An Active Set Strategy for Solving Optimization Problems with up to 200,000,000 Nonlinear Constraints

Klaus Schittkowski
Department of Computer Science, University of Bayreuth
95440 Bayreuth, Germany
e-mail: klaus.schittkowski@uni-bayreuth.de

Abstract

Numerical test results are presented for solving smooth nonlinear programming problems with a large number of constraints, but a moderate number of variables. The active set method proceeds from a given bound for the maximum number of expected active constraints at an optimal solution, which must be less than the total number of constraints. A quadratic programming subproblem is generated with a reduced number of linear constraints from the so-called working set, which is internally changed from one iterate to the next. Only for active constraints, i.e., a certain subset of the working set, new gradient values must be computed. The line search is adapted to avoid too many active constraints which do not fit into the working set. The active set strategy is an extension of an algorithm described earlier by the author together with a rigorous convergence proof. Numerical results for some simple academic test problems show that nonlinear programs with up to 200,000,000 nonlinear constraints are efficiently solved on a standard PC.

Keywords: SQP; sequential quadratic programming; nonlinear programming; many constraints; active set strategy

1 Introduction

We consider the general optimization problem to minimize an objective function under nonlinear equality and inequality constraints,

$$\begin{aligned} & \min \quad f(x) \\ x \in I\!R^n: \quad & g_j(x) = 0\ , \quad j = 1, \ldots, m_e\ , \\ & g_j(x) \geq 0\ , \quad j = m_e + 1, \ldots, m\ . \end{aligned} \tag{1}$$

To simplify the notation, upper and lower bounds of the variables are omitted.

It is assumed that the functions $f(x)$ and $g_j(x)$, $j = 1, \ldots, m$, are continuously differentiable on $\mathbb{R}^n$, and that the nonlinear programming problem possesses a very large number of nonlinear inequality constraints on the one hand, but a much lower number of variables. A typical situation is the discretization of an infinite number of constraints, as indicated by the first two case studies.

1. Semi-infinite optimization: Constraints must be satisfied for all $y \in Y$, where y is an additional variable and $Y \subset \mathbb{R}^r$,

$$x \in \mathbb{R}^n : \begin{array}{ll} \min & f(x) \\ & g(x,y) \geq 0 \quad \text{for all } y \in Y \quad . \end{array} \tag{2}$$

Here we assume for simplicity that there is only one scalar restriction of inequality type. If we discretize the set Y, we get a standard nonlinear programming problem, but with a large number of constraints depending on the desired accuracy.

2. Min-max optimization: We minimize the maximum of a function f depending now on two variables $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^r$,

$$\min_{x \in X} \max_{y \in Y} \quad f(x,y) \tag{3}$$

with suitable subsets $X \subset \mathbb{R}^n$ and $Y \subset \mathbb{R}^r$, respectively. (3) is easily transformed into an equivalent semi-infinite nonlinear programming problem with one additional variable.

3. L_∞-Approximation: The situation is similar to min-max optimization, but we want to minimize the maximum of absolute values of a given set of functions,

$$\min_{x \in \mathbb{R}^n} \max_{i=1,\ldots,r} \quad |f_i(x)| \quad . \tag{4}$$

Typically, the problem describes the approximation of a nonlinear function by a simpler one, i.e., $f_i(x) = f(t_i) - p(t_i, x)$. In this case, $f(t)$ is a given function depending on a variable $t \in \mathbb{R}$ and $p(t,x)$ a member of a class of approximating functions, e.g., a polynomial in t with coefficients $x \in \mathbb{R}^n$. The problem is non-differentiable, but can be transformed into a smooth one assuming that all functions $f_i(x)$ are smooth.

In addition, also optimal control problems often possess a very large number of constraints, but a reasonably small number of variables. The total number of constraints m can become so large that even the linearized constraints cannot be stored in memory.

Our main application we have in mind, is structural mechanical optimization with a limited number of design variables, typically not depending on the number of Finite Elements. On the other hand, constraints often depend on the FE discretization, e.g., for bounding stresses and displacements at each node or for taking a large number of load cases into account, where each single one could double the number of constraints. The proposed active set strategy has been used, for example, in an FE-based optimization system of EADS called Lagrange for many years, see Kneppe, Kramer, and Winkler [2] or Schittkowski, Zillober, and Zotemantel [7].

The basic idea is to proceed from a user-provided value m_w with $n \leq m_w \leq m$ by which we estimate the maximum number of active constraints expected at the optimal solution. Only quadratic programming subproblems with m_w linear constraints are created which require lower storage and allow faster numerical solution. Thus, one has to develop a strategy to decide, which constraint indices are added to a working set of size m_w

$$W \doteq \{j_1, \ldots, j_{m_w}\} \subset \{1, \ldots, m\}$$

and which ones have to leave the working set.

It is difficult to estimate a priori the size of the working set, i.e., m_w, without knowing anything about the internal structure of the mathematical optimization problem. It is possible, that too many constraints are violated at a starting point even if it is known that the optimal solution possesses only very few active constraints. To avoid an unnecessary blow-up of the working set, it would be possible to extend the given optimization problem by an additional artificial variable x_{n+1}, which, if chosen sufficiently large at the start, decreases the number of active constraints. (1) is then replaced by

$$x \in I\!R^{n+1}: \quad \begin{array}{ll} \min f(x) + \rho x_{n+1} & \\ g_j(x) = 0\,, & j = 1, \ldots, m_e\ , \\ g_j(x) + x_{n+1} \geq 0\,, & j = m_e + 1, \ldots, m\ , \\ x_{n+1} \geq 0\ . & \end{array} \tag{5}$$

The choice of the penalty parameter ρ and the starting value for x_{n+1} is crucial. A too rapid decrease of x_{n+1} to zero must be prevented to avoid

too many active constraints, which is difficult to achieve in general. But if adapted to a specific situation, the transformation works very well and can become extremely helpful.

To further reduce the number of gradient evaluations, the active set strategy is extended. Gradients are calculated at a new iterate only for a certain subset of the estimated active constraints, i.e., for a subset of W. The underlying SQP algorithm is described in Schittkowski [8], and the presented active set approach in Schittkowski [10], which contains more details, a convergence analysis, and a listing of the test problems used again in this paper. The existing procedure is slightly extended based on numerical experiments performed since then, especially to make the line search more efficient and robust.

Active set strategies are widely discussed in the nonlinear programming literature and have been implemented in most of the available codes. A computational study for linear constraints was even conducted in the 70's, see Lenard [5], and Google finds about 100,000 hits for `active set strategy nonlinear programming`. It is out of the scope of this paper to give a review. Some of these strategies are quite complex and a typical example is the one included in the KNITRO package for large scale optimization, see Byrd, Gould, Nocedal, and Waltz [1], based on linear programming and equality constrained subproblems, whereas Tits [14] published an active set strategy for linear programming.

The modified SQP-algorithm is outlined in Section 2, and some numerical test results based on a few academic examples are found in Section 3, where the number of nonlinear constraints is very large, i.e., up to 200,000,000. The amazing observation is that the large number of constraints is obtained by discretization leading to nonlinear programs with a large number of nearly dependent active constraints, which are nevertheless efficiently solved.

2 An Active-Set Sequential Quadratic Programming Method

Sequential quadratic programming methods construct a sequence of quadratic programming subproblems by approximating the Lagrangian function

$$L(x,u) \doteq f(x) - \sum_{j=1}^{m} u_j g_j(x) \tag{6}$$

quadratically and by linearizing the constraints. A typical dense SQP code takes all constraints into account, and requires a double precision working array of length $O(n^2 + n \cdot m)$. In particular, we need $m \cdot n$ double precision real numbers to store the gradients of the constraint functions.

We assume now that n is of reasonable size, say below 100, but that m is very large compared to n, say 1,000,000 or even more. Then either the available memory is insufficient to store the total Jacobian matrix of size $n \cdot m$, or the large set of linear constraints in the subproblem slows down the quadratic programming solver because of internal IO loads of the runtime system of the compiler. It is furthermore assumed that there are no sparsity patterns in the Jacobian matrix of the constraints which could be exploited.

Our goal is to replace m by m_w in the quadratic programming subproblem, where m_w is a user-provided number depending on the available memory and the expected number of active constraints, and which satisfies $n \leq m_w \leq m$. It is supposed that a double precision array of size $n \cdot m_w$ can be allocated. Moreover, it has to be guaranteed that the active-set algorithm is identical with a standard SQP method if $m = m_w$.

Thus, we formulate a quadratic programming subproblem with m_w linear constraints. If x_k denotes an iterate of the algorithm, v_k the corresponding multiplier estimate and B_k a positive definite estimate of the Hessian of the Lagrangian function (6), we solve quadratic programs of the form

$$\begin{aligned} & \min \ \tfrac{1}{2} d^T B_k d + \bigtriangledown f(x_k)^T d \ + \tfrac{1}{2}\sigma_k^2 \delta^2 \ , \\ d \in I\!R^n, \delta \in I\!R : \ & \bigtriangledown g_j(x_k)^T d + (1-\delta) g_j(x_k) \ \left\{ \begin{matrix} = \\ \geq \end{matrix} \right\} \ 0 \ , \ j \in J_k^\star \ , \\ & \bigtriangledown g_j(x_{j(k)})^T d + g_j(x_k) \geq 0 \ , \ j \in \overline{K}_k^\star \ . \end{aligned} \tag{7}$$

An additional variable δ is introduced to prevent infeasible linear constraints, see Schittkowski [8] for details. The index set $J_k^\star$ is called the set of active constraints and is defined by

$$J_k^\star \doteq \{1, \ldots, m_e\} \cup \{j : m_e < j \leq m, \ g_j(x_k) < \epsilon \text{ or } v_j^{(k)} > 0\} \ . \tag{8}$$

It is assumed that $|J_k^\star| \leq m_w$, i.e., that all active constraints are part of the working set

$$W_k \doteq J_k^\star \cup \overline{K}_k^\star \tag{9}$$

with m_w elements. The working set contains the active constraints plus a certain subset of the non-active ones, $\overline{K}_k^\star \subset K_k^\star$, defined by

$$K_k^\star := \{1, \ldots, m\} \setminus J_k^\star \ . \tag{10}$$

$v_k = (v_1^{(k)}, \ldots, v_m^{(k)})^T$ is the actual multiplier estimate and ϵ a user provided error tolerance. The indices $j(k)$ in (7) denote previously computed gradients of constraints. The idea is to recalculate only gradients of active constraints and to fill the remaining rows of the constraint matrix with previously computed ones.

We have to assume that there are not more than m_w active constraints in each iteration step. In addition, we need some safeguards in the line search algorithm to prevent this situation. We do not support the idea to include some kind of automatized *phase I* procedure to project an iterate back to the feasible region whenever this assumption is violated. If, for example at the starting point, more than m_w constraints are active, it is preferred to stop the algorithm and to leave it to the user either to change the starting point or to establish an outer constraint restoration procedure depending on the problem structure.

After solving the quadratic programming subproblem (7) we get a search direction d_k and a corresponding multiplier vector u_k. The new iterate is obtained by

$$x_{k+1} \doteq x_k + \alpha_k d_k \quad , \qquad v_{k+1} \doteq v_k + \alpha_k (u_k - v_k) \tag{11}$$

for approximating the optimal solution and the corresponding optimal multiplier vector. The steplength parameter α_k is the result of an additional line search sub-algorithm, by which we want to achieve a sufficient decrease of an augmented Lagrangian merit function

$$\psi_r(x, v) \doteq f(x) - \sum_{j \in J(x,v)} (v_j g_j(x) - \frac{1}{2} r_j g_j(x)^2) - \frac{1}{2} \sum_{j \in K(x,v)} v_j^2 / r_j \quad . \tag{12}$$

The index sets $J(x, v)$ and $K(x, v)$ are defined by

$$\begin{aligned} J(x,v) &\doteq \{1, \ldots, m_e\} \cup \{j : m_e < j \leq m, g_j(x) \leq v_j / r_j\} \quad , \\ K(x,v) &\doteq \{1, \ldots, m\} \setminus J(x,v) \quad . \end{aligned} \tag{13}$$

The corresponding penalty parameters $r_k \doteq (r_1^k, \ldots, r_m^k)^T$ that control the degree of constraint violation, must carefully be chosen to guarantee a sufficient descent direction of a merit function

$$\phi_{r_k}(\alpha_k) \leq \phi_{r_k}(0) + \alpha_k \mu \phi'_{r_k}(0) \quad , \tag{14}$$

where

$$\phi_{r_k}(\alpha) \doteq \psi_{r_k}\left(\begin{pmatrix} x_k \\ v_k \end{pmatrix} + \alpha \begin{pmatrix} d_k \\ u_k - v_k \end{pmatrix}\right) \quad . \tag{15}$$

It is essential to understand that in contrast to most other SQP algorithms which exist, we perform a simultaneous line search subject to the primal and the dual variables. The advantage is that another separate update of the multipliers is avoided.

An important requirement is that at each intermediate step of the line search algorithm, at most m_w constraints are active. If this condition is violated, the steplength is further reduced until satisfying this condition. From the definition of our index sets, we have

$$J_k^\star \supset J(x_k, v_k) \quad . \tag{16}$$

The starting point x_0 is crucial from the viewpoint of numerical efficiency and must be predetermined by the user. It has to satisfy the assumption that not more than m_w constraints are active, i.e., that $J_0 \subset W_0$. The remaining indices of W_0 are to be set in a suitable way and must not overlap with the active ones. Also W_0 must be provided by the user to have the possibility to exploit pre-existing knowhow about the position of the optimal solution and its active constraints.

The basic idea of the algorithm can be described in the following way: We determine a working set W_k and perform one step of a standard SQP-algorithm, NLPQLP [12] in our case, with m_w nonlinear constraints. Then the working set is updated and the whole procedure repeated.

One particular advantage is that the numerical convergence conditions for the reduced problem are applicable for the original one as well, since all constraints not in the working set W_k are inactive, i.e., satisfy $g_j(x_k) > \epsilon$ for $j \in \{1, \ldots, m\} \setminus W_k$.

The line search procedure described in Schittkowski [8] can be used to determine a steplength parameter α_k. The algorithm is a combination of an Armijo-type steplength reduction with a quadratic interpolation of $\phi_k(\alpha)$. The proposed approach guarantees theoretical convergence results, is very easy to implement and works satisfactorily in practice. But in our case we want to achieve the additional requirement that all intermediate iterates do not possess more than m_w violated constraints. By introducing an additional loop reducing the steplength by a constant factor, it is always possible to guarantee this condition. An artificial penalty term is added to the objective

function consisting of violated constraints. The modification of the line search procedure prevents iterates of the modified SQP-method that violate too many constraints.

Since a new restriction is included in the working set W_{k+1} only if it belongs to $J^\star_{k+1}$, we always get new and actual gradients for the quadratic programming subproblem (7). But gradients can be reevaluated for any larger set, e.g., for all indices of W_{k+1}. In this case, we can even expect a better performance of the algorithm.

The proposed modification of the standard SQP-technique is straightforward and easy to implement, see Schittkowski [10] for details. However, we want to stress that its practical performance depends mainly on the heuristics used to determine the working set W_k. The first idea could be to take out those constraints from the working set which have the largest function values. But the numerical size of a constraint depends on its internal scaling. In other words, we cannot conclude from a *large* restriction function value that the constraint is probably inactive.

By the subsequent algorithm, we summarize in short the main steps of our active set strategy. One important ingredient is an extended merit function used only for an additional subiteration during the line search,

$$\overline{\phi}_{r_k}(\alpha) \doteq \phi_{r_k}(\alpha) + \frac{\nu}{2} \sum_{j=m_e+1}^{m} g_j^-(x_k)^2 \quad , \tag{17}$$

where $\nu > 0$ is a penalty constant and $g_j^-(x_k) \doteq \min\{0, g_j(x_k)\}$, $m_e < j \leq m$.

Algorithm 2.1 *Active Set Strategy*

Start: *Choose m_w with $n \leq m_w \leq m$ and $x_0 \in I\!R^n$ with*

$$|J^\star_0| = m_e + |\{j : m_e < j \leq m, g_j(x_0) < \epsilon\}| \leq m_w \quad .$$

Moreover, let $W_0 \doteq J^\star_0 \cup \overline{K}^\star_0$ such that $g_j(x_0) \geq \epsilon$ for all $j \in \overline{K}^\star_0$.

Main Loop: *For $k = 0, 1, \ldots$ let $x_k \in I\!R^n$ and $v_k \in I\!R^m$ be given iterates and W_k the corresponding working set with $|W_k| \leq m_w$. Compute new iterates $x_{k+1} \in I\!R^n$ and $v_{k+1} \in I\!R^m$ and a new working set W_{k+1} as follows:*

1. *Search Direction: Solve the quadratic program (7) with m_w linear constraints to get a new search direction $d_k \in I\!R^n$ for the primal and a new guess $u_k \in I\!R^m$ for the dual variable. Note that (7) contains only $|J^\star_k|$ newly computed constraint gradients $\bigtriangledown g_j(x_k), j \in J^\star_k$.*

2. *Line Search: Let* $\alpha_k^0 \doteq 1$. *For* $i = 0, 1, \ldots$ *compute* α_k^{i+1} *from* α_k^i *as follows:*

   (a) *Compute all constraint values* $g_j(x_k + \alpha_k^i d_k)$, $j = 1, \ldots, m$. *If too many constraints are active, i.e., if* $|J_{k,i}^\star| > m_w$ *for*

   $$J_{k,i}^\star \doteq \{1, \ldots, m_e\} \cup \{j : m_e < j \leq m,\ g_j(x_k + \alpha_k^i d_k) < \epsilon \text{ or } v_j^{(k)} > 0\},$$

   replace α_k^i *by* $\tau \alpha_k^i$, $0 < \tau < 1$. *Repeat this loop until* $|J_{k,i}^\star| \leq m_w$.

   (b) *Determine* α_k^{i+1} *in the usual way by a combination of an Armijo-type steplength reduction and a quadratic interpolation subject to* $\overline{\phi}_{r_k}(0)$, $\overline{\phi}'_{r_k}(0)$, *and* $\overline{\phi}_{r_k}(\alpha_k^i)$, *see [8].*

3. *Working Set: Select a new working set* $W_{k+1} \doteq J_{k+1}^\star \cup \overline{K}_{k+1}^\star$, *where the indices of* $\overline{K}_{k+1}^\star$ *are selected according to the following guidelines:*

   (a) *Among the constraints feasible at* x_k *and* x_{k+1}, *keep those in the working set that are violated during the line search. If there are too many of them according to some given constant, select constraints for which*

   $$\frac{g_j(x_{k+1}) - \epsilon}{g_j(x_{k+1}) - g_j(x_k + \alpha_{k,i-1} d_k)} \tag{18}$$

   is minimal, where $\alpha_{k,i-1}$ *is an iterate of the line search procedure. The decision whether a constraint is feasible or not, is performed with respect to the given tolerance* ϵ.

   (b) *In addition, keep the restriction in the working set for which* $g_j(x_k + d_k)$ *is minimal.*

   (c) *Take out those feasible constraints from the working set, which are the oldest ones with respect to their successive number of iterations in the working set.*

Step 3 is chosen to get a decision on constraints in the working set W_k that is independent from the scaling of the functions as much as possible.

Under the assumptions mentioned so far, it is shown in Schittkowski [10], Section 3, that

$$\phi'_{r_k}(0) = \nabla \psi_{r_k}(x_k, v_k)^T \begin{pmatrix} d_k \\ u_k - v_k \end{pmatrix} < -\frac{1}{4} \gamma \|d_k\|^2 \tag{19}$$

for all k and a positive constant γ, i.e., we get a sufficient decrease of a merit function. If we assume, that the additional modification of the steplength is avoided, e.g., by a sufficiently large m_w, or is at least bounded away from zero, convergence of the algorithm follows in the sense that a stationary point is approximated, see Schittkowski [8].

It is possible that we get a descent direction of the merit function, but that $\phi_r'(0)$ is extremely small. To avoid interruption of the whole iteration process, the idea is to repeat the line search with another stopping criterion. Instead of testing (19), we accept a stepsize α_k as soon as the inequality

$$\phi_{r_k}(\alpha_k) \leq \max_{k-p(k)<=j<=k} \phi_{r_j}(0) + \alpha_k \mu \phi'_{r_k}(0) \tag{20}$$

is satisfied, where $p(k)$ is a predetermined parameter with $p(k) = \min\{k, p\}$, p a given tolerance. Thus, we allow an increase of the reference value $\phi_{r_k}(0)$ in a certain error situation, i.e., an increase of the merit function value, see Dai and Schittkowski [3] for details and a convergence proof.

3 Numerical Tests

The modified SQP-algorithm is implemented in the form of a Fortran subroutine with name NLPQLB, see Schittkowski [13]. As pointed out in the previous section, an iteration consists of one step of a standard SQP-method, in our case of the code NLPQLP [12], with exactly m_w constraints. Basically, only the definition of the working set and some rearrangements of index sets must be performed. Then NLPQLP is called to perform only one iteration proceeding from the iterates x_k, v_k, B_k, r_k, and $J_k^\star$.

The algorithm requires an estimate of the maximum size of the working set, m_w, a starting point $x_0 \in I\!R^n$, and an initial working set W_0 with $J_0^\star \subset W_0$, see (8) for a definition of the active set $J_k^\star$. A straightforward idea is to sort the constraints according to their function values at x_0, and to take the first m_w constraints in increasing order. However, one would have to assume that all constraints are equally scaled, a reasonable assumption in case of scalar semi-infinite problems of the form (2). Otherwise, an alternative proposal could be to include all constraints in the initial working set for which $g_j(x) \leq \epsilon$, and to fill the remaining position with indices for which $g_j(x) > \epsilon$.

Some test problem parameters are summarized in Table 1, see also the subsequent list of abbreviations. The examples are small academic test prob-

lems taken from the literature, e.g., semi-infinite or min-max problems. They have been used before to get the results published in Schittkowski [10], and are now formulated with up to 200,000,000 constraints. P1F is identical to P1, but formulated with soft constraints, see (5). Gradients are evaluated analytically. More details are found in Schittkowski [10, 13], and the software report [13] can be downloaded from `http://www.klaus-schittkowski.de/`.

The number of test cases is by far too small to draw general conclusions. But we cannot find a large set of nonlinear programming test examples of the type *few variables and very many constraints*, especially in the area structural mechanical optimization, where we see an important domain of application. Nevertheless, we believe that these examples illustrate at least typical behavior and also typical difficulties.

name	n	m	m_w	$f^\star$
P1	3	60,000,000	30,000,000	5.33469
P1F	4	200,000,000	2,000	5.33469
P3	3	200,000,000	200,000	4.30118
P4	3	200,000,000	200	1.00000
TP332	2	200,000,000	100	398.587
TP374	10	200,000,000	50,000	0.2917
U3	6	200,000,000	200	0.00012399
L5	8	200,000,000	40,000	0.0952475
E5	5	200,000,000	50,000	125.619

Table 1: Test Examples

The Fortran codes are compiled by the Intel Visual Fortran Compiler, Version 10.1, EM64T, under Windows Vista and Intel(R) Core(TM) Duo CPU E8500, 3.16 GHz, and 8 GB RAM. The working arrays of the routine calling NLPQLB are dynamically allocated. Quadratic programming subproblems are solved by the primal-dual method of Goldfarb and Idnani [4] based on numerically stable orthogonal decompositions, see Schittkowski [11]. NLPQLB is executed with termination accuracy $\epsilon = 10^{-8}$. Numerical experiments are reported in Table 2 and Table 3 where we use the following notation:

$name$	-	identification of the test example
n	-	number of variables corresponding to formulation (1)
m	-	total number of constraints corresponding to formulation (1)
m_w	-	number of constraints in the working set
$f^\star$	-	final objective function value, i.e., $f(x_k)$ for the last computed iterate x_k
$\|J_{max}\|$	-	maximum number of active constraints, i.e., the maximum value $\|J_k^\star\|$ taken over all $k = 0, 1, \ldots$
n_f	-	number of simultaneous function computations, i.e., of objective function and all constraints at a given iterate
n_g^{tot}	-	total number of gradient computations, i.e., of all individual constraint gradient evaluations
n_{its}	-	number of iterations or simultaneous gradient computations, i.e., of gradients of objective function and all constraints at a given iterate
i_{fail}	-	termination reason, i.e., 0 : successful return 1 : address space exceeded 2 : zero search direction 4 : error in line search 3 : too many active constraints
t_{calc}	-	calculation time in seconds

$name$	$\|J_{max}\|$	n_f	n_g^{tot}	n_{its}	t_{calc}
P1	8,368,516	22	41,431,865	14	160
P1F	801	23	6,728	15	283
P3	50,002	10	550,023	8	64
P4	1	4	203	4	20
TP332	1	12	110	11	463
TP374	37,229	29	647,144	53	2,038
U3	285,901	129	2,376	44	57
L5	39,976	80	207,960	24	1,224
E5	41,674	30	212,245	21	388

Table 2: Numerical Results

It is obvious that the efficiency of an active set strategy strongly depends

on how close the active set at the starting point to that of the optimal solution is. If dramatic changes of active constraints are expected as in the case of P1, i.e., if intermediate iterates with a large number of violated constraints are generated, the success of the algorithm is marginal. On the other hand, practical optimization problems often have special structures from where good starting points can be predetermined. Examples P1F and especially P4 and TP332 show a dramatic reduction of derivative calculations, which is negligible compared to the number of function calls.

To show that the starting points are not chosen close to the optimal solution, we display function values for E5. In this case, we minimize the maximum of

$$f(x,t) \doteq (x_1 + x_2 t - \exp t)^2 + (x_3 + x_4 \sin t - \cos t)^2$$

over all $t \in [0,4]$. After an equidistant discretization of this interval with m break points and the introduction of an auxiliary variable, we obtain a nonlinear programming problem in five variables and m constraints. Function $f(x,t)$ is plotted in Figure 1 for the starting point $x = x_0$ and the computed optimal solution $x = x^\star$. We observe a drastic change of the active set from the initial guess to the final solution.

Since the constraints are nonlinear and non-convex, we have to compute all m function values at each iteration to check feasibility and to predict the new active set. The total number of individual constraint function evaluations is therefore $n_f \cdot m$.

Note that the code NLPQLB requires additional working space in the order of $2m$ double precision real numbers plus $m_w \cdot (n+1)$ double precision numbers for the partial derivatives of constraints in the working set. Thus, the total memory to run a test problem with $m = 2 \cdot 10^8$ constraints requires at least $6 \cdot 10^8$ double precision numbers and in addition at least $4 \cdot 10^8$ logical values.

Although limited to a very small test set, the results show that the proposed active set strategy might be helpful in situations, where the full Jacobian matrix of a nonlinear programming problem cannot be kept in memory. There remains the question whether it might be reasonable to benefit from an active set information also in case of a somewhat lower number of constraints without exceeding the available address space, or whether one should better proceed from the full set of constraints.

Table 3 shows numerical results for $m = 2 \cdot 10^7$, where m_w is reduced from $m_w = m$ down to $m_w = 20$ successively. P1 and TP374 do not yield

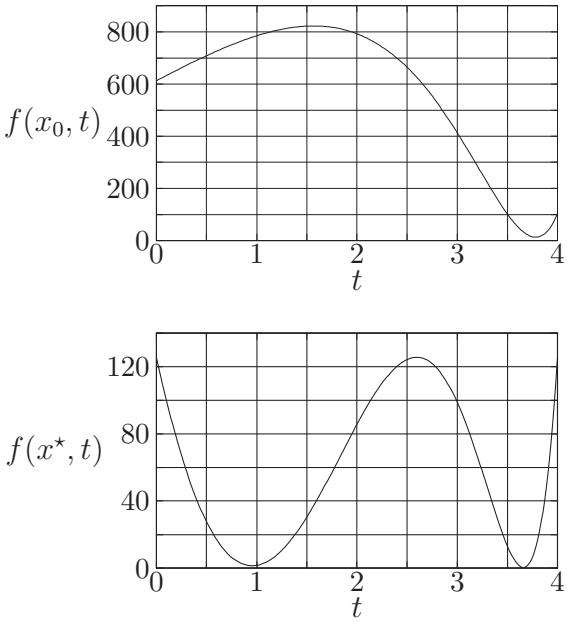

Figure 1: Function Plot for E5 at Start and Optimal Solution

interpretable results in this case because of too many intermediate active constraints, and are therefore not taken into account.

The number of gradient evaluations decreases linearly with m_w. In a few situations, i.e., for P3 and L5, the number of iterations and the calculation time increase the smaller m_w becomes.

There are quite a few test runs, where the number of objective function calls is significantly higher than the number of iterations. In these cases, e.g., for U3, there are numerical instabilities in the search direction leading to a large number of line search steps. Quite often, a line search needs to be restarted to accept an uphill search direction.

In some situations, e.g., for E5, NLPQLB needs even more iterations for m or $m/10$ constraints in the working set, where the active set is not activated at all or where m_w is too big to influence the solution process significantly. There are two possible explanations. First, there might be numerical instabilities when solving quadratic programs with a large number of linear constraints. We apply a primal-dual method, where staring from a solution of the unconstrained problem violated constraints are successively

name	m_w	i_{fail}	$f^\star$	n_f	n_g^{tot}	n_{its}	t_{calc}
P1F	20,000,000	0	5.3347	44	20,051,088	27	179
	2,000,000	0	5.3347	36	4,162,898	29	58
	200,000	0	5.3347	48	1,038,331	28	60
	20,000	0	5.3347	38	92,031	23	46
	2,000	0	5.3347	35	8,836	22	42
	200	0	5.3347	33	867	21	40
	20	0	5.3347	33	138	21	40
P3	20,000,000	0	4.3012	4	20,016,153	4	27
	2,000,000	0	4.3012	7	2,859,108	6	9
	200,000	0	4.3012	9	487,562	8	6
	20,000	0	4.3012	6	45,009	5	4
	2,000	0	4.3012	10	6,024	8	6
	200	0	4.3012	12	782	10	8
	20	0	4.3012	17	148	13	10
P4	20,000,000	0	1.0000	4	19,998,787	4	15
	2,000,000	0	1.0000	4	1,999,881	4	3
	200,000	0	1.0000	4	199,990	4	2
	20,000	0	1.0000	4	20,001	4	2
	2,000	0	1.0000	4	2,002	4	2
	200	0	1.0000	4	202	4	2
	20	0	1.0000	4	22	4	2
TP332	20,000,000	0	279.64	12	20,000,010	11	83
	2,000,000	0	279.64	12	2,000,010	11	47
	200,000	0	279.64	12	200,010	11	43
	20,000	0	279.64	12	20,010	11	43
	2,000	0	279.64	12	2,010	11	43
	200	0	279.64	12	210	11	37
	20	0	279.64	10	29	10	37
U3	20,000,000	2	0.14338	156	93,395,906	90	738
	2,000,000	2	0.38232	151	7,660,736	123	153
	200,000	0	0.00012399	75	1,736,633	46	36
	20,000	0	0.00012399	172	214,611	48	67
	2,000	0	0.00012399	152	23,021	44	59
	200	0	0.00012399	124	2,376	44	48
	20	3	0.04083535	27	189	15	12
L5	20,000,000	1	-	-	-	-	-
	2,000,000	0	0.95248	44	3,161,443	30	41
	200,000	0	0.95248	49	613,887	33	11
	20,000	0	0.95248	35	97,339	23	6
	2,000	4	0.95248	94	4,195	10	14
	200	0	0.95248	24	728	18	51
	20	3	0.57489	3	23	2	1
E5	20,000,000	0	125.62	162	104,518,562	86	699
	2,000,000	0	125.62	118	16,523,416	63	186
	200,000	0	125.62	28	732,484	20	38
	20,000	0	125.62	30	83,762	21	39
	2,000	0	125.62	30	8,792	21	39
	200	0	125.62	52	1,102	22	64
	20	3	789.49	7	28	6	10

Table 3: Numerical Results for Varying m_w

added to an internal working set. It seems that in these cases, new constraints might be too close to the active ones, such that a large number of exchanges of working set entries are required, each requiring an update of the orthogonal decomposition.

Another reason could be that we have an additional *active set* strategy on a lower level by requesting the update of gradient information only for a certain subset of W_k. For a large number of constraints with *close* function and gradient values, this might result in a larger number of iterations because a larger number of gradient exchanges, until the final active set is found.

4 Conclusions

We present some numerical results for an active set strategy for nonlinear optimization problems with a relatively small number of variables, but a large number of nonlinear constraints. Rigorous convergence results are available and have been proved in a previous paper. Numerical results are presented first for a situation where the full Jacobian of all constraints cannot be kept in memory. The algorithm converges within an average of 22 iterations (Table 2) and the superlinear convergence rate is not spoiled by the active set strategy.

From another set of test runs for a large number of constraints without overflow of the address space, we conclude that it might be profitable to apply the active set strategy as well. In particular, we observe a dramatic reduction of the number of gradient evaluations, which is extremely important in case of an expensive underlying simulation code. However, too small or too large working sets can lead to instabilities during the line search. We conclude that the proposed active set strategy even stabilizes an SQP algorithm in case of very many constraints, if the working set is carefully chosen.

It is amazing that numerical instabilities due to degeneracy are prevented. The huge number of constraints indicates that derivatives are extremely close to each other, making the optimization problem unstable. Note that the LICQ (linear independency constraint qualification) is more or less violated. If we discretize the interval $[0, 1]$ by $2 \cdot 10^8$ grid pints to get the same number of constraints, see, e.g., problem P3, we can expect that the gradients of neighbored active constraints coincide up to the first eight digits.

We benefit from the fact that derivatives are analytically given, and that we apply an extremely stable primal-dual quadratic programming solver based on orthogonal decompositions, see Schittkowski [11], Goldfarb and

Idnani [4], and especially Powell [6].

We know that the number of test examples is too small to draw general conclusions. But all observations indicate the advantages of an active set set strategy, if carefully implemented.

References

[1] Byrd R., Gould N., Nocedal J., Waltz R. (2004): *An Active-Set Algorithm for Nonlinear Programming Using Linear Programming and Equality Constrained Subproblems*, Mathematical Programming B, Vol. 100, 27 - 48, with R. Byrd, N. Gould and R. Waltz, 2004

[2] Kneppe G., Krammer J., Winkler E. (1987): *Structural optimization of large scale problems using MBB-LAGRANGE*, Report MBB-S-PUB-305, Messerschmitt-Bölkow-Blohm, Munich

[3] Dai Y.H., Schittkowski K. (2006): *A sequential quadratic programming algorithm with non-monotone line search*, Pacific Journal of Optimization, Vol. 4, 335-351

[4] Goldfarb D., Idnani A. (1983): *A numerically stable method for solving strictly convex quadratic programs*, Mathematical Programming, Vol. 27, 1-33

[5] Lenard M. (1979): *A computational study of active set strategies in nonlinear programming with linear constraints* , Mathematical Programming, Vol. 16, 81 - 97

[6] Powell M.J.D. (1983): *ZQPCVX, A FORTRAN subroutine for convex quadratic programming*, Report DAMTP/1983/NA17, University of Cambridge, England

[7] Schittkowski K., Zillober C., Zotemantel R. (1994): *Numerical comparison of nonlinear programming algorithms for structural optimization*, Structural Optimization, Vol. 7, No. 1, 1-28

[8] Schittkowski K. (1983): *On the convergence of a sequential quadratic programming method with an augmented Lagrangian search direction,* Optimization, Vol. 14, 197-216

[9] Schittkowski K. (1985/86): *NLPQL: A Fortran subroutine solving constrained nonlinear programming problems,* Annals of Operations Research, Vol. 5, 485-500

[10] Schittkowski K. (1992): *Solving nonlinear programming problems with very many constraints*, Optimization, Vol. 25, 179-196

[11] Schittkowski K. (2003): *QL: A Fortran code for convex quadratic programming - user's guide*, Report, Department of Computer Science, University of Bayreuth

[12] Schittkowski K. (2006): *NLPQLP: A Fortran implementation of a sequential quadratic programming algorithm with distributed and nonmonotone line search - user's guide, version 2.2*, Report, Department of Computer Science, University of Bayreuth

[13] Schittkowski K. (2007): *NLPQLB: A Fortran implementation of an SQP algorithm with active set strategy for solving optimization problems with a very large number of nonlinear constraints - user's guide, version 2.0*, Report, Department of Computer Science, University of Bayreuth

[14] Tits A. (1999): *An interior point method for linear programming, with an active set flavor*, Report, Department of Electrical and Computer Engineering, University of Maryland, College Park