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

We propose a numerical algorithm 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 m_w for the maximum number of expected violated constraints, where m_w is a user-provided parameter less than the total number of constraints. A quadratic programming subproblem is generated with m_w linear constraints, the so-called working set, which are internally exchanged 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 takes the active constraints into account. Numerical results for some simple academic test problems show that nonlinear programs with up to 200,000,000 nonlinear constraints can be efficiently solved on a standard PC.

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

1 Introduction

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

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

It is assumed that all problem functions $f(x)$ and $g_j(x)$, $j = 1, \ldots, m$, are continuously differentiable on the whole $I\!\!R^n$. We assume now 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 following case studies.

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

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

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 $x \in I\!R^n$ and $y \in I\!R^r$,

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

with suitable subsets $X \subset I\!R^n$ and $Y \subset I\!R^r$, 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,

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

Typically, the problem describes the approximation of a nonlinear functions by a simpler function, 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 $t \in I\!R$ and $p(t,x)$ a member of a class of approximating functions, e.g., a polynomial in t with coefficients $x \in I\!R^n$. The problem is non-differentiable, but can be transformed into a smooth one assuming that all functions $f_i(x)$ are smooth.

But also optimal control or mechanical structural optimization 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 either the linearized constraints cannot be stored in memory.

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 expected active constraints. 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 recommended to keep as many constraints as possible in a working set, i.e., to keep m_w as large as possible, since also non-active constraint could contribute an important influence on the computation of search direction.

It is, however, 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 is also possible to extend the given optimization problem by an additional artificial variable x_{n+1}, which, if chosen sufficiently large at 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 be extremely helpful.

To further reduce the number of gradient evaluations, an additional active set strategy is introduced. Gradients are calculated at a new iterate only for a certain subset of estimated active constraints. The underlying SQP algorithm is described in Schittkowski [5], and the presented active set approach for solving problems with a large number of constraints in Schittkowski [7].

Active set strategies are widely discussed in the nonlinear programming literature and have been implemented in most of the available codes. A computation study for linear constraints was even conducted in the 70's, see

Lenard [4], and Google finds 267,000 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.

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.

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 $O(n^2 + nm)$. In particular, we need mn 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 gradient matrix of size nm, 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 gradient matrix 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 nm_w can be addressed in memory. 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 an positive definite estimate of the Hessian of the Lagrangian function (6), we solve quadratic programs of the form

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

An additional variable δ is introduced to prevent infeasible linear constraints, see Schittkowski [5] 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 at each iterate of the algorithm. In addition, we have some safeguards in the line search algorithm to prevent this situation. But 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

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

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

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

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 [8] 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 [5] 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 get always new and actual gradients in the quadratic programming subproblem (7). But gradients can be reevaluated for any larger set, e.g., W_{k+1}. In this case we can expect even a better performance of the algorithm.

The proposed modification of the standard SQP-technique is straightforward and easy to implement. However, we want to stress out 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 got 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.

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, we propose the following rules:

- Among the constraints feasible at x_k and x_{k+1}, keep those in the working set that were violated during the line search. If there are too many of them according to some given constant, select constraints for which
$$\frac{g_j(x_{k+1}) - \epsilon}{g_j(x_{k+1}) - f_j(x_k + \alpha_{k,i-1} d_k)}$$
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.

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

- 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.

Under the assumptions mentioned so far, it is shown in Schittkowski [7] that
$$\phi'_{r_k}(0) = \bigtriangledown \psi_{r_k}(x_k, v_k)^T \begin{pmatrix} d_k \\ u_k - v_k \end{pmatrix} < -\frac{1}{4}\gamma \|d_k\|^2 \tag{17}$$
for all k and a positive constant γ.

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 (17), we accept a stepsize α_k as soon as the inequality
$$\phi_{r_k}(\alpha_k) \leq \max_{k-p(k)<=j<=k} \phi_{r_j}(0) + \alpha_k \mu \phi'_{r_k}(0) \tag{18}$$
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 [2] for details and a convergence proof.

3 Numerical Tests

The modified SQP-algorithm was implemented in form of a Fortran subroutine with name NLPQLB, see Schittkowski [9]. 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 [8], 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 as described in the previous section, 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 very reasonable assumption in case of scalar semi-infinite problems of the the form (2). Otherwise, an alternative, much simpler 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 data characterizing their structure, are summarized in Table 1. The examples are small academic test problems taken from the literature, e.g., semi-infinite or min-max problems. They have been used before to get the results published in Schittkowski [7], 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.

The Fortran codes are compiled by the Intel Visual Fortran Compiler, Version 9.1, under Windows XP64, and executed on a Dual Core ADM Opteron processor 265 with 1.81 GHz and 4 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 [3] based on numerically stable orthogonal decompositions. NLPQLB is executed with termination accuracy $\epsilon = 10^{-8}$. Numerical experiments are reported in Table 2 where we use the following notation:

name	n	m	m_w	$f^\star$
P1	3	40,000,000	20,000,000	5.33469
P1F	4	200,000,000	2,000	5.33469
P3	3	200,000,000	500,000	4.30118
P4	3	200,000,000	200	1.00000
TP332	2	200,000,000	100	398.587
TP374	10	100,000,000	2,000,000	0.434946
U3	6	100,000,000	500,000	0.00012399
L5	8	200,000,000	40,000	0.0952475
E5	5	200,000,000	50,000	125.619

Table 1: Test Examples

name	-	identification of test example
n	-	number of variables in standard form (1)
m	-	total number of constraints in standard form (1)
m_w	-	number of constraints in the working set
$f^\star$	-	final objective function value
$\lvert J_{max} \rvert$	-	maximum number of active constraints
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
time	-	calculation time in seconds
$f^\star$	-	final objective function value

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 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

name	$\lvert J_{max}\rvert$	n^f	n_g^{tot}	n_{its}	t_{calc}
P1	5,579,011	20	27,597,939	13	256
P1F	801	23	6,728	15	942
P3	129,237	12	1,558,343	10	210
P4	1	4	203	4	62
TP332	1	12	110	11	1,229
TP374	584,004	150	20,238,000	89	7,470
U3	357,153	140	4,832,326	49	899
L5	39,976	80	207,960	24	3,005
E5	41,674	30	209,519	21	1,244

Table 2: Numerical Results

is negligible compared to the number of function calls.

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. Thus, the total number of individual constraint function evaluations is $n_f \cdot m$.

Calculation times are excessive and depend mainly on data transfer operations from and to the standard swap file of Windows, and the available memory in core, which is 4 GB in our case. To give an example, test problem TP332 requires 110 sec for $m = 2 \cdot 10^7$ constraints and only 8 sec for $m = 2 \cdot 10^6$ constraints.

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 $m = 2 \cdot 8$ constraints requires at least 600,000,000 double precision numbers and in addition at least 400,000,000 logical values.

It is amazing that numerical instabilities due to degeneracy are prevented. The huge number of constraints indicates that the derivatives are extremely close to each other, making the optimization problem unstable. The constraint qualification, i.e., the linear independence of active constraints, is more or less violated. We benefit from the fact that derivatives are analytically given.

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

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

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

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

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

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

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

[9] 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