Math. Prog. Comp. (2012) 4:383–412
DOI 10.1007/s12532-012-0045-0

FULL LENGTH PAPER

A comparative study of SQP-type algorithms for nonlinear and nonconvex mixed-integer optimization

Oliver Exler · Thomas Lehmann · Klaus Schittkowski

Received: 24 March 2012 / Accepted: 11 July 2012 / Published online: 5 August 2012

Abstract We present numerical results of a comparative study of codes for nonlinear and nonconvex mixed-integer optimization. The underlying algorithms are based on sequential quadratic programming (SQP) with stabilization by trust-regions, linear outer approximations, and branch-and-bound techniques. The mixed-integer quadratic programming subproblems are solved by a branch-and-cut algorithm. Second order information is updated by a quasi-Newton update formula (BFGS) applied to the Lagrange function for continuous, but also for integer variables. We do not require that the model functions can be evaluated at fractional values of the integer variables. Thus, partial derivatives with respect to integer variables are replaced by descent directions obtained from function values at neighboring grid points, and the number of simulations or function evaluations, respectively, is our main performance criterion to measure the efficiency of a code. Numerical results are presented for a set of 100 academic mixed-integer test problems. Since not all of our test examples are convex, we reach the best-known solutions in about 90 % of the test runs, but at least feasible solutions in the other cases. The average number of function evaluations of the new mixed-integer SQP code is between 240 and 500 including those needed for one- or two-sided approximations of partial derivatives or descent directions, respectively. In

Sponsored by Shell GameChanger, SIEP Rijswijk, under project number 4600003917.

O. Exler · K. Schittkowski (✉)
Department of Computer Science, University of Bayreuth,
95440 Bayreuth, Germany
e-mail: klaus@schittkowski.de

T. Lehmann
Konrad-Zuse-Zentrum (ZIB), Takustr. 7,
14195 Berlin, Germany

addition, we present numerical results for a set of 55 test problems with some practical background in petroleum engineering.

Keywords MINLP · Mixed-integer nonlinear programming · SQP · Sequential quadratic programming · Trust region method · Linear outer approximation · MIQP · Mixed-integer quadratic programming · Numerical algorithms · Performance evaluation · Mixed-integer test problems · Engineering optimization

Mathematics Subject Classification 90-08 · 90C10 · 90C30 · 90C55 · 65K05

1 Introduction

We consider the general mixed-integer nonlinear program to minimize a scalar objective function under nonlinear equality and inequality constraints,

$$\begin{aligned} \underset{x \in \mathbb{R}^{n_c},\ y \in \mathbb{Z}^{n_i}}{\text{minimize}} \quad & f(x, y) \\ \text{subject to} \quad & g_j(x, y) = 0, \quad j = 1, \ldots, m_e, \\ & g_j(x, y) \geq 0, \quad j = m_e + 1, \ldots, m, \\ & x \in X,\ y \in Y. \end{aligned} \tag{1}$$

x denotes the continuous and y the integer variables including binary variables. The two sets X and Y are defined by upper and lower bounds of variables,

$$\begin{aligned} X &:= \{x \in \mathbb{R}^{n_c} : x_l \leq x \leq x_u\}, \\ Y &:= \{y \in \mathbb{Z}^{n_i} : y_l \leq y \leq y_u\}, \end{aligned} \tag{2}$$

where n_c is the number of continuous variables and n_i is the number of integer variables. It is assumed that the problem functions $f(x, y)$ and $g_j(x, y)$, $j = 1, \ldots, m$, are twice continuously differentiable with respect to x for all $x \in X$.

Numerous algorithms to address mixed-integer nonlinear programs have been proposed in the past, see for example Floudas [25] or Grossmann and Kravanja [29] for review papers. Comparative results of a variety of solvers are found in Bonami et al. [8], and a review on available software is published by Bussieck and Vigerske [11].

Typically, these approaches require 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 be computed for all $y \in Y_{\mathbb{R}}$, where

$$Y_{\mathbb{R}} := \{y \in \mathbb{R}^{n_i} : y_l \leq y \leq y_u\}. \tag{3}$$

In this case, we require in addition that $f(x, y)$ and $g_j(x, y)$, $j = 1, \ldots, m$, are also twice continuously differentiable subject to all $y \in Y_{\mathbb{R}}$.

However, many real-life mixed-integer problems are not relaxable. Functions are often highly nonlinear and nonconvex and sometimes their values can only be computed by complex simulation software. Examples arise in mechanical, electrical, aerospace, chemical, automotive, petroleum and many other areas. An industrial case study is considered by Bünner et al. [10], where typical integer variables are the number of fingers and layers of an electronic filter, which cannot be relaxed due to the underlying simulation tools. Other typical non-relaxable integer variables are the number of rips and rills of a corrugated horn antenna, see Hartwanger et al. [32], or the number and position of trays of a distillation column, see Thomas and Kröner [54]. In Exler, et al. [18], the authors consider a wastewater treatment plant for nitrogen removal. The feed layer in a settler is an integer variable with physical meaning that is not relaxable due to the underlying Simulink implementation.

When considering non-relaxable integer variables, we do not have in mind categorial variables, e.g., the series of planets of flyby missions or the type and the number of boosters of a launch vehicle, see Schlueter [52], where a change of an integer value leads to a structural change of the underlying model. In this case, there is not a natural or intrinsic ordering of the decision variables in a category. We are more interested in applications, especially in engineering sciences, which are modeled by non-relaxable integer variables with some *physical* meaning. In other words, function values implicitly depend on each other, please see the examples mentioned above. Consider, e.g., the number of fingers of an electronic filter. A change from 36 to 37 would not lead to a dramatic change of the objective or constraint functions. A plot of objective function values over two of these integer variables, evaluated only at integer values, would look like a smooth grid. This observation is the main motivation for our idea to apply quadratic approximations.

We suppose that a *black box* simulation code provides function values, i.e., that we do not know anything about the internal analytical structure of the model functions nor how the model equations are implemented. Since, in addition, most simulation systems in engineering sciences are highly complex, calculation times of these black box simulation codes are often excessive. Typical requirements for an optimization method in this environment are to limit the number of function evaluations to fewer than 1000 and to operate efficiently without any partial derivative information.

In the past decades, several heuristic and deterministic methods to solve optimization problems with integer variables were proposed. Search algorithms were developed that explore the integer space, as pattern search algorithms, see, e.g., Audet and Dennis [2]. Other approaches transform the mixed-integer problem into a continuous problem and make use of nonlinear optimization techniques. Li and Chou [39] for instance, replace the integrality condition by continuous nonlinear constraints, and the resulting highly nonconvex program is solved by a global optimization algorithm.

Branch-and-bound methods also transform the problem. Here a series of relaxed nonlinear programs obtained by restricting the variable range of the relaxed integer variables must be solved, see Gupta and Ravindran [30] or Borchers and Mitchell [9]. When applying an SQP algorithm at a node of the search tree, it is possible to apply early branching as described by Leyffer [38].

Another frequently used solution method for solving mixed-integer nonlinear programming problems is based on linear outer approximations. The idea is introduced by

Duran and Grossmann [17] and is extended by Fletcher and Leyffer [24]. Convergence towards the global optimal solution of a convex program is guaranteed by considering a gradually improving mixed-integer linear relaxation of the original mixed-integer nonlinear program. A related approach was proposed by Westerlund and Pörn [57]. The ECP technique applies cutting planes as in linear programming.

Several codes were implemented that address mixed-integer problems. BARON solves nonconvex optimization problems with continuous and integer variables based on the *Branch And Reduce Optimization Navigator* combining constraint propagation, interval analysis, and duality with enhanced branch and bound concepts, see Sahinidis and Tawarmalani [47] and Tawarmalani and Sahinidis [53]. Convex and polyhedral relaxations are used by Nowak et al. [42] to generate inner and outer approximations, where the resulting code is called LAGO. Moreover, the code DICOPT realizes an extension of the outer approximation method for an equality relaxation strategy, and comes with some heuristics for solving nonconvex problems, see Viswanathan and Grossmann [56].

Open-source solvers have been created within the COIN-OR initiative: BONMIN consists of sophisticated branch-and-bound, outer approximation, branch-and-cut and hybrid codes, see Bonami et al. [7]. Another implementation is COUENNE based on convex over- and under-envelopes and a branch-and-bound algorithm to solve nonconvex mixed-integer nonlinear programs, see Belotti et al. [6,5].

A first version of our mixed-integer sequential quadratic programming method (MISQP) was discussed and implemented by Exler and Schittkowski [21]. The algorithm proceeds from the SQP-based trust region method of Yuan [58], see also Schittkowski and Yuan [51], and is adapted to solve nonlinear mixed-integer optimization problems by solving a sequence of mixed-integer quadratic subproblems. The algorithm is outlined in Sect. 2 in more detail.

A possible stabilization of the method of Exler et al. [21] is achieved by adding linear outer approximations as proposed by Fletcher and Leyffer [24] and Duran and Grossmann [17]. These modifications are presented in Sect. 3 together with a general outline of linear outer approximation algorithms.

For a sequential quadratic programming (SQP) method, but also for applying outer approximations, the availability of first partial derivatives is crucial. An important question is how to replace partial derivatives with respect to non-relaxable integer variables. Since $f(x, y)$ and $g_1(x, y), \ldots, g_m(x, y)$ cannot be evaluated at small perturbations of an integer variable value, we compute a descent direction instead. For the objective function and given $x \in X$ and $y \in Y$, we apply a two-sided difference formula,

$$\frac{\partial f(x, y)}{\partial y_j} \approx \frac{1}{2}\left(f(x, y_1, \ldots, y_j + 1, \ldots, y_{n_i}) - f(x, y_1, \ldots, y_j - 1, \ldots, y_{n_i})\right) \tag{4}$$

for $j = 1, \ldots, n_i$, where $y = (y_1, \ldots, y_{n_i})^T$. The same formula is used for constraint functions. Since our algorithms guarantee satisfaction of box constraints, the formula is adapted at a bound. For binary variables or for variables at a bound, (4) is replaced by a forward or backward difference formula, respectively. Partial derivatives with

respect to continuous variables are computed by standard difference formulae, but can be provided by a user also in analytical form.

There is a very attractive advantage of approximating integer derivatives at grid points. The additional function evaluations for gradient approximations are not wasted. We keep track of the best feasible iterate regardless whether it has been used for computing a descent direction or for the main iteration of our MISQP algorithm, and return to this point whenever it seems to be profitable.

It is important to understand that our SQP-type algorithm is also applicable to solve relaxable mixed-integer programming problems. If analytical partial derivatives are available, the user may pass them to the MISQP code. Moreover, if the number of integer variables is set to zero, i.e., if $n_i = 0$, the algorithm behaves exactly like a standard SQP-code for continuous optimization stabilized by trust-regions.

An interesting question is whether the usage of analytical derivatives, if available, leads to a better performance or not. Our numerical results indicate that numerical approximations at grid points lead to a somewhat more reliable implementation, i.e., more problems are successfully solved. A possible explanation might be that the main goal is to achieve a descent direction with respect to a suitable merit function from one grid point to the next, not locally around the current iterate.

Section 4 outlines the test framework and contains numerical results. To compare robustness and efficiency of our codes, we develop criteria to decide whether the result of a test run can be considered as a successful one or not. Two sets of test problems are considered, a selection of 100 academic mixed-integer nonlinear test problems published in Schittkowski [50], and a set of 55 test problems with practical background in petroleum engineering. Both are used to compare our codes MISQP [20], MISQPN [19], MISQPOA [36], and MINLPB4 [35]. Some of them are executed with different parameters to test alternative formulations. It turns out that different versions of MISQP solve between 70 and 90 problems out of a set of 100 academic test examples, whereas the average number of function evaluations is between 240 and 500 including those needed for approximating derivatives. Numerical results are presented in form of mean values, priority factors, and performance profiles.

2 A sequential quadratic programming algorithm with trust region stabilization

In this section, we present an SQP-type algorithm developed by Exler and Schittkowski [21]. The algorithm extends the concept of a trust region SQP algorithm as described by Yuan [58], Conn et al. [14], Schittkowski and Yuan [51] and many others, to mixed-integer nonlinear optimization. Instead of solving continuous quadratic programs, we solve a sequence of mixed-integer convex quadratic optimization problems to approximate a solution of (1).

The Lagrange function of the mixed-integer nonlinear program (1) is

$$L(x, y, u, v_l, v_u, w_l, w_u) := f(x, y) - \sum_{j=1}^{m} u_j g_j(x, y) - v_l^T (x - x_l) \\ -v_u^T (x_u - x) - w_l^T (y - y_l) - w_u^T (y_u - y), \qquad (5)$$

where u_j is the Lagrange multiplier for the j-th constraint, $j = 1, \ldots, m$, where v_l and v_u are multipliers for lower and upper bounds of the continuous variables, and where w_l and w_u are multipliers for lower and upper bounds of the integer variables. Note that in the presence of integer variables, local optimality criteria based on the Lagrange function (5) do not exist. In the mixed-integer case, multipliers are only used for updating a quasi-Newton matrix. They become more important in case of continuous optimization ($n_i = 0$) to serve as local optimality criteria.

To simplify the notation, we define

$$g(x, y) := (g_1(x, y), \ldots, g_m(x, y))^T,$$
$$\nabla g(x, y) := (\nabla g_1(x, y), \ldots, \nabla g_m(x, y)).$$

The method is based on the exact L_∞-penalty function

$$P_\sigma(x, y) := f(x, y) + \sigma \|g(x, y)^-\|_\infty, \tag{6}$$

where $g(x, y)^- \in \mathbb{R}^m$ is defined as

$$g_j(x, y)^- := g_j(x, y), \quad 1 \leq j \leq m_e \tag{7}$$

and

$$g_j(x, y)^- := \min(g_j(x, y), 0), \quad m_e + 1 \leq j \leq m. \tag{8}$$

The penalty parameter σ is adapted by the algorithm. It is guaranteed that our algorithm always stays within the bounds given by X and Y, see (2), which are therefore not included in $P_\sigma(x, y)$.

In the remainder of this section, the subscript k denotes the iteration index. The basic idea of trust region methods is to approximate the original problem by a simpler one, in our case a convex mixed-integer quadratic program. In addition, we add a *trusted neighborhood* to avoid steps that are too large. The solution of the quadratic subproblem subject to the trust region is a potential step towards a new iterate. Depending on the quality of the predicted improvement compared to the actual change in the merit function (6), the trial point is accepted or rejected and the trust region is enlarged or reduced, respectively.

To approximate $P_{\sigma_k}(x_k, y_k)$ in the k-th iteration step, where (x_k, y_k) is a current iterate and σ_k a suitable penalty parameter, we successively solve the subproblem

$$\begin{aligned} \underset{d^c \in \mathbb{R}^{n_c},\ d^i \in \mathbb{Z}^{n_i}}{\text{minimize}} \quad & \nabla f(x_k, y_k)^T d + \tfrac{1}{2} d^T C_k d + \sigma_k \left\| \left(g(x_k, y_k) + \nabla g(x_k, y_k)^T d \right)^- \right\|_\infty \\ \text{subject to} \quad & \|d^c\|_\infty \leq \Delta_k^c, \ \|d^i\|_\infty \leq \Delta_k^i, \\ & x_k + d^c \in X, \, y_k + d^i \in Y, \end{aligned} \tag{9}$$

with $d := (d^c, d^i)$. Note that (9) is feasible and that $C_k \in \mathbb{R}^{(n_c+n_i)\times(n_c+n_i)}$ is positive definite for all k. Moreover, it is guaranteed that the subsequent iterate $x_{k+1} := x_k + d_k^c$, $y_{k+1} := y_k + d_k^i$, if accepted, satisfies the bounds given by (2). $\Delta_k^c > 0$ and $\Delta_k^i \geq 0$

denote the trust region radii for the continuous and the integer search space, respectively.

We use the L_∞-norm to be able to replace (9) by a standard mixed-integer quadratic program, see below. To simplify the notation, binary variables are considered as integer variables with bounds 0 and 1. Our code MISQP handles both types of variables separately.

During the remainder of this section, we denote the objective function of the mixed-integer subproblem (9) by

$$\Phi_k(d) := \nabla f(x_k, y_k)^T d + \frac{1}{2} d^T C_k d + \sigma_k \left\| \left(g(x_k, y_k) + \nabla g(x_k, y_k)^T d \right)^- \right\|_\infty . \tag{10}$$

Since (9) is non-smooth, we introduce a slack variable $\eta \in \mathbb{R}$ to replace (9) by an equivalent mixed-integer quadratic programming problem

$$\begin{aligned} \underset{d^c \in \mathbb{R}^{n_c},\, d^i \in \mathbb{Z}^{n_i},\, \eta \in \mathbb{R}}{\text{minimize}} \quad & \nabla f(x_k, y_k)^T d + \tfrac{1}{2} d^T C_k d + \sigma_k \eta \\ \text{subject to} \quad & \eta + g_j(x_k, y_k) + \nabla g_j(x_k, y_k)^T d \geq 0,\ j = 1, \ldots, m_e, \\ & \eta - g_j(x_k, y_k) - \nabla g_j(x_k, y_k)^T d \geq 0,\ j = 1, \ldots, m_e, \\ & \eta + g_j(x_k, y_k) + \nabla g_j(x_k, y_k)^T d \geq 0,\ j = m_e + 1, \ldots, m, \\ & \|d^c\|_\infty \leq \Delta_k^c,\ \|d^i\|_\infty \leq \Delta_k^i, \\ & x_k + d^c \in X,\ y_k + d^i \in Y,\ \eta \geq 0, \end{aligned} \tag{11}$$

where $d := \left(d^c, d^i\right)$. If we fix the integer variables, we get the continuous convex quadratic program

$$\begin{aligned} \underset{d^c \in \mathbb{R}^{n_c},\, \eta \in \mathbb{R}}{\text{minimize}} \quad & \nabla_x f(x_k, y_k)^T d^c + \tfrac{1}{2} d^{cT} C_k^c d^c + \sigma_k \eta \\ \text{subject to} \quad & \eta + g_j(x_k, y_k) + \nabla_x g_j(x_k, y_k)^T d^c \ \geq\ 0,\ j = 1, \ldots, m_e, \\ & \eta - g_j(x_k, y_k) - \nabla_x g_j(x_k, y_k)^T d^c \ \geq\ 0,\ j = 1, \ldots, m_e, \\ & \eta + g_j(x_k, y_k) + \nabla_x g_j(x_k, y_k)^T d^c \ \geq\ 0,\ j = m_e + 1, \ldots, m, \\ & \|d^c\|_\infty \leq \Delta_k^c, \quad \eta \geq 0, \quad x_k + d^c \in X. \end{aligned} \tag{12}$$

In this case, the matrix C_k^c is the $n_c \times n_c$ upper left part of $C_k \in \mathbb{R}^{(n_c+n_i)\times(n_c+n_i)}$. C_k is updated by a BFGS quasi-Newton formula, see e.g. Fletcher [22], subject to gradients of the Lagrange function (5). Partial derivatives subject to integer variables are approximated, e.g., by the difference formula (4), if the functions $f(x, y)$ and $g_1(x, y), \ldots, g_m(x, y)$ are not relaxable.

However, for updating C_k we need Lagrange multipliers with respect to all constraints at a new iterate $x_{k+1} \in X$, $y_{k+1} \in Y$. They can be calculated by solving the following least squares problem related to the KKT conditions,

$$
\begin{aligned}
\underset{\substack{u \in \mathbb{R}^m,\, v_l,\, v_u \in \mathbb{R}^{n_c},\\ w_l,\, w_u \in \mathbb{R}^{n_i}}}{\text{minimize}} \quad & \left\| \nabla f(x_{k+1}, y_{k+1}) - \nabla g(x_{k+1}, y_{k+1})\, u - \begin{pmatrix} v_l \\ w_l \end{pmatrix} + \begin{pmatrix} v_u \\ w_u \end{pmatrix} \right\|_2^2 \\
\text{subject to} \quad & u_j \geq 0 \text{ for all } j \in J_k,\ j > m_e, \text{ and } u_j = 0 \text{ for all } j \in \overline{J}_k, \\
& v_j^l \geq 0 \text{ for all } j \in V_k^l, \text{ and } v_j^l = 0 \text{ for all } j \in \overline{V}_k^l, \\
& v_j^u \geq 0 \text{ for all } j \in V_k^u, \text{ and } v_j^u = 0 \text{ for all } j \in \overline{V}_k^u, \\
& w_j^l \geq 0 \text{ for all } j \in W_k^l, \text{ and } w_j^l = 0 \text{ for all } j \in \overline{W}_k^l, \\
& w_j^u \geq 0 \text{ for all } j \in W_k^u, \text{ and } w_j^u = 0 \text{ for all } j \in \overline{W}_k^u,
\end{aligned} \tag{13}
$$

where $J_k := \{1, \ldots, m_e\} \cup \{j \ : g_j(x_{k+1}, y_{k+1}) \leq 0,\ j = m_e + 1, \ldots, m\}$ denotes the index set of active constraints and $\overline{J}_k := \{1, \ldots, m\} \backslash J_k$ its complement. The other multipliers are related to active bounds of variables with $v_l = (v_1^l, \ldots, v_{n_c}^l)^T$, $v_u = (v_1^u, \ldots, v_{n_c}^u)^T$, $w_l = (w_1^l, \ldots, w_{n_i}^l)^T$, and $w_u = (w_1^u, \ldots, w_{n_i}^u)^T$. The set V_k^l is defined by $V_k^l := \{j \in \{1, \ldots, n_c\} \ : x_j = x_j^l\}$ with complement $\overline{V}_k^l := \{1, \ldots, n_c\} \backslash V_k^l$. The other sets are defined accordingly.

All index sets may change from one iteration to the next. We denote the solution of (13) by $u_k \in \mathbb{R}^m$, $v_k^l, v_k^u \in \mathbb{R}^{n_c}$, and $w_k^l, w_k^u \in \mathbb{R}^{n_i}$. These multiplier approximations are then used to compute gradients of the Lagrange function (5). Together with the difference vectors $(x_{k+1}, y_{k+1}) - (x_k, y_k)$ and corresponding differences of gradients of the Lagrange function, we are able to update C_k by the BFGS-formula leading to C_{k+1}. The inner product of one of the two denominators is modified to guarantee positive definite matrices. Lagrange multipliers are calculated by (13) to ensure their independence from the branch-and-cut strategy for solving MIQP subproblems of the form (11). Otherwise, it might happen that multipliers of active artificial bounds introduced by the selected branching strategy influence the multipliers subject to the linear constraints of (9).

In the continuous case, a drawback of using the L_∞-penalty function is the Maratos [41] effect, see Fukushima [26] or Yuan and Sun [59], that prevents superlinear convergence even arbitrarily close to a stationary point under certain circumstances. To overcome this difficulty, a second order correction (SOC) is proposed by Fletcher [23] and Yuan [58] by solving an additional quadratic programming problem in certain situations. However, we apply the SOC step for the continuous variables only. Integer variables are fixed, i.e., are set to d_k^i obtained by (11), and we obtain

$$
\begin{aligned}
\underset{d^c \in \mathbb{R}^{n_c}}{\text{minimize}} \ & \nabla f(x_k, y_k)^T \left(d_k + \begin{pmatrix} d^c \\ 0 \end{pmatrix} \right) + \frac{1}{2} \left(d_k + \begin{pmatrix} d^c \\ 0 \end{pmatrix} \right)^T C_k \left(d_k + \begin{pmatrix} d^c \\ 0 \end{pmatrix} \right) \\
& + \sigma_k \left\| \left(g((x_k, y_k) + d_k) + \nabla g(x_k, y_k)^T \begin{pmatrix} d^c \\ 0 \end{pmatrix} \right)^- \right\|_\infty \\
\text{subject to} \ & \|d_k^c + d^c\|_\infty \leq \Delta_k^c, \\
& x_k + d_k^c + d^c \in X,
\end{aligned} \tag{14}
$$

where $d_k = (d_k^c, d_k^i)$ is the solution of (11). The non-smooth problem (14) can also be rewritten as a smooth quadratic programming problem in standard form similar to (11). Let $\hat{d}_k := (\hat{d}_k^c, 0)$ denote its optimal solution.

To prevent a change in the notation, we use in the sequel $\nabla_y f(x, y)$ or $\nabla_y g(x, y)$ either for exact analytical gradients and Jacobians, if available, or alternatively for numerical approximations computed by the subsequent algorithm. The data returned, are descent directions evaluated at grid points. For the sake of completeness, we state the detailed calculations in Procedure 2.1.

We would like to highlight that additional function evaluations for gradient approximations are not wasted. We keep track of the best feasible point subject to a tolerance $\epsilon > 0$ that has been evaluated. We may return to this neighboring grid point whenever it seems to be profitable. Thus, Procedure 2.1 can be interpreted as a direct neighborhood search. It returns the best feasible neighbor of (x_k, y_k), if one exists, which is denoted by (x^{bn}, y^{bn}) and f^{bn}, respectively. We omit the iteration index k to improve readability.

Procedure 2.1 *Given $x \in X$, $y \in Y$, $f(x, y)$ and $g(x, y)$. Let $\epsilon > 0$ be a small tolerance and $f^{bn} := \infty$, $(x^{bn}, y^{bn}) = (x, y)$.*
Output: *Approximations to $\nabla_y f(x, y)$, $\nabla_y g(x, y)$, f^{bn} and (x^{bn}, y^{bn}).*
begin
for $i = 1$ **to** n_i **do**
$z^{+1} := (x, y_1, \ldots, y_i + 1, \ldots, y_{n_i})$ *and* $z^{-1} := (x, y_1, \ldots, y_i - 1, \ldots, y_{n_i})$.
if $y_i^l < y_i < y_i^u$ **then**
Evaluate $f(z^{+1})$, $g(z^{+1})$ *and* $f(z^{-1})$, $g(z^{-1})$.
if $\|g(z^{+1})^-\|_\infty \leq \epsilon$ **and** $f(z^{+1}) < f^{bn}$ **then** $f^{bn} := f(z^{+1})$ *and* $(x^{bn}, y^{bn}) := z^{+1}$.
if $\|g(z^{-1})^-\|_\infty \leq \epsilon$ **and** $f(z^{-1}) < f^{bn}$ **then** $f^{bn} := f(z^{-1})$ *and* $(x^{bn}, y^{bn}) := z^{-1}$.
Set $\dfrac{\partial f(x, y)}{\partial y_i} := \dfrac{1}{2}\left(f(z^{+1}) - f(z^{-1})\right)$.
for $j = 1$ **to** m **do** *Set* $\dfrac{\partial g_j(x, y)}{\partial y_i} := \dfrac{1}{2}\left(g_j(z^{+1}) - g_j(z^{-1})\right)$.
else if $y_i = y_i^l$ **then**
Evaluate $f(z^{+1})$ *and* $g(z^{+1})$.
if $\|g(z^{+1})^-\|_\infty \leq \epsilon$ **and** $f(z^{+1}) < f^{bn}$ **then** $f^{bn} := f(z^{+1})$ *and* $(x^{bn}, y^{bn}) := z^{+1}$.
Set $\dfrac{\partial f(x, y)}{\partial y_i} := f(z^{+1}) - f(x, y)$.
for $j = 1$ **to** m **do** *Set* $\dfrac{\partial g_j(x, y)}{\partial y_i} := g_j(z^{+1}) - g_j(x, y)$.
else if $y_i = y_i^u$ **then**
Evaluate $f(z^{-1})$ *and* $g(z^{-1})$.
if $\|g(z^{-1})^-\|_\infty \leq \epsilon$ **and** $f(z^{-1}) < f^{bn}$ **then** $f^{bn} := f(z^{-1})$ *and* $(x^{bn}, y^{bn}) := z^{-1}$.
Set $\dfrac{\partial f(x, y)}{\partial y_i} := f(x, y) - f(z^{-1})$.
for $j = 1$ **to** m **do** *Set* $\dfrac{\partial g_j(x, y)}{\partial y_i} := g_j(x, y) - g_j(z^{-1})$.
end if
end do
end

We now state the mixed-integer sequential quadratic programming algorithm with trust region stabilization which is an extension of Yuan's [58] trust region method.

Algorithm 2.1 *Let $\Delta_0^c > 0$, $\Delta_0^i \geq 1$, $\sigma_0 > 0$, $\bar{\sigma} > 0$, and $\epsilon > 0$ be given constants, choose starting values $x_0 \in X$, $y_0 \in Y$ and a positive definite matrix $C_0 \in \mathbb{R}^{(n_c+n_i)\times(n_c+n_i)}$. Let $f^* := \infty$, $(x^*, y^*) = (x_0, y_0)$ be the current best known solution. Evaluate function and partial derivative values $f(x_0, y_0)$, $g(x_0, y_0)$, $\nabla_x f(x_0, y_0)$ and $\nabla_x g(x_0, y_0)$ with respect to the continuous variables. Set $k := 0$.*

1. *Approximate $\nabla_y f(x_k, y_k)$ and $\nabla_y g(x_k, y_k)$ with respect to integer variables using Procedure 2.1 and obtain (x_k^{bn}, y_k^{bn}) and f_k^{bn}.*

   if $\|g(x_k^{bn}, y_k^{bn})^-\|_\infty \leq \epsilon$ **and** $f(x_k^{bn}, y_k^{bn}) < f^*$ **then** *update* $f^* := f(x_k^{bn}, y_k^{bn})$ *and* $(x^*, y^*) := (x_k^{bn}, y_k^{bn})$.
2. *Solve the mixed-integer quadratic programming problem (11) to get $d_k = (d_k^c, d_k^i)^T$.*

   if $\Phi_k(0) - \Phi_k(d_k) \leq \epsilon$ **and** *(*$\|g(x_k, y_k)^-\|_\infty \leq \epsilon$ **or** $\sigma_k > \bar{\sigma}$*),* **then goto** *Step 10.*
3. **if** $\|g(x_k, y_k)^-\|_\infty - \|(g(x_k, y_k) + \nabla g(x_k, y_k)^T d_k)^-\|_\infty < \epsilon$

   and $\|(g(x_k, y_k) + \nabla g(x_k, y_k)^T d_k)^-\|_\infty > \epsilon$ **then** $\sigma_{k+1} := 10\sigma_k$

   else $\sigma_{k+1} := \sigma_k$.

   if $\Phi_k(0) - \Phi_k(d_k) < \sigma_k \min[\Delta_k^c, \|g(x_k, y_k)^-\|_\infty]$ **then** $\sigma_{k+1} := 2\sigma_{k+1}$.
4. *Evaluate new function values $f(x_k + d_k^c, y_k + d_k^i)$, $g_j(x_k + d_k^c, y_k + d_k^i)$, $j = 1, \ldots, m$, and compute the ratio of the actual and the predicted improvements*

$$r_k := \frac{P_{\sigma_{k+1}}(x_k, y_k) - P_{\sigma_{k+1}}(x_k + d_k^c, y_k + d_k^i)}{\Phi_k(0) - \Phi_k(d_k)} \quad . \tag{15}$$

5. **if** $r_k \leq 0.75$ **then** *solve the SOC problem (14) to obtain a solution $\hat{d}_k = (\hat{d}_k^c, 0)^T$ and evaluate new function values*

$$f(x_k + d_k^c + \hat{d}_k^c, y_k + d_k^i) \ , \quad g_j(x_k + d_k^c + \hat{d}_k^c, y_k + d_k^i), \ j = 1, \ldots, m.$$

   if $P_{\sigma_{k+1}}(x_k + d_k^c + \hat{d}_k^c, y_k + d_k^i) < P_{\sigma_{k+1}}(x_k + d_k^c, y_k + d_k^i)$ **then** *update r_k by*

$$r_k := \frac{P_{\sigma_{k+1}}(x_k, y_k) - P_{\sigma_{k+1}}(x_k + d_k^c + \hat{d}_k^c, y_k + d_k^i)}{\Phi_k(0) - \Phi_k(d_k)} \tag{16}$$

   and replace d_k by $d_k + \hat{d}_k$.
6. *Update trust region radii by*

$$\Delta_{k+1}^c := \begin{cases} \min[\|d_k\|_\infty/2, \ \Delta_k^c], & \text{if} \quad 0.25 > r_k, \\ \Delta_k^c, & \text{if} \quad 0.25 \leq r_k \leq 0.75, \\ \max[2\|d_k\|_\infty, \ \Delta_k^c], & \text{if} \quad 0.75 < r_k, \end{cases} \tag{17}$$

and

$$\Delta_{k+1}^i := \begin{cases} \lfloor \|d_k^i\|_\infty/2 \rfloor, & \text{if} \quad 0.25 > r_k, \\ \Delta_k^i, & \text{if} \quad 0.25 \le r_k \le 0.75, \\ \max[2\|d_k^i\|_\infty,\ \Delta_k^i,\ 1], & \text{if} \quad 0.75 < r_k. \end{cases} \tag{18}$$

7. **if** $r_k \le 0$, **then** *set* $(x_{k+1}, y_{k+1}) := (x_k, y_k)$, $C_{k+1} := C_k$, $k := k+1$, *and* **goto** *Step 2.*
   else *set* $(x_{k+1}, y_{k+1}) := (x_k, y_k) + d_k$.
8. *Evaluate* $\nabla_x f(x_{k+1}, y_{k+1})$ *and* $\nabla_x g(x_{k+1}, y_{k+1})$ *with respect to continuous variables.*
   Approximate $\nabla_y f(x_{k+1}, y_{k+1})$ *and* $\nabla_y g(x_{k+1}, y_{k+1})$ *with respect to integer variables using Procedure 2.1 and obtain* $(x_{k+1}^{bn}, y_{k+1}^{bn})$ *and* f_{k+1}^{bn}.
   if $\|g(x_{k+1}^{bn}, y_{k+1}^{bn})^-\|_\infty \le \epsilon$ **and** $f(x_{k+1}^{bn}, y_{k+1}^{bn}) < f^*$ **then** *update* $f^* := f(x_{k+1}^{bn}, y_{k+1}^{bn})$ *and* $(x^*, y^*) := (x_{k+1}^{bn}, y_{k+1}^{bn})$.
9. *Solve the bound-constrained least squares problem (13) to get multiplier approximations* $u_k \in \mathbb{R}^m$, $v_k^l, v_k^u \in \mathbb{R}^{n_c}$, *and* $w_k^l, w_k^u \in \mathbb{R}^{n_i}$. *Generate* C_{k+1} *by the BFGS update formula, let* $k := k+1$, *and* **goto** *Step 2.*
10. **if** $\|g(x_k, y_k)^-\|_\infty \le \epsilon$ **and** $f^* \ge f(x_k, y_k)$ **then** *set* $f^* := f(x_k, y_k)$, $(x^*, y^*) := (x_k, y_k)$, *return the best solution* f^* *and* (x^*, y^*) *and* **stop**.
    if $\|g(x_k, y_k)^-\|_\infty > \epsilon$ **and** $f^\star = \infty$ **then** *report that the problem might be infeasible and* **stop**.
    otherwise *set* $(x_{k+1}, y_{k+1}) := (x^*, y^*)$, *evaluate function values* $f(x_{k+1}, y_{k+1})$, $g(x_{k+1}, y_{k+1})$ *and gradients* $\nabla_x f(x_{k+1}, y_{k+1})$, $\nabla_x g(x_{k+1}, y_{k+1})$ *for continuous variables. Set* $k := k+1$ *and* **goto** *Step 1.*

Algorithm 2.1 is implemented and the code is called MISQP. However, MISQP contains additional heuristics to improve the robustness of the described algorithm. In the remainder of this section, we describe them in more detail.

In mixed-integer nonlinear programming, we do not know local optimality conditions comparable to the KKT conditions in continuous optimization. Convexity of the mixed-integer program (1) is not assumed, and we stop the algorithm as soon as sufficient reduction of the merit function (10) is no longer possible.

Note that in Step 2 the penalty parameter σ_k might grow arbitrarily large, in particular if the underlying mixed-integer program is infeasible. If σ_k is greater than a threshold $\bar{\sigma}$ and the predicted reduction of the merit function is small, the algorithm is supposed to terminate at an infeasible stationary point, see Yuan [58]. The parameter $\bar{\sigma}$ should be set to a sufficiently large value, e.g., 10^{20}.

In Step 6, the trust region update for the continuous trust-region radius Δ_k^c uses the norm of the complete step d_k including the integer part, see (17), to guarantee that $\Delta_k^c > 0$. Expression $\lfloor \|d_k^i\|_\infty/2 \rfloor$ in (18) denotes the largest integer value smaller than $\|d_k^i\|_\infty/2$. Thus, the trust-region radius Δ_k^i is integer for all k.

Procedure 2.1 in Step 1 and Step 8 is not executed if exact gradients for integer variables are provided or are approximated externally. The corresponding changes of Algorithm 2.1 are straightforward. $f^\star$ remains unchanged and restarts are not performed in Step 10.

We allow a non-monotone decrease of penalty function values $P_{\sigma_k}(x_k, y_k)$. The idea of accepting new iterates which eventually increase the penalty function, is investigated in the context of trust region algorithms by several authors, see, e.g., Toint [55], Chen et al. [13], and Deng et al. [15]. We choose an integer constant $M > 0$ and compare an actual penalty function value always with the highest one obtained during the previous M successful iterations. We call an iteration k a successful iteration, if d_k is used to update an iterate, i.e., if $(x_{k+1}, y_{k+1}) = (x_k, y_k) + d_k$. Let $\overline{K}_k$ be the set of iterates that correspond to the last M successful iterations. Whenever $(x_{k+1}, y_{k+1}) = (x_k, y_k) + d_k$, the iterate (x_{k+1}, y_{k+1}) substitutes the element with the lowest iteration index in set $\overline{K}_{k+1}$. The alternative formulation of Step 4 is

4. Evaluate new function values $f(x_k + d_k^c, y_k + d_k^i)$, $g_j(x_k + d_k^c, y_k + d_k^i)$, $j = 1, \ldots, m$, and compute the quotient of the actual and the predicted improvements

$$r_k := \frac{P_{\sigma_{k+1}}(x_{l_k}, y_{l_k}) - P_{\sigma_{k+1}}(x_k + d_k^c, y_k + d_k^i)}{\Phi_k(0) - \Phi_k(d_k)}, \tag{19}$$

where $P_{\sigma_{k+1}}(x_{l_k}, y_{l_k}) := \max_{(x,y) \in \overline{K}_k} P_{\sigma_{k+1}}(x, y)$.

Although we introduce this non-monotone reduction condition on the penalty function, we still apply the SOC steps in Step 5. Our numerical tests indicate that this strategy improves efficiency in some situations. Moreover, we retain superlinear convergence when applying the mixed-integer algorithm to solve continuous problems.

The standard procedure for updating C_k is a modified BFGS update formula, as outlined above, which guarantees positive definite matrices. However, we modify C_k if the conditions of Step 2 are satisfied and if Step 10 is reached, to get

$$\|C_k\|_\infty \leq \frac{1}{n_c + n_i} \|\nabla f(x_k, y_k)\|_\infty. \tag{20}$$

All entries in C_k are scaled by the same value. The scaling strategy is also motivated by the fact that large values in C_k result in void integer steps. Numerical tests show that this heuristic scaling strategy (20) improves the robustness of the algorithm significantly.

Numerical tests indicate that restarts are highly profitable. If the termination criterion in Step 10 is fulfilled, the relaxed quadratic program (11) is solved subject to a continuous vector $\tilde{d}_k^i \in \mathbb{R}^{n_i}$. The integer variable values of the solution $\tilde{d}_k = (\tilde{d}_k^c, \tilde{d}_k^i)$ are rounded to get $\tilde{d}_k^i \in \mathbb{Z}^{n_i}$ and we restart Algorithm 2.1 from the obtained new iterate.

Finally, we apply different trust region norms for the integer variables. For binary variables, an L_1 norm is applied and the trust region constraint is reformulated by one linear inequality constraint. The aim is to obtain more freedom in restricting the search step in the binary space. The update rules of the trust region parameters are the same for both types of variables.

3 A linear outer approximation algorithm combined with SQP and trust region stabilization

In this section, we introduce two algorithms combining linear outer approximations and the SQP-type mixed-integer nonlinear programming algorithm described in the previous section. To motivate our method, we briefly present the theoretical background of the method of linear outer approximation described by Fletcher and Leyffer [24] and Duran and Grossmann [17], see also Quesada and Grossmann [44] for an alternative approach.

To simplify the notation and the analysis of this section, we assume that there are no equality constraints, i.e., $m_e = 0$. The mixed-integer nonlinear program (1) is then written in the form

$$\begin{aligned} &\underset{x \in X,\, y \in Y}{\text{minimize}} && f(x, y) \\ &\text{subject to} && g(x, y) \geq 0, \end{aligned} \tag{21}$$

where the corresponding continuous relaxation is

$$\begin{aligned} &\underset{x \in X,\, y \in Y_{\mathbb{R}}}{\text{minimize}} && f(x, y) \\ &\text{subject to} && g(x, y) \geq 0. \end{aligned} \tag{22}$$

For a given $y \in Y$, we denote the nonlinear program

$$\begin{aligned} &\underset{x \in X}{\text{minimize}} && f(x, y) \\ &\text{subject to} && g(x, y) \geq 0, \end{aligned} \tag{23}$$

by NLP(y) and its solution by $x(y)$.

In the following, we denote the set of integer values leading to feasible nonlinear subproblems by

$$T := \{y \in Y : \text{ NLP}(y) \text{ feasible}\}. \tag{24}$$

Analogously, we denote the set of integer values y leading to infeasible subproblems by

$$S := \{y \in Y : \text{ NLP}(y) \text{ infeasible}\}. \tag{25}$$

Note that $Y = T \cup S$.

Consider now $y \in S$. Let $J(y)$ be the set of all indices from $\{1, \ldots, m\}$ such that there exists an $x \in X$ with $g_j(x, y) \geq 0$ for all $j \in J(y)$. With $J^{\perp}(y) := \{1, ..., m\} \backslash J(y)$, we obtain a feasibility problem F(y) for any fixed $y \in Y$,

$$\begin{aligned} &\underset{x \in X}{\text{minimize}} && -\sum_{j \in J^{\perp}(y)} w_j g_j(x, y)^- \\ &\text{subject to} && g_j(x, y) \geq 0, \;\; j \in J(y), \end{aligned} \tag{26}$$

where w_j are appropriate nonnegative weights, which are not simultaneously equal to zero and where $g_j(x, y)^-$ is defined by (8). We denote the solution of F(y) by $x^F(y)$ for $y \in S$, and obtain additional constraints of the form

$$g\left(x^F(y), y\right)+\nabla g\left(x^F(y), y\right)^T \begin{pmatrix} x - x^F(y) \\ z - y \end{pmatrix} \geq 0, \quad \text{for all } y \in S, \quad z \in Y. \tag{27}$$

To be able to prove convergence, Fletcher and Leyffer [24] or Duran and Grossmann [17], e.g., assumed that the relaxed program (22) is convex, i.e., that $f(x, y)$ is convex and that $g(x, y)$ is concave over X and $Y_{\mathbb{R}}$. Moreover, the linear independency constraint qualification (LICQ) is supposed to hold at a solution $x(y)$ of the nonlinear program NLP(y), $y \in T$, and at a solution $x^F(y)$ of the feasibility problem $F(y)$, $y \in S$.

Linear outer approximation algorithms approach a solution of (21) by alternately solving continuous nonlinear programs NLP(y), see (23), with fixed integer variables $y \in Y$, followed by a mixed-integer linear program called *master program*. The idea is to decouple the continuous nonlinear and the integer optimization parts and to apply efficient nonlinear programming and linear mixed-integer programming solvers separately. The master program is a linear relaxation of (21), where the number of linearized constraints grows successively. Since Y is finite, a linear outer approximation method terminates after finitely many steps. Each solution of a master program provides a lower bound for (21).

The methods of Fletcher and Leyffer [24] and Duran and Grossmann [17] are based on the idea that (21) is equivalent to

$$\underset{y \in T}{\text{minimize}} \quad f(x(y), y). \tag{28}$$

After introducing an artificial variable η for minimizing the maximum linearized objective function value over T, the outer approximation *master problem* is given in the form

$$\begin{aligned} \underset{x \in X,\, z \in Y,\, \eta \in \mathbb{R}}{\text{minimize}} \quad & \eta \\ \text{subject to} \quad & f(x(y), y) + \nabla f(x(y), y)^T \begin{pmatrix} x - x(y) \\ z - y \end{pmatrix} \leq \eta, \qquad \text{for all } y \in T, \\ & g(x(y), y) + \nabla g(x(y), y)^T \begin{pmatrix} x - x(y) \\ z - y \end{pmatrix} \geq 0, \qquad \text{for all } y \in T, \\ & g(x^F(y), y) + \nabla g(x^F(y), y)^T \begin{pmatrix} x - x^F(y) \\ z - y \end{pmatrix} \geq 0, \quad \text{for all } y \in S. \end{aligned} \tag{29}$$

Since the sets S defined by (25) and T defined by (24) are not known a priori, they are approximated by

$$\begin{aligned} T_k &:= \{y_i : i \leq k, \ \text{NLP}(y_i) \text{ is feasible}\}, \\ S_k &:= \{y_j : j \leq k, \ \text{NLP}(y_j) \text{ is infeasible}\} \end{aligned} \tag{30}$$

in the k-th step of an outer approximation algorithm, by which $y_0, y_1, \ldots, y_k \in Y$ were computed. The corresponding relaxed linear master program, where we replace z by y to simplify the subsequent notation, is

$$\begin{aligned}
&\underset{x \in X,\, y \in Y,\, \eta \in \mathbb{R}}{\text{minimize}} \quad \eta \\
&\text{subject to} \quad f(x(y_i), y_i) + \nabla f(x(y_i), y_i)^T \begin{pmatrix} x - x(y_i) \\ y - y_i \end{pmatrix} \leq \eta, \qquad \text{for all } i \in T_k, \\
&\qquad\qquad g(x(y_i), y_i) + \nabla g(x(y_i), y_i)^T \begin{pmatrix} x - x(y_i) \\ y - y_i \end{pmatrix} \geq 0, \qquad \text{for all } i \in T_k, \\
&\qquad\qquad g(x(y_j), y_j) + \nabla g(x(y_j), y_j)^T \begin{pmatrix} x - x^F(y_j) \\ y - y_j \end{pmatrix} \geq 0, \quad \text{for all } j \in S_k.
\end{aligned} \tag{31}$$

The methods of Fletcher and Leyffer [24] and Grossmann [28] require exact partial derivatives with respect to the continuous and integer variables in (31). If exact gradients are not available and if they have to be approximated numerically, see e.g. (4), the resulting approximations might not underestimate or overestimate, respectively, objective function and constraints. Moreover, for non-convex problems linear outer approximations might cut off the optimal solution and additional safeguards must be attached.

In the remainder of this section, we propose two algorithms, which combine the well-known linear outer approximation techniques with Algorithm 2.1. Both are applicable also to nonconvex mixed-integer programs (21).

Our first approach is to add a linear outer approximation master program (31) to the trust region algorithm of Section 2. To prevent cycling, we alternate between mixed-integer nonlinear programming subproblems and continuous nonlinear programs with fixed integer variables, as done in available linear outer approximation algorithms, i.e., between NLP(y) or F(y), respectively. Furthermore, we introduce additional constraints cutting off previous solutions. Note that, if one knew that the given mixed-integer nonlinear program is convex, one should better apply a linearized objective function cut.

The first step of the modified algorithm is to apply Algorithm 2.1 to solve the mixed-integer nonlinear program (21), with some additional constraints to prevent cycling,

$$\begin{aligned}
&\underset{x \in X,\, y \in Y}{\text{minimize}} \quad f(x, y) \\
&\text{subject to} \quad g(x, y) \geq 0, \\
&\qquad\qquad \|y - y_l\|_2^2 \geq 1, \quad \text{for all } 1 \leq l < k.
\end{aligned} \tag{32}$$

We denote this problem by MINLP(k). In the first iteration with $k = 0$, Algorithm 2.1 is applied to the original problem without artificial constraints. If (32) turns out to be infeasible, as, e.g., measured by too large penalty parameters, the feasibility problem F(y_k) is solved. Subsequently, the master program (31) is set up and the solution of (31) provides a new starting point for (32) in the next iterate.

The additional constraints

$$\|y - y_l\|_2^2 \geq 1, \quad \text{for all } 1 \leq l < k \tag{33}$$

lead to nonconvex and thus more difficult subproblems. However, numerical tests show that they improve the overall performance and robustness of the algorithm drastically. The L_2-norm is chosen instead of the L_1-norm to get differentiable constraints.

The modified algorithm is summarized as follows.

Algorithm 3.1 *Let $x_0 \in X$ and $y_0 \in Y$ be given starting values, $\epsilon > 0$ a termination tolerance, and $R > 1$ a cycling rate and $\bar{\sigma} > 0$ an upper bound for the penalty parameter in Algorithm 2.1. Set $f^* := \infty$, $(x^*, y^*) := (x_0, y_0)$, and $k := 0$.*

1. **if** $k = 0$ *or if k is a multiple of R,* **then goto** *Step 2*

   else *solve NLP(y_k), or, if NLP(y_k) is infeasible, $F(y_k)$ subject to the stopping tolerance ϵ. Denote the solution by $\overline{x}_k$, let $\overline{y}_k := y_k$ and* **goto** *Step 3.*
2. *Solve MINLP(k) (32) by Algorithm 2.1 subject to stopping tolerance ϵ and denote the solution by $\overline{x}_k$ and $\overline{y}_k$.*

   if *MINLP(k) cannot be solved, i.e., stops at an infeasible point, or if $\sigma_k > \bar{\sigma}$ in Step 4 of Algorithm 2.1,* **then** *solve $F(y_k)$ and denote the solution again by $\overline{x}_k$.*
3. *Evaluate new function and derivative values, either analytically or by Procedure 2.1, at $(\overline{x}_k, \overline{y}_k)$.*

   if $\|g\left(\overline{x}_k, \overline{y}_k\right)^- \|_\infty \leq \epsilon$ *and* $f\left(\overline{x}_k, \overline{y}_k\right) < f^*$, **then** *set* $f^* := f\left(\overline{x}_k, \overline{y}_k\right)$ *and* $(x^*, y^*) := (\overline{x}_k, \overline{y}_k)$.
4. *Add new constraints to the linear master program (31) formulated at $(\overline{x}_k, \overline{y}_k)$, and solve (31). Denote the solution by $(x_{k+1}, y_{k+1}, \eta_k)$.*

   if $\eta_k \geq f^* - \epsilon$, **then stop** *and return the best solution f^* and (x^*, y^*).*

   else *set $k := k + 1$ and* **goto** *Step 1.*

Note, that the numerical solution of the mixed-integer nonlinear program (32) requires substantial additional computational overhead. Thus, R is a constant parameter to reduce the number of solutions of (32). In Step 1 of Algorithm 3.1, we solve continuous nonlinear optimization problems with given integer variables. Step 2 offers more freedom in computing new search steps, since possible changes in continuous and integer variables are simultaneously taken into account.

An alternative idea is to add linear outer approximations directly to the internal cycle of the SQP-type Algorithm 2.1. Our goal is to modify the linear outer approximation method as outlined before, by allowing a subsequent adaption of integer variables also outside of the master problem (29). First, we define a condition under which a mixed-integer search step obtained from (11) is acceptable compared to a continuous step $\tilde{d}_k^c$ obtained by solving (12).

Definition 3.1 Denote the mixed-integer solution of the quadratic program (11) at $(x_k, y_k) \in X \times Y$ by $(d_k^c, d_k^i) \in \mathbb{R}^{n_c} \times \mathbb{Z}^{n_i}$ and let $\tilde{d}_k^c \in \mathbb{R}^{n_c}$ be the solution of the continuous quadratic program (12) for a fixed y_k. If

$$P_{\sigma_k}(x_k + d_k^c, y_k + d_k^i) < P_{\sigma_k}(x_k + \tilde{d}_k^c, y_k), \tag{34}$$

holds, then (d_k^c, d_k^i) is an *improved mixed-integer search step* subject to $\tilde{d}_k^c$, where P_{σ_k} denotes the L_∞-penalty function (5) with penalty parameter σ_k.

A change in the integer variables according to Definition 3.1 is considered as an improved direction, if the search step is more profitable than the continuous step $\tilde{d}_k^c$ with respect to the L_∞-penalty function (5).

To avoid cycling, we record the previous solutions of the master problem (31) and skip the computation of mixed-integer search-directions, i.e., we do not solve MIQP (11), whenever cycling is detected. As soon as the master problem yields an unexplored integer value, mixed-integer search directions are again computed.

Note that the best current solution f^*, (x^*, y^*) of the subsequent algorithm is updated, whenever the conditions

$$\|g(x_k, y_k)^-\|_\infty \leq \epsilon \tag{35}$$

$$f(x_k, y_k) < f^* \tag{36}$$

are satisfied for function values evaluated at (x_k, y_k) subject to a feasibility tolerance $\epsilon > 0$.

Algorithm 3.2 *Let $\Delta_0^c > 0$, $\Delta_0^i \geq 1$, $\sigma_0 > 0$, and $\epsilon > 0$ be given constants, choose starting values $x_0 \in X$, $y_0 \in Y$ and a positive definite matrix $C_0 \in \mathbb{R}^{(n_c+n_i)\times(n_c+n_i)}$. Evaluate function values $f(x_0, y_0)$, $g(x_0, y_0)$ and derivatives $\nabla f(x_0, y_0)$, $\nabla g(x_0, y_0)$, either analytically or by Procedure 2.1. Set $f^* := \infty$, $(x^*, y^*) = (x_0, y_0)$ and $k := 0$.*

1. *Solve the continuous quadratic program (12) and denote its solution by $\tilde{d}_k^c$. Evaluate new function values $f(x_k + \tilde{d}_k^c, y_k)$ and $g_j(x_k + \tilde{d}_k^c, y_k)$, $j = 1, \ldots, m$, and update the best solution values, if conditions (35) and (36) are satisfied. Update the penalty parameter σ_{k+1} according to Step 3 of Algorithm 2.1 with respect to σ_k, Δ_k^c and $g(x_k + \tilde{d}_k^c, y_k)$, compute r_k^c according to (15), and update the trust region radius Δ_{k+1}^c by (17).*
2. *Solve the mixed-integer quadratic program (11) and denote its solution by $d_k^c \in \mathbb{R}^{n_c}$ and $d_k^i \in \mathbb{Z}^{n_i}$. Evaluate new function values $f(x_k + d_k^c, y_k + d_k^i)$ and $g_j(x_k + d_k^c, y_k + d_k^i)$, $j = 1, \ldots, m$. Update the best solution, if (35) and (36) are satisfied. Compute r_k^i according to (15), and update the trust region radius Δ_{k+1}^i by (18).*
3. **if** *(d_k^c, d_k^i) is an improved mixed-integer search direction,* **then** *let $d_k := (d_k^c, d_k^i)$, $r_k := r_k^i$*
   else *let $d_k := (\tilde{d}_k^c, 0)$ and $r_k := r_k^c$.*

   if *x_k is a stationary point of NLP(y_k) subject to the termination tolerance ϵ, either feasible or non-feasible,* **then goto** *Step 6.*
4. **if** *$r_k \leq 0$,* **then** *set $(x_{k+1}, y_{k+1}) := (x_k, y_k)$, $C_{k+1} := C_k$, $k := k + 1$ and* **goto** *Step 1.*

   else *set $(x_{k+1}, y_{k+1}) := (x_k, y_k) + d_k$.*
5. *Evaluate gradients $\nabla f(x_{k+1}, y_{k+1})$ and $\nabla g(x_{k+1}, y_{k+1})$ either analytically or by Procedure 2.1. Solve the bound-constrained least squares problem (13) to get*

multiplier approximations. Generate C_{k+1} *by the BFGS update formula, set* $k := k + 1$ *and* **goto** *Step* 1.

6. *Update the linear master program* (31) *by adding new constraints subject to* (x_k, y_k) *and denote the solution by* $(x_{k+1}, y_{k+1}, \eta_k)$.

   if $\eta_k \geq f^* - \epsilon$, **then** **stop** *and return the best solution* f^* *and* (x^*, y^*).

   else *evaluate new function values* $f(x_{k+1}, y_{k+1})$, $g_j(x_{k+1}, y_{k+1})$ *and new gradient values* $\nabla f(x_{k+1}, y_{k+1})$, $\nabla g_j(x_{k+1}, y_{k+1})$, $j = 1, \ldots, m$, *either analytically or by Procedure* 2.1 *and update best solution, if conditions* (35) *and* (36) *are satisfied. Set* $k := k + 1$ *and* **goto** *Step* 1.

4 Comparative numerical results

4.1 Test environment

Our codes are part of a modular toolbox, which allows to switch easily from one solver to another. They are implemented in thread-safe Fortran as close to F77 as possible, to be able to convert the codes to C directly.

We intend to evaluate the numerical performance of four solvers, some of them executed with alternative parameter settings, on a set of 100 academic mixed-integer and a set of 55 mixed-binary test problems provided by our industrial cooperation partner Shell SIEP Rijswijk. In both cases, we have nonlinear and also nonconvex objective functions and nonconvex feasible domains, especially also nonlinear equality constraints.

Evaluating statistical comparative scores by mean values for a series of test problems and different computer codes is often misleading. It might happen that the less reliable codes do not solve the more complex test problems successfully, but the more advanced ones solve them with additional numerical efforts, say calculation time or number of function calls. A direct evaluation of mean values over successful test runs would thus penalize the more reliable algorithms.

A more realistic possibility is to compute mean values of the criteria we are interested in, and to compare the codes pairwise over sets of test examples, which are successfully solved by the two codes. We then get a reciprocal $n_{code} \times n_{code}$ matrix, where n_{code} is the number of codes under consideration. The largest eigenvalue of this matrix is positive and we compute its normalized eigenvector from where we retrieve priority scores. The idea is known under the name *priority theory*, see Saaty [46] or the appendix, and has been used by Schittkowski [48] and Hock and Schittkowski [33] for comparing 27 optimization codes. In a final step, we normalize the eigenvectors so that the smallest coefficient gets the value one.

The following codes are implemented based on the algorithms outlined in the previous sections, and are tested with different parameter settings:

MISQP [20] - Mixed-integer SQP-based trust region method, i.e., Algorithm 2.1, partial derivatives approximated by Procedure 2.1

MISQP/bmod [20]	- Same as MISQP, but quasi-Newton updates not scaled in order to satisfy (20)
MISQP/fwd [20]	- Same as MISQP, but integer variables treated as relaxable, i.e., partial derivatives with respect to integer variables computed by difference formulae analogously to the procedure used for continuous variables
MISQP/rst0 [20]	- Same as MISQP, but no restarts
MISQPOA [36]	- Implementation of Algorithm 3.1, i.e., additional stabilization by outer approximations, successive solution of mixed-integer nonlinear problems by MISQP. Every 6th subproblem is formulated as a mixed-integer nonlinear program (32)
MISQPN [19]	- Implementation of Algorithm 3.2, i.e., by an SQP-based outer approximation method, successive solution of mixed-integer quadratic programs extended by linear outer approximation constraints
MINLPB4 [35]	- Branch-and-bound method based on MISQP with branching subject to integer and binary variables, i.e., generation of a sequence of nonlinear continuous programs solved by MISQP

For solving continuous quadratic programming problems, we use the code QL of Schittkowski [49], which is based on an implementation of Powell [43]. The underlying primal-dual method of Goldfarb and Idnani [27] is particularly useful for designing a branch-and-cut algorithm for mixed-integer quadratic programs, e.g., by exploiting dual information for early branching. The corresponding code is called MIQL, see Lehmann and Schittkowski [34], and is used in all situations where we have to solve mixed-integer quadratic programs.

Note that all continuous nonlinear programs are solved by MISQP by setting the number of integer variables to zero. The algorithm then behaves like an SQP algorithm with trust region stabilization and quasi-Newton updates, see Exler et al. [20] for details.

The quadratic programming code MIQL mentioned above and the nonlinear mixed-integer programming code MINLPB4 call the branch-and-bound subroutine BFOUR, see Lehmann et al. [37], where several branching and node selection strategies are implemented.

For executing the above mentioned optimization codes, we apply default parameter settings and tolerances, see the corresponding user guides for details, with termination tolerance 10^{-6}. Maximum number of iterations is 2,000, and the number of branch-and-bound nodes is bounded by 10,000. We backup the last 10 iterations for applying the non-monotone trust region updates.

All test examples are provided with the best objective function value $f^\star$ we know, either obtained from analytical solutions, literature, or extensive numerical testing. Derivatives with respect to continuous variables are always approximated by forward differences subject to a small tolerance (10^{-6}), whereas integer derivatives are replaced by descent directions, see Procedure 2.1. For binary variables or for variables at a bound, a forward or backward difference formula is applied, respectively.

Exceptions are the codes MISQP/fwd and MINLPB4. In both cases the derivatives with respect to the integer variables are approximated by the same procedure used for the continuous variables.

The Fortran codes are compiled by the Intel Visual Fortran Compiler 10.1 under Windows 7 and executed on an Intel Core(TM)2 i7 64 bit processor with 3.16 GHz and 8 GB RAM.

First we need a criterion to decide whether the result of a test run can be considered as a successful return or not. Let $\epsilon_t > 0$ be a tolerance for defining the relative accuracy, and (x_k, y_k) the final iterate of a test run. If $f^\star = 0$, as in some of the academic test instances, we add the value one to the objective function. We call (x_k, y_k) a *successful* solution, 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

$$f(x_k, y_k) - f^\star < \epsilon_t |f^\star| \tag{37}$$

and $\|g(x_k, y_k)^-\|_\infty < \epsilon_t^2$, where $g(x_k, y_k)^-$ represents the vector of constraint violations. We take into account that a code returns a solution with a better function value than the best known one, subject to the error tolerance of the allowed constraint violation. The tolerance is smaller for measuring the constraint violation than for the error in the objective function, since we apply a relative measure in the latter case, whereas constraint functions of our test problems are often badly scaled.

Moreover, we would like to distinguish between feasible, but non-global solutions, successful solutions as defined above, and false terminations. We call the return of a test run, say x_k and y_k, an *acceptable* solution, if the internal termination conditions are satisfied subject to a reasonably small tolerance $\epsilon = 10^{-6}$ and if instead of (37)

$$f(x_k, y_k) - f^\star \geq \epsilon_t |f^\star| \tag{38}$$

holds. For our numerical tests, we use $\epsilon_t = 0.01$.

Note again that our main paradigm is to proceed from non-relaxable integer variables and corresponding descent directions, which could be considered as a very crude numerical approximations of partial derivatives by forward differences. To be as close to complex practical engineering applications as possible, we apply a forward difference formula for approximating partial derivatives subject to the continuous variables.

We use the subsequent criteria to compare the robustness and efficiency of our codes on two sets of test problems:

- n_{succ} - number of successful test runs according to above definition,
- n_{acc} - number of acceptable, i.e., of non-global feasible solutions, see above definition,
- Δ_{err} - average relative deviation of computed solution from the best known known one, $(f(x_k, y_k) - f^\star)/|f^\star|$, taken over all acceptable solutions,
- n_{err} - number of test runs terminated by an error message,
- n_{func} - average number of equivalent function calls including function calls used for computing a descent direction or gradient approximations, evaluated over all successful test runs, where one function call

consists of one evaluation of the objective function and all constraint functions at a given x and y,

p_{func} - relative priority of equivalent function calls including function calls used for gradient approximations,

$time$ - average execution times in seconds, evaluated over all successful test runs,

p_{time} - relative priority of execution times.

For an alternative way to present comparative numerical results, Dolan and Moré [16] developed performance profiles which are frequently applied in comparative computational studies. We present them for our main measure, namely the number of equivalent function calls.

4.2 Academic test problems

First, we evaluate the performance of the presented solvers on a test set of 100 academic test examples published in Schittkowski [50]. Each test problem comes with a function value which has been found in the literature or which has been obtained by extensive testing over several years. They are believed to represent the optimal solution, at least we did not find better results by our test runs. There are at most 23 continuous, 100 integer, and 24 binary variables. Moreover, there are up to 17 equality constraints and the total number of constraints is at most 75. 65 test problems are taken from the GAMS MINLPLib, see Bussieck et al. [12].

Table 1 shows numerical results obtained for the mixed-integer trust region method MISQP, the branch-and-bound solver MINLPB4, and the outer approximation solvers MISQPOA and MISQPN, see the previous subsection for more details.

In a few cases, the codes are unable to find a feasible solution and an error message is generated. In a couple of other situations, the codes are unable to improve a current iterate and report that a feasible solution is obtained, but which is not the global optimum.

Table 1 shows that scaling the quasi-Newton matrix is extremely important to improve the robustness of our implementation. Otherwise, the number of successfully solved test runs decreases by almost 20 per cent. The significant reduction of the

Table 1 Performance results for a set of 100 academic test problems

Code	n_{succ}	n_{acc}	Δ_{err}	n_{err}	p_{func}	n_{func}	p_{time}	Time
MISQP	89	11	0.84	0	1.3	500	3.4	0.39
MISQP/bmod	71	29	3.48	0	1.0	340	1.0	0.20
MISQP/fwd	81	19	0.64	0	1.9	396	4.0	0.11
MISQP/rst0	69	30	0.36	1	1.0	241	2.3	0.14
MISQPOA	91	9	0.31	0	3.1	1,093	10.7	0.65
MISQPN	74	24	1.17	2	4.4	1,139	8.5	0.17
MINLPB4	88	8	1.00	4	51.6	218,881	1.8	4.11

function calls indicates that the solver terminates too early. The results of MISQP/rst0 emphasize the importance of internal restarts. Although the MISQP variant with nearly exact partial derivatives, MISQP/fwd, needs less function evaluations, it is on the other hand less reliable. There is no evidence to conclude that the one or the other variant is better.

MISQPOA calls MISQP within an outer approximation framework. Thus, the obtained solution is at least as good as the one found by MISQP. The added safeguards result in a higher number of function evaluations, but more problems can be solved successfully.

The new outer approximation algorithm implemented under the name MISQPN, is still an experimental implementation and less reliable than any of the other codes. The average number of function calls of MISQPN is more than twice higher than that of MISQP, and the priority level indicates that n_{func} is even about four times higher when compared over the set of test runs which terminated successfully for both codes. One possible explanation is the effect of a heuristic scaling of the BFGS updates which is not implemented for MISQPN. Thus, one should better compare MISQPN to MISQP/bmod. Both codes solve almost the same number of test problems, but MISQP/bmod requires less function evaluations.

As expected, the branch-and-bound solver MINLPB4 is much less efficient than the other solvers, since a large number of continuous nonlinear optimization problems must be solved. To prevent exhaustive computation times, MINLPB4 is provided with the information that all test problems are supposed to be convex. Thus, the number of successful test runs is comparable to those of MISQP indicating again that at least ten test problems of our collection are nonconvex.

Performance profiles for the number of function evaluations, n_{func}, are presented in Fig. 1. For each solver the profile shows the percentage of test problems the code solved successfully without exceeding an upper bound on the number of function evaluations. The upper bound is a multiple of the number of function calls the best solver on an instance needed. The factor is given along the abscissa. For example, MISQP solves 80 % of the problems with at most four times the number of function evaluations compared to the best solvers. According to the performance profiles MISQP/rst0 is the most efficient solver. In 45 % of the problems MISQP/rst0 needs the fewest number of function evaluations to solve the problem. MISQP/fwd and MISQP/bmod follow. If all heuristics are activated then MISQP becomes more robust. The performance profiles can be interpreted similarly to the results presented in Table 1.

4.3 Test problems from petroleum engineering

A large variety of applications of mixed-integer nonlinear programming is found in the petroleum industry. We select two classes of problems known as well relinking and gas lift problems for our numerical tests, which differ by their dimensions and data and which are collected in another set of 55 test examples. The case studies have been provided by Shell SIEP Rijswijk together with optimal solutions found by extensive numerical tests with global optimization solvers. These applications are based

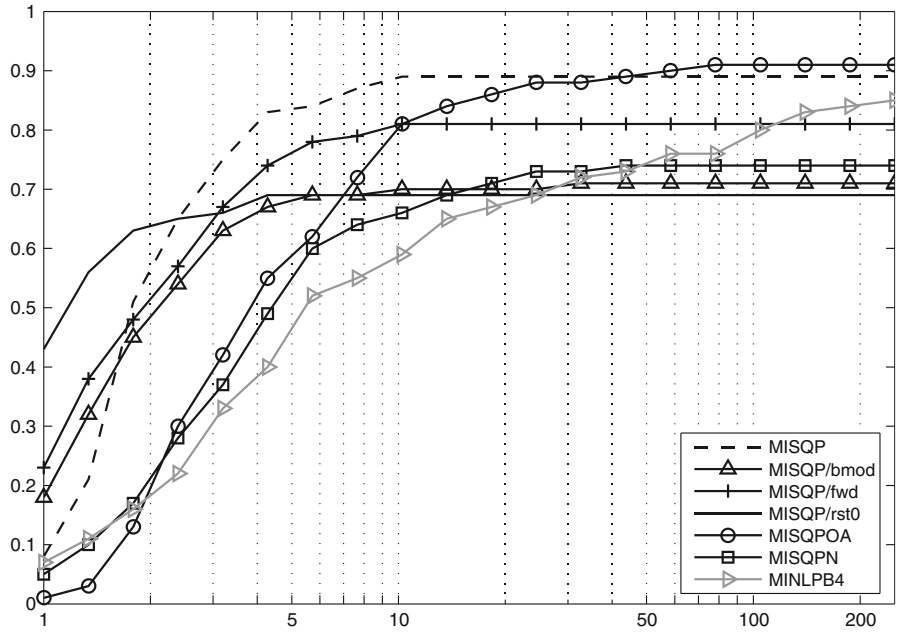

Fig. 1 Performance profiles for academic test problems

on complex simulators, but simplified algebraic description are provided reproducing typical problem characteristics.

To give a typical example, we introduce a simple well relinking model, where the total flow in a given network is to be maximized. The network consists of a number of source nodes and some sink nodes. The flow from each source node is to be directed to exactly one sink node, and the total capacity at the sinks is limited in terms of pressure and flow. A source node has a special pressure-flow characteristic and the total flow within the network is bounded.

Let us assume that there are m_s sinks and n_s sources, and that we want to maximize the total flow

$$\sum_{i=1}^{n_s} x_i$$

under so-called split-factor constraints, i.e., a set of switching conditions for each source i, $i = 1, \ldots, n_s$, of the form

$$\sum_{j=1}^{m_s} y_j^i = 1.$$

Moreover, we have pressure constraints at source i, $i = 1, \ldots, n_s$,

$$\sum_{j=1}^{m_s} c_j^i y_j^i \leq a_i - b_i x_i,$$

Table 2 Performance results for a set of 55 well relinking and gas lift test problems

Code	n_{succ}	n_{acc}	Δ_{err}	n_{err}	p_{func}	n_{func}	p_{time}	Time
MISQP	50	5	0.041	0	2.7	1,964	3.6	0.91
MISQP/bmod	49	6	0.044	0	2.0	1,430	2.0	0.50
MISQP/fwd	45	10	0.041	0	2.8	1,901	3.5	0.88
MISQP/rst0	37	17	0.104	1	1.0	630	1.0	0.14
MISQPOA	52	3	0.033	0	23.5	17,786	31.5	7.78
MISQPN	33	9	0.087	13	10.4	5,331	11.6	1.29
MINLPB4	55	0	0.0	0	204.2	154,898	2.0	0.45

and upper bounds Q_j for mass rates at the sinks, $j = 1, \ldots, m_s$,

$$\sum_{i=1}^{n_s} x_i y_j^i \leq Q_j,$$

with appropriate positive constants c_j^i, Q_j, $j = 1, \ldots, m_s$, and a_i, b_i, $i = 1, \ldots, n_s$. The well relinking test examples are defined by their dimensions $m_s = 3$ and $n_s = 3$, $n_s = 6$, or $n_s = 9$, and the constants mentioned above.

It would be possible to replace the bilinear mass rate constraints by linear ones or to apply a perspective reformulation of Günlük and Linderoth [31] to strengthen the relaxation. However, we do not want to apply any simplifying transformation and prefer the nonlinear formulation as it stands for our numerical tests. Note that the true real-life model is much more complex by taking different topologies, compressors, or any other technical systems into account. The mass rate and pressure constraints are based on computationally intensive simulations.

In a very similar way, some gas lift test problems are created, see Ray and Sarker [45] or Ayatollahi et al. [3] for related models. We finally get a set of 55 test problems, where the number of continuous variables varies between 3 and 10, the number of binary variables between 9 and 27, the number of linear equality constraints between 0 and 9, and the number of inequality constraints between 1 and 21. Table 2 contains performance results for the solvers under consideration.

The code MISQP is, in any of the four different versions tested, by far the most efficient one in terms of number of function evaluations. Even if an optimal solution is not reached, MISQP stops at least at an acceptable solution.

The additional stabilizations of MISQPOA by linear outer approximations require a significant amount of additional iterations and, consequently, a much larger number of function calls. The code is an upgrade of MISQP in the sense that proceeding from an initial optimal solution of MISQP, linear outer approximation constraints are added successively.

The idea behind the code MISQPN is similar. We get less acceptable solutions compared to MISQP, but, however, a couple of false terminations, which are in some cases very close to a solution. Also the number of function evaluations is significantly larger than those of MISQP. We have to note again that MISQPN is still an experimental implementation without all the additional heuristics developed for MISQP.

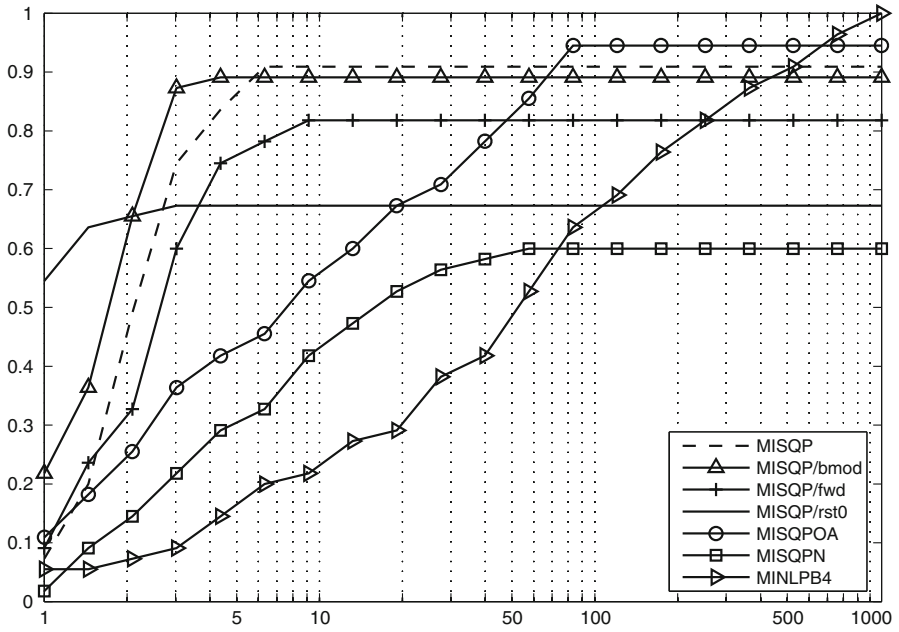

Fig. 2 Performance profiles for petroleum engineering problems

The branch-and-bound code MINLPB4 solves all test problems, but requires about 100 times as many function calls as MISQP.

Performance profiles are shown in Fig. 2, where the performance ratio of n_{func} is displayed over those logarithmic numbers, for which the performance ratio is below the displayed number, see also Fig. 1 and the corresponding comments.

5 Conclusions

It is well known that nonconvex nonlinear mixed-integer optimization problems are extremely difficult to solve. In general, even simple concepts like local solutions or convexity of functions are not available, especially if appropriate relaxations do not exist, i.e., continuous representations of model functions where integer variables can be treated as continuous. In highly complex technical simulation codes especially for engineering applications, however, it is often not possible to evaluate an objective or constraint function value for fractional values of an integer variable. In this situation, most of the known optimization algorithms fail to find a solution or cannot even be applied. Thus, the complexity of an optimization problem depends heavily on the structure of integer variables.

We conclude from our numerical results that SQP-based algorithms provide an efficient way to solve nonlinear mixed-integer programs, if the main performance criterion is the number of function evaluations. Since convex mixed-integer quadratic programming problems must be solved successively, the total computational effort, e.g., calculation times, might become costly. We recommend our approach also in

situations, where function evaluations are extremely expensive. The limitations of our approach are that integer variables must have an internal *smooth* structure, i.e., do not behave like categorial variables, and that the total number of variables is not too large.

We investigate the possibility to extend our mixed-integer SQP-type algorithm by adding a mixed-integer linear master program with linear outer approximation constraints, by which the SQP-type method is stabilized. The safeguards require a significant amount of additional function calls.

On the other hand, it is possible to apply the outer approximation idea directly to the mixed-integer SQP methods mentioned above, by combining the master program and the continuous solver for fixed integer variables. Drawback in this case is a larger number of feasible, but non-optimal solutions and more terminations in error situations. The corresponding code MISQPN is still a preliminary implementation and we will investigate its mathematical background in the future.

Our experience is based on numerical tests obtained by a set of 100 academic examples and 55 examples with some background in petroleum engineering. All objective and constraint functions consist of analytical expressions, and all integer variables are relaxable. However, this information is only exploited for testing certain variants of our codes, especially when we need partial derivative values with respect to integer variables. Our main motivation is to provide software in form of a toolbox for complex practical applications, where integer variables are not required to be relaxable and where function evaluations are costly. Since we assume that partial derivatives with respect to integer variables are not explicitly available, we replace them by descent directions computed by two-sided differences at neighbored grid points. Surprisingly enough, these crude approximations work extremely well in our numerical tests.

We suppose that a 'black box' simulation code provides the function values, i.e., that we do not need to know anything about the internal structure of variables or constraints. But if some structures are known in advance, as, e.g., special linear constraints based on special ordered set (SOS) variables, we have a very simple way to handle them. Since constraints are linearized, they are passed directly to the underlying MIQP. Due to the modular structure, our own MIQP solver is easily exchanged by another one which, after passing the structural information, takes them into account.

Despite of using inaccurate approximations of partial derivatives with respect to integer variables, our solvers efficiently compute feasible solutions for most test problems. Objective function values are often close to the best known optimal solutions.

Acknowledgments The authors are extremely grateful to the constructive comments of the referees and of the associated editor, and to their support to improve quality and readability of the manuscript.

Appendix: priority theory

To explain possible irregularities when evaluating statistical comparative scores, let us consider a simple example. Suppose we want to evaluate one performance criterion, e.g., calculation time, for comparing three optimization codes, say C_1, C_2, C_3, on five test problems, $TP_1, \ldots, TP_5$. The results of the test runs might be given by the data of Table 3. An asterix ìndicates that the code could not solve the corresponding test problem successfully.

Table 3 Some fictitious calculation times for 5 test problems and 3 codes

Code	TP_1	TP_2	TP_3	TP_4	TP_5	Mean
C_1	*	5.2	1.3	4.0	7.0	4.4
C_2	0.3	*	1.5	*	8.2	3.3
C_3	3.0	11.2	*	*	12.2	8.8

The mean values of calculation times are taken over all successful test runs. We get the impression that C_2 is somewhat faster than C_1, but the calculation times for test problems successively solved by both codes, are the other way round. On the other hand, the mean value for C_3 is twice as larger than the one for C_1 and a bit more than the one for C_2.

We have to expect that in particular the higher dimensional, time-consuming test problems could not be solved successfully by all programs. To avoid the difficulties as outlined in the above example, we exploit the priority theory of Saaty [46], which was used by Lootsma [40] for comparing optimization software.

Now we assume that we want to compare N codes C_i, $i = 1, \ldots, N$, on a set of M test problems TP_j, $j = 1, \ldots, M$. Let S_i denote the set of test problems that could be solved successfully by code C_i, $i = 1, \ldots, N$, i.e.,

$$S_i := \{j : TP_j \text{ could be solved successfully by } C_i, 1 \leq j \leq M\}. \tag{39}$$

Priority theory is based on a pairwise comparison of the N programs with respect to the performance criterion under consideration, e.g., calculation time. Let t_{ij} be the performance result obtained by code C_i on test problem TP_j, $t_{ij} > 0$. Then we use the expressions

$$r_{ik} := \frac{\sum_{j \in S_i \cap S_k} t_{ij}}{\sum_{j \in S_i \cap S_k} t_{kj}} \tag{40}$$

for $i = 1, \ldots, N$ and $k = 1, \ldots, N$, to define a reciprocal matrix

$$R := (r_{ik})_{i,k=1,N}, \tag{41}$$

where the elements of R satisfy the condition

$$r_{ik} = r_{ki}^{-1} > 0. \tag{42}$$

Matrix R is to be considered as an approximation of the matrix

$$P := \left(\frac{w_i}{w_k}\right)_{i,k=1,N}, \tag{43}$$

where the entries $w_1, \ldots, w_N$ are the true mean values of the stochastic variables we are considering, say expectation value of execution time for code C_i. To simplify the subsequent analysis, we assume that

$$\sum_{i=1}^{N} w_i = 1$$

and let $w := (w_1, \dots w_N)^T$. Then P is a rank-one matrix with

$$Pw = Nw, \tag{44}$$

i.e., N is the only positive eigenvalue of P and w is the uniquely determined normalized eigenvector with positive elements.

On the other hand, we can apply a theorem of Perron–Frobenius, see Bellman [4] for example, which states that the largest eigenvalue of R, which is considered as an approximation of P, is real and positive, and that there is a uniquely determined eigenvector with positive elements.

To sum up, the performance evaluation consists of establishing the matrix R, see (40) and (41), and of computing the maximum eigenvalue of R with positive eigenvector $\overline{w}$, which is considered as a suitable approximation of w, see (44). The entries of the eigenvector are scaled so that the smallest coefficient becomes the value one.

To give an example, consider the data of Table 3. Then

$$R = \begin{pmatrix} 1.0 & 0.86 & 0.52 \\ 1.17 & 1.0 & 0.56 \\ 1.92 & 1.79 & 1.0 \end{pmatrix}$$

and the eigenvector corresponding to the largest eigenvalue of R is

$$w = (1.0, 1.1, 2.0)^T.$$

These scores can be used to estimate the performance criterion which is to be evaluated. We observe that C_1 is slightly faster than C_2, as can be retrieved from Table 3. On the other hand, C_3 is about two times slower than C_1 and, approximately also twice slower than C_2. Whereas the first conclusion cannot be retrieved from the mean values, that last one corresponds quite exactly with the mean values, see Table 3.

References

1. Asaadi, J.: A computational comparison of some non-linear programs. Math. Program. **4**, 144–154 (1973)
2. Audet, C., Dennis, J.E.: Pattern search algorithm for mixed variable programming. SIAM J. Optim. **11**, 573–594 (2001)
3. Ayatollahi, S., Narimani, M., Moshfeghiam, M.: Intermittent gas lift in Aghjari Oil 488 Field, a mathematical study. J. Petroleum Sci. Eng. **42**, 245–255 (2004)
4. Bellman, R.: Introduction to Matrix Analysis. McGraw-Hill, India (1960)
5. Belotti, P., Lee, J., Liberti, L., Margot, F., Wächter, A.: Branching and bounds tightening techniques for non-convex MINLP. Optim. Methods Softw. **24**, 597–634 (2009)
6. Belotti, P.: Couenne: a user's manual, Technical Report, Department of Mathematical Sciences, Clemson University, Clemson SC 29643, USA (2009)
7. Bonami, P., Biegler, L.T., Conn, A.R., Cornuejols, G., Grossmann, I.E., Laird, C.D., Lee, J., Lodi, A., Margot, F., Sawaya, N., Waechter, A.: An algorithmic framework for convex mixed integer nonlinear programs. IBM Research Report RC23771 (2005)

8. Bonami, P., Kilinç, M., Linderoth, J.: Algorithms and software for convex mixed-integer nonlinear programs, Technical Report No. 1664, Computer Science Department, University of Wisconsin, Madison (2009)
9. Borchers, B., Mitchell, J.E.: An improved branch-and-bound algorithm for mixed integer nonlinear programming. Comput. Oper. Res. **21**(4), 359–367 (1994)
10. Bünner, M.J., Schittkowski, K., van de Braak, G.: Optimal design of electronic components by mixed-integer nonlinear programming. Optim. Eng. **5**, 271–294 (2004)
11. Bussieck, M.R., Vigerske, S. : MINLP Solver Software, submitted for publication (2010)
12. Bussieck, M.R., Drud, A.S., Meeraus, A.: MINLPLib—a collection of test models for mixed integer nonlinear programming. GAMS Development Corp, Washington D.C., USA (2007)
13. Chen, Z., Zhang, X.: A nonmonotone trust-region algorithm with nonmonotone penalty parameters for constrained optimization. J. Comput. Appl. Math. **172**, 7–39 (2004)
14. Conn, A.R., Gould, N.M., Toint, P.L.: Trust-region methods. MPS-SIAM Series on Optimization, Philadelphia (2000)
15. Deng, N.Y., Xiao, Y., Zhou, F.J.: Nonmonotonic trust-region algorithm. J. Optim. Theory Appl. **26**, 259–285 (1993)
16. Dolan, E.D., Moré, J.J.: Benchmarking optimization software with performance profiles. Math. Program. **91**, 201–213 (2002)
17. Duran, M., Grossmann, I.E.: An outer-approximation algorithm for a class of Mixed Integer Nonlinear Programs. Math. Program. **36**, 307–339 (1986)
18. Exler, O., Antelo, L.T., Egea, J.A., Alonso, A.A., Banga, J.R.: Tabu search-based algorithm for mixed-integer nonlinear problems and its application to integrated process and control system design. Comput. Chem. Eng. **32**(8), 1877–1891 (2008)
19. Exler, O., Lehmann, T., Schittkowski, K.: MISQPN : a Fortran subroutine for mixed-integer nonlinear optimization by outer approximation supported by mixed-integer search steps - user's guide, version 1.0, Report, Department of Computer Science, University of Bayreuth (2009)
20. Exler, O., Lehmann, T., Schittkowski, K.: MISQP: A Fortran subroutine of a trust region SQP algorithm for mixed-integer nonlinear programming—user's guide, Report, Department of Computer Science, University of Bayreuth (2012)
21. Exler, O., Schittkowski, K.: A trust region SQP algorithm for mixed integer nonlinear programming. Optim. Lett. **1**(3), 269–280 (2007)
22. Fletcher, R.: A new approach to variable metric algorithms. Comput. J. **13**, 317–322 (1970)
23. Fletcher, R.: Second order correction for nondifferentiable optimization. In: Watson, G.A. (eds.) Numerical analysis, pp. 85–114, Springer, Berlin (1982)
24. Fletcher, R., Leyffer, S.: Solving mixed integer nonlinear programs by outer approximation. Math. Program. **66**, 327–349 (1994)
25. Floudas, C.A.: Nonlinear and Mixed-Integer Optimization. Oxford University Press, New York (1995)
26. Fukushima, M.: A successive quadratic programming algorithm with global and superlinear convergence properties. Math. Program. **35**, 253–264 (1986)
27. Goldfarb, D., Idnani, A.: A numerically stable method for solving strictly convex quadratic programs. Math. Program. **27**, 1–33 (1983)
28. Grossmann, I.E.: Review of nonlinear mixed-integer and disjunctive programming techniques. Optim. Eng. **3**, 227–252 (2002)
29. Grossmann, I.E., Kravanja, Z.: 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 (1997)
30. Gupta, O.K., Ravindran, V.: Branch-and-bound experiments in convex nonlinear integer programming. Manag. Sci. **31**, 1533–1546 (1985)
31. Günlük, O., Linderoth, J.: Perspective relaxation of mixed integer nonlinear programs with indicator variables. Math. Program. Series B **104**, 186–203 (2010)
32. Hartwanger, C., Schittkowski, K., Wolf, H.: Computer aided optimal design of horn radiators for satellite communication. Eng. Optim. **33**, 221–244 (2000)
33. Hock, W., Schittkowski, K.: A comparative performance evaluation of 27 nonlinear programming codes. Computing **30**, 335–358 (1983)
34. Lehmann, T., Schittkowski, K.: MIQL: A Fortran subroutine for convex mixed-integer quadratic programming—user's guide, version 1.0, Report, Department of Computer Science, University of Bayreuth (2009)

35. Lehmann, T., Schittkowski, K.: MINLPB4: A Fortran code for nonlinear mixed-integer quadratic programming by branch-and-bound - user's guide, version 1.0, Report, Department of Computer Science, University of Bayreuth (2009)
36. Lehmann, T., Schittkowski, K.: MISQPOA: a Fortran subroutine for mixed-integer nonlinear optimization by outer approximation—user's guide, version 1.0, Report, Department of Computer Science, University of Bayreuth (2009)
37. Lehmann, T., Schittkowski, K., Spickenreuther, T.: BFOUR: a Fortran subroutine for integer optimization by branch-and-bound—user's guide, Report, Department of Computer Science, University of Bayreuth, Germany (2009)
38. Leyffer, S.: Integrating SQP and branch-and-bound for mixed integer nonlinear programming. Comput. Optim. Appl. **18**, 295–309 (2001)
39. Li, H.L., Chou, C.T.: A global approach for nonlinear mixed discrete programming in design optimization. Eng. Optim. **22**, 109–122 (1994)
40. Lootsma, F.A.: Performance evaluation of nonlinear optimization methods via multi-criteria analysis and via linear model analysis. In: Powell, M.J.D. (ed.) Nonlinear Optimization, vol. 82. Academic Press, San Diego (1982)
41. Maratos, N.: Exact penalty function algorithms for finite-dimensional and control optimization problems, Ph.D. thesis, University of London, England (1978)
42. Nowak, I., Alperin, H., Vigerske, S.: LaGO—an object oriented library for solving MINLPs. In: Bliek, C., Jermann, C., Neumaier, A. (eds.) Global Optimization and Constraint Satisfaction. Lecture Notes in Computer Science, vol. 2861, pp. 32–42. Springer, Berlin (2003)
43. Powell, M.J.D.: ZQPCVX, A FORTRAN subroutine for convex quadratic programming, Report DAMTP/1983/NA17, University of Cambridge, England (1983)
44. Quesada, I., Grossmann, I.E.: An LP/NLP based branch-and-bound algorithm for convex MINLP optimization problems. Comput. Chem. Eng. **16**, 937–947 (1992)
45. Ray, T., Sarker, R.: Genetic algorithm for solving a gas lift optimization problem. J. Petroleum Sci. Eng. **59**, 84–96 (2007)
46. Saaty, T.L.: A scaling method for priorities in hierarchical structures. J. Math. Psychol. **15**, 234–281 (1977)
47. Sahinidis, N.V., Tawarmalani, M.: BARON 9.0.4: Global Optimization of Mixed-Integer Nonlinear Programs, User's Manual (2010). http://www.gams.com/dd/docs/solvers/baron.pdf
48. Schittkowski, K.: Nonlinear Programming Codes - Information, Tests, Performance. Lecture Notes in Economics and Mathematical Systems, vol. 183. Springer, Berlin (1980)
49. Schittkowski, K.: QL: A Fortran code for convex quadratic programming—User's guide, Report, Department of Mathematics, University of Bayreuth, Germany (2003)
50. Schittkowski, K.: A collection of 100 test problems for nonlinear mixed-integer programming in Fortran—user's guide, Report, Department of Computer Science, University of Bayreuth (2010)
51. Schittkowski, K., Yuan, Y.-X.: Sequential quadratic programming methods to appear: Wiley Encyclopedia of Operations Research and Management Science (2010)
52. Schlueter, M.: Nonlinear Mixed Integer Based Optimization Techniques for Space Applications. Dissertation, School of Mathematics, University of Birmingham (2012)
53. Tawarmalani, M., Sahinidis, N.V.: A polyhedral branch-and-cut approach to global optimization. Math. Program. **103**, 225–249 (2005)
54. Thomas I., Kröner, A.: Mixed-integer optimization of distillation column tray position in industrial practice. In: Marquardt, W., Pantelides, C. (eds.) 16th European Symposium on Computer Aided Engineering ans 9th International Symposium on Porcess Systems Engineering. Elsevier, Amsterdam, pp. 1015–1020 (2006)
55. Toint, P.L.: A non-monotone trust region algorithm for nonlinear optimization subject to convex constraints. Math. Program. **77**(1), 69–94 (1997)
56. Viswanathan, J., Grossmann, I.E.: A combined penalty function and outer approximation method for MINLP optimization. Comput. Chem. Eng. **14**, 769–782 (1990)
57. Westerlund, T., Pörn, R.: Solving pseudo-convex mixed integer optimization problems by cutting plane techniques. Optim. Eng. **3**, 253–280 (2002)
58. Yuan, Y.X.: On the convergence of a new trust region algorithm. Numerische Mathematik **70**, 515–539 (1995)
59. Yuan, Y.X., Sun, W.: Optimization Theory and Methods. Springer, Berlin (2006)