A Trust Region SQP Algorithm for Mixed-Integer Nonlinear Programming [1]

Oliver Exler, Klaus Schittkowski

Address:	Oliver Exler Process Engineering Group IIM - CSIC E - 36208 Vigo (Pontevedra)
E-mail:	oexler@iim.csic.es
Address:	Klaus Schittkowski Department of Computer Science University of Bayreuth D - 95440 Bayreuth
E-mail:	klaus.schittkowski@uni-bayreuth.de
Web:	http://www.klaus-schittkowski.de
Date:	July 7, 2006

Abstract

We propose a modified sequential quadratic programming (SQP) method for solving mixed-integer nonlinear programming problems. Under the assumption that integer variables have a *smooth* influence on the model functions, i.e., that function values do not change drastically when in- or decrementing an integer value, successive quadratic approximations are applied. The algorithm is stabilized by a trust region method with Yuan's second order corrections. It is not assumed that the mixed-integer program is relaxable or, in other words, function values are evaluated only at integer points. The Hessian of the Lagrangian function is approximated by a quasi-Newton update formula subject to the continuous and integer variables. Numerical results are presented for a set of 80 mixed-integer test problems taken from the literature. The surprising result is that the number of function evaluations, the most important performance criterion in practice, is less than the number of function calls needed for solving the corresponding relaxed problem without integer variables.

Keywords: mixed-integer nonlinear programming, sequential quadratic programming, SQP, trust region methods

[1]Sponsored by the EU Marie-Curie Training Network under project number MRTN-CT-2004-512233

1 Introduction

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

$$
x \in \mathbb{R}^{n_c}, y \in \mathbb{Z}^{n_i} : \quad
\begin{array}{ll}
\min f(x,y) & \\
g_j(x,y) = 0 \;, \quad j = 1, \ldots, m_e & , \\
g_j(x,y) \geq 0 \;, \quad j = m_e + 1, \ldots, m & , \\
x_l \leq x \leq x_u & , \\
y_l \leq y \leq y_u & ,
\end{array}
\tag{1}
$$

where x and y denote the vectors of the continuous and integer variables, respectively. It is assumed that the problem functions $f(x,y)$ and $g_j(x,y)$, $j = 1, \ldots, m$, are continuously differentiable subject to all $x \in \mathbb{R}^{n_c}$.

Trust region methods have been invented many years ago first for unconstrained optimization, especially for least squares optimization, see for example Powell [22], or Moré [20]. Extensions were developed for non-smooth optimization, see Fletcher [11], and for constrained optimization, see Celis [6], Powell and Yuan [23], Byrd et al. [5], Toint [28] and many others. A comprehensive review on trust region methods is given by Conn, Gould, and Toint [7].

On the other hand, sequential quadratic programming or SQP methods belong to the most frequently used algorithms to solve practical optimization problems. The theoretical background is described e.g. in Stoer [27], Fletcher [10], or Gill et al. [15].

However, the situation becomes much more complex if additional integer variables must be taken into account. Numerous algorithms have been proposed in the past, see for example Floudas [13] or Grossmann and Kravanja [16] for review papers. Typically, these approaches require convex model functions and continuous relaxations of integer variables. By a continuous relaxation, we understand that integer variables can be treated as continuous variables, i.e., function values can also be computed between successive integer points. There are branch-and-bound methods where a series of relaxed nonlinear programs must be solved obtained by restricting the variable range of the relaxed integer variables, see Gupta and Ravindran [17] or Borchers and Mitchell [2]. When applying an SQP algorithm for solving a subproblem, it is possible to apply early branching, see also Leyffer [18]. Pattern search algorithms are available to search the integer space, see Audet and Dennis [1]. After replacing the integrality condition by continuous nonlinear constraints, it is possible to solve the resulting highly nonconvex program by a global optimization algorithm, see e.g. Li and Chou [19]. Outer approximation methods are investigated by Duran and Grossmann [8] and Fletcher and Leyffer [12], where a sequence of alternating mixed-integer linear and nonlinear programs must be solved. Alternatively, it is also possible to apply cutting planes as in linear programming, see Westerlund and Pörn [29].

But fundamental assumptions are often violated in practice. Many real-life mixed-integer problems are not relaxable, and model functions are highly nonlinear and nonconvex. Moreover, some approaches require detection of infeasibility of nonlinear programs, a highly unattractive feature from the computational point of view. We assume now that integer variables are not relaxable, that there are relatively large ranges for integer values, and that the integer variables possess some kind of *physical* meaning, i.e., are *smooth* in a certain sense. It is supposed that a slight alteration of an integer value, say by one, changes the model functions only slightly, at least much less than a more drastic change. Typically, relaxable programs satisfy this requirement. In contrast to them, there are *categorical* variables which are introduced to enumerate certain situations and where any change leads to a completely different category and thus to a completely different response.

A practical situation is considered by Bünner, Schittkowski, and van de Braak [3], where typical integer variables are the number of fingers and the number of layers of an electrical filter. By increasing or decreasing the number of fingers by one, we expect only a small alteration in the total response, the transmission energy. The more drastically the variation of the integer variable is, the more drastically model function values are supposed to change.

Thus, we propose an alternative idea in Section 2 where we try to approximate the Lagrangian subject to the continuous and integer variables by a quadratic function based on a quasi-Newton update formula. Instead of a line search as is often applied in the continuous case, we use trust regions to stabilize the algorithm and to enforce convergence following the continuous trust region method of Yuan [32] with second order corrections. The specific form of the quadratic programming subproblem avoids difficulties with inconsistent linearized constraints and leads to a convex mixed-integer quadratic programming problem, which can be solved by any available algorithm, for example, a branch-and-bound method. Though we are unable to provide a convergence proof, our numerical experiments reported in Section 3 demonstrate the performance of the algorithm. They are based on a collection of 80 mixed-integer test problems, which have been widely used in the past to develop and test mixed-integer programming algorithms.

2 The Mixed-Integer Trust Region SQP Method

Integer variables lead to extremely difficult optimization problems. Even if we assume that the integer variables are relaxable and that the resulting continuous problem is strictly convex, it is possible that the integer solution is not unique. There is no numerically applicable criterion to decide whether we are close to an optimal solution nor can we retrieve any information about the position of the optimal integer solution from the continuous solution of the corresponding relaxed problem.

To illustrate the basic algorithmic ideas, we proceed from the continuous case and will subsequently introduce integer variables again. To facilitate the notation, we neglect upper and lower bounds x_u and x_l, see (1), and we consider the somewhat simpler formulation

$$x \in \mathbb{R}^n : \begin{array}{l} \min f(x) \\ g_j(x) = 0 \ , \quad j = 1, \ldots, m_e \ \ , \\ g_j(x) \geq 0 \ , \quad j = m_e + 1, \ldots, m \ \ . \end{array} \tag{2}$$

It is assumed that all problem functions $f(x)$ and $g_j(x)$, $j = 1, \ldots, m$, are continuously differentiable on $\mathbb{R}^n$.

The basic procedure of a trust region method is to compute a new iterate x_{k+1} by a second order model or a close approximation. The stepsize is restricted by a trust region radius Δ_k, where k denotes the k-th iteration step. Subsequently, a ratio r_k of the actual and the predicted improvement subject to a certain merit function is computed. The trust region radius is either enlarged or decreased depending on the deviation of r_k from the ideal value $r_k = 1$. If sufficiently close to a solution, the artificial bound Δ_k should not become active, so that the new trial proposed by the second order model can always be accepted.

The Lagrangian function of (2) is

$$L(x, u) := f(x) - \sum_{j=1}^{m} u_j g_j(x) \tag{3}$$

and the so-called exact penalty function is given by

$$P_\sigma(x) := f(x) + \sigma \left\| g(x)^- \right\|_\infty \ \ . \tag{4}$$

Here we combine all constraints in one vector, $g(x) = (g_1(x), \ldots, g_m(x))^T$, and the minus-sign defines the constraint violation

$$g_j(x)^- := \begin{cases} g_j(x) \ , & \text{if } j \leq m_e \ , \\ \min(0, g_j(x)) \ , & \text{otherwise} \ , \end{cases}$$

$j = 1, \ldots, m$. $\sigma > 0$ denotes the penalty parameter, which must be sufficiently large and which is adapted by the algorithm. (4) is also called a merit function and is often applied to enforce convergence, see for example Fletcher [11], Burke [4], or Yuan [32].

To obtain an SQP or sequential quadratic programming method, we compute a trial step towards the next iterate by

$$d \in \mathbb{R}^n : \begin{array}{l} \min \frac{1}{2} d^T B_k d + \nabla f(x_k)^T d + \sigma_k \left\| \left(\nabla g_j(x_k)^T d + g_j(x_k) \right)^- \right\|_\infty \\ \|d\|_\infty \leq \Delta_k \ \ . \end{array} \tag{5}$$

The constraints are linearized and are treated as a penalty term subject to the maximum norm. A particular advantage is that we do not need further safeguards in case of inconsistent linear systems. Note that (5) is equivalent to the quadratic program

$$d \in \mathbb{R}^n, \delta \in \mathbb{R} : \begin{array}{l} \min \frac{1}{2} d^T B_k d + \nabla f(x_k)^T d + \sigma_k \delta \\ -\delta \leq \nabla g_j(x_k)^T d + g_j(x_k) \leq \delta \ , \quad j = 1, \ldots, m_e \ , \\ -\delta \leq \nabla g_j(x_k)^T d + g_j(x_k) \ , \quad j = m_e + 1, \ldots, m \ , \\ \|d\|_\infty \leq \Delta_k \ , \quad 0 \leq \delta \ , \end{array} \tag{6}$$

which can be solved by any available *black-box* quadratic programming solver. The optimal solution is denoted by d_k and u_k is the corresponding multiplier. σ_k and Δ_k are suitable parameters which are iteratively adapted.

The second key ingredient of a trust region method is the prediction of a new radius for the next iteration. The idea is to check the quotient of the actual and the predicted improvements of the merit function by

$$r_k := \frac{P_k(x_k) - P_k(x_k + d_k)}{\phi_k(0) - \phi_k(d_k)} \ , \tag{7}$$

where $P_k(x) := P_{\sigma_k}(x)$ denotes the penalty function (4) to measure the actual improvement, and where

$$\phi_k(d) := \nabla f(x_k)^T d + \frac{1}{2} d^T B_k d + \sigma_k \left\| \left(g(x_k) + \nabla g(x_k)^T d \right)^- \right\|_\infty \tag{8}$$

is used to estimate the linearly predicted improvement, i.e., the objective function of subproblem (5). If r_k is close to one or even greater than one, then Δ_k is enlarged and if r_k is very small, Δ_k is decreased. If r_k remains in the intermediate range, Δ_k is not changed at all. More formally, we use the same constants proposed by Yuan [32], and set

$$\Delta_{k+1} = \begin{cases} \max\left[2\Delta_k, \ 4\|d_k\|_\infty\right] \ , & \text{if} \quad r_k > 0.9 \ , \\ \Delta_k \ , & \text{if} \quad 0.1 \leq r_k \leq 0.9 \ , \\ \min[\Delta_k/4, \ \|d_k\|_\infty/2] \ , & \text{if} \quad 0 < r_k < 0.1 \ . \end{cases} \tag{9}$$

If, on the other hand, $r_k < 0$, then Δ_k is decreased and we solve subproblem (5) again. The penalty parameter σ_k is updated by testing

$$\phi_k(d_k) \leq \phi_k(0) - \delta_k \sigma_k \min\left(\Delta_k, \|g(x_k)^-\|_\infty\right) \ . \tag{10}$$

If this condition is not satisfied, the penalty parameter σ_k is increased and Δ_k is decreased.

Finally, one has to approximate the Hessian matrix of the Lagrangian function in a suitable way by a matrix B_k. To avoid calculation of second derivatives and to obtain a final superlinear convergence rate, the standard approach is to update B_k by the BFGS quasi-Newton formula. The update formula is based on the Lagrangian function (3), where the multiplier vector u_k obtained from (6) is inserted.

Because of the non-differentiable merit function, however, superlinear convergence cannot be guaranteed and it is even possible that the algorithm only converges linearly, see Yuan [30]. To avoid this situation, Fletcher [11] introduced a second order correction, for which superlinear convergence can be shown, see Yuan [31]. Thus, the *classical* trust region method as outlined before is combined with an additional correction which can be interpreted as a feasible direction step. We define a new approximation at $x_k + d_k$, if the basis step obtained form solving (5) is not as good as expected,

$$\bar{\phi}_k(d) := \frac{1}{2}(d+d_k)^T B_k (d+d_k) + \nabla f(x_k)^T (d+d_k) + \sigma_k \left\| \Big(\nabla g_j(x_k)^T d + g_j(x_k+d_k) \Big)^- \right\|_\infty \tag{11}$$

and get the modified subproblem

$$d \in \mathbb{R}^n : \begin{array}{l} \min \bar{\phi}_k(d) \\ \|d+d_k\|_\infty \le \Delta_k \end{array} . \tag{12}$$

Let the solution be $\hat{d}_k$. Since the subproblem possesses the same structure as (5), it is easily transformed into an equivalent problem which can be solved by an available *black box* quadratic programming solver, see (6). If the quadratic programming subproblems are solved by a primal-dual method as in our case, an available Cholesky decomposition of B_k can be passed from (5) to (12).

The quotient r_k needed to adopt the trust region, is replaced by

$$\hat{r}_k := \frac{P_k(x_k) - P_k(x_k + d_k + \hat{d}_k)}{\phi_k(0) - \phi_k(d_k)} . \tag{13}$$

Moreover, we need

$$\bar{r}_k := r_k + \frac{\bar{\phi}_k(0) - \bar{\phi}_k(\hat{d}_k)}{\phi_k(0) - \phi_k(d_k)} , \tag{14}$$

to predict the actual value of $\hat{r}_k$.

The resulting trust region SQP algorithm with second order correction of Yuan [32] is somewhat more complex and consists of the following steps:

Algorithm 2.1 *0: Let $x_0 \in \mathbb{R}^n$, $\Delta_0 > 0$, $B_0 \in \mathbb{R}^{n\times n}$ positive definite, $\delta_0 > 0$, $\sigma_0 > 0$, $\epsilon > 0$, and let $k := 1$.*

1: Solve subproblem (5) or (6), respectively, to get d_k and the multiplier u_k. If $\phi_k(0) - \phi_k(d_k) < \epsilon$ and $g(x_k)^- < \epsilon$, then stop. If $\|g(x_k)^-\|_\infty - \|(g(x_k)^- + \nabla g(x_k) d_k\|_\infty < \epsilon$ and $\|(g(x_k)^- + \nabla g(x_k) d_k\|_\infty > \epsilon$, let $\sigma_k := 11\sigma_k$ and $\delta_k := \delta_k/11$. If (10) is violated, let $\sigma_{k+1} := 2\sigma_k$ and $\delta_{k+1} := \delta_k/4$, else $\sigma_{k+1} := \sigma_k$ and $\delta_{k+1} := \delta_k$.

2: Compute r_k by (7). If $r_k > 0.75$, goto Step 5. Solve subproblem (12) to get $\hat{d}_k$ and the corresponding multiplier $\hat{u}_k$, and compute $\bar{r}_k$ by (14). Let $u_k := \hat{u}_k$. If $r_k < 0.25$, goto Step 3. If $0.9 < \bar{r}_k < 1.1$, let $\Delta_{k+1} := 2\Delta_k$, else $\Delta_{k+1} := \Delta_k$ and goto Step 6.

3: If $\bar{r}_k < 0.75$, goto Step 4. Otherwise, compute $f(x_k + d_k + \hat{d}_k)$ and $g(x_k + d_k + \hat{d}_k)$. If $P_k(x_k + d_k + \hat{d}_k) \ge P_k(x_k + d_k)$, goto Step 4. Calculate $\hat{r}_k$ by (13) and let $d_k := d_k + \hat{d}_k$, $r_k := \hat{r}_k$. If $r_k \ge 0.75$, goto Step 5. If $r_k \ge 0.25$, goto Step 6.

4: Let $\Delta_{k+1} := \|d_k\|_\infty/2$ and goto Step 6.

5: If $\|d_k\|_\infty < \Delta_k$, then $\Delta_{k+1} := \Delta_k$ and goto Step 6. If $r_k > 0.9$, then $\Delta_{k+1} := 4\Delta_k$, else $\Delta_{k+1} := 2\Delta_k$.

6: If $r_k > 0$, goto Step 7. Otherwise, let $x_{k+1} := x_k$, $B_{k+1} := B_k$, increment k, and goto Step 1.

7: Define a new iterate $x_{k+1} := x_k + d_k$, compute $f(x_{k+1})$ and $g(x_{k+1})$, update B_{k+1} by the BFGS formula applied to d_k and $\nabla L(x_{k+1}, u_k) - \nabla L(x_k, u_k)$. Increment k and goto Step 1.

For Algorithm 2.1, the superlinear convergence rate can be proved, see Fletcher [11] and Yuan [31]. The individual steps depend on a large number of constants, which are carefully selected based on numerical experience.

Now we introduce the integer variables again and consider the mixed-integer nonlinear program (1). The goal is to apply the trust region SQP algorithm Algorithm 2.1 outlined before and adapt it to the mixed-integer case with as few alterations as possible. Due to the integer variables, the quadratic programming subproblems that have to be solved are mixed-integer quadratic programming problems and can be solved by any available algorithm. Since the generated subproblems are always convex, we apply a branch-and-bound algorithm, see Spickenreuther [26]. Thus, the mixed-integer quadratic programming problems of Step 1 and Step 2 are of the form

$$\begin{array}{ll} & \min \frac{1}{2} \begin{pmatrix} d \\ e \end{pmatrix}^T B_k \begin{pmatrix} d \\ e \end{pmatrix} + \nabla_x f(x_k, y_k)^T d + d_y f(x_k, y_k)^T e + \sigma_k \delta \\ d \in \mathbb{R}^{n_c}, & -\delta \le \nabla_x g_j(x_k, y_k)^T d + d_y g_j(x_k, y_k)^T e + g_j(x_k, y_k) \le \delta \ , \quad j = 1, \ldots, m_e \ , \\ e \in \mathbb{Z}^{n_i}, \quad : & -\delta \le \nabla_x g_j(x_k, y_k)^T d + d_y g_j(x_k, y_k)^T e + g_j(x_k, y_k) \ , \quad j = m_e+1, \ldots, m \ , \\ \delta \in \mathbb{R} & \max(x_l, -\Delta_k^c) \le d \le \min(x_u, \Delta_k^c) \ , \\ & \max(y_l, -\Delta_k^i) \le e \le \min(y_u, \Delta_k^i) \ , \\ & 0 \le \delta \ . \end{array} \tag{15}$$

The solution is denoted by d_k and e_k.

Since we do not assume that (1) is relaxable, i.e., that f and $g_1, \ldots, g_m$ can be evaluated at any fractional parts of the integer variables, we approximate the first derivatives at $f(x, y)$ by the difference formula

$$d_y f(x, y) = \frac{1}{2} \left(f(x, y_1, \ldots, y_j + 1, \ldots, y_{n_i}) - f(x, y_1, \ldots, y_j - 1, \ldots, y_{n_i}) \right) \tag{16}$$

for $j = 1, \ldots, n_i$, at neighbored grid points. If either $y_j + 1$ or $y_j - 1$ violates a bound, we apply a non-symmetric difference formula. Similarly, $d_y g_j(x, y)$ denote a difference formula for first derivatives at $g_j(x, y)$ computed at neighbored grid points.

The adaption of the trust region radius must be modified, since a trust region radius smaller than one does not allow any further change of integer variables. Therefore, two different radii are defined, Δ_k^c for the continuous and Δ_k^i for the integer variables. We prevent Δ_k^i from falling below one by $\Delta_{k+1}^i := \max(1, 2\Delta_k^i)$ and $\Delta_{k+1}^i := \max(1, 4\Delta_k^i)$ in Step 2 and Step 5 of Algorithm 2.1, respectively. Note, however, that Step 4 allows a decrease of Δ_k^i below one. In this situation, integer variables are fixed and a new step is made only subject to the continuous variables. As soon as a new iterate is accepted, we set Δ_k^i to one in order to be able to continue optimization over the whole range of variables.

Furthermore, we use a non-monotone trust region adaption to reduce the probability of Δ_k^i falling below one, see Toint [28] for a more sophisticated version. We allow an increase of penalty function values to accept new iterates earlier and to reduce the number of reductions of Δ_k^i. Thus, we replace formula (7) by

$$r_k := \frac{P_k(x_{l_k}) - P_k(x_k + d_k)}{\phi_k(0) - \phi_k(d_k)} \quad , \tag{17}$$

where x_{l_k} is defined by

$$P_k(x_{l_k}) := \max_{0 \le j \le m_k} P_k(x_{k-j}) \tag{18}$$

with $m_k := \min(k, M)$. M defines the number of successful iterations to be considered for computing x_{l_k}. $\hat{r}_k$ is then calculated by

$$\hat{r}_k := \frac{P_k(x_{l_k}) - P_k(x_k + d_k + \hat{d}_k)}{\phi_k(0) - \phi_k(d_k)} \quad . \tag{19}$$

The robustness of our algorithm is further improved by adding a new step to Algorithm 2.1 to handle functions with narrow curved valleys. In these cases, Δ_k^i is often lower than one. *Step 8* is executed whenever the stopping condition of Step 1 is fulfilled.

8: *If no improvement since last execution of Step 8, then stop.*
Else solve relaxed subproblem (5) and round integer variables. Define a new iterate $(x_{k+1}, y_{k+1}) := (x_k + d_k, y_k + e_k)$, *compute* $f(x_{k+1}, y_{k+1})$ *and* $g(x_{k+1}, y_{k+1})$, *update* B_{k+1} *by the BFGS formula applied to* (d_k, e_k) *and* $\nabla L(x_{k+1}, y_{k+1}, u_k) - \nabla L(x_{k+1}, y_{k+1}, u_k)$, *increment* k *and goto Step 1.*

By accepting iterates leading to an increase of the penalty function P, we hope to escape from local minima.

We do not have a convergence proof for the proposed trust region method for solving nonlinear mixed-integer programming problems, even not for the convex case. However, the computational method is reliable and efficient as shown in the subsequent section.

3 Numerical Tests

We evaluate the performance of the trust region mixed-integer code called MISQP on a set of 80 mixed-integer nonlinear test problems found in the literature. Some of them are widely used to develop and test new algorithms, see for example Floudas et al. [14]. More details about the test problems, individual numerical results, and also of comparative results for continuous test problems are found in Exler and Schittkowski [9] together with a documentation of the Fortran code MISQP. The number of continuous variables is between 0 and 16, the number of integer variables between 1 and 48, the number of equality constraints between 0 and 17, and the total number of constraints between 0 and 53.

Mixed-integer quadratic programming subproblems are solved by the branch-and-bound code MIQLB4, see Spickenreuther [26], where the corresponding continuous programs are solved by the primal-dual code QL, see Schittkowski [25]. Derivatives subject to continuous variables are approximated by forward differences. The test examples are provided with exact solutions, either known from analytical solutions or from the best numerical data known to our knowledge. The Fortran codes are compiled by the Intel Visual Fortran Compiler, Version 9.1, under Windows XP64, and executed on a Dual Core AMD Opteron processor 265 with 1.81 GHz and 4 GB of RAM.

First we need a criterion to decide whether the result of a test run is considered as a successful return or not. Let $\epsilon_t > 0$ be a tolerance for defining the relative accuracy, x_k the final iterate of a test run, and $x^\star$ the supposed exact solution. Then we call the output a successful return, if the relative error in the objective function is less than ϵ_t and if the maximum constraint violation is less than ϵ_t^2, i.e., if

$$f(x_k) - f(x^\star) < \epsilon_t |f(x^\star)| \tag{20}$$

and $\|g(x_k)^-\|_\infty < \epsilon_t^2$. We take into account that a code returns a solution with a better function value than the exact one, subject to the error tolerance of the allowed constraint violation. However, there is still the possibility that an algorithm terminates at a local solution. Thus, we call the return of a test run a local one, if the internal termination conditions are satisfied subject to a reasonably small tolerance $\epsilon = 10^{-8}$ and if instead of (20)

$$f(x_k) - f(x^\star) \geq \epsilon_t |f(x^\star)| \tag{21}$$

holds. For our numerical tests, we use $\epsilon_t = 0.0001$.

Numerical results are summarized in Table 1 where we use the following notation:

Table 1: Comparison of Mixed-Integer Version versus Continuous Relaxation

code	p_{succ}	n_{bett}	n_{loc}	n_{err}	n_{grad}	n_{func}	*time*
MISQP	98 %	1	4	2	18	239	0.47
MISQP/RX	100 %	58	4	0	23	288	0.02

p_{succ}	-	percentage of successful test runs (according to above definition)
n_{bett}	-	number of better solutions, i.e., of feasible returns with function value less than $f(x^\star) - \epsilon_t$
n_{loc}	-	number of local solutions obtained
n_{err}	-	number of test runs with error messages (IFAIL>0)
n_{grad}	-	average number of gradient evaluations subject to the continuous variables or iterations, respectively
n_{func}	-	average number of equivalent function calls (including function calls used for gradient approximations)
time	-	average execution time in seconds

In two cases, the solver for the mixed-integer quadratic program is unable to find a feasible solution and an error message is generated. For four other test problems, the algorithm is unable to improve an actual iterate and reports that a local solution is obtained. However, it might be possible that the optimal solution found in the literature, is inaccurate. It must be expected that the relaxed solution is better than the mixed-integer solution. However, we observe that in about 27 % of all test runs the same solution is approached. One reason is that bounds of the integer variables become active. Surprisingly, the average number of iterations for mixed-integer solutions is less than the number of iterations needed to solve its continuous relaxation.

Because of a large number of branch-and-bound steps, the calculation time for solving mixed-integer quadratic programs is much higher than in case of relaxed problems. But for practical applications with time consuming function calls we have in mind, these additional efforts are tolerable and in many situations negligible compared to a large amount of internal calculations of a complex simulation code. We observe that there are only very few problems for which calculation times are excessively large dominating the time evaluation, see Figure 1, where shown values are in seconds multiplied by 100. The branch-and-bound solver for mixed-integer quadratic programs slows down in these cases because of a relatively large number of integer variables.

4 Conclusions

We present a new trust region SQP method for nonlinear mixed-integer optimization, where convexity and relaxation are not required. However, we assume that the model functions are *smooth* in the sense that an increment of an integer variable by one leads

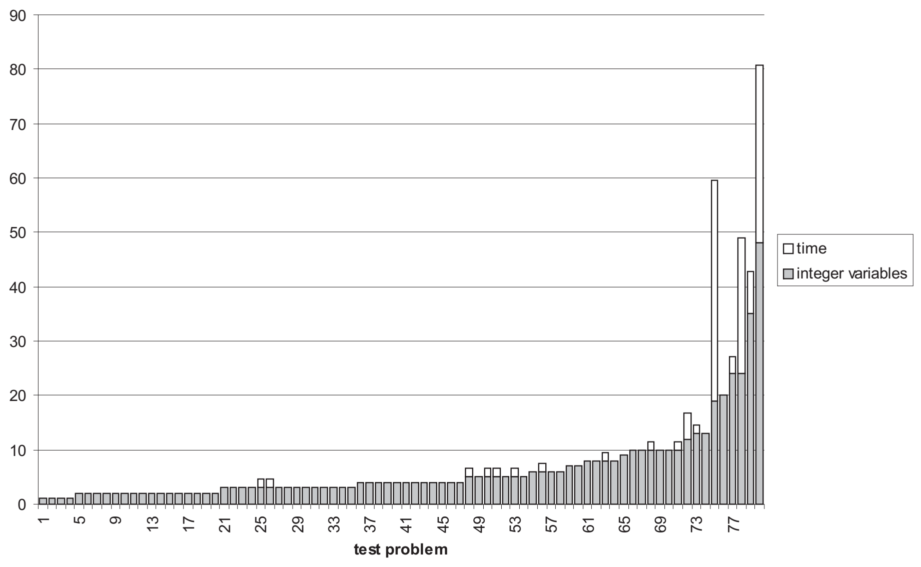

Figure 1: Integer Variables and Calculation Times

to a small change of function values. The Hessian of the Lagrangian is approximated by quasi-Newton updates for continuous and for integer variables.

Numerical tests for a set of 80 mixed-integer test problems show the efficiency of the proposed algorithm. The average number of iterations and function evaluations, respectively, is less than the corresponding figures obtained for relaxed problems starting from the same point. Thus, the proposed algorithm is expected to be more efficient than any other method which starts from a solution of the relaxed problem. Gradient values must be provided by the calling program only subject to the continuous variables. Since we assume that the problem is not relaxable, function values subject to integer variables are evaluated only at grid points.

However, we do not have a convergence proof, even not for the convex case, and we observe that the code might stop at an iterate from where a further local improvement is not possible (*local solution*). More research is required to stabilize the implementation, to analyze convergence, to understand the role of multipliers, and to improve stopping criteria. Also there is a need to replace the presently used branch-and-bound algorithm for solving mixed-integer quadratic programming problems by a more efficient method, e.g., based on branch-and-cut.

References

[1] Audet C., Dennis J.E. (2001): *Pattern search algorithm for mixed variable programming*, SIAM Journal on Optimization, Vol. 11, 573–594

[2] Borchers B., Mitchell J.E. (1994): *An improved branch and bound algorithm for mixed integer nonlnear programming,* Computers and Operations Research, Vol. 21, No. 4, 359-367

[3] Bünner M.J., Schittkowski K., van de Braak G. (2004): *Optimal design of electronic components by mixed-integer nonlinear programming*, Optimization and Engineering, Vol. 5, 271-294

[4] Burke J.V. (1992): *A robust trust region method for constrained nonlinear programming problems*, SIAM Journal on Optimization, Vol. 2, 325–347

[5] Byrd R., Schnabel R.B., Schultz G.A. (1987): *A trust region algorithm for nonlinearly constrained optimization*, SIAM Journal of Numerical Analysis, Vol. 24, 1152–1170

[6] Celis M.R. (1983): *A trust region strategy for nonlinear equality constrained optimization*, Ph.D. Thesis, Department of Mathematics, Rice University, USA

[7] Conn A.R., Gould I.M., Toint P.L. (2000): *Trust-Region Methods*, SIAM, Philadelphia

[8] Duran M., Grossmann I.E. (1986): *An outer-approximation algorithm for a class of mixed-integer nonlinear programs*, Mathematical Programming, Vol. 36, 307–339

[9] Exler O., Schittkowksi K. (2006): *MISQP: A Fortran implementation of a trust region SQP algorithm for mixed-integer nonlinear programming - user's guide, version 2.1*, Report, Department of Computer Science, University of Bayreuth
`http://www.uni-bayreuth.de/departments/math/~kschittkowski/MISQP.pdf`

[10] Fletcher R. (1981): *Practical Methods of Optimization. Volume 2, Constrained Optimization*, Wiley, Chichester

[11] Fletcher R. (1982): *Second order correction for nondifferentiable optimization*, in: Watson G.A. (Hrsg.): *Numerical analysis*, Springer Verlag, Berlin, 85–114

[12] Fletcher R., Leyffer S. (1994): *Solving mixed integer nonlinear programs by outer approximation*, Mathematical Programming, Vol. 66, 327–349

[13] Floudas C.A. (1995): *Nonlinear and Mixed-Integer Optimization*, Oxford University Press, New York, Oxford

[14] Floudas C.A., Pardalos P.M., Adjiman C.S., Esposito W.R., Gumus Z.H., Harding S.T., Klepeis J.L., Meyer C.A., Schweiger C.A. (1999): *Handbook of Test Problems in Local and Global Optimization*, Kluwer Academic Publishers

[15] Gill P.E., Murray W., Wright M. (1981): *Practical Optimization*, Academic Press, New York

[16] Grossmann I.E., Kravanja Z. (1997): *Mixed-integer nonlinear programming: A survey of algorithms and applications*, in: Conn A.R., Biegler L.T., Coleman T.F., Santosa F.N. (eds.): *Large-Scale Optimization with Applications, Part II: Optimal Design and Control*, Springer, New York, Berlin

[17] Gupta O. K., Ravindran V. (1985): *Branch and bound experiments in convex nonlinear integer programming*, Management Science, Vol. 31, 1533–1546

[18] Leyffer S. (2001): *Integrating SQP and branch-and-bound for mixed integer nonlinear programming*, Computational Optimization and Applications, Vol. 18, 295–309

[19] Li H.-L., Chou C.-T. (1994): *A global approach for nonlinear mixed discrete programming in design optimization*, Engineering Optimization, Vol. 22, 109–122

[20] Moré J.J. (1983) *Recent developments in algorithms and software for trust region methods*, in: Bachem A., Grötschel M., Korte B. (eds.): *Mathematical Programming: The State of the Art*, Springer, Berlin, 258–287

[21] Powell M.J.D. (1978): *The convergence of variable metric methods for nonlinearly constrained optimization calculations*, in: O.L. Mangasarian, R.R. Meyer, S.M. Robinson (eds.): *Nonlinear Programming 3*, Academic Press

[22] Powell M.J.D. (1984): *On the global convergence of trust region algorithms for unconstrained minimization*, Mathematical Programming, Vol. 29, 297–303

[23] Powell M.J.D., Yuan Y. (1991): *A trust region algorithm for equality constrained optimization*, Mathematical Programming, Vol. 49, 189–211

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

[25] Schittkowski K. (2003): *QL: A Fortran code for convex quadratic programming - user's guide, version 2.11*, Report, Department of Mathematics, University of Bayreuth
`http://www.uni-bayreuth.de/departments/math/~kschittkowski/QL.pdf`

[26] Spickenreuther T. (2005): *Entwicklung eines allgemeinen Branch & Bound Ansatzes zur gemischt-ganzzahligen Optimierung*, Diploma Thesis, Department of Mathematics, University of Bayreuth

[27] Stoer J. (1985): *Foundations of recursive quadratic programming methods for solving nonlinear programs,* in: K. Schittkowski (ed.): *Computational Mathematical Programming*, NATO ASI Series, Series F: Computer and Systems Sciences, Vol. 15, Springer

[28] Toint P.L. (1997): *A nonmonotone trust-region algorithm for nonlinear optimization subject to convex constraints*, Mathematical Programming, Vol. 77, 69–94

[29] Westerlund, Pörn (2002): *Solving pseudo-convex mixed integer optimization problems by cutting plane techniques*, Optimization and Engineering, Vol. 3, 253-280

[30] Yuan Y.-X. (1984): *An example of only linearly convergence of trust region algorithms for nonsmooth optimization*, IMA Journal on Numerical Analysis, Vol. 4, 327–335

[31] Yuan Y.-X. (1985): *On the superlinear convergence of a trust region algorithm for nonsmooth optimization*, Mathematical Programming, Vol. 31, 269–285

[32] Yuan Y.-X. (1995): *On the convergence of a new trust region algorithm*, Numerische Mathematik, Vol. 70, 515–539