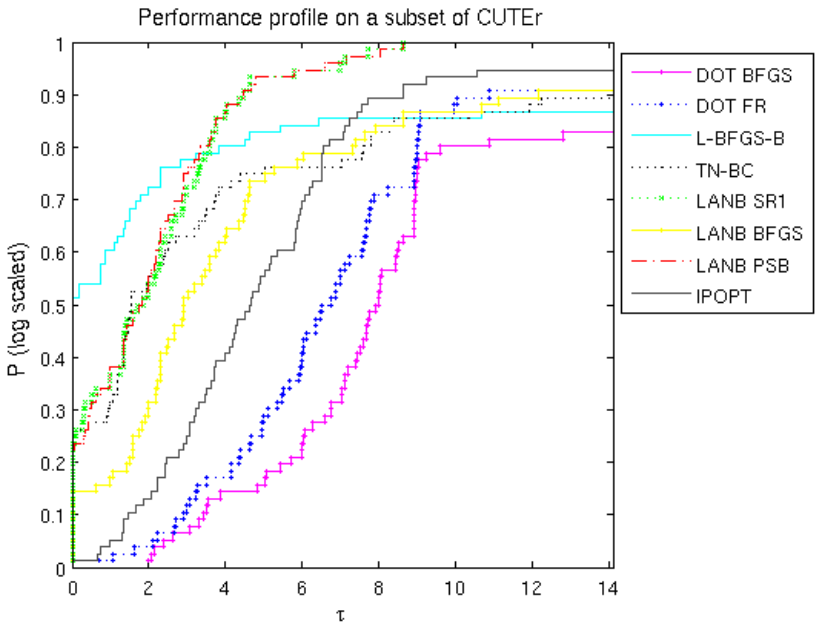

Figure 1: Results in terms of CPU time

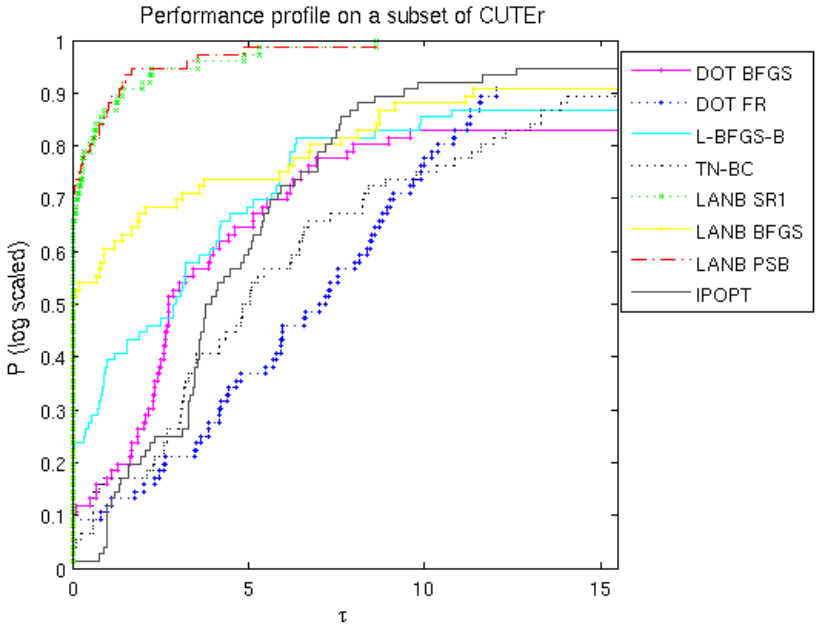

Figure 2: Results in terms of function+gradient evaluations

In Table 1 can be seen that L-BFGS-B was the fastest solver in 37 of the test cases and that IPOPT and both methods of DOT have a very bad performance in terms of CPU time. In terms of function and gradient evaluations, Lancelot B outperforms the other methods. The version which uses PSB update solved 53 of the test cases with the lowest number of evaluations. Obviously, Lancelot B takes advantage of trust region method in which it is sufficient to compute the function value only once per iteration in contrast to line search method where it is necessary to compute it several times during the search for the step length.

4.2 Experiment with high accuracy

In this experiment, every run was terminated when the infinity norm of the projected gradient of the objective function was reduced below 10^{-5}.

Solver	**CPU time**	**Nbr.funct.+grad.evaluations**
DOT-BFGS	0	2
DOT-FR	0	1
L-BFGS-B	28	9
TN-BC	16	1
Lancelot B SR1	19	49
Lancelot B BFGS	9	38
Lancelot B PSB	13	48
IPOPT	1	0

Table 2: Results with required high accuracy

Table 2 shows nearly the same picture as the experiment with required low accuracy. L-BFGS-B performs again best in terms of CPU time and Lancelot B needs again the smallest number of function and gradient evaluations more often than other methods to solve the problems of the test set. This time, the version using SR1 update is slighly better than that using PSB update.

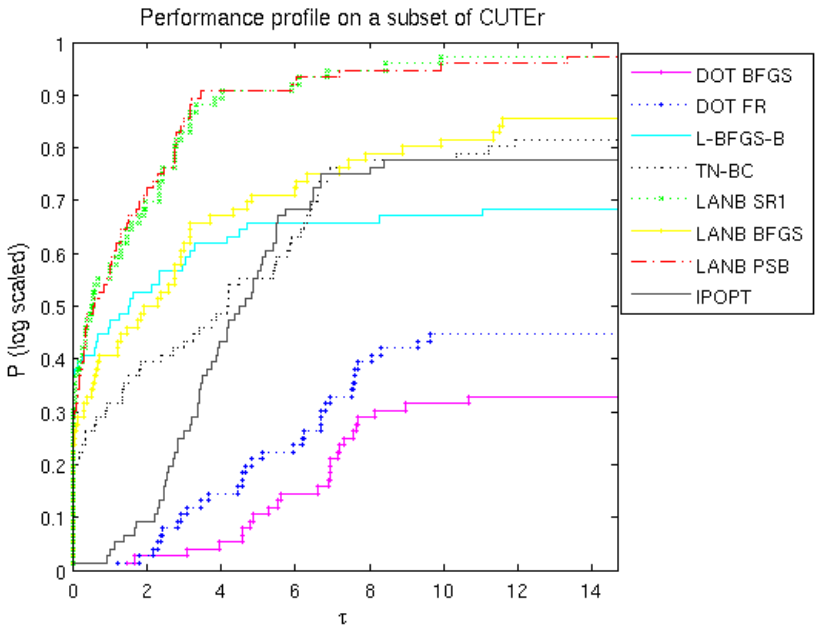

Figure 3: Results in terms of CPU time

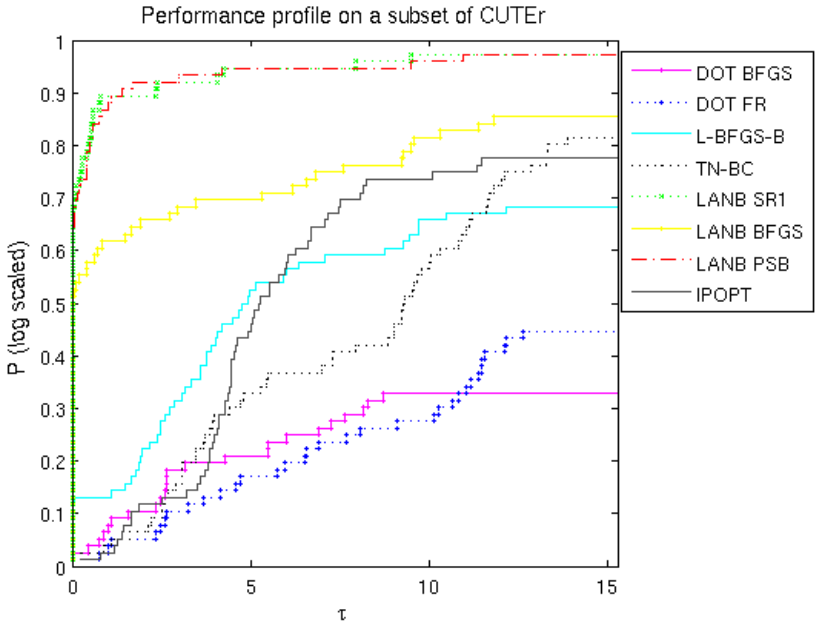

Figure 4: Results in terms of function+gradient evaluations

4.3 Case of failure due to demanded accuracy

Here, we describe for which reasons each solver failed to solve the complete set of test problems. In Table 3, the number of failures are counted and displayed as a comparison of the two different required accuracies. The term "error" stands for failures of the solver itself, for example errors in line search. The term "stuck" stands for a lack of accuracy because in this case the solver was in the near of the solution but didn't manage to reduce the projected gradient at that point to get the required accuracy. In both cases the solver itself stopped the iteration with one of these messages. Termination by user means that we have chosen the both limits for maximum iteration count and CPU time limit. Although, we thought of a quite generous choice of these limits with an iteration limit at 100000 and CPU time limit at 1800 seconds, the solvers might have solved the problem afterwards exactly.

	Low accuracy 10^{-3}		**High accuracy** 10^{-5}	
Termination	by solver	by user	by solver	by user
For reason	error / stuck	maxit / CPU	error / stuck	maxit / CPU
DOT-BFGS	0 / 5	0 / 8	0 / 26	1 / 24
DOT-FR	0 / 6	0 / 1	0 / 34	8 / 1
L-BFGS-B	1 / 8	1 / 0	2 / 14	8 / 0
TN-BC	3 / 2	2 / 1	6 / 4	2 / 2
Lancelot B SR1	0 / 0	0 / 0	0 / 1	0 / 1
Lancelot B BFGS	0 / 2	3 / 2	0 / 2	4 / 5
Lancelot B PSB	0 / 0	0 / 0	0 / 0	0 / 2
IPOPT	8 / 0	3 / 6	4 / 0	0 / 0

Table 3: Case of failure

Naturally, less failures occured when lower accuracy was required. Oftentimes, the problem was solved before the errors could occur. The interesting fact of the table is that some solvers had more differences between the two runs than others. Namely must be said that Lancelot B in general had nearly no problems to solve all test cases in an accurate way. Especially with PSB update, the solver had only two times exceeded the CPU time limit with required high accuracy in comparison with no failure for required low accuracy. Nearly the same results were obtained using SR1 update.

4.4 Experiment with maximum cost 70

Here, another experiment with an extra implemented stopping criterion was done. The run of a solver was terminated if a sum of 70 function and gradient evaluations or the former stopping criterion 10^{-5} was reached. As result, the lowest function values were observed.

Solver	lowest function value
DOT-BFGS	8
DOT-FR	9
L-BFGS-B	10
TN-BC	4
Lancelot B SR1	39
Lancelot B BFGS	35
Lancelot B PSB	39
IPOPT	8

Table 4: Results with maximum cost 70

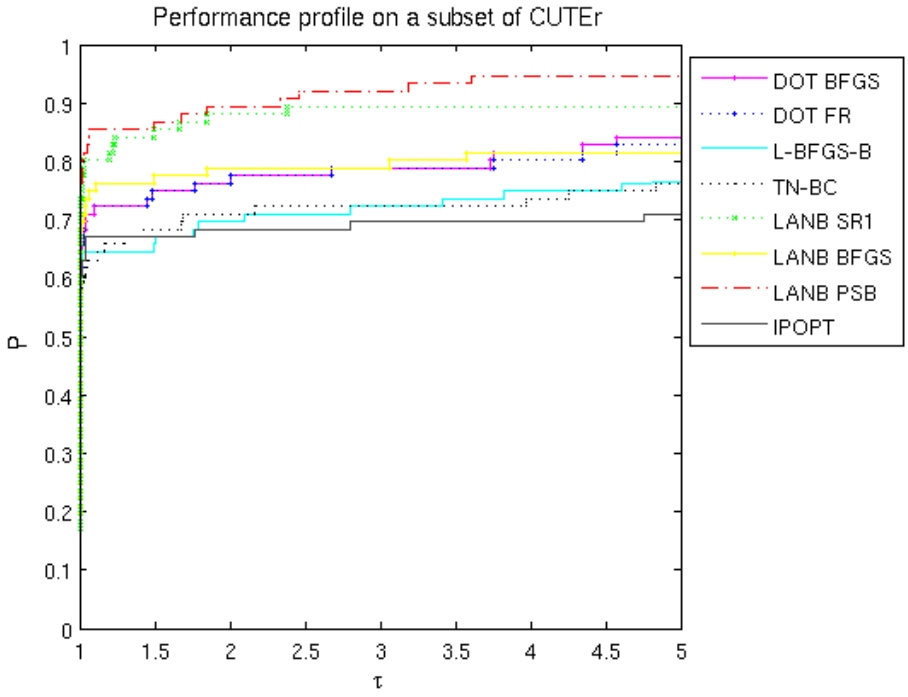

Figure 5: Results in terms of function value

It can be seen in Table 4 that the winner of test cases is again Lancelot B. It solved the most test cases with lowest function value compared to the others. In the corresponding Figure 5 can be seen that also the other solvers are not far away of being the best. Both DOT methods perform not bad in this type of experiment. But it should be mentioned that for instance at $\tau = 2$ they solved 75% of the test cases reaching only twice the best function value. In this experiment, TN-BC performed worst.

Besides, out of Figure 5 can't be drawn conclusions about the robustness

of the solvers because this performance profile concentrates on the first part and was truncated after $\tau = 5$. What means, it can be only seen the improvements of the solvers in five times the best function value.

4.5 Experiment with Airbus model function

Another experiment was done to compare the different solvers on a model function which describes a problem which is originally considered at Airbus. The function describes a model of a coaction of physical draw and lift in an aerodynamical manner. The model function which was to minimize contained two variables and is depicted on Figure 6.

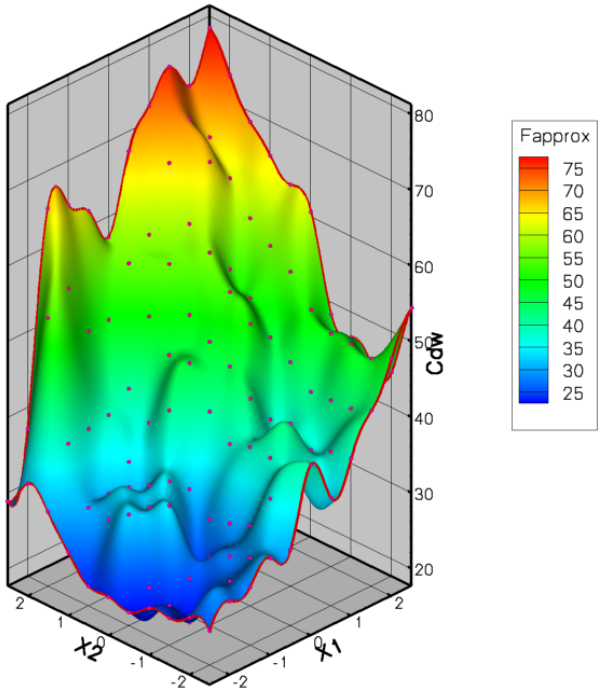

Figure 6: Model function of an aerodynamical problem

The problem is constrained by the given box

$$l = \begin{pmatrix} -2.5 \\ -2.5 \end{pmatrix} \leq x \leq \begin{pmatrix} 2.5 \\ 2.5 \end{pmatrix} = u.$$

To compare the considered solvers, a starting point had to be selected. In the first run, the arbitrary point $x_0 = (0.0, 0.0)$ was used as starting point. The required accuracy of the projected gradient at the solution was 10^{-5}. Results

in terms of final function values, CPU time (for the model function without real CFD calculations) and the sums of function and gradient evaluations are displayed in Table 5.

Solver	**final f value**	**nbr.func+grad.eval.**	**Time**
DOT-BFGS	21.6895	44	14.40 s
DOT-FR	21.6895	49	14.67 s
L-BFGS-B	24.8152	4	1.40 s
TN-BC	21.6895	46	9.66 s
Lancelot B SR1	20.5656	40	6.19 s
Lancelot B BFGS	24.8152	104	13.06 s
Lancelot B PSB	21.6895	40	5.36 s
IPOPT	21.6895	25	6.13 s

Table 5: Results of aerodynamical model function

In Table 5 is to be seen that the solvers converged to different local minima so that a direct comparison is not possible. For this reason, we chose another starting point $x_0 = (-2.0, -1.0)$ which lies closer to the global minimum $x^* = (-2.3, -0.6)$. This time, all solvers converged to the same minimum.

Solver	**final f value**	**nbr.func+grad.eval.**	**Time**
DOT-BFGS	20.5656	66	18.83 s
DOT-FR	20.5656	64	18.45 s
L-BFGS-B	20.5656	38	4.76 s
TN-BC	20.5656	35	7.20 s
Lancelot B SR1	20.5656	35	4.56 s
Lancelot B BFGS	20.5656	31	4.05 s
Lancelot B PSB	20.5656	31	4.14 s
IPOPT	20.5656	37	8.31 s

Table 6: Results of aerodynamical model function second run

Table 6 shows that in terms of function plus gradient evaluations all solvers performed nearly the same, except for both DOT methods which needed the double amount of evaluations. In terms of CPU time it shows the same picture, but it can be said that Lancelot B and L-BFGS-B outperform the other solvers.

4.6 Conclusions

We have compared 8 solvers and variations for bound-constrained optimization problems. The comparisons were made using a model function provided

by Airbus and a set of 76 test problems taken out of the CUTEr test collection. We have considered two accuracy requirements on the solution (low and high accuracy), and we have compared the solvers in terms of number of function plus gradient evaluations as well as in terms of CPU time. We believe that function and gradient evaluation counts are especially of interest in situations were then CPU cost is large, as it may be the case at Airbus when 3D simulations are involved.

Our experiments show very different behaviours of the compared solvers. The trust region based solver Lancelot B is clearly the best of the considered solvers, both for CPU time and number of function and gradient evaluations. It is the most robust solver too, since it can solve more than 95% of the test cases within the given time and iteration limits.

To compare the line search based solvers we have to consider two cases: low and high accuracy is requested. For high accuracy, L-BFGS-B is faster and more robust than DOT. For low accuracy, it turns out that L-BFGS-B is still faster but DOT is more robust and solved 90% of the test cases within the limits.

In terms of function and gradient evaluations, the interior point method IPOPT is slightly faster than DOT and slower than L-BFGS-B. The IPOPT code solved more problems, that means, it is at the end more robust than DOT and L-BFGS-B.

TN-BC is faster than IPOPT and DOT and has about the same robustness as IPOPT.

A Experiment with generally constrained optimization problems

In addition to the main target of this work about solving bound-constrained optimization problems as described in the previous chapters, we had also a closer look to optimization problems with more general constraints because these problems are also of interest in the optimization framework of solving problems in aircraft design at Airbus. But this is to be seen as a preliminary work which needs further investigation.

In this Appendix, we describe the comparison of the two already implemented solvers IPOPT and Lancelot B which can handle general constraints.

A.1 Problem statement

The optimization problem with general constraints can be written as

$$\min_{x \in R^n} f(x_i) \tag{51}$$

subject to

$$c_i(x) = 0, \quad i \in \mathcal{E} \quad \text{and} \quad c_i(x) \leq 0, \quad i \in \mathcal{I} \tag{52}$$

where f is a nonlinear function with n variables and c denotes the constraint functions of the problem. The components of c are either inequality constraints for each i in the index set $\mathcal{I}$ or represent equality constraints for each i in the index set $\mathcal{E}$.

Constraints in general can be of different nature. This definition includes the case where only simple bounds are on the variables. Further, constraints can be linear functions and smooth or non-smooth nonlinear functions.

A.2 Solvers

The solvers considered in these experiments are IPOPT and Lancelot B. Both are freely available for academic use and the source code is also available. IPOPT uses a primal-dual interior point method and was compiled with the freely available Harwell subroutines MA27 and MC19. Second order information was approximated by limited memory BFGS update as in the previous experiments. Lancelot B uses a sequential augmented Lagrangian method to solve problems with general constraints and was run with the SR1 update which seemed to perform best in the previous tests. More information about the algorithms can be found in (Wächter and Biegler, 2006) and (Conn et al., 1991).

In addition, we planned to consider the optimization tool DOT in this experiment because the target of this work was the comparison of other codes to the methods in DOT. This could be not managed in the given time-frame for different reasons. Firstly, the tool DOT needs a huge amount of memory so that it solves only a few problems of the test set without encountering memory problems. This is the main reason wherefore it was not possible to consider DOT in the comparison for the moment. A second reason is, that in two of the three provided methods for solving optimization problems with general constraints (SLP and SQP), the stopping criterion is only based on stagnation of the function values and not on optimality. It is necessary to put more time and effort to implement an objective criterion for terminating a run successfully.

A.3 Methodology

Stopping criteria

Unfortunatly, the stopping criteria of the two considered methods IPOPT and LANCELOT B were also not directly comparable, but these codes are really checking optimality using a KKT system. For both methods their default termination rules were taken in the experiments. The default termination tolerance for IPOPT, a residual of the interior point system, had

to met 10^{-8}, whereas the infinity norms of the projected gradient and the constraints, used by Lancelot B, had to be equal or below 10^{-5}.

In addition, a CPU time limit of 1 hour and an iteration count limit of 10000 were imposed for the test.

Test case statistics

The test cases used for this comparison are taken out of the CUTEr collection (Gould et al., 2003). In both codes, problems with general inequality constraints of the form

$$d_l \leq d(x) \leq d_u \tag{53}$$

are reformulated into equality constraints by adding slack variables $d_l \leq s \leq d_u$ and replacing the inequality constraints by

$$d(x) - s = 0. \tag{54}$$

For the numerical experiment, initially all 731 test cases which have linear or nonlinear constraints but not only simple bounds on the variables were considered. The problems vary in size from $n = 2$ to 123200 variables and from $m = 1$ to 123200 constraints (after introduction of slack variables). For problems with variable size, the default was taken.

After running all test problems, 29 test cases were excluded for which the final values of the objective functions were not close in an attempt to avoid comparisons of runs to different local solutions. Those problems were discarded from a performance plot for which

$$\frac{f^{max} - f^{min}}{1 + max\{|f^{max}|, |f^{min}|\}} > 10^{-1}, \tag{55}$$

where f^{max} is the bigger and f^{min} is the smaller final objective function value of the two methods. If one or both solvers terminated with an error message, there was paid no attention to the function values whether they were different or not.

Finally, 71 test problems could not be solved by both solvers for different reasons but they were not removed from the performance profile.

A.4 Results and conclusions

In Table 7 can be seen the number of test cases where one of the solver was faster (for CPU time) or needed less evaluations than the other one. It is also counted if both had the same time or number of evaluations.

Solver	**CPU time**	**Nbr.funct.+grad.evaluations**
Lancelot B SR1	231	260
IPOPT	271	236

Table 7: Results with generally constrained problems

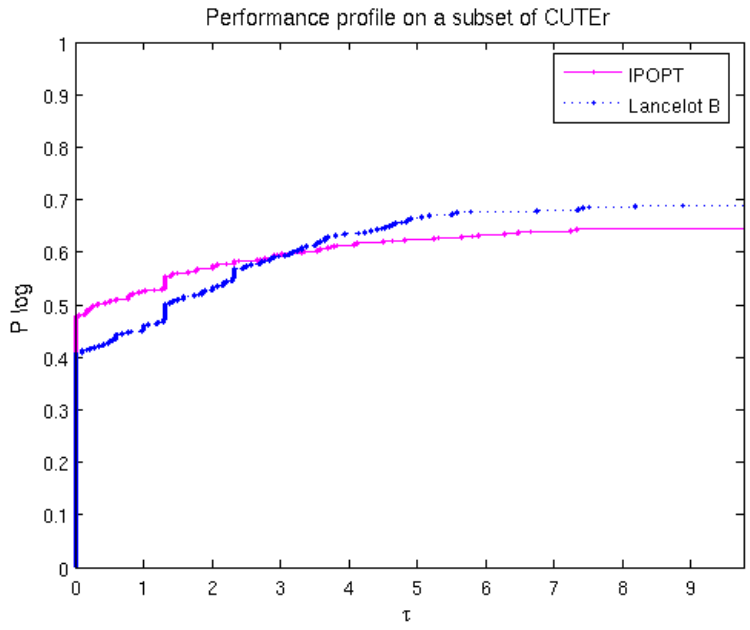

Figure 7: Results in terms of CPU time

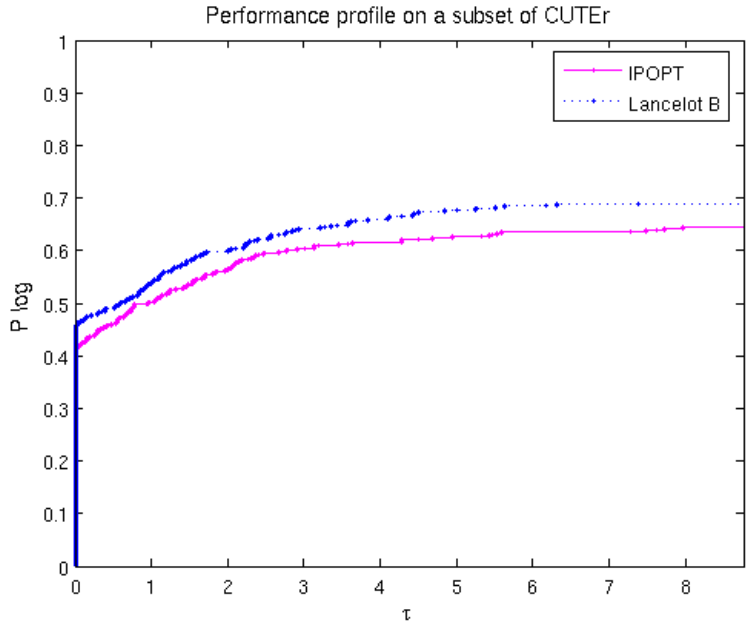

Figure 8: Results in terms of function+gradient evaluations

The two Figures 7 and 8 show that Lancelot performs better for the test set in terms of function and gradient evaluations than IPOPT and in terms of CPU time IPOPT performs better than Lancelot B. In Table 7 can be seen that the difference between the two methods is not that big. Lancelot B solved 231 test cases fastest and for 271 problems IPOPT was faster. In 260 cases Lancelot B needed less function and gradient evaluations and in 236 problems this was the case for IPOPT.

About robustness of the solvers can be said that Lancelot B solved about 70% of the test cases where IPOPT solved only 64% of the test cases successfully. This leads to the question, why couldn't they solve all 567 problems of the test set. In Table 8 is listed for what reasons the methods didn't solve the problems.

Termination	by solver	by user
for reason	stuck / NaN / other	maxit / CPU
Lancelot B SR1	70 / 3 / 17	45 / 38
IPOPT	3 / 3 / 100	59 / 37

Table 8: Case of failure

Lancelot B terminated for 173 and IPOPT terminated for 202 problems not successfully. In some cases, the solvers failed to converge within the user defined time or iteration limit. In some other cases, the solvers aborted the run itself due to different reasons. Lancelot B could make no progress to reach the required accuracy in 70 cases. IPOPT was stuck only in 3 cases. In 17 problems Lancelot B aborted because it could not further reduce the constraint norm during the run and assumed the problems maybe infeasible. IPOPT solved 7 of these problems successfully but stated possible infeasibility for one of it. The 100 test problems which IPOPT terminated for other reasons can be separated in three parts. In 17 cases, IPOPT was not able to start the optimization process because there were too few degress of freedom (more equality constraints than free variables) but Lancelot B solved 10 of them successfully. For 4 problems, IPOPT converged to a stationary point for infeasibility. In the restoration phase, IPOPT is minimizing the constraint violation and tries to further improve feasibility. In 79 test cases, this stage could not be completed successfully by the algorithm.

Conclusions

Again, Lancelot B scores well at this comparison. It performs better than IPOPT in terms of number of function plus gradient evaluations. IPOPT performs better in terms of CPU time but at the end, Lancelot B is more robust than IPOPT because it is able to solve more test problems in the given time and iteration limits. Further, the biggest part of Lancelot's unsolved problems are due to the required accuracy. So, if high accuracy was

not that important, the outcome of Lancelot would be even better.

The next step of this study will consist in implementing a unified stopping criteria of the considered solvers to ensure a better comparability of the methods. To give the opportunity to run DOT with all generally constrained test problems of CUTEr, the whole testing environment could be installed on a bigger machine. We would also like to make a realistic software comparison in the Airbus OPTALIA optimization suite.

Acknowledgements

We would like to thank Jean-Francois Boussuge, Pascal Larrieu, Julien Laurenceau and Matthieu Meaux for helpful discussion concerning the use of optimization software in aircraft design.

List of Algorithms

References

J. V. Burke, Jorge J. Moré, and G. Toraldo. Convergence properties of trust region methods for linear and convex constraints. *Math. Program.*, 47(3): 305–336, 1990.

Richard H. Byrd, G. Liu, and Jorge Nocedal. On the local behavior of an interior point method for nonlinear programming. In D. F. Griffiths and D. J. Higham, editors, *Numerical Analysis 1997*, pages 37–56. Addison Wesley Longman, 1998.

Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM J. Sci. Comput.*, 16(5):1190–1208, 1995.

Richard H. Byrd, Jorge Nocedal, and Robert B. Schnabel. Representations of quasi-newton matrices and their use in limited memory methods. *Mathematical Programming*, 63(2):129–156, 1994.

Paul H. Calamai and Jorge J. Moré. Projected gradient methods for linearly constrained problems. *Math. Program.*, 39(1):93–116, 1987.

Andrew R. Conn, Nicholas I. M. Gould, and Philippe L. Toint. Global convergence of a class of trust region algorithms for optimization with simple bounds. *SIAM J. Numer. Anal.*, 25(2):433–460, 1988a.

Andrew R. Conn, Nicholas I. M. Gould, and Philippe L. Toint. Testing a class of methods for solving minimization problems with simple bounds on the variables. *Mathematics of Computation*, 50(182):399–430, 1988b.

Andrew R. Conn, Nicholas I. M. Gould, and Philippe L. Toint. A globally convergent augmented lagrangian algorithm for optimization with general constraints and simple bounds. *SIAM J. Numer. Anal.*, 28(2):545–572, 1991.

Andrew R. Conn, Nicholas I. M. Gould, and Philippe L. Toint. Numerical experiments with the lancelot package (release a) for large-scale nonlinear optimization, 1993.

Andrew R. Conn, Nicholas I. M. Gould, and Philippe L. Toint. *Trust-region methods*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000. ISBN 0-89871-460-5.

J. Encarnation Dennis and Robert B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Classics in Applied Mathematics. Siam, Philadelphia, 1996.

Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. Ser. A 91, Math. Program., p.201-203, Oct 2001.

R. Fletcher. *Practical methods of optimization; (2nd ed.).* Wiley-Interscience, New York, NY, USA, 1987.

Philip E. Gill and Walter Murray. Conjugate-gradient methods for large-scale nonlinear optimization, report sol 79-15, department of operations research, stanford university, stanford, ca., 1979.

Philip E. Gill, Walter Murray, and Margaret H. Wright. *Practical Optimization.* Elsevier, June 1981.

G. H. Golub and C. F. Van Loan. *Matrix Computations.* The Johns Hopkins University Press, Baltimore, MD, USA, second edition, 1989.

Nicholas I. M. Gould, Dominique Orban, Annick Sartenaer, and Philippe L. Toint. Superlinear convergence of primal-dual interior point algorithms for nonlinear programming. *SIAM J. on Optimization*, 11(4):974–1002, 2000.

Nicholas I. M. Gould, Dominique Orban, and Philippe L. Toint. Cuter and sifdec: A constrained and unconstrained testing environment, revisited. *ACM Trans. Math. Softw.*, 29(4):373–394, 2003.

Nicholas I. M. Gould, Dominique Orban, and Philippe L. Toint. Numerical methods for large-scale optimization. *Acta Numerica*, 14:299–361, 2005.

C. Tim Kelley. *Iterative Methods for Optimization.* Frontiers in Applied Mathematics. Siam, Philadelphia, 1999.

Jorge J. Moré. Trust regions and projected gradients. In M. Iri and K. Yajima, editors, *Systems Modelling and Optimization*, Lecture Notes in Control and Information Sciences, volume 113, pages 1–13, Berlin, Heidelberg, New York, Tokyo, 1988. Springer-Verlag.

Jorge J. Moré and D. Thuente. Line search algorithms with guaranteed sufficient decrease, acm transactions on mathematical software 20, no. 3, pp. 286-307., 1994.

Jorge J. Moré and Stephen J. Wright. *Optimization Software Guide.* Frontiers in Applied Mathematics. Siam, Philadelphia, 1993.

Stephen G. Nash. Newton-type minimization via the Lánczos method. *SIAM J. Numer. Anal.*, 21(4):770–788, 1984a.

Stephen G. Nash. User's guide for TN/TNBC: Fortran routines for nonlinear optimization. Report 397, Mathematical Sciences Dept., The Johns Hopkins Univ., Baltimore, MD, 1984b.

Stephen G. Nash. Preconditioning of truncated-newton methods. *SIAM Journal on Scientific and Statistical Computing*, 6(3):599–616, 1985.

Stephen G. Nash. A survey of truncated-newton methods. *J. Comput. Appl. Math.*, 124(1-2):45–59, 2000.

Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer Series in Operations Research. Springer, Berlin, Heidelberg, 1999.

G. N. Vanderplaats. *Numerical Optimization Techniques for Engineering Design: With Applications*. McGraw Hill, New York, NY, USA, 1984.

G. N. Vanderplaats. *DOT Users Manual, Version 4.20*. Vanderplaats Research & Development, Inc., Colorado Springs, USA, 1995.

Curtis R. Vogel. *Computational Methods for Inverse Problems*. Frontiers in Applied Mathematics. SIAM, Philadelphia, 2002.

Andreas Wächter and Lorenz T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Math. Program.*, 106(1):25–57, 2006.

Stephen J. Wright. *Primal-Dual Interior-Point Methods*. Siam, Philadelphia, 1997.

Ciyou Zhu, Richard H. Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Trans. Math. Softw.*, 23(4):550–560, December 1997.