An Implementation of an Algorithm for Nonlinear Programming Based on Piecewise Linear Models

Richard Byrd Jorge Nocedal Richard Waltz Yuchen Wu

April 21, 2011

Abstract

This is a progress report on an implementation of the active-set method for nonlinear programming proposed in [6] that employs piecewise linear models in the active-set prediction phase. The motivation for this work is to develop an algorithm that is capable of solving large-scale problems, including those with a large reduced space. Unlike SQP methods, which solve a general quadratic program at each iteration, the proposed algorithm solves linear programs and equality constrained quadratic programs – both of which scale up well in the number of variables and constraints. The algorithm is implemented in the KNITRO software package and contains a variety of features to handle difficulties occurring in practice, such as Jacobian rank deficiencies, infeasibility, and ill-conditioning. Particular attention is given to the implementation of the piecewise linear models to achieve economy of computation and to conform with the theoretical guidelines given in [6]. Numerical results comparing the new algorithm with two established active-set solvers, SNOPT and KNITRO/ACTIVE, are presented.

1 Introduction

In [6] the authors presented a method for large-scale nonlinear programming that uses piecewise linear models in the active-set identification phase. [6] gives a general description of the method and studies it global and local convergence properties. In this report, we describe ongoing work on a practical implementation of this approach and study its numerical performance.

We consider nonlinear programming problems of the form

$$\begin{aligned} \min_x \quad & f(x) && \text{(1.1a)}\\ \text{s.t.} \quad & h^{[i]}(x) = 0, \quad i \in \mathcal{E} && \text{(1.1b)}\\ & g^{[i]}(x) \geq 0, \quad i \in \mathcal{I} && \text{(1.1c)}\\ & \underline{x} \leq x \leq \overline{x}, && \text{(1.1d)} \end{aligned}$$

where the objective function $f : \mathbb{R}^n \to \mathbb{R}$, the equality constraint functions $h^{[i]} : \mathbb{R}^n \to \mathbb{R}$, $i \in \mathcal{E}$, and the inequality constraint functions $g^{[i]} : \mathbb{R}^n \to \mathbb{R}$, $i \in \mathcal{I}$ are assumed to be twice differentiable, and $\underline{x}, \overline{x} \in \mathbb{R}^n$ are lower and upper bounds on the optimization variables x.

A variety of algorithms and software is available for solving nonlinear programming problems of the form (2.2). Most of them fall into three categories: interior point, active set and augmented Lagrangian methods. Generally speaking, interior point methods [22, 5, 23, 21] are very effective for solving large problems, but are not always robust and can be inefficient when a series of related problems must be solved. Active-set SQP methods [14, 11] are perhaps the most robust techniques for small and medium-size problems, particularly when the problem contains constraint degeneracies. The more recent SLP-QP [8, 12, 7, 2] methods, in which the active set prediction phase is performed by solving a linear program, have emerged as serious competitors to SQP methods as they scale up better with the dimension of the problem, but are not yet as robust as SQP methods. Augmented Lagrangian methods are effective for special classes of problems, such as those containing only a few constraints or so many variables and constraints to render interior point methods impractical.

The algorithm discussed in this paper aims to combine some of the properties of SQP and SLP-QP methods. Like SQP methods, it incorporates curvature information about the Hessian of the Lagrangian in the active set prediction phase. But since this curvature information is provided in the form of a piecewise linear model, which can be recast as a linear program, the proposal algorithm also shares some important characteristics with SLP-QP methods. The novelty of the approach lies in the construction of the piecewise linear models, which must be simple enough to make the iteration affordable, yet useful enough to provide significantly better information than a linear model. We will refer to the new algorithm as the *PLA algorithm*, where the acronym stands for "piecewise linear approximation".

The paper is organized into 5 sections. A detailed description of the algorithm is presented in sections 2 and 3. Some algorithmic options are discussed in section 4, and a comparison with several well known solvers is presented in section 5.

Notation. We denote the component of a vector by a superscript, i.e., $d^{[i]}$, and reserve the subscript k (as in x_k) to denote the iteration number.

2 The PLA Algorithm

Let us begin by summarizing the algorithm described in [6]. We denote the Lagrangian of the nonlinear program (2.2) as

$$L(x, \lambda, \mu) = f(x) - \sum_{i \in \mathcal{E}} \lambda^{[i]} h^{[i]}(x) - \sum_{i \in \mathcal{I}} \mu^{[i]} g^{[i]}(x) - \iota^T (x - \underline{x}) + \kappa^T (x - \overline{x}), \tag{2.1}$$

where $\lambda \in \mathbb{R}^{|\mathcal{E}|}$, $\mu \in \mathbb{R}^{|\mathcal{I}|}$, $\iota \in \mathbb{R}^n$, $\kappa \in \mathbb{R}^n$ are vectors of multipliers.

Each iteration of the algorithm consists of two phases – an active-set identification phase and an equality constrained (EQP) phase. Given a primal-dual iterate (x_k, λ_k, μ_k), the algorithm starts the active-set identification phase by building a convex piece-wise linear

problem that normally has the form

$$\begin{aligned} \min_{d} \quad & \nabla f(x_k)^T d + \Gamma_k(d) && \text{(2.2a)} \\ \text{s.t.} \quad & h^{[i]}(x_k) + \nabla h^{[i]}(x_k)^T d = 0, \quad i \in \mathcal{E} && \text{(2.2b)} \\ & g^{[i]}(x_k) + \nabla g^{[i]}(x_k)^T d \geq 0, \quad i \in \mathcal{I} && \text{(2.2c)} \\ & \underline{x} - x_k \leq d \leq \overline{x} - x_k, && \text{(2.2d)} \end{aligned}$$

where $\Gamma_k(d)$ is a convex, nonnegative, piece-wise linear function. In our practical implementation (see section 2.1), we make two important changes to this subproblem. In order to ensure that the constraints in (2.2) are always feasible, we reformulate this problem via an exact penalty approach. In addition, we include a trust region constraint in (2.2) for greater control and robustness.

We denote the solution of (2.2) as d_k^{pla}, where the superscript is an abbreviation of "piecewise linear approximation". The PLA step d_k^{pla} defines a working set $\mathcal{W}_k$, which is a linearly independent subset of the constraints in (2.2b)-(2.2d) that are active at the solution of (2.2). Based on the step d_k^{pla}, we also compute a Cauchy point d_k^{c} (as discussed below) that ensures global convergence of the iteration.

Upon completion of the active-set identification phase, the algorithm solves an equality constrained quadratic problem (EQP) in which the constraints in the working set $\mathcal{W}_k$ are imposed as equalities, while the rest of the constraints are temporarily ignored. We solve the EQP subproblem by applying a projected conjugate gradient method, and denote its solution as d_k^{E}.

The trial step of the algorithm is defined to lie on the line segment from d_k^{E} to d_k^{c}. The trial step d_k^{trial} is accepted only if it gives sufficient decrease in the ℓ_1 merit function

$$\phi_\pi(x) = f(x) + \pi \sum_{i \in \mathcal{E}} |h^{[i]}(x)| + \pi \sum_{i \in \mathcal{I}} \max(0, -g^{[i]}(x)), \tag{2.3}$$

where the scalar $\pi > 0$ is the penalty parameter. Finally, based on whether d_k^{trial} is accepted, we adjust the trust region size to be used in the next iteration.

In [6], we described a procedure for constructing the piecewise linear model Γ_k in (2.2) at each iteration, and have given conditions under which global and local convergence guarantees can be established. The goal of this paper is to provide a full-fledged implementation of the algorithm that is equipped to deal with the many pitfalls that may arise in practice. Our implementation is written within the KNITRO software package, and is designed for efficiency in the large scale case.

In the following sections we address they key algorithmic components of the algorithm and its software implementation. In the description of the algorithm that follows, we denote the Hessian of this Lagrangian function (2.1) by

$$H(x, \lambda, \mu) \triangleq \nabla^2 f(x) + \sum_{i=1}^{|\mathcal{E}|} \lambda^{[i]} \nabla^2 h^{[i]}(x) + \sum_{i=1}^{|\mathcal{I}|} \mu^{[i]} \nabla^2 g^{[i]}(x). \tag{2.4}$$

2.1 The Working Set Prediction Phase

To ensure that the PLA algorithm is always well defined, we first need to deal with the possible inconsistency of the linearized constraints in the PLA subproblem (2.2). We employ an ℓ_1 penalty approach to do so. Specifically, we move the linearized constraints (2.2b)-(2.2c) into the objective function in the form of penalty terms, i.e., we define the function

$$\begin{aligned}\ell_{\pi_k}(d) = \ & \nabla f(x_k)^T d + \Gamma_k(d) + \\ & \pi_k \left[\sum_{i \in \mathcal{E}} \left| h^i(x_k) + \nabla h^i(x_k)^T d \right| + \sum_{i \in \mathcal{I}} \max\left(0, -g^i(x) - \nabla g^i(x)^T d\right) \right],\end{aligned} \tag{2.5}$$

and pose the new PLA subproblem as

$$\begin{aligned}\min_d \ & \ell_{\pi_k}(d) \\ \text{s.t.} \ & \underline{x} - x_k \leq d \leq \overline{x} - x_k,\end{aligned} \tag{2.6}$$

where $\pi_k > 0$ is a penalty parameter used to balance the goal of decreasing the objective (2.2a) versus making progress on satisfying the linearized constraints. The objective function of (2.6) is nonsmooth, but (2.6) can be written as an equivalent linear program. The solution methodology and precise form for (2.6), as well as the construction of the piecewise linear function Γ_k, are discussed in greater detail in Section 3.

Since we are using a penalty formulation to ensure that the PLA subproblem is always feasible, we need to develop a strategy for updating the penalty parameter π_k in an effective way. If the algorithm is struggling to get feasible it may be necessary to increase the penalty parameter. However, increasing the penalty parameter too much can lead to inefficiencies.

We employ the so-called *steering rules*, which were introduced in [2, 4]. The steering rules impose two main requirements on the penalty parameter π_k. First, it should be chosen large enough to enforce that sufficient progress is made in reducing a measure of linearized feasibility, which is defined as

$$m_x(d) = \sum_{i \in \mathcal{E}} |h_i(x) + \nabla h_i(x)^T d| + \sum_{i \in \mathcal{I}} [g_i(x) + \nabla g_i(x)^T d]^-, \tag{2.7}$$

at each iteration. More specifically, the steering rules require that, when the linearized constraints are not feasible, π_k should ensure that

$$m_x(0) - m_x(d_k^{\text{pla}}(\pi_k)) \geq \epsilon_1 [m_x(0) - m_x(d_k^{\text{pla}}(\infty))], \tag{2.8}$$

where $\epsilon_1 \in (0, 1)$ is a prescribed parameter, and $d_k^{\text{pla}}(\pi)$ is the solution of the PLA subproblem (2.6) given penalty parameter π. In addition, the steering rules require that we increase π_k further (if necessary) to ensure that

$$\ell_{\pi_k}(0) - \ell_{\pi_k}((d_k^{\text{pla}}(\pi_k)) \geq \epsilon_2 \pi_k [m_x(0) - m_x(d_k^{\text{pla}}(\pi_k)], \tag{2.9}$$

where $\epsilon_2 \in (0, 1)$.

Algorithm 2.1 Steering Rules for Updating the Penalty Parameter

Input: x_k, π_{k-1} and the parameters ϵ_1 and ϵ_2. Output π_k.

1. Solve (2.6) with $(x_k,\ \pi_{k-1})$ to get $d_k^{\text{pla}}(\pi_{k-1})$. If $d_k^{\text{pla}}(\pi_{k-1})$ is linearly feasible (i.e., $m_x(d_k^{\text{pla}}(\pi_{k-1})) = 0$), set $\pi_+ \leftarrow \pi_{k-1}$ and proceed to step 4.

2. Solve (2.6) with $(x_k,\ \infty)$ to get $d_k^{\text{pla}}(\infty)$. If $d_k^{\text{pla}}(\infty)$ is linearly feasible, choose some $\pi_+ \in (\pi_{k-1}, \infty]$ such that $m_x(d_k^{\text{pla}}(\pi_+)) = 0$ and proceed to step 4.

3. Choose some $\pi_+ \in [\pi_{k-1}, \infty]$ such that (2.8) is satisfied.

4. If π_+ satisfies (2.9), set $\pi_k \leftarrow \pi_+$, else choose, $\pi_k > \pi_+$ to satisfy (2.9).

We describe the steering rules routine in Algorithm 2.1, which we use in our implementation of the PLA algorithm. This routine is adapted from Algorithm 9.1 in [2], where it was presented in the context of SLP-QP methods.

After we have chosen an appropriate value for our penalty parameter using Algorithm 2.1 and have solved the PLA subproblem (2.6) to obtain a step $d_k^{\text{pla}}(\pi_k)$, we define our working set $\mathcal{W}_k$ to be some linearly independent set of the active constraints (2.2b)-(2.2d). It is important that are working set be linearly independent to avoid numerical difficulties for our EQP subproblem (2.10), which will impose these constraints as equalities. Extracting this linearly independent subset is computational challenging in general. Fortunately, modern linear programming software, such as CPLEX[16] and CLP [17] produce an independent working set as a byproduct of the solution process.

2.2 Trust Region Framework and Cauchy Step

The algorithm follows a trust region framework and promotes convergence by imposing decrease in the ℓ_1 merit function (2.3). Notice that we have ignored the bounds $\underline{x}$ and $\overline{x}$ in the definition of ϕ, which is safe to do as all PLA iterates will be restricted to these bounds.

In the trust region framework, every iteration of the PLA algorithm is endowed with a trust region. Let us consider the k-th iteration of the algorithm, and denote the trust region size as Δ_k.

After we have obtained the step d_k^{pla} by solving the PLA subproblem as described in the previous section, we then compute a Cauchy point x_k^{c} that can be used to establish global convergence of the algorithm ([6]). In order to compute the Cauchy point, first, we define the quadratic model function

$$\mathcal{Q}_{\pi_k}(d) = \ell_{\pi_k}(d) + \tfrac{1}{2} d^T W_k d,$$

where W_k is the Hessian of the Lagrangian (2.4) at iteration k, or some symmetric approximation of it. This model function is a quadratic model of our ℓ_1 merit function (2.3). We then define the Cauchy step $d_k^{\text{c}} \triangleq \alpha_k^{\text{c}} d_k^{\text{pla}}$ with the step length α_k^{c} chosen to be the

approximate solution of

$$\min_{0\leq\alpha\leq\Delta_k/\|d_k^{\mathrm{pla}}\|_2} \mathcal{Q}_{\pi_k}(d),$$

and set the Cauchy point as

$$x_k^{\mathrm{c}} = x_k + \alpha_k^{\mathrm{c}} d_k^{\mathrm{pla}}.$$

The Cauchy point is a crucial component to ensure global convergence in any trust region method [9].

2.3 The Equality Constrained QP Subproblem

Upon completion of the PLA working set determination phase, the algorithm enters the EQP phase where it seeks to solve the following equality constrained quadratic problem using the working set $\mathcal{W}_k$

$$\begin{aligned}
\min_{d} \quad & \tfrac{1}{2} d^T W_k d + \nabla f(x_k)^T d && \text{(2.10a)}\\
\text{s.t.} \quad & h^{[i]}(x_k) + \nabla h^{[i]}(x_k)^T d = 0, \quad i \in \mathcal{E} \cap \mathcal{W}_k && \text{(2.10b)}\\
& g^{[i]}(x_k) + \nabla g^{[i]}(x_k)^T d = 0, \quad i \in \mathcal{I} \cap \mathcal{W}_k && \text{(2.10c)}\\
& (\underline{x} - x_k - d)^{[i]} = 0, \quad i \in \mathcal{B}^L \cap \mathcal{W}_k && \text{(2.10d)}\\
& (\overline{x} - x_k - d)^{[i]} = 0, \quad i \in \mathcal{B}^U \cap \mathcal{W}_k && \text{(2.10e)}\\
& \|d\|_2 \leq \Delta_k. && \text{(2.10f)}
\end{aligned}$$

Here we have defined the sets $\mathcal{L}$ and $\mathcal{U}$ to be the sets of simple lower and upper bound constraints, respectively, given by (2.2d).

Note, however, that this problem may be infeasible due to the presence of the trust region constraint (2.10f). To overcome this obstacle, we relax the constraints (2.10b)-(2.10e), if necessary, by replacing the zeros on the right-hand side of (2.10b)-(2.10e), by

$$\begin{aligned}
r_{\mathcal{E}}^{[i]} &= h^{[i]}(x_k) + \nabla h^{[i]}(x_k)^T d_k^{\mathrm{N}}, \quad i \in \mathcal{E} \cap \mathcal{W}_k, && \text{(2.11)}\\
r_{\mathcal{I}}^{[i]} &= g^{[i]}(x_k) + \nabla g^{[i]}(x_k)^T d_k^{\mathrm{N}}, \quad i \in \mathcal{I} \cap \mathcal{W}_k, && \text{(2.12)}\\
r_{\mathcal{L}}^{[i]} &= (\underline{x} - x_k - d_k^{\mathrm{N}})^{[i]}, \quad i \in \mathcal{B}^L \cap \mathcal{W}_k, && \text{(2.13)}\\
r_{\mathcal{U}}^{[i]} &= (\overline{x} - x_k - d_k^{\mathrm{N}})^{[i]}, \quad i \in \mathcal{B}^U \cap \mathcal{W}_k, && \text{(2.14)}
\end{aligned}$$

where d_k^{N} is an orthogonal projection of x_k onto the subspace defined by $\mathcal{W}_k$, which has been truncated, if necessary, to satisfy $\|d_k^{\mathrm{N}}\| \leq 0.8\Delta_k$. This modification ensures that the subproblem is always feasible since d_k^{N} is feasible for the relaxed EQP.

We then approximately solve the resulting EQP subproblem using a projected conjugate gradient (PCG) approach, and define the EQP point as

$$x_k^{\mathrm{E}} = x_k + d_k^{\mathrm{E}},$$

where d_k^{E} is the (approximate) optimal solution to the (potentially relaxed) EQP subproblem. See [5, 2, 19] for details on using a PCG approach to solve a problem of this form.

2.4 The Trial Point

After both the PLA and EQP steps are computed, the trial point of the algorithm, denoted as x_k^{trial}, is then defined to lie on the line segment from x_k^{c} to x_k^{E} and satisfy all the bound constraints (1.1d). This trial step is accepted if it gives sufficient decrease in an ℓ_1 merit function ϕ.

More specifically, let us define the segment from the Cauchy point to the EQP point as $d_k^{\text{CE}} = d_k^{\text{E}} - d_k^{\text{c}}$. Then we define the trial step as

$$d_k^{\text{trial}} = d_k^{\text{c}} + \alpha_k \beta_k d_k^{\text{CE}}, \tag{2.15}$$

where $\beta_k \in [0, 1]$ is a truncation factor used to ensure that the trial point satisfies all the simple problem bounds (1.1d), and $\alpha_k \in [0, 1]$ is chosen to satisfy the sufficient decrease condition

$$\phi_{\pi_k}(x_k + p_k) \leq \phi_{\pi_k}(x_k) - \sigma qred_{\pi_k}(p_k), \quad \sigma \in (0, 1), \tag{2.16}$$

where

$$qred_{\pi_k}(d) = \mathcal{Q}_{\pi_k}(0) - \mathcal{Q}_{\pi_k}(d). \tag{2.17}$$

The steplength α_k is computed using a simple backtracking linesearch and is automatically set to zero after a fixed number of backtracks. Setting $\alpha_k = 0$ has the affect of taking the Cauchy point as the trial point (where the Cauchy point has been defined to satisfy the sufficient decrease condition).

After the trial point is accepted or rejected, the trust region radius is adjusted based on established techniques.

2.5 The Second Order Correction (SOC) Step

The Maratos effect [18, 20], is a well known phenomenon where steps that make good progress toward the optimum could still be rejected by the penalty function, causing sluggish convergence even when the iterates are close to the optimal solution. In our implementation of PLA we adopt a common remedy to the Maratos effect based on the employment of so-called second order correction steps [19], which can be computed and used in the following manner. At iteration k, we recall that x_k is the primal iterate, and $\mathcal{W}_k$ is the working set identified by the PLA subproblem. We let x_k^{trial} denote the trial point computed at the end of the EQP phase (in the context of Algorithm 2.2 that is $x_k + d_k^{\text{c}} + \alpha_k \beta_k d_k^{\text{CE}}$), and consider the case when x_k^{trial} does not render sufficient decrease of the penalty function ϕ_{π_k}. Instead of rejecting x_k^{trial}, we first compute d_k^{S}, the second-order correction (SOC) step, to be the minimum norm solution of the system

$$A^{\mathcal{W}_k}(x_k) d + c^{\mathcal{W}_k}(x_k^{\text{trial}}) = 0,$$

where $c^{\mathcal{W}_k}(x_k^{\text{trial}})$ is the value of the working set constraints at the trial point x_k^{trial}, and $A^{\mathcal{W}_k}(x_k)$ is the Jacobian of the working set constraints at the current point. We then make a second test on the penalty function with the new trial point defined as

$$x_k^{\text{trial}} \leftarrow x_k^{\text{trial}} + \tau_k^{\text{S}} d_k^{\text{S}},$$

where the scalar $\tau_k^S \in [0, 1]$ is a backtracking parameter to ensure that the new trial point satisfies all the bounds of (1.1d).

2.6 Other Implementation Details

The Lagrange multipliers for the constraints in the working set are computed using a least squares approach. Least squares multipliers corresponding to inequality constraints which are negative are reset to zero. Lagrange multipliers corresponding to constraints not in the working set are set to zero.

2.7 The Complete Algorithm

By adopting this trust region framework and incorporating the various enhancements described above, we obtain a practical PLA algorithm, which we summarize in Algorithm 2.2. The novelty of the PLA algorithm lies in Step 1-2, which is yet to be specified. This is the subject of the next section.

Algorithm 2.2 A Practical PLA Algorithm

Initialize: $x_0, \lambda_0, \mu_0, \iota_0, \kappa_0, f_0, c_0, \nabla f_0, A_0, W_0$.

for iteration $k = 0, \cdots,$

1. Construct the PLA subproblem (2.6);

2. Solve (2.6) to obtain the step d_k^{pla} and working set $\mathcal{W}_k$ while applying Algorithm 2.1 to update the penalty parameter π_k;

3. Approximately minimize $\mathcal{Q}_k(d)$ along d_k^{pla} subject to the trust region Δ_k to find the Cauchy point x_k^{c};

4. Find the EQP point x_k^{E} by solving a (potentially relaxed) EQP subproblem (2.10) with trust region Δ_k;

5. Find $\alpha_k \in [0, 1]$, $\beta_k \in [0, 1]$ and define the trial point $x_k^{\text{trial}} \triangleq \alpha_k \beta_k x_k^{\text{E}} + (1 - \alpha_k \beta_k) x_k^{\text{C}}$;

6. Evalute the objective f and constraints c at the trial point, x_k^{trial};

7. Compute $qred = \mathcal{Q}_{\pi_k}(0) - \mathcal{Q}_{\pi_k}(d_k^{\text{trial}})$, $ared = \phi_{\pi_k}(x_k) - \phi_{\pi_k}(x_k^{\text{trial}})$;

   if $\frac{ared}{qred} \geq$ tolerance /* successful step */

   $x_{k+1} \leftarrow x_k^{\text{trial}}$ and possibly increase the trust region size;

   go to Step 9;

   end(if)

8. Compute the SOC step and update x_k^{trial}, $f(x_k^{\text{trial}})$, $c(x_k^{\text{trial}})$, $ared = \phi_{\pi_k}(x_k) - \phi_{\pi_k}(x_k^{\text{trial}})$;

   if $\frac{ared}{qred} \geq$ tolerance /* successful step */

   $x_{k+1} \leftarrow x_k^{\text{trial}}$ and possibly increase the trust region size;

   else

   $x_{k+1} \leftarrow x_k$ and decrease trust region size;

   end(if)

9. If step was successful, update the gradient information, Lagrange multipliers, and the Hessian of the Lagrangian (or an approximation) at the new point, x_{k+1};

end(for)

3 Construction of the Piecewise Linear Model

The piece-wise linear function $\Gamma_k(d)$, which appears in the objective function of (2.2), is defined to be an approximation to the quadratic model $\frac{1}{2}d^T H_k d$. For our approach to be practical, we require that Γ_k be a separable function (possibly after a change of variables). This implies that for any $d \in \mathbb{R}^n$, we can write

$$\Gamma_k(d) = \sum_{j=1}^{n} \Gamma_k^{[j]}(d^{[j]}).$$

Each one-dimensional function $\Gamma_k^{[j]}$, $j = 1, \cdots, n$, is constructed by interpolating a quadratic function $d^T B_k d$ at $2r+1$ interpolation points, where B_k is a diagonal matrix

For each component of Γ_k, we want $\Gamma_k^{[j]}$, $j = 1, \cdots, n$, to approximate a one dimensional quadratic function, which we define as

$$\gamma_k^{[j]}(t) \triangleq \frac{1}{2} b_k^{[j]} t^2. \tag{3.1}$$

3.1 Interpolation Points

Given $\{b_k^{[j]}\}$ and $\gamma_k^{[j]}$, we would like $\Gamma_k^{[j]}$ to be the piece-wise linear function whose values and slopes coincide with the quadratic function $\gamma_k^{[j]}$ at $2r+1$ *interpolation points* denoted as $t_{k,l}^{[j]}$, $l = 0, \cdots, 2r$. Without loss of generality, we assume that $t_{k,l}^{[j]}$ increases with l. In the following we consider three different cases of $x_k^{[j]}$ and define the interpolation points $\{t_{k,l}^{[j]}\}_{l=0}^{2r}$ accordingly.

Case 1. If $|x_k^{[j]} - \underline{x}^{[j]}| \leq 10^{-6}$, then we let $t_{k,0}^{[j]} = D_k^{[j]} = 0$, choose $U_k^{[j]} > 0$, and set

$$t_{k,l}^{[j]} = 0.3^{2r-l} U_k^{[j]}, \text{ for } l = 1, \cdots, 2r;$$

Case 2. If $|x_k^{[j]} - \overline{x}^{[j]}| \leq 10^{-6}$, then we let $t_{k,2r}^{[j]} = U_k^{[j]} = 0$, choose $D_k^{[j]} > 0$, and set

$$t_{k,l}^{[j]} = -0.3^{l} D_k^{[j]}, \text{ for } l = 0, \cdots, 2r-1;$$

Case 3. Otherwise, we choose $U_k^{[j]} > 0$, $D_k^{[j]} > 0$, and set

$$t_{k,l}^{[j]} = \begin{cases} -0.3^l D_k^{[j]} & \text{for } l = 0, \cdots, r-1, \\ 0 & \text{for } l = r, \\ 0.3^{2r-l} U_k^{[j]} & \text{for } l = r+1, \cdots, 2r, \end{cases}$$

where the positive scalars $D_k^{[j]}, U_k^{[j]}$ represent the *interpolation range* of negative and positive interpolation points respectively, which are yet to be determined.

In broad terms, we would like $U_k^{[j]}$ and $D_k^{[j]}$ to yield a piecewise linear problem (2.2) that is bounded, and estimates a good working set $\mathcal{W}_k$. In our implementation, they are computed in the following steps. First, an initial guess is given such that

$$U_k^{[j]} \leftarrow \tilde{U}_k^{[j]} \triangleq \min(2\Delta_k/n,\ \overline{x}^{[j]} - x_k^{[j]}), \quad D_k^{[j]} \leftarrow \tilde{D}_k^{[j]} \triangleq \min(2\Delta_k/n,\ x_k^{[j]} - \underline{x}^{[j]})$$

where, as we recall, Δ_k is the ℓ_2 trust region size at iteration k. This initial guess ensures that any point in the Cartesian product of the sets of interpolation points is at most $2\Delta_k$ measured in its ℓ_2 norm, and resides inside the bounds of the PLA subproblem (2.2).

Then, we make adjustments to the initial interpolation range guesses so that the resulting PLA subproblem (2.2) yields a step of *reasonable length*. Specifically, by computing $v_k^{[j]} = -\nabla^{[j]} f(x_k)/b_k^{[j]}$ to be the constrained minimum of the quadratic function $\gamma_k^{[j]}$, we observe that

$$2v_k^{[j]} < U_k^{[j]} \quad \Rightarrow \quad b_k^{[j]} U_k^{[j]} + \nabla^{[j]} f(x_k) > |\nabla^{[j]} f(x_k)|,$$

and that

$$-D_k^{[j]} < 2v_k^{[j]} \quad \Rightarrow \quad -b_k^{[j]} D_k^{[j]} + \nabla^{[j]} f(x_k) > |\nabla^{[j]} f(x_k)|.$$

Thus, if $-D_k^{[j]} < 2v_k^{[j]} < U_k^{[j]}$, we can deduce that the PLA objective (2.2) is bounded below, in which case no further modification is needed on the interpolation region. However, if $2v_k^{[j]} > U_k^{[j]}$, we consider three possible cases:

Case 1. If $2v_k^{[j]} < \overline{x}^{[j]} - x_k^{[j]}$ and $2v_k^{[j]} < 2\tilde{U}_k^{[j]}$, which means that $2v_k^{[j]}$ is not too much larger than initial guess of positive interpolation range, we will increase the positive interpolation range such that $U_k^{[j]} \leftarrow 2v_k^{[j]}$;

Case 2. Else, if $\overline{x}^{[j]} - x_k^{[j]} < 2\tilde{U}_k^{[j]}$, which means that the upper bound on $d^{[j]}$ is not too far away, we simply equate the positive interpolation region to the upper bound, i.e., $U_k^{[j]} \leftarrow \overline{x}^{[j]} - x_k^{[j]}$;

Case 3. Otherwise, we let $U_k^{[j]} \leftarrow 2\tilde{U}_k^{[j]}$, and add j to the set $\mathcal{U}_k$. The set $\mathcal{U}_k$ remembers the indices j for which we will take further measures later on in the algorithm to prevent elongated PLA step along the positive orthant of $d^{[j]}$.

Similarly, if $2v_k^{[j]} < -D_k^{[j]}$, we also consider three possible cases:

Case 1. If $2v_k^{[j]} > \underline{x}^{[j]} - x_k^{[j]}$ and $2v_k^{[j]} > -2\tilde{D}_k^{[j]}$, which means that the magnitude of $2v_k^{[j]}$ is not too much larger than the initial guess of negative interpolation range, we will increase the negative interpolation range such that $D_k^{[j]} \leftarrow -2v_k^{[j]}$;

Case 2. Else, if $\underline{x}^{[j]} - x_k^{[j]} > -2\tilde{D}_k^{[j]}$, which means that the lower bound on $d^{[j]}$ is not too far away, we simply equate the negative interpolation region to the magnitude of the lower bound, i.e., $D_k^{[j]} \leftarrow x_k^{[j]} - \underline{x}^{[j]}$;

Case 3. Otherwise, we let $D_k^{[j]} \leftarrow 2\tilde{D}_k^{[j]}$, and add the index j to the set $\mathcal{D}_k$. The set $\mathcal{D}_k$ remembers the indices j for which we will take further measures later on in the algorithm to prevent elongated PLA step along the negative orthant of $d^{[j]}$.

This concludes the procedure of choosing of interpolation points.

3.2 Break Points and Slopes

So far, we have chosen the interpolation points $t_{k,l}^{[j]}$ for $l = 0, \cdots, 2r$. Recall that $\Gamma_k^{[j]}$ is a convex piecewise linear function whose values and slopes coincide with the quadratic function $\gamma_k^{[j]}$ at these $2r+1$ interpolation points. As a result, the j-th component of the PLA objective (2.2) has the following form

$$\max_{l=0\cdots 2r} \left\{ (\nabla^{[j]} f(x_k) + b_k^{[j]} t_{k,l}^{[j]}) d^{[j]} - \frac{1}{2} b_k^{[j]} (t_{k,l}^{[j]})^2 \right\},$$

which consists of $2r+1$ linear segments, with each of the two adjacent segments divided by a *break point*. These linear segments and the break points motivate us to express the j-th component of the PLA objective equivalently as

$$\nabla^{[j]} f(x_k) d^{[j]} + \Gamma_k^{[j]}(d^{[j]}) = \begin{cases} s_{k,0}^{[j]} d^{[j]} - \frac{1}{2} b_k^{[j]} (t_{k,0}^{[j]})^2 & d^{[j]} < z_{k,0}^{[j]} \\ s_{k,1}^{[j]} d^{[j]} - \frac{1}{2} b_k^{[j]} (t_{k,1}^{[j]})^2 & z_{k,0}^{[j]} \le d^{[j]} \le z_{k,1}^{[j]} \\ \quad \vdots & \\ s_{k,2r-1}^{[j]} d^{[j]} - \frac{1}{2} b_k^{[j]} (t_{k,2r-1}^{[j]})^2 & z_{k,2r-2}^{[j]} \le d^{[j]} \le z_{k,2r-1}^{[j]} \\ s_{k,2r}^{[j]} d^{[j]} - \frac{1}{2} b_k^{[j]} (t_{k,2r}^{[j]})^2 & d^{[j]} \ge z_{k,2r-1}^{[j]} \end{cases},$$

where the break points are defined as

$$z_{k,l}^{[j]} = \frac{1}{2}(t_{k,l}^{[j]} + t_{k,l+1}^{[j]}), \quad l = 0, \cdots, 2r-1,$$

and the slopes are set to be

$$s_{k,l}^{[j]} = \nabla^{[j]} f(x_k) + b_k^{[j]} t_{k,l}^{[j]}, \quad l = 0, \cdots, 2r.$$

Here we need to revisit the boundedness issue of the PLA subproblem (2.2). As discussed earlier when choosing interpolation points, the only case in which the PLA step is left unguarded is when $j \in \mathcal{U}_k \cup \mathcal{D}_k$. If $j \in \mathcal{U}_k$, which means that $d^{[j]}$ is not guarded from increasing, we will ensure that the last segment of $\nabla^{[j]} f(x_k) + \Gamma_k^{[j]}$ has sufficiently positive slope, which is achieved by overriding

$$s_{k,2r}^{[j]} \leftarrow \max\{\nabla^{[j]} f(x_k) + b_k^{[j]} t_{k,2r}^{[j]},\ \|\nabla f(x_k)\|\};$$

Similarly, if $j \in \mathcal{D}_k$, which means that $d^{[j]}$ is not guarded from decreasing, we will ensure that the first segment of $\nabla^{[j]} f(x_k) + \Gamma_k^{[j]}$ has sufficiently negative slope, which is achieved by overriding

$$s_{k,0}^{[j]} \leftarrow \min\{\nabla^{[j]} f(x_k) + b_k^{[j]} t_{k,0}^{[j]},\ -\|\nabla f(x_k)\|\}.$$

All the parts are now prepared to assemble the LP reformulation of the PLA subproblem (2.2), which we describe as follows.

3.3 Assembling the Delta-form LP

With the above stipulation of Γ_k, we can transform the PLA subproblem into a so-called Delta-form LP. The key step is to perform a substitution of variables

$$d = \sum_{l=0}^{2r} \delta_l,$$

such that

$$\delta_0 \le z_{k,0}, \quad \delta_{2r} \ge 0, \quad \text{and } 0 \le \delta_l \le z_{k,l} - z_{k,l-1} \text{ for } l = 1, \cdots 2r-1,$$

which transforms the PLA subproblem (2.2) into

$$\begin{aligned}
\min_{\delta_0,\cdots,\delta_{2r}} \quad & \sum_{l=0}^{2r} s_{k,l}^T \delta_l \\
\text{s.t.} \quad & h^{[i]}(x_k) + \nabla h^{[i]}(x_k)^T \sum_{l=0}^{2r} \delta_l = 0, \quad i \in \mathcal{E} \\
& g^{[i]}(x_k) + \nabla g^{[i]}(x_k)^T \sum_{l=0}^{2r} \delta_l \ge 0, \quad i \in \mathcal{I} \\
& \underline{x} - x_k \le \delta_0 \le z_{k,0} \\
& 0 \le \delta_{2r} \le \overline{x} - x_k - z_{k,2r-1} \\
& 0 \le \delta_l \le z_{k,l} - z_{k,l-1} \qquad\qquad l = 1, \cdots 2r-1.
\end{aligned} \tag{3.2}$$

Furthermore, since the constraints in (3.2) are not guaranteed to be compatible, we employ the penalty relaxation approach and obtain

$$\begin{aligned}
\min_{\delta_0,\cdots,\delta_{2r}} \quad & \sum_{l=0}^{2r} s_{k,l}^T \delta_l + \pi_k (u+v)^T \mathbf{1} + \pi_k w^T \mathbf{1} \\
\text{s.t.} \quad & h^{[i]}(x_k) + \nabla h^{[i]}(x_k)^T \sum_{l=0}^{2r} \delta_l = u - v, \quad i \in \mathcal{E} \\
& g^{[i]}(x_k) + \nabla g^{[i]}(x_k)^T \sum_{l=0}^{2r} \delta_l \ge w, \qquad i \in \mathcal{I} \\
& \underline{x} - x_k \le \delta_0 \le z_{k,0} \\
& 0 \le \delta_{2r} \le \overline{x} - x_k - z_{k,2r-1} \\
& 0 \le \delta_l \le z_{k,l} - z_{k,l-1} \qquad\qquad l = 1, \cdots 2r-1, \\
& u, v, w \ge 0,
\end{aligned} \tag{3.3}$$

which is the LP solved in Step 2 of Algorithm 1 to obtain d_k^{pla}.

4 Exploring the Algorithmic Options

As mentioned above, the matrix B_k can be defined as a diagonal approximation to the Hessian of the Lagrangian H_k or as a limited memory approximation. In this section we describe our choice for the matrix B_k that will be used in the numerical results to follow.

The first choice we consider is

$$b_k^{[j]} = \begin{cases} \text{mid}(b_{\min}, H_k^{[jj]}, b_{\max}) & j \notin \mathcal{Z}, \\ \|H_k\|_{\text{F}}/n^2 & j \in \mathcal{Z}, \end{cases} \tag{4.1}$$

where $H_k^{[jj]}$ is the j-th diagonal component of H_k; $b_{\max} = 10^5$ and $b_{\min} = 10^{-5}$ are the *upper* and *lower* limits for the quadratic coefficient; $\text{mid}(\cdot,\cdot,\cdot)$ is an operator that takes three scalar arguments and returns the middle value; $\|\cdot\|_{\text{F}}$ denotes the Frobenius norm; and $\mathcal{Z}$ is the index set of *structurally zero* variables such that

$$j \in \mathcal{Z} \quad \Leftrightarrow \quad \frac{\partial^2 L(x,\lambda,\mu)}{(\partial x^{[j]})^2} \equiv 0 \text{ for all } (x,\lambda,\mu).$$

Other choices of B are the subject of ongoing investigation.

5 Numerical Comparison with Other Solvers

In order to assess the potential of the proposed approach, which will henceforth be referred to as the PLA method, we test it here on the CUTEr [1, 15] set of problems coded in AMPL[13] format, and compare it with the state-of-the-art solvers KNITRO/ACTIVE [3, 24] and SNOPT [14]. Given that PLA is an active-set method, it is appropriate to compare it only with active-set solvers.

In all results reported in this section, the PLA algorithm uses the IBM CPLEX 12.0 solver [16] running the default dual simplex approach to solve the LP subproblems. KNITRO/ACTIVE implements a sequential linear-quadratic programming method with trust regions. SNOPT 7.2-1 is a line search SQP method in which the search direction is determined by an active-set method for convex quadratic programming. SNOPT employs only first derivatives of the objective function and constraints, and maintains a (limited memory) BFGS approximation to the reduced Hessian of a Lagrangian function, while PLA and KNITRO/ACTIVE both make use of exact second derivatives in the EQP phase. Even though the benchmark algorithms tested here are different from the PLA algorithm, they are useful for our purposes since they are generally regarded as some of the most effective active-set codes available for large-scale nonlinear optimization.

Our implementation of PLA follows the specifications in Algorithm 2.2, where we set $r = 3$, and thus the number of interpolation points per coordinate direction is $2r + 1 = 7$.

The results reported in this section were obtained under the following computational environment. We use an `Intel Core(TM)2 Quad CPU Q9400 2.66GHz` machine with 8,127,936 KB of total physical memory running the 64 bit version of Ubuntu Linux. Both the PLA method and KNITRO/ACTIVE were written in C, and compiled using the `gcc` compiler.

SNOPT is written in Fortran and compiled using the `f90` compiler. We impose an iteration limit of 10000 for both solvers for each problem, and for SNOPT we impose a major iteration limit of 50000. For all codes, we use a CPU limit of 8 hours. We mark the instance as a failure if one of these limits was reached by a solver.

The stopping conditions for PLA are the same as those in KNITRO/ACTIVE, which are described in detail in the KNITRO User's Manual [25]. We use the default stopping test for SNOPT, and the default stopping tolerances of 10^{-6} for all solvers.

5.1 Robustness

To evaluate the robustness of PLA, we compare it with KNITRO/ACTIVE on 3 problem categories: i) quadratic programs (QP), ii) bound constrained problems (BC), i.e., whose only constraints are bounds on the variables, and iii) general constrained problems (GC) that are neither QP nor BC. It is possible for a problem to be both a QP and a BC. In that case, the problem is categorized as QP. We summarize the distribution of the testing problems based on their categories and sizes in Table 1. We have used the value $n + m$ as an indicator of problem size, where n is the number of variables and m is the total number of constraints, i.e. $m = |\mathcal{E}| + |\mathcal{I}|$. As we can see, the 514 testing problems contain sufficient variety and cover a wide span of scales.

Problem size	QP	BC	GC	Total
VS ($1 \leq n + m < 100$)	49	49	200	298
S ($100 \leq n + m < 1000$)	10	10	56	76
M ($1000 \leq n + m < 10000$)	33	6	22	61
L ($10000 \leq n + m$)	49	3	27	79
Total	141	68	305	514

Table 1: Distribution of test problems based on sizes and types.

We measure robustness by counting in each category the number of problems for which each solver reported finding a (local) optimal solution. Note that there are a few problems in CUTEr for which no optimal solution exists. For example, the problem can be infeasible or unbounded. Being able to recognize and properly handle these cases is important for practical optimization software. However, these considerations are outside the scope of this paper. Since such instances affect both competing solvers in the same way, it is fair for us to simply treat them as failure for both solvers. In addition, it is possible that different solvers converge to different local minima of a problem. In this case, we count both solvers as having achieved optimality. The effect of this is negligible since the vast majority of test problems were found to converge to the same optimal point by both solvers.

In Table 2, we summarize the number (# opt) and percentage (% opt) of problems for which each solver found the solution, for each problem category. In Table 3, we compare

Problem category	Sample size	PLA # opt	PLA % opt	KNITRO/A # opt	KNITRO/A % opt
QP	141	134	95.0	131	92.9
BC	68	64	94.1	65	95.6
GC	305	282	92.5	284	93.1
Total	514	480	93.4	480	93.4

Table 2: Robustness of PLA and KNITRO/ACTIVE by problem category.

the robustness of PLA and KNITRO/ACTIVE based on different problem sizes. We observe that the robustness of PLA is almost the same as that of KNITRO/ACTIVE across all different problem sizes and problem types. This is encouraging as KNITRO/ACTIVE is considered to be a fairly robust algorithm and its implementation is mature.

Problem size	Sample size	PLA # opt	PLA % opt	KNITRO/A # opt	KNITRO/A % opt
VS	298	289	97.0	290	97.3
S	76	68	89.5	66	86.8
M	61	55	90.2	56	91.8
L	79	68	86.1	68	86.1
Total	514	480	93.4	480	93.4

Table 3: Robustness of PLA by problem size.

5.2 Comparing PLA and KNITRO/ACTIVE in Terms of Function Evaluations

We now study the performance of PLA and KNITRO/ACTIVE in terms of the number of function evaluations (FEV) taken to achieve optimality. We present the results in this section using the Dolan and Moré performance profiles[10]. In broad outline, $\pi_s(\tau)$ in the y-axis of a performance profile is defined as follows

$$\pi_s(\tau) = \frac{\text{number of problems where } r_{p,s} \leq \tau}{\text{total number of problems}}, \qquad \tau \geq 0, \tag{5.1}$$

where $r_{p,s}$ is the ratio between the cost (e.g., FEV) to solve problem p by solver s over the lowest cost required by any of the solvers.

The results are summarized in Figures 1, 2, 3. We observe that in all three categories, PLA outperforms KNITRO/ACTIVE in terms of requiring fewer function evaluations. This is highly encouraging and shows that the piecewise linear programs contain useful curvature information, which helps to produce better steps than the steps of the SLP-QP approach. From Figure 1, we note that PLA performs particularly well on QPs. Although there exist specialized approaches for solving quadratic programs, we find this result useful for the purpose of demonstrating the effect of the curvature information incorporated in the piecewise linear model when the Hessian of the Lagrangian is constant.

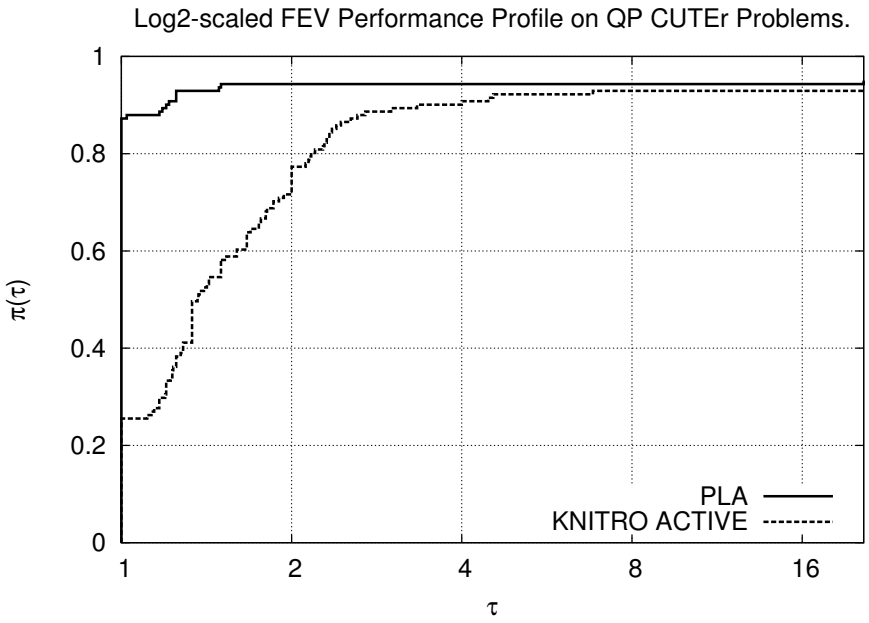

Figure 1: FEV comparison on QP CUTEr problems.

5.3 Comparing PLA and SNOPT in Terms of Function Evaluations

In Figures 4-5 we compare the performance of PLA and SNOPT in terms of CPU time and function evaluations for general nonlinear programs with a number of variables greater than or equal to 500 (only for larger problems are CPU times meaningful). We obtained a set of general nonlinear programs by omitting unconstrained problems, linear programs and quadratic programs. We also omitted indefinite quadratic programs because SNOPT with the AMPL interface does not work correctly on this class of problems. In Figure 6 we compare the two solvers, in terms of function evaluation, on the whole set of general nonlinear programs.

5.4 Final Remarks

This is a progress report on the development of a production-quality implementation of the PLA method. Improvements in the algorithm and in the implementation can be expected to lead to gains in performance. The following comments summarize some of the main observations made in this report.

- The PLA algorithm is already very robust.
- Timing results comparing PLA and KNITRO/ACTIVE have not yet been reported because the implementation has not been optimized in terms of speed. This is one of

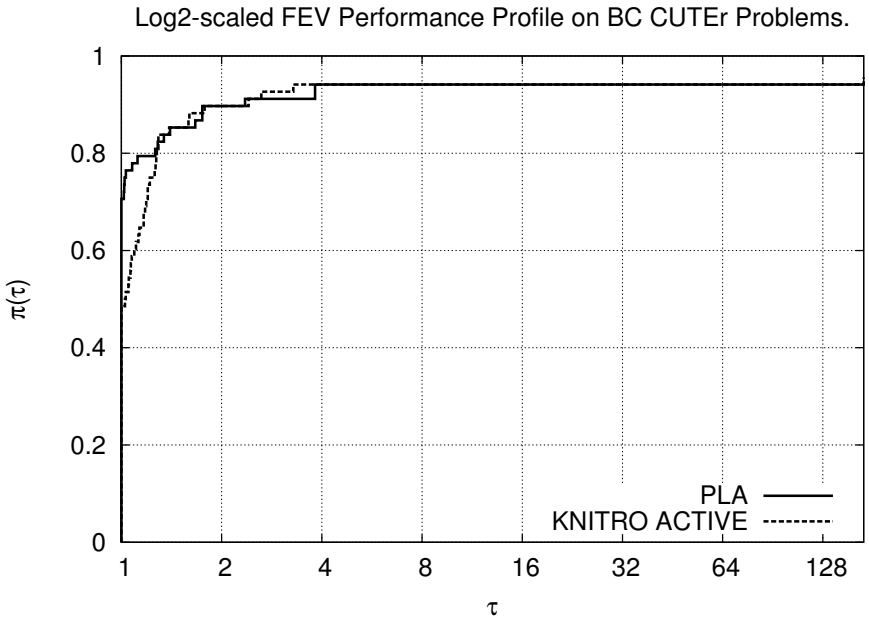

Figure 2: FEV comparison on BC CUTEr problems.

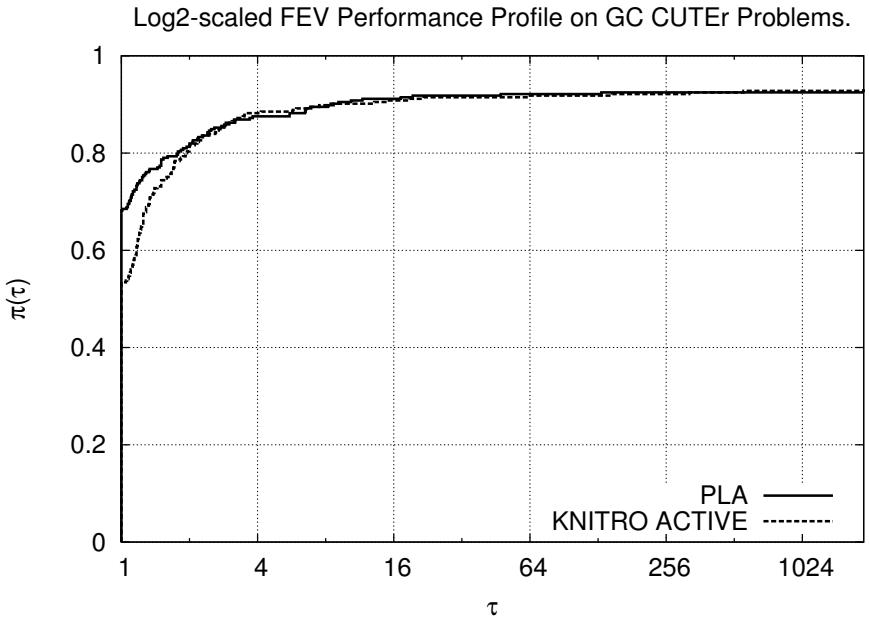

Figure 3: FEV comparison on GC CUTEr problems.

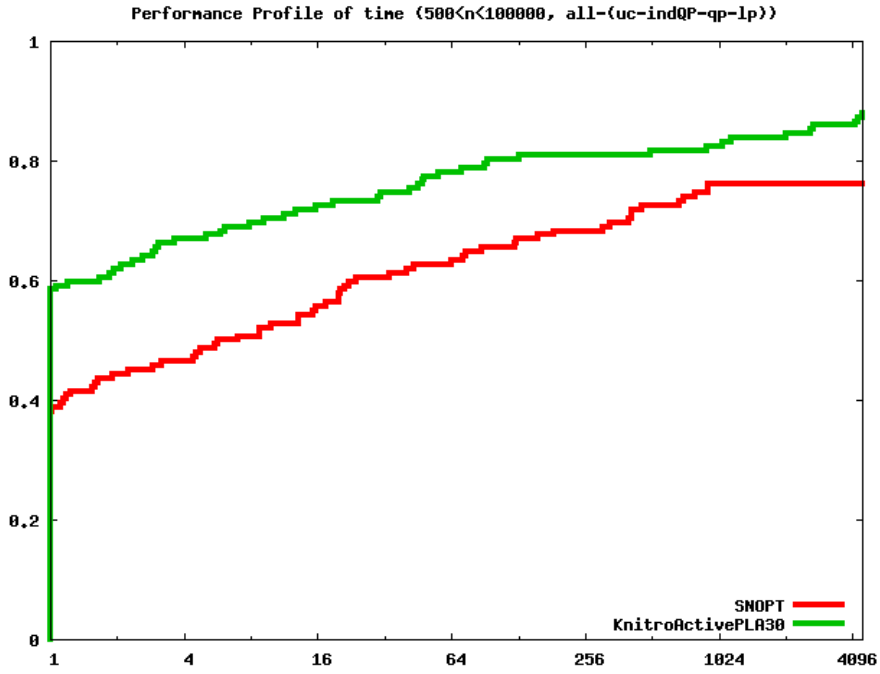

Figure 4: Comparison of PLA and SNOPT in terms of CPU time, for $n \geq 500$.

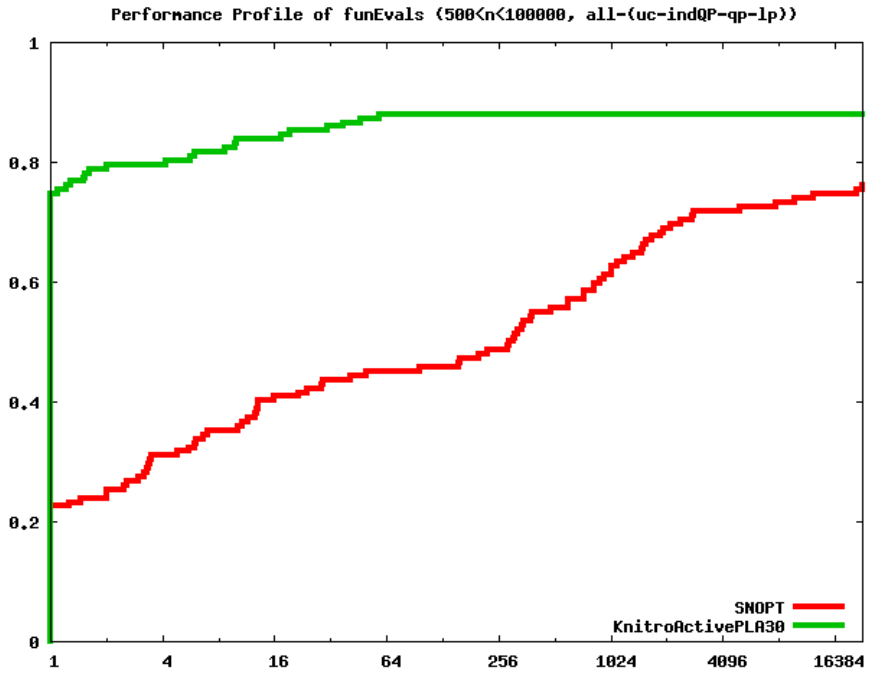

Figure 5: Comparison of PLA and SNOPT in terms of function evaluations time, for $n \geq 500$.

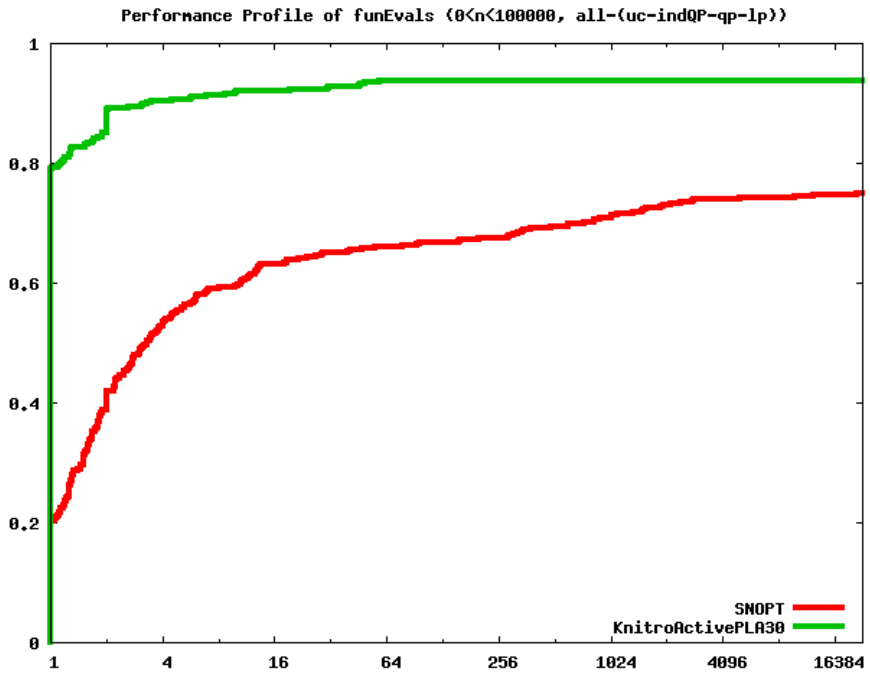

Figure 6: Comparison of PLA and SNOPT in terms of function evaluations on all general nonlinear programs.

the ongoing tasks that requires refinement in the algorithm, particularly in the identification phase. Nevertheless, we are already able to show improvements over SNOPT, both in terms of function evaluations and CPU time.

- The PLA method requires fewer iterations/function evaluations than both KNITRO/ACTIVE and SNOPT.
- The diagonal Hessian approximation used for the numerical results reported in this paper is not very sophisticated. One of the main questions under investigation is how to design an efficient implementation of the (diagonalized) L-BFGS Hessian approximation described in [6].

References

[1] I. Bongartz, A. R. Conn, N. I. M. Gould, and Ph. L. Toint. CUTE: Constrained and Unconstrained Testing Environment. *ACM Transactions on Mathematical Software*, 21(1):123–160, 1995.

[2] R. H. Byrd, N. I. M. Gould, J. Nocedal, and R. A. Waltz. An algorithm for nonlinear optimization using linear programming and equality constrained subproblems. *Mathematical Programming, Series B*, 100(1):27–48, 2004.

[3] R. H. Byrd, M. E. Hribar, and J. Nocedal. An interior point algorithm for large scale nonlinear programming. *SIAM Journal on Optimization*, 9(4):877–900, 1999.

[4] R. H. Byrd, J. Nocedal, and R. A. Waltz. Steering exact penalty methods. *Optimization Methods and Software*, 23(2), 2008.

[5] R. H. Byrd, J. Nocedal, and R.A. Waltz. KNITRO: An integrated package for nonlinear optimization. In G. di Pillo and M. Roma, editors, *Large-Scale Nonlinear Optimization*, pages 35–59. Springer, 2006.

[6] R.H. Byrd, J. Nocedal, R.A. Waltz, and Y. Wu. On the use of piecewise linear models in nonlinear programming. Technical report, Optimization Center, Northwestern University, 2009.

[7] R.H. Byrd and R.A. Waltz. An active-set algorithm for nonlinear programming using parametric linear programming. *Optimization Methods and Software*, pages 1029–4937, 2009.

[8] C. M. Chin and R. Fletcher. On the global convergence of an SLP-filter algorithm that takes EQP steps. *Mathematical Programming, Series A*, 96(1):161–177, 2003.

[9] A. R. Conn, N. I. M. Gould, and Ph. Toint. *Trust-region methods*. MPS-SIAM Series on Optimization. SIAM publications, Philadelphia, Pennsylvania, USA, 2000.

[10] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming, Series A*, 91:201–213, 2002.

[11] R. Fletcher and S. Leyffer. Nonlinear programming without a penalty function. *Mathematical Programming*, 91:239–269, 2002.

[12] R. Fletcher and E. Sainz de la Maza. Nonlinear programming and nonsmooth optimization by successive linear programming. *Mathematical Programming*, 43(3):235–256, 1989.

[13] R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Scientific Press, 1993. `www.ampl.com`.

[14] P. E. Gill, W. Murray, and M. A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Journal on Optimization*, 12:979–1006, 2002.

[15] N. I. M. Gould, D. Orban, and Ph. L. Toint. CUTEr (and SifDec), a Constrained and Unconstrained Testing Environment, revisited. Technical Report TR/PA/01/04, CERFACS, Toulouse, France, 2003. To appear in Transactions on Mathematical Software.

[16] ILOG CPLEX 8.0. *User's Manual*. ILOG SA, Gentilly, France, 2002.

[17] R. Lougee-Heimer. The common optimization interface for operations research. *IBM Journal of Research and Development*, pages 57–66, 2003.

[18] N. Maratos. *Exact penalty function algorithms for finite-dimensional and control optimization problems*. PhD thesis, University of London, London, England, 1978.

[19] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Series in Operations Research. Springer, 1999.

[20] M. J. D. Powell. Convergence properties of algorithms for nonlinear optimization. *SIAM Review*, 28(4):487–500, 1986.

[21] R. Silva, M. Ulbrich, S. Ulbrich, and L. N. Vicente. A globally convergent primal-dual interior-point filter method for nonlinear programming: new filter optimality measures and computational results. Technical Report 08-49,, Department of Mathematics, University of Coimbra, 2009.

[22] R. J. Vanderbei and D. F. Shanno. An interior point algorithm for nonconvex nonlinear programming. *Computational Optimization and Applications*, 13:231–252, 1999.

[23] A. Wächter and L. T. Biegler. On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006.

[24] R. A. Waltz and J. Nocedal. KNITRO user's manual. Technical Report OTC 2003/05, Optimization Technology Center, Northwestern University, Evanston, IL, USA, April 2003.

[25] R. A. Waltz and T. D. Plantenga. KNITRO 5.0 User's Manual. Technical report, Ziena Optimization, Inc., Evanston, IL, USA, February 2006.