USER'S GUIDE FOR TOMLAB /MINLP[1]

Kenneth Holmström[2], Anders O. Göran[3] and Marcus M. Edvall[4]

February 26, 2007

[1]More information available at the TOMLAB home page: `http://tomopt.com`. E-mail: **tomlab@tomopt.com**.

[2]Professor in Optimization, Mälardalen University, Department of Mathematics and Physics, P.O. Box 883, SE-721 23 Västerås, Sweden, kenneth.holmstrom@mdh.se.

[3]Tomlab Optimization AB, Västerås Technology Park, Trefasgatan 4, SE-721 30 Västerås, Sweden, anders@tomopt.com.

[4]Tomlab Optimization Inc., 855 Beech St #121, San Diego, CA, USA, medvall@tomopt.com.

Contents

1 Introduction

1.1 Overview

Welcome to the TOMLAB /MINLP User's Guide. TOMLAB /MINLP includes a suite of 4 solvers in dense and sparse version and interfaces between The MathWorks' MATLAB and the MINLP solver package. The solver package includes the following solvers:

bqpd - For sparse and dense linear and quadratic programming (nonconvex).

filterSQP - For sparse and dense linear, quadratic and nonlinear programming.

minlpBB - For sparse and dense mixed-integer linear, quadratic and nonlinear programming.

miqpBB - For sparse and dense mixed-integer linear and quadratic.

Please visit `http://tomopt.com/tomlab/products/minlp/` for more information.

The interface between TOMLAB /MINLP, Matlab and TOMLAB consists of two layers. The first layer gives direct access from Matlab to MINLP, via calling one Matlab function that calls a pre-compiled MEX file (DLL under Windows, shared library in UNIX) that defines and solves the problem in MINLP. The second layer is a Matlab function that takes the input in the TOMLAB format, and calls the first layer function. On return the function creates the output in the TOMLAB format.

1.2 Contents of this Manual

- Section 2 gives the basic information needed to run the Matlab interface.
- Section 3 provides all the solver references for bqpd, filterSQP, minlpBB, and miqpBB.
- Section 4 presents the fields used in the special structure *Prob.DUNDEE*.

1.3 Prerequisites

In this manual we assume that the user is familiar with MINLP, TOMLAB and the Matlab language.

2 Using the Matlab Interface

The main routines in the two-layer design of the interface are shown in Table 1. Page and section references are given to detailed descriptions on how to use the routines.

Table 1: The interface routines.

Function	**Description**	**Section**	**Page**
bqpd	The layer one Matlab interface routine, calls the MEX-file interface *bqpd.dll*	3.1.1	7
bqpdTL	The layer two interface routine called by the TOMLAB driver routine *tomRun*. This routine then calls *bqpd.m*.	3.1.2	11
filterSQP	The layer one Matlab interface routine, calls the MEX-file interface *filterSQP.dll*	3.2.1	14
filterSQPTL	The layer two interface routine called by the TOMLAB driver routine *tomRun*. This routine then calls *filterSQP.m*.	3.2.2	19
minlpBB	The layer one Matlab interface routine, calls the MEX-file interface *minlpBB.dll*	3.3.1	24
minlpBBTL	The layer two interface routine called by the TOMLAB driver routine *tomRun*. This routine then calls *minlpBB.m*.	3.3.2	29
miqpBB	The layer one Matlab interface routine, calls the MEX-file interface *miqpBB.dll*	3.4.1	34
miqpBBTL	The layer two interface routine called by the TOMLAB driver routine *tomRun*. This routine then calls *minlpBB.m*.	3.4.2	38

The MINLP control parameters are possible to set from Matlab.

They can be set as inputs to the interface routine *bqpd* and the others. The user sets fields in a structure called *Prob.DUNDEE.optPar*. The following example shows how to set the print level.

```
Prob.DUNDEE.optPar(1)   = 1;    % Setting the print level
```

3 TOMLAB /MINLP Solver Reference

The MINLP solvers are a set of Fortran solvers that were developed by Roger Fletcher and Sven Leyffer, University of Dundee, Scotland. All solvers are available for sparse and dense continuous and mixed-integer quadratic programming (**qp**,**miqp**) and continuous and mixed-integer nonlinear constrained optimization.

Table 2 lists the solvers included in TOMLAB /MINLP. The solvers are called using a set of MEX-file interfaces developed as part of TOMLAB. All functionality of the MINLP solvers are available and changeable in the TOMLAB framework in Matlab.

Detailed descriptions of the TOMLAB /MINLP solvers are given in the following sections. Extensive TOMLAB m-file help is also available, for example *help minlpBBTL* in Matlab will display the features of the minlpBB solver using the TOMLAB format.

TOMLAB /MINLP package solves **mixed-integer nonlinear programming** (**minlp**) problem defined as

$$
\begin{array}{ll}
\min\limits_{x} & f(x) \\
 & \\
s/t & \begin{array}{rcccccl} -\infty < & x_L & \leq & x & \leq & x_U & < \infty \\ & b_L & \leq & Ax & \leq & b_U & \\ & c_L & \leq & c(x) & \leq & c_U, & x_j \in \mathbb{N} \quad \forall j \in \mathrm{I}, \end{array}
\end{array}
\tag{1}
$$

where $x, x_L, x_U \in \mathbb{R}^n$, $f(x) \in \mathbb{R}$, $A \in \mathbb{R}^{m_1 \times n}$, $b_L, b_U \in \mathbb{R}^{m_1}$ and $c_L, c(x), c_U \in \mathbb{R}^{m_2}$. The variables $x \in I$, the index subset of $1, ..., n$, are restricted to be integers.

mixed-integer quadratic programming (**miqp**) problems defined as

$$
\begin{array}{ll}
\min\limits_{x} & f(x) = \frac{1}{2}x^T F x + c^T x \\
 & \\
s/t & \begin{array}{ccccc} x_L & \leq & x & \leq & x_U, \\ b_L & \leq & Ax & \leq & b_U \end{array}
\end{array}
\tag{2}
$$

where $c, x, x_L, x_U \in \mathbb{R}^n$, $F \in \mathbb{R}^{n \times n}$, $A \in \mathbb{R}^{m_1 \times n}$, and $b_L, b_U \in \mathbb{R}^{m_1}$. The variables $x \in I$, the index subset of $1, ..., n$, are restricted to be integers.
as well as sub-types of these problems.

Table 2: Solver routines in TOMLAB /MINLP.

Function	**Description**	**Reference**
bqpd	Quadratic programming using a null-space method.	*bqpdTL.m*
miqpBB	Mixed-integer quadratic programming using *bqpd* as subsolver.	*miqpBBTL.m*
filterSQP	Constrained nonlinear minimization using a Filtered Sequential QP method.	*filterSQPTL.m*
minlpBB	Constrained, mixed-integer nonlinear minimization using a branch-and-bound search scheme. *filterSQP* is used as NLP solver.	*minlpBBTL.m*

3.1 bqpd

The BQPD code solves quadratic programming (minimization of a quadratic function subject to linear constraints) and linear programming problems. If the Hessian matrix Q is positive definite, then a global solution is found. A global solution is also found in the case of linear programming (Q=0). When Q is indefinite, a Kuhn-Tucker point that is usually a local solution is found.

The code implements a null-space active set method with a technique for resolving degeneracy that guarantees that cycling does not occur even when round-off errors are present. Feasibility is obtained by minimizing a sum of constraint violations. The Devex method for avoiding near-zero pivots is used to promote stability. The matrix algebra is implemented so that the algorithm can take advantage of sparse factors of the basis matrix. Factors of the reduced Hessian matrix are stored in a dense format, an approach that is most effective when the number of free variables is relatively small. The user must supply a subroutine to evaluate the Hessian matrix Q, so that sparsity in Q can be exploited. An extreme case occurs when Q=0 and the QP reduces to a linear program. The code is written to take maximum advantage of this situation, so that it also provides an efficient method for linear programming.

3.1.1 Direct Solver Call

A direct solver call is not recommended unless the user is 100 % sure that no other solvers will be used for the problem. Please refer to Section 3.1.2 for information on how to use bqpd with TOMLAB.

Purpose
bqpd solves quadratic optimization problems defined as

$$
\begin{array}{cl}
\min\limits_{x} & f(x) = \frac{1}{2}x^T F x + c^T x \\
 & \\
s/t & \begin{array}{ccccc} & & x & & , \\ b_L & \leq & Ax & \leq & b_U \end{array}
\end{array}
\tag{3}
$$

where $c, x \in \mathbb{R}^n$, $F \in \mathbb{R}^{n\times n}$, $A \in \mathbb{R}^{m_1 \times n}$, and $b_L, b_U \in \mathbb{R}^{n+m_1}$.

Calling Syntax

[Inform, x_k, Obj, g, Iter, k, ls, e, peq, lp, v_k] = bqpds(A, x_0, bl, bu, H, fLowBnd, mlp, mode, kmax, PriLev, PrintFile, k, ls, e, peq, lp, optPar, Prob, moremem);

The sparse version MEX is bqpds, the dense is bqpdd.

Description of Inputs

The following fields are used:

A	Constraint matrix, n x m+1 (SPARSE).
bl	Lower bounds on (x, Ax).
bu	Upper bounds on (x, Ax).
x_0	Initial x vector (if empty set as 0).
H	Quadratic matrix, n x n, SPARSE or DENSE, empty if LP problem. If H is a string, H should be the name of a function routine, e.g if H = 'HxComp' then the function routine. function Hx = HxComp(x, nState, Prob) should compute H * x. The user must define this routine nState == 1 if calling for the first time, otherwise 0. Third argument, the Prob structure, should only be used if calling BQPD with the additional input parameter Prob, see below. Tomlab implements this callback to the predefined Matlab function HxFunc.m, using the call if Prob.DUNDEE.callback == 1.
fLowBnd	Lower bound on optimal f(x).
mlp	Maximum number of levels of recursion.
mode	Mode of operation, default set as 2*Prob.WarmStart.
kmax	Max dimension of reduced space (k), default n, set as 0 if LP.
PriLev	Print Level. (0 = off, 1 = summary, 2 = scalar information, 3 = verbose)
PrintFile	Name of the Print file. Unit 9 is used. Name includes the path, maximal number of characters = 500. Output is written on file bqpd.txt, if not given. To make bqpd to not open and not write anything to file: Set PriLev = 0.

For Warm Start:

The following fields are used:, continued

k	Dimension of the reduced space (Warm Start).
ls	Indices of active constraints, first n-k used for warm start.
e	Steepest-edge normalization coefficients (Warm Start).
peq	Pointer to the end of equality constraint indices in ls (Warm Start).
lp	List of pointers to recursion information in ls (Warm Start).
optPar	Vector of optimization parameters. If -999, set to default. Length from 0 to 20 allowed.
optPar(1): iprint	Print level in BQPD, default 0.
optPar(2): tol	Relative accuracy in solution, default 1E-10.
optPar(3): emin	Use cscale (constraint scaling) 0.0 no scaling, default 1.0.
optPar(4): sgnf	Max rel error in two numbers equal in exact arithmetic, default 5E-4.
optPar(5): nrep	Max number of refinement steps, default 2.
optPar(6): npiv	No repeat if no more than npiv steps were taken, default 3.
optPar(7): nres	Max number of restarts if unsuccessful, default 2.
optPar(8): nfreq	The max interval between refactorizations, default 500.
Prob	Sending the Prob structure is optional, only of use if sending H as a function string, see input H.
moremem	Scalar or 2x1-vector with workspace increase. If <0, use default strategy. If scalar, use same increase for both real and integer workspaces. If vector, first element is for real workspace, second for integer.

Description of Outputs

The following fields are used:

Inform	Result of BQPD run, 0 = Optimal solution found. See the same parameter in section 3.1.2.
x_k	Solution vector with n decision variable values.
Obj	Objective function value at optimum. If infeasible, the sum of infeasibilities
g	Gradient at solution.
Iter	Number of iterations.

The following fields are used:, continued

For Warm Start:

k	Dimension of the reduced space (Warm Start).
ls	Indices of active constraints, first n-k used for warm start.
e	Steepest-edge normalization coefficients (Warm Start).
peq	Pointer to the end of equality constraint indices in ls (Warm Start).
lp	List of pointers to recursion information in ls (Warm Start).
v_k	Lagrange parameters.

3.1.2 Using TOMLAB

Purpose

bqpdTL solves nonlinear optimization problems defined as

$$
\begin{array}{cc}
\min\limits_{x} & f(x) = \frac{1}{2}x^T F x + c^T x \\
\\
s/t & \begin{array}{lcccl} x_L & \leq & x & \leq & x_U, \\ b_L & \leq & Ax & \leq & b_U \end{array}
\end{array}
\qquad (4)
$$

where $c, x, x_L, x_U \in \mathbb{R}^n$, $F \in \mathbb{R}^{n \times n}$, $A \in \mathbb{R}^{m_1 \times n}$, and $b_L, b_U \in \mathbb{R}^{m_1}$.

Calling Syntax

Using the driver routine *tomRun*:

Prob = ⋄Assign(...);
Result = tomRun('bqpd', Prob ...);

or

Prob = ProbCheck(Prob,'bpqd'); Result = bqpdTL(Prob);

Call Prob = ⋄Assign(...) or Prob=ProbDef; to define the Prob for the second option.

Description of Inputs

Prob, The following fields are used:

x_L, x_U	Bounds on variables.
b_L, b_U	Bounds on linear constraints.
A	Linear constraint matrix.
QP.c	Linear coefficients in objective function.
QP.F	Quadratic matrix of size n x n.
PriLevOpt	Print Level (0 = off, 1 = summary, 2 = scalar information, 3 = verbose).
WarmStart	If TRUE (=1), use warm start, otherwise cold start.
LargeScale	If TRUE (=1), use sparse version, otherwise dense.
DUNDEE.QPmin	Lower bound for the QP subproblems. Default: -1E300.
DUNDEE.callback	If 1, use a callback to Matlab to compute QP.F * x for different x. Faster when F is very large and almost dense, avoiding copying of F from Matlab to MEX.

Prob, The following fields are used:, continued

DUNDEE.kmax	Max dimension of reduced space (k), default n, set as 0 if LP.
DUNDEE.mlp	Maximum number of levels of recursion.
DUNDEE.mode	Mode of operation, default set as 2*Prob.WarmStart.
DUNDEE.x	Solution (Warm Start).
DUNDEE.k	Dimension of the reduced space (Warm Start).
DUNDEE.e	Steepest-edge normalization coefficients (Warm Start).
DUNDEE.ls	Indices of active constraints, first n-k used for warm start.
DUNDEE.lp	List of pointers to recursion information in ls (Warm Start).
DUNDEE.peq	Pointer to the end of equality constraint indices in ls (Warm Start).
DUNDEE.PrintFile	Name of print file. Amount/print type determined by optPar(1). Default name bqpd.txt.
DUNDEE.optPar	Vector of optimization parameters. If -999, set to default Length from 0 to 20 allowed. Elements used:

DUNDEE.optPar(1): iprint	Print level in PrintFile, default 0.
DUNDEE.optPar(2): tol	Relative accuracy in solution, default 1E-10.
DUNDEE.optPar(3): emin	1.0 Use cscale (constraint scaling) 0.0 no scaling, default 1.0.
DUNDEE.optPar(4): sgnf	Max rel error in two numbers equal in exact arithmetic, default 5E-4.
DUNDEE.optPar(5): nrep	Max number of refinement steps, default 2.
DUNDEE.optPar(6): npiv	No repeat if no more than npiv steps were taken, default 3.
DUNDEE.optPar(7): nres	Max number of restarts if unsuccessful, default 2.
DUNDEE.optPar(8): nfreq	The max interval between refactorizations, default 500.

Description of Outputs

Result, The following fields are used:

Result	The structure with results (see ResultDef.m).
f_k	Function value at optimum or constraint deviation if infeasible.
x_k	Solution vector.
x_0	Initial solution vector.
g_k	Exact gradient computed at optimum.

Result, The following fields are used:, continued

xState	State of variables. Free == 0; On lower == 1; On upper == 2; Fixed == 3;
bState	State of linear constraints. Free == 0; Lower == 1; Upper == 2; Equality == 3;
v_k	Lagrangian multipliers (for bounds + dual solution vector).
ExitFlag	Exit status from bqpd.m (similar to TOMLAB).
Inform	BQPD information parameter. 0 - Solution obtained 1 - Unbounded problem detected (f(x)¡=fLow occurred) 2 - Lower bound bl(i) > bu(i) (upper bound) for some i 3 - Infeasible problem detected in Phase 1 4 - Incorrect setting of m, n, kmax, mlp, mode or tol 5 - Not enough space in lp 6 - Not enough space for reduced Hessian matrix (increase kmax) 7 - Not enough space for sparse factors 8 - Maximum number of unsuccessful restarts taken
Iter	Number of iterations.
MinorIter	Number of minor iterations. Always set to 0.
FuncEv	Number of function evaluations. Set to Iter.
GradEv	Number of gradient evaluations. Set to Iter.
ConstrEv	Number of constraint evaluations. Set to 0.
QP.B	Basis vector in TOMLAB QP standard.

3.2 filterSQP

The solver filterSQP is a Sequential Quadratic Programming solver suitable for solving large, sparse or dense linear, quadratic and nonlinear programming problems. The method avoids the use of penalty functions. Global convergence is enforced through the use of a trust-region and the new concept of a "filter" which accepts a trial point whenever the objective or the constraint violation is improved compared to all previous iterates. The size of the trust-region is reduced if the step is rejected and increased if it is accepted (provided the agreement between the quadratic model and the nonlinear functions is sufficiently good).

This method has performed very well in comparative numerical testing, and has the advantage that the user does not need to supply any estimates of penalty parameters. The NLP problem is specified by means of user subroutines, and it is necessary to provide information about both first and second derivatives of the nonlinear functions in the problem.

It must be used in conjunction with the bqpd solver.

3.2.1 Direct Solver Call

A direct solver call is not recommended unless the user is 100 % sure that no other solvers will be used for the problem. Please refer to Section 3.2.2 for information on how to use filterSQP with TOMLAB.

Purpose
filterSQP solves constrained nonlinear optimization problems defined as

$$\begin{array}{cccccc} \min\limits_x & f(x) & & & & \\ & & & & & \\ & & & x & & , \\ s/t & b_L & \leq & Ax & \leq & b_U \\ & & & c(x) & & \end{array} \tag{5}$$

where $x \in \mathbb{R}^n$, $f(x) \in \mathbb{R}$, $A \in \mathbb{R}^{m_1 \times n}$, $b_L, b_U \in \mathbb{R}^{n+m_1+m_2}$ and $c(x) \in \mathbb{R}^{m_2}$.

The full input matrix A has three parts A = A = [g ConsPattern' A'];
Where g is a vector of length n, values irrelevant, ConsPattern is the 0-1 pattern of the nonlinear constraint gradients and A is the linear constraint coefficient matrix.

Calling Syntax

The file 'funfdf.m' must be defined and contain: function [mode, f, g] = funfdf(x, Prob, mode, nstate) to compute the objective function f and the gradient g at the point x.

The file 'funcdc.m' must be defined and contain: function [mode ,c ,dcS] = funcdc(x, Prob, mode, nstate) to compute the nonlinear constraint value c and the constraint Jacobian dcS for the nonlinear constraints at the point x.

[ifail, x_k, f_k, c_k, v_k, lws, istat, rstat] = filSQPs(A, bl, bu, nnCon, x_0, Scale, scmode, fLow, MaxIter, rho, mlp, kmax, maxf, WarmStart, lws, istat, PriLev, pname, optPar, Prob, moremem);

The sparse version MEX is filSQPs, the dense is filSQPd.

Description of Inputs

The following fields are used:

A	Gradient matrix [g ConsPattern' A'] (sparse or dense).
bl	Lower bounds on (x,c(x),Ax).
bu	Upper bounds on (x,c(x),Ax).
nnCon	Number of nonlinear constraints (i.e. length(c(x)).
x_0	Initial x vector (if empty set as 0).
Scale	n+m vector scale factors for variables and constraints (same ordering as bl, bu).
scmode	Scale mode: 0 - unit variable and constraint scaling (Scale can be set empty). 1 - User provided scale factors for variables. Scale must be of length n. 2 - Unit variable scaling, user provided constraint scaling. Scale must be of length n+m, but only the last m elements are used. 3- User provided variable AND constraint scaling. Scale must be of length n+m (n+nnCon+nnLin)
fLow	A lower bound on the objective function value.
MaxIter	Maximum number of iterations.
rho	Initial trust-region radius.
mlp	Maximum level parameter for resolving degeneracy in BQPD.
kmax	Maximum size of null-space (at most n).
maxf	Maximum size of the filter.
WarmStart	Set to 1 to restart the solver. If a warmstart is requested, the input parameters lws and istat must be provided. Also, n and m (the number of variables and constraints) may not change.
lws	Used only when doing a warmstart. This must be the lws vector returned by the previous call to filterSQP. Otherwise, set to empty.

The following fields are used:, continued

lam	Multipliers, n+m values required for warmstarts. If wrong length, zeros are set in the MEX interface.
istat	Used only when doing a warmstart. Must be the first element of the istat vector returned by the previous call to filterSQP. Otherwise, set to empty.
PriLev	Print level. See also input parameter pname. 0 - Silent, except for minor output into ¡pname¿.out 1 - One line per iteration 2 - Scalar information printed 3 - Scalar and vector information printed
pname	Problem name, at most 10 characters. The output files are named <pname>.sum and <pname>.out.
optPar	Vector of max length 20 with optimization parameters: If any element is -999, default value is assigned. Elements 2-8 are BQPD parameters, 1,9-11,19-20 for filterSQP.
optPar(1): iprint	Print level in filterSQP, default 0.
optPar(2): tol	Relative tolerance for BQPD subsolver, default 1E-10.
optPar(3): emin	1=Use cscale in BQPD, 0=no scaling , default 1.0.
optPar(4): sgnf	Max rel error in two numbers equal in exact arithmetic (BQPD), default 5E-4.
optPar(5): nrep	Max number of refinement steps (BQPD), default 2.
optPar(6): npiv	No repeat if no more than npiv steps were taken, default 3.
optPar(7): nres	Max number of restarts if unsuccessful, default 2.
optPar(8): nfreq	The max interval between refactorizations, default 500.
optPar(9): NLP_eps	NLP subproblem tolerance, default 1E-6.
optPar(10): ubd	Upper bound on constraint violation used in the filter, default 1E-2.
optPar(11): tt	Parameter related to ubd. The actual upper bound is defined by the maximum of ubd and tt multiplied by the initial constraint violation., default 0.125.
optPar(19): infty	A large value representing infinity, default 1E20.
optPar(20): Nonlin	If 1, skip linear feasibility tests filterSQP treating all constraints as nonlinear, default 0.
Prob	The Tomlab problem definition structure.
moremem	Scalar or 2x1-vector with workspace increase. If <0, use default strategy. If scalar, use same increase for both real and integer workspaces. If vector, first element is for real workspace, second for integer.

Description of Outputs

The following fields are used:

Inform	Exit flag indicating success or failure: 0 - Solution found 1 - Unbounded: feasible point x with f(x)<=fmin found 2 - Linear constraints are infeasible 3 - (Locally) nonlinear infeasible, optimal solution to feasibility problem found 4 - Terminated at point with h(x)<=eps but QP infeasible 5 - Terminated with rho<=eps 6 - Terminated due to too many iterations 7 - Crash in user routine could not be resolved 8 - Unexpected failure in QP solver 9 - Not enough real workspace 10 - Not enough integer workspace
x_k	Solution vector (n+m by 1) with n decision variable values together with the m slack variables.
f_k	Function value at optimum x_k
c_k	Nonlinear constraints vector at optimum.
v_k	Lagrange multipliers vector (bounds, nonlinear, linear).
lws	Integer vector (used as input when doing warmstarts).
istat	Solution statistics, integer values. First element is required as input if doing a warmstart.
istat(1)	Dimension of nullspace at solution
istat(2)	Number of iterations
istat(3)	Number of feasibility iterations
istat(4)	Number of objective evaluations
istat(5)	Number of constraint evaluations
istat(6)	Number of gradient evaluations
istat(7)	Number of Hessian evaluations
istat(8)	Number of QPs with mode<=2
istat(9)	Number of QPs with mode>=4
istat(10)	Total number of QP pivots
istat(11)	Number of SOC steps
istat(12)	Maximum size of filter
istat(13)	Maximum size of Phase 1 filter
istat(14)	Number of QP crashes
rstat	Solution statistics, real values.

The following fields are used:, continued

rstat(1)	l_2 norm of KT residual
rstat(3)	Largest modulus multiplier
rstat(4)	l_inf norm of final step
rstat(5)	Final constraint violation h(x)

3.2.2 Using TOMLAB

Purpose
filterSQPTL solves constrained nonlinear optimization problems defined as

$$
\begin{array}{cccccc}
\min\limits_{x} & f(x) & & & & \\
\\
 & x_L & \leq & x & \leq & x_U, \\
s/t & b_L & \leq & Ax & \leq & b_U \\
 & c_L & \leq & c(x) & \leq & c_U
\end{array}
\tag{6}
$$

where $x, x_L, x_U \in \mathbb{R}^n$, $f(x) \in \mathbb{R}$, $A \in \mathbb{R}^{m_1 \times n}$, $b_L, b_U \in \mathbb{R}^{m_1}$ and $c_L, c(x), c_U \in \mathbb{R}^{m_2}$.

Calling Syntax
Using the driver routine *tomRun*:

Prob = ⋄Assign(...);
Result = tomRun('filterSQP', Prob ...);

or

Result = filterSQPTL(funfdf, funcdc, Prob), where the inputs are:

funfdf	Name of routine [f, gradf] = funfdf(x, Prob, mode, nstate). Normally funfdf = nlp_fg, included in TOMLAB.
funcdc	Name of routine [g, gJac] = funcdc(x, Prob, mode, nstate). Normally funcdc = nlp_cdcS, included in TOMLAB.
Prob	Problem structure in TOMLAB format.

Call Prob = ⋄Assign(...) or Prob=ProbDef; to define the Prob for the second option.

Description of Inputs

Prob, The following fields are used:

x_L, x_U	Bounds on variables.
b_L, b_U	Bounds on linear constraints.
c_L, c_U	Bounds on nonlinear constraints. For equality constraints (or fixed variables), set e.g. b_L(k) == b_U(k).
LargeScale	If 1 use sparse version of solver. The default is 0, the dense version.
PriLevOpt	Print level in the filterSQP solver.
WarmStart	Indicates that the solver should be warmstarted. See Prob.DUNDEE for necessary arguments when doing warmstarts.

Prob, The following fields are used:, continued

optParam	Structure with optimization parameters. Fields used:
MaxIter	Maximum number of iterations.
DUNDEE	Structure with special fields for filterSQP optimization parameters. The following fields are used:
DUNDEE.QPmin	Lower bound for the QP subproblems. Default: -1E300.
DUNDEE.rho	Initial trust region radius. Default: 10.0 (REAL).
DUNDEE.kmax	Maximum size of the null-space, less than or equal to no. of variables. Default: n (INTEGER).
DUNDEE.maxf	Maximum size of the filter. Default: 100 (INTEGER).
DUNDEE.mlp	Maximum level parameter for resolving degeneracy in BQPD QP subsolver. Default: 100 (INTEGER).
DUNDEE.Name	Problem name, at most 10 characters. The output files are named <pname>.sum and <pname>.out. Default name filterSQP, i.e. files filterSQP.sum, filterSQP.out.
DUNDEE.optPar	Vector of max length 20 with optimization parameters: If any element is -999, default value is assigned.
DUNDEE.optPar(1): iprint	Print level in filterSQP, default 0.
DUNDEE.optPar(2): tol	Relative tolerance for BQPD subsolver, default 1E-10.
DUNDEE.optPar(3): emin	1=Use cscale in BQPD, 0=no scaling , default 1.0.
DUNDEE.optPar(4): sgnf	Max rel error in two numbers equal in exact arithmetic (BQPD), default 5E-4.
DUNDEE.optPar(5): nrep	Max number of refinement steps (BQPD), default 2.
DUNDEE.optPar(6): npiv	No repeat if no more than npiv steps were taken, default 3.
DUNDEE.optPar(7): nres	Max number of restarts if unsuccessful, default 2.
DUNDEE.optPar(8): nfreq	The max interval between refactorizations, default 500.
DUNDEE.optPar(9): NLP_eps	NLP subproblem tolerance, default 1E-6.
DUNDEE.optPar(10): ubd	Upper bound on constraint violation used in the filter, default 1E-2.
DUNDEE.optPar(11): tt	Parameter related to ubd. The actual upper bound is defined by the maximum of ubd and tt multiplied by the initial constraint violation., default 0.125.
DUNDEE.optPar(19): infty	A large value representing infinity, default 1E20.
DUNDEE.optPar(20): Nonlin	If 1, skip linear feasibility tests filterSQP treating all constraints as nonlinear, default 0.
lws	If doing warmstarts, this field is set to the Result.DUNDEE.lws field from the previous run.

Prob, The following fields are used:, continued

istat	Similarly, for warmstarts, set istat to Result.DUNDEE.istat from the previous run. Only the first element is used.
lam	Vector of initial multipliers. Necessary for warmstarts, but can always be given if desired. Must be n+m elements in order to be used.
morereal	Increase of REAL workspace. A problem dependent default value is used if <0 or empty.
moreint	Increase of INTEGER workspace. A problem dependent default value is used if <0 or empty.
	Scaling parameters: It is possible to supply scale factors for the variables and/or the constraints. Normally, the DUNDEE solvers does not differentiate between linear and nonlinear constraints with regard to scaling, but the Tomlab interface handles this automatically. Thus is it possible to give scale factors e.g. for the nonlinear constraints only. All scaling values must be greater than zero.
xScale	Vector of scale factors for variables. If less than n values given, 1's are used for the missing elements.
bScale	Vector of scale factors for the linear constraints. If length(bScale) is less than the number of linear constraints (size(Prob.A,1)), 1's are used for the missing elements.
cScale	Vector of scale factors for the nonlinear constraints. If length(cScale) is less than the number of nonlinear constraints, 1's are used for the missing elements.

Description of Outputs

Result, The following fields are used:

Result	The structure with results (see ResultDef.m).
f_k	Function value at optimum.
g_k	Gradient of the function.
x_k	Solution vector.
x_0	Initial solution vector.
c_k	Nonlinear constraint residuals.
cJac	Nonlinear constraint gradients.
xState	State of variables. Free == 0; On lower == 1; On upper == 2; Fixed == 3;

Result, The following fields are used:, continued

bState	State of linear constraints. Free == 0; Lower == 1; Upper == 2; Equality == 3;
cState	State of nonlinear constraints. Free == 0; Lower == 1; Upper == 2; Equality == 3;
v_k	Lagrangian multipliers (for bounds + dual solution vector).
ExitFlag	Exit status from filterSQP MEX.
Inform	filterSQP information parameter: 0 - Solution found 1 - Unbounded: feasible point x with f(x)<=fmin found 2 - Linear constraints are infeasible 3 - (Locally) nonlinear infeasible, optimal solution to feasibility problem found 4 - Terminated at point with h(x)<=eps but QP infeasible 5 - Terminated with rho<=eps 6 - Too many iterations 7 - Crash in user routine could not be resolved 8 - Unexpected ifail from QP solver. This is often due to too little memory being allocated and is remedied by setting appropriate values in the Prob.DUNDEE.morereal and Prob.DUNDEE.moreint parameters. 9 - Not enough REAL workspace 10 - Not enough INTEGER workspace
Iter	Number of iterations.
FuncEv	Number of function evaluations.
GradEv	Number of gradient evaluations.
ConstrEv	Number of constraint evaluations.
Solver	Name of the solver (filterSQP).
SolverAlgorithm	Description of the solver.
DUNDEE.lws	Workspace vector, should be treated as integer valued. Required if doing warmstarts.
DUNDEE.lam	Vector of multipliers, required if doing warmstarts.
istat	Solution statistics, integer values. First element is required as input if doing a warmstart.
istat(1)	Dimension of nullspace at solution
istat(2)	Number of iterations
istat(3)	Number of feasibility iterations

Result, The following fields are used:, continued

istat(4)	Number of objective evaluations
istat(5)	Number of constraint evaluations
istat(6)	Number of gradient evaluations
istat(7)	Number of Hessian evaluations
istat(8)	Number of QPs with mode<=2
istat(9)	Number of QPs with mode>=4
istat(10)	Total number of QP pivots
istat(11)	Number of SOC steps
istat(12)	Maximum size of filter
istat(13)	Maximum size of Phase 1 filter
istat(14)	Number of QP crashes

rstat	Solution statistics, floating point values.

rstat(1)	l_2 norm of KT residual
rstat(3)	Largest modulus multiplier
rstat(4)	l_inf norm of final step
rstat(5)	Final constraint violation h(x)

3.3 minlpBB

The solver minlpBB solves large, sparse or dense mixed-integer linear, quadratic and nonlinear programming problems. minlpBB implements a branch-and-bound algorithm searching a tree whose nodes correspond to continuous nonlinearly constrained optimization problems. The user can influence the choice of branching variable by providing priorities for the integer variables.

The solver must be used in conjunction with both filterSQP and bqpd.

3.3.1 Direct Solver Call

A direct solver call is not recommended unless the user is 100 % sure that no other solvers will be used for the problem. Please refer to Section 3.2.2 for information on how to use filterSQP with TOMLAB.

Purpose
filterSQP solves constrained nonlinear optimization problems defined as

$$
\begin{array}{cccccc}
\min\limits_{x} & f(x) & & & & \\
 & & & & & \\
 & & & x & & , \\
s/t & b_L & \leq & Ax & \leq & b_U \\
 & & & c(x) & &
\end{array}
\tag{7}
$$

where $x \in \mathbb{R}^n$, $f(x) \in \mathbb{R}$, $A \in \mathbb{R}^{m_1 \times n}$, $b_L, b_U \in \mathbb{R}^{n+m_1+m_2}$ and $c(x) \in \mathbb{R}^{m_2}$.

The full input matrix A has three parts A = A = [g ConsPattern' A'];
Where g is a vector of length n, values irrelevant, ConsPattern is the 0-1 pattern of the nonlinear constraint gradients and A is the linear constraint coefficient matrix.

Calling Syntax

The file 'funfdf.m' must be defined and contain: function [mode, f, g] = funfdf(x, Prob, mode, nstate) to compute the objective function f and the gradient g at the point x.

The file 'funcdc.m' must be defined and contain: function [mode ,c ,dcS] = funcdc(x, Prob, mode, nstate) to compute the nonlinear constraint value c and the constraint Jacobian dcS for the nonlinear constraints at the point x.

[ifail, x_k, f_k, c_k, v_k, lws, istat, rstat] = filSQPs(A, bl, bu, nnCon, x_0, Scale, scmode, fLow, MaxIter, rho, mlp, kmax, maxf, WarmStart, lws, istat, PriLev, pname, optPar, Prob, moremem);

The sparse version MEX is filSQPs, the dense is filSQPd.

Description of Inputs

The following fields are used:

A	Gradient matrix [g ConsPattern' A'] (sparse or dense).

The following fields are used:, continued

bl	Lower bounds on (x,c(x),Ax).
bu	Upper bounds on (x,c(x),Ax).
nnCon	Number of nonlinear constraints (i.e. length(c(x)).
x_0	Initial x vector (if empty set as 0).
Scale	n+m vector scale factors for variables and constraints (same ordering as bl, bu).
scmode	Scale mode: 0 - unit variable and constraint scaling (Scale can be set empty). 1 - User provided scale factors for variables. Scale must be of length n. 2 - Unit variable scaling, user provided constraint scaling. Scale must be of length n+m, but only the last m elements are used. 3- User provided variable AND constraint scaling. Scale must be of length n+m (n+nnCon+nnLin)
fLow	A lower bound on the objective function value.
MaxIter	Maximum number of iterations.
rho	Initial trust-region radius.
mlp	Maximum level parameter for resolving degeneracy in BQPD.
kmax	Maximum size of null-space (at most n).
maxf	Maximum size of the filter.
WarmStart	Set to 1 to restart the solver. If a warmstart is requested, the input parameters lws and istat must be provided. Also, n and m (the number of variables and constraints) may not change.
lws	Used only when doing a warmstart. This must be the lws vector returned by the previous call to filterSQP. Otherwise, set to empty.
lam	Multipliers, n+m values required for warmstarts. If wrong length, zeros are set in the MEX interface.
istat	Used only when doing a warmstart. Must be the first element of the istat vector returned by the previous call to filterSQP. Otherwise, set to empty.

The following fields are used:, continued

PriLev	Print level. See also input parameter pname. 0 - Silent, except for minor output into ¡pname¿.out 1 - One line per iteration 2 - Scalar information printed 3 - Scalar and vector information printed
pname	Problem name, at most 10 characters. The output files are named <pname>.sum and <pname>.out.
optPar	Vector of max length 20 with optimization parameters: If any element is -999, default value is assigned. Elements 2-8 are BQPD parameters, 1,9-11,19-20 for filterSQP.
optPar(1): iprint	Print level in filterSQP, default 0.
optPar(2): tol	Relative tolerance for BQPD subsolver, default 1E-10.
optPar(3): emin	1=Use cscale in BQPD, 0=no scaling , default 1.0.
optPar(4): sgnf	Max rel error in two numbers equal in exact arithmetic (BQPD), default 5E-4.
optPar(5): nrep	Max number of refinement steps (BQPD), default 2.
optPar(6): npiv	No repeat if no more than npiv steps were taken, default 3.
optPar(7): nres	Max number of restarts if unsuccessful, default 2.
optPar(8): nfreq	The max interval between refactorizations, default 500.
optPar(9): NLP_eps	NLP subproblem tolerance, default 1E-6.
optPar(10): ubd	Upper bound on constraint violation used in the filter, default 1E-2.
optPar(11): tt	Parameter related to ubd. The actual upper bound is defined by the maximum of ubd and tt multiplied by the initial constraint violation., default 0.125.
optPar(19): infty	A large value representing infinity, default 1E20.
optPar(20): Nonlin	If 1, skip linear feasibility tests filterSQP treating all constraints as nonlinear, default 0.
Prob	The Tomlab problem definition structure.
moremem	Scalar or 2x1-vector with workspace increase. If <0, use default strategy. If scalar, use same increase for both real and integer workspaces. If vector, first element is for real workspace, second for integer.

Description of Outputs

The following fields are used:

Inform	Exit flag indicating success or failure: 0 - Solution found 1 - Unbounded: feasible point x with f(x)<=fmin found

The following fields are used:, continued

	2 - Linear constraints are infeasible 3 - (Locally) nonlinear infeasible, optimal solution to feasibility problem found 4 - Terminated at point with h(x)<=eps but QP infeasible 5 - Terminated with rho<=eps 6 - Terminated due to too many iterations 7 - Crash in user routine could not be resolved 8 - Unexpected failure in QP solver 9 - Not enough real workspace 10 - Not enough integer workspace
x_k	Solution vector (n+m by 1) with n decision variable values together with the m slack variables.
f_k	Function value at optimum x_k
c_k	Nonlinear constraints vector at optimum.
v_k	Lagrange multipliers vector (bounds, nonlinear, linear).
lws	Integer vector (used as input when doing warmstarts).
istat	Solution statistics, integer values. First element is required as input if doing a warmstart.
istat(1)	Dimension of nullspace at solution
istat(2)	Number of iterations
istat(3)	Number of feasibility iterations
istat(4)	Number of objective evaluations
istat(5)	Number of constraint evaluations
istat(6)	Number of gradient evaluations
istat(7)	Number of Hessian evaluations
istat(8)	Number of QPs with mode<=2
istat(9)	Number of QPs with mode>=4
istat(10)	Total number of QP pivots
istat(11)	Number of SOC steps
istat(12)	Maximum size of filter
istat(13)	Maximum size of Phase 1 filter
istat(14)	Number of QP crashes
rstat	Solution statistics, real values.
rstat(1)	l_2 norm of KT residual
rstat(3)	Largest modulus multiplier
rstat(4)	l_inf norm of final step
rstat(5)	Final constraint violation h(x)

The following fields are used:, continued

3.3.2 Using TOMLAB

Purpose
minlpBBTL solves mixed-integer nonlinear optimization problems defined as

$$\begin{array}{cccccc} \min\limits_{x} & f(x) & & & & \\ & & & & & \\ & x_L & \leq & x & \leq & x_U, \\ s/t & b_L & \leq & Ax & \leq & b_U \\ & c_L & \leq & c(x) & \leq & c_U \end{array} \tag{8}$$

where $x \in \mathbb{R}^n$, $f(x) \in \mathbb{R}$, $A \in \mathbb{R}^{m_1 \times n}$, $b_L, b_U \in \mathbb{R}^{n+m_1+m_2}$ and $c(x) \in \mathbb{R}^{m_2}$.
The variables $x \in I$, the index subset of $1, ..., n$, are restricted to be integers.

In addition, Special Ordered Sets of type 1 (SOS1) can be defined.
The algorithm uses a branch-and-bound scheme with a depth-first search strategy. The NLP relaxations are solved using the solver filterSQP by R. Fletcher and S. Leyffer.

Calling Syntax
Using the driver routine *tomRun*:

Prob = ⋄Assign(...);
Result = tomRun('minlpBB', Prob ...);

or

Result = minlpBBTL(Prob).

Call Prob = ⋄Assign(...) or Prob=ProbDef; to define the Prob for the second option.

Description of Inputs

Prob, The following fields are used:

A	Linear constraints coefficient matrix.
x_L, x_U	Bounds on variables.
b_L, b_U	Bounds on linear constraints.
c_L, c_U	Bounds on nonlinear constraints.
LargeScale	If 1 use sparse version of solver. The default is 0, the dense version.
PriLevOpt	Print level in the minlpBB solver.
optParam.MaxIter	Maximum number of iterations.

Prob, The following fields are used:, continued

MIP	Structure with fields defining the integer properties of the problem. The following fields are used:
IntVars	Vector designating which variables are restricted to integer values. This field is interpreted differently depending on the length. If length(IntVars) = length(x), it is interpreted as a zero-one vector where all non-zero elements indicate integer values only for the corresponding variable. A length less than the problem dimension indicates that IntVars is a vector of indices for the integer variables, for example [1 2 3 6 7 12]
VarWeight	Defines the priorities of the integer variables. Can be any values, but minlpBB uses integer priorities internally, with higher values implying higher priorities.
sos1	Structure defining the Special Ordered Sets of Type 1 (SOS1). If there are k sets of type sos1, then sos1(1).var is a vector of indices for variables in sos1, set 1. sos1(1).row is the row number for the reference row identifying the ordering information for the sos1 set, i.e. A(sos1(1).row,sos1(1).var) identifies this information sos1(1).prio sets the priority for sos1 test 1. sos1(2).var is a vector of indices for variables in sos1, set 2. sos1(2).row is the row number for the reference row of sos1 set 2. sos1(2).prio is the priority for sos1 set 2. sos1(k).var is a vector of indices for variables in sos1, set k. sos1(k).row is the row number for the reference row of sos1 set k. sos1(k).prio is the priority for sos1 set k.
DUNDEE	Structure with special fields for minlpBB optimization parameters. The following fields are used:
stackmax	Maximum size of the LIFO stack storing info about B&B tree. Default: 10000.
QPmin	Lower bound for the QP subproblems. Default: -1E300.
rho	Initial trust region radius. Default: 10.0 (REAL).
kmax	Maximum size of the null-space, less than or equal to no. of variables Default: n (INTEGER).
maxf	Maximum size of the filter Default: 100 (INTEGER).
mlp	Maximum level parameter for resolving degeneracy in BQPD QP subsolver..

Prob, The following fields are used:, continued

lam	Multipliers (n+m) on entry (NOTE: Experimental parameter).
Name	Problem name, at most 10 characters. The output files are named <pname>.sum and <pname>.out. Default name minlpBB, i.e. files minlpBB.sum, minlpBB.out.
optPar	Vector of max length 20 with optimization parameters. If any element is -999, default value is assigned. The elements used by minlpBB are:
optPar(1): iprint	Print level in minlpBB Summary on file minlpBB.sum. More printout on file minlpBB.out. Default 0.
optPar(2): tol	Relative tolerance for BQPD subsolver. Default 1E-10.
optPar(3): emin	1=Use cscale in BQPD, 0=no scaling. Default 1.0.
optPar(4): sgnf	Max rel error in two numbers equal in exact arithmetic (BQPD). Default 5E-4.
optPar(5): nrep	Max number of refinement steps (BQPD). Default 2.
optPar(6): npiv	No repeat if no more than npiv steps were taken. Default 3.
optPar(7): nres	Max number of restarts if unsuccessful. Default 2.
optPar(8): nfreq	The max interval between refactorizations. Default 500.
optPar(9): NLP_eps	NLP subproblem tolerance. Default 1E-6.
optPar(10): ubd	Upper bound on constraint violation used in the filter. Default 1E-2.
optPar(11): tt	Parameter related to ubd. The actual upper bound is defined by the maximum of ubd and tt multiplied by the initial constraint violation. Default 0.125.
optPar(12): epsilon	Tolerance for x-value tests. Default 1E-6.
optPar(13): MIopttol	Tolerance for function value tests. Default 1E-4.
optPar(17): branchtype	Branch strategy. Currently only 1 strategy.
optPar(19): infty	A large value representing infinity. Default 1E20.
optPar(20): Nonlin	If 1, skip linear feasibility tests minlpBB treating all constraints as nonlinear. Default 0.
morereal	Number of extra REAL workspace locations. Set to <0 for problem dependent default strategy.
moreint	Number of extra INTEGER workspace locations. Set to <0 for problem dependent default strategy.
	Scaling parameters. It is possible to supply scale factors for the variables and/or the constraints. Normally, the DUNDEE solvers does not differentiate between linear and nonlinear constraints with regard to scaling, but the TOMLAB interface handles this automatically. Thus is it possible to give scale factors e.g. for the nonlinear constraints only. The three parameters in the Prob.DUNDEE substructure that control scaling are:
xScale	Vector of scale factors for variables. If less than n values given, 1's are used for the missing elements.

Prob, The following fields are used:, continued

bScale	Vector of scale factors for the linear constraints. If length(bScale) is less than the number of linear constraints (size(Prob.A,1)), 1's are used for the missing elements.
cScale	Vector of scale factors for the nonlinear constraints. If length(cScale) is less than the number of nonlinear constraints, 1's are used for the missing elements.

Description of Outputs

Result, The following fields are used:

Result	The structure with results (see ResultDef.m).
f_k	Function value at optimum.
g_k	Gradient of the function.
x_k	Solution vector.
x_0	Initial solution vector.
c_k	Nonlinear constraint residuals.
cJac	Nonlinear constraint gradients.
xState	State of variables. Free == 0; On lower == 1; On upper == 2; Fixed == 3;
bState	State of linear constraints. Free == 0; Lower == 1; Upper == 2; Equality == 3;
cState	State of nonlinear constraints. Free == 0; Lower == 1; Upper == 2; Equality == 3;
v_k	Lagrangian multipliers (for bounds + dual solution vector).
ExitFlag	Exit status.
Inform	minlpBB information parameter: 0 - Optimal solution found 1 - Root problem infeasible 2 - Integer infeasible 3 - Stack overflow - some integer solution. obtained 4 - Stack overflow - no integer solution obtained 5 - SQP termination with rho < eps 6 - SQP termination with iter > max_iter 7 - Crash in user supplied routines 8 - Unexpected ifail from QP solvers This is often due to too little memory being allocated and is remedied by setting appropriate values in the Prob.DUNDEE.morereal and Prob.DUNDEE.moreint parameters.

Result, The following fields are used:, continued

	9 - Not enough REAL workspace or parameter error
	10 - Not enough INTEGER workspace or parameter error
rc	Reduced costs. If ninf=0, last m == -v_k.
Iter	Number of iterations.
FuncEv	Number of function evaluations.
GradEv	Number of gradient evaluations.
ConstrEv	Number of constraint evaluations.
QP.B	Basis vector in TOMLAB QP standard.
Solver	Name of the solver (minlpBB).
SolverAlgorithm	Description of the solver (sparse or dense, mainly).

3.4 miqpBB

The solver miqpBB solves sparse and dense mixed-integer linear and quadratic programs. The package implements the Branch and Bound method with some special features such as the computation of improved lower bounds and hot starts for the QP subproblems. miqpBB allows the user to influence the choice of branching variable in two ways: Firstly by employing user supplied priorities in the branching decision and secondly by supplying a choice of branching routines. The package is also efficient as an MILP solver.

3.4.1 Direct Solver Call

A direct solver call is not recommended unless the user is 100 % sure that no other solvers will be used for the problem. Please refer to Section 3.4.2 for information on how to use miqpBB with TOMLAB.

Purpose
miqpBB solves mixed-integer quadratic optimization problems defined as

$$\begin{array}{cccccl} \min\limits_{x} & f(x) = \frac{1}{2}x^T F x + c^T x & & & & \\ & & & & & \\ s/t & & & x & , & \\ & b_L & \leq & Ax & \leq & b_U \end{array} \qquad (9)$$

where $c, x \in \mathbb{R}^n$, $F \in \mathbb{R}^{n\times n}$, $A \in \mathbb{R}^{m_1\times n}$, and $b_L, b_U \in \mathbb{R}^{n+m_1}$.

The variables $x \in I$, the index subset of $1, ..., n$, are restricted to be integers.

If F is empty, an LP or MILP problem is solved.

Calling Syntax

[Inform, x_k, Obj, Iter] = miqpbb(A, bl, bu, IntVars, Priority, Func, mlp, kmax, stackmax, optPar, PriLev, PrintFile, Prob, moremem);

Description of Inputs

The following fields are used:

[c A']	Linear constraint matrix, dense or sparse n x (m+1) matrix. miqpBB requires the transpose of the constraint matrix, and also with the linear part of the objective function as the first column.
bl, bu	Lower and upper bounds on variables and constraints. Length must be n+m where the first n elements are simple bounds on the variables.
IntVars	Vector with integer variable indices.
Priority	Priorities for the integer variables. Length must the same as that of IntVars.

The following fields are used:, continued

Func	Name of MATLAB callback function that performs the Hessian - vector multiplication F*x. A standard routine is supplied in tomlab/lib/HxFunc.m, using the Prob.QP.F matrix. If the user for some reason wants to write his own callback function, it must take arguments similar to those of HxFunc.m. The second argument nState is always 0.0 in the current version of the solver.
mlp	Maximum level parameter for resolving degeneracy in BQPD which is used as sub-problem solver. If empty, the MEX interface sets mlp to m, the number of constraints.
kmax	Maximum dimension of reduced space. Default (and maximum) value is n, the number of variables.
stackmax	Size of the stack storing information during the tree-search. Default value if empty: 5000.
optPar	Vector of optimization parameters. If -999, set to default. Length from 0 to 20 allowed. The following elements are used by miqpBB:
optPar(1): iprint	Print level in miqpbb, default 0.
optPar(2): tol	Relative accuracy in solutio, default 1E-10.
optPar(3): emin	1.0 Use cscale (constraint scaling) 0.0 no scaling, default 1.0.
optPar(4): sgnf	Max rel error in two numbers equal in exact arithmetic, default 5E-4.
optPar(5): nrep	Max number of refinement steps, default 2.
optPar(6): npiv	No repeat if no more than npiv steps were taken, default 3.
optPar(7): nres	Max number of restarts if unsuccessful, default 2.
optPar(8): nfreq	The max interval between refactorizations, default 500.
optPar(12): epsilon	Tolerance used for x value tests, default 1E-5.
optPar(13): MIopttol	Tolerance used for function value tests, default 1E-4.
optPar(14): fIP	Upper bound on f(x). Only consider solutions < fIP - epsilon, default 1E20.
optPar(15): timing	No Timing = 1, Use timing, = 0. Default, no timing
optPar(16): max_time	Maximal time allowed for the run in seconds, default 4E3.
optPar(17): branchType	Branch on variable with highest priority. If tie: = 1. Variable with largest fractional part, among those branch on the variable giving the largest increase in the objective. = 2. Tactical Fletcher (PMO) branching. The var that solves max(min(e+,e-)) is chosen. The problem than corresponding to min(e+,e-) is placed on the stack first.

The following fields are used:, continued

	= 3. Tactical branching, Padberg/Rinaldi,91, Barahona et al.,89 (i) Choose the branching variable the one that most violates the integrality restrictions. i.e. find max(i)min(pi+,pi-) pi+ = int(x(i)+1) - x(i) , pi- = x(i) - int(x(i)) (ii) among those branch on the variable that gives the greatest increase in the obj. function (iii) Finally a LOWER BOUND is computed on the branched problems using the bounding method of Fletcher and Leyffer (dual active set step) DEFAULT = 1.
optPar(18): ifsFirst	If 1, then only search for first ifs (ifail=6), Default 0.
optPar(19): infty	Real value for infinity, default 1E20.
PriLev	Print level in the MEX interface: 0 = off, 1 = only result is printed, 2 = result and intermediate steps are printed. scalar information, 3 = verbose).
PrintFile	Name of print file. Amount/print type determined by PriLev parameter. Default name miqpbbout.txt.
Prob	The Tomlab problem description structure. This is a necessary argument if the standard HxFunc.m callback routine is used. HxFunc uses Prob.QP.F to calculate the Hessian*vector multiplication.
moremem	Scalar or 2x1-vector giving values for extra work memory allocation. If scalar, the value given is added to both the INTEGER and REAL workspaces. If a vector is given, the first element controls the REAL workspace increase and the second the INTEGER workspace. Set one or both elements to values ¡0 for problem dependent memory increases.

Description of Outputs

The following fields are used:

ifail Status code: the following values are defined:

0 - Solution found
1 - Error in parameters for BQPD
2 - Unbounded QP encountered
3 - Stack overflow - no integer solution found
4 - Stack overflow - some integer solution found
5 - Integer infeasible
6 - (on I/O) only search for first ifs and stop
7 - Infeasible root problem

x_k The solution vector, if any found. If ifail is other than 0 or 4, the contents of x is undefined.

The following fields are used:, continued

Obj	The value of the objective function at x_k.
iter	The number of iterations used to solve the problem.

3.4.2 Using TOMLAB

Purpose
miqpBBTL solves mixed-integer quadratic optimization problems defined as

$$
\begin{array}{cl}
\min\limits_{x} & f(x) = \frac{1}{2}x^T F x + c^T x \\
 & \\
s/t & \begin{array}{lcccl} x_L & \leq & x & \leq & x_U, \\ b_L & \leq & Ax & \leq & b_U \end{array}
\end{array}
\tag{10}
$$

where $c, x, x_L, x_U \in \mathbb{R}^n$, $F \in \mathbb{R}^{n\times n}$, $A \in \mathbb{R}^{m_1\times n}$, and $b_L, b_U \in \mathbb{R}^{m_1}$.
The variables $x \in I$, the index subset of $1, ..., n$, are restricted to be integers.

If F is empty, an LP or MILP problem is solved.

miqpBBTL converts the problem from the Tomlab structure format and calls either miqpBBs (sparse) or miqpBBd (dense). On return converts the result to the Tomlab structure format.

Calling Syntax
Using the driver routine *tomRun*:

Prob = ⋄Assign(...);
Result = tomRun('miqpBB', Prob ...);

or

Result = miqpBBTL(Prob);

Call Prob = ⋄Assign(...) or Prob=ProbDef; to define the Prob for the second option.

Description of Inputs

Prob, The following fields are used:

x_L, x_U	Lower and upper bounds on variables.
b_L, b_U	Lower and upper bounds on linear constraints.
A	Linear constraint matrix, dense or sparse m x n matrix.
QP.c	Linear coefficients in objective function, size n x 1.
QP.F	Quadratic matrix of size n x n.
PriLevOpt	Print Level (0 = off, 1 = summary, 2 = scalar information, 3 = verbose). > 10 Pause statements, and maximal printing (debug mode)
LargeScale	If TRUE (=1), use sparse version, otherwise dense.

Prob, The following fields are used:, continued

optParam.MaxIter	Limit of iterations.
MIP.IntVars	Defines which variables are integers. Variable indices should be in the range [1,...,n]. IntVars is a logical vector ==> x(find(IntVars > 0)) are integers. IntVars is a vector of indices ==> x(IntVars) are integers (if [], then no integers of type I or B are defined).
MIP.VarWeight	Variable priorities. Lower value means higher priority.
DUNDEE.kmax	Max dimension of reduced space (k), default n, set as 0 if LP.
DUNDEE.mlp	Maximum number of levels of recursion.
DUNDEE.stackmax	Maximum size of the LIFO stack storing info about B&B tree. Default 5000.
DUNDEE.PrintFile	Name of print file. Amount/print type determined by optPar(1) Default name miqpBBout.txt.
DUNDEE.optPar	Vector of optimization parameters. If -999, set to default Length from 0 to 20 allowed.
DUNDEE.optPar(1)	Print level in miqpBB. = 0 Silent = 1 Warnings and Errors = 2 Summary information = 3 More detailed information
DUNDEE.optPar(2): tol	Relative accuracy in solution, default 1E-10.
DUNDEE.optPar(3): emin	1.0 Use cscale (constraint scaling) 0.0 no scaling, default 1.0.
DUNDEE.optPar(4): sgnf	Max rel error in two numbers equal in exact arithmetic, default 5E-4.
DUNDEE.optPar(5): nrep	Max number of refinement steps, default 2.
DUNDEE.optPar(6): npiv	No repeat if no more than npiv steps were taken, default 3.
DUNDEE.optPar(7): nres	Max number of restarts if unsuccessful, default 2.
DUNDEE.optPar(8): nfreq	The max interval between refactorizations, default 500.
DUNDEE.optPar(12): epsilon	Tolerance used for x value tests, default 1E-5.
DUNDEE.optPar(13): MIopttol	Tolerance used for function value tests, default 1E-4.
DUNDEE.optPar(14): fIP	Upper bound on f(x). Only consider solutions < fIP - MIopttol, default 1E20.
DUNDEE.optPar(15): timing	If 1 - use timing, if 0 no timing (default).
DUNDEE.optPar(16): max_time	Maximal time allowed for the run in seconds, default 4E3.
DUNDEE.optPar(17): branchType	Branch on variable with highest priority. If tie: = 1. Variable with largest fractional part, among those branch on the variable giving the largest increase in the objective. = 2. Tactical Fletcher (PMO) branching. The var that solves max(min(e+,e-)) is chosen. The problem than corresponding to min(e+,e-) is placed on the stack first.

Prob, The following fields are used:, continued

	= 3. Tactical branching, Padberg/Rinaldi,91, Barahona et al.,89. (i) Choose the branching variable the one that most violates the integrality restrictions. i.e. find max(i)min(pi+,pi-) pi+ = int(x(i)+1) - x(i) , pi- = x(i) - int(x(i)) (ii) among those branch on the variable that gives the greatest increase in the obj. function (iii) Finally a LOWER BOUND is computed on the branched problems using the bounding method of Fletcher and Leyffer (dual active set step) DEFAULT = 1.
DUNDEE.optPar(18): ifsFirst	If 1, then only search for first ifs (ifail=6), DEFAULT 0.
DUNDEE.optPar(19): infty	Real value for infinity (default 1E20).
DUNDEE.morereal	Number of extra REAL workspace locations. Set to <0 for problem dependent default value.
DUNDEE.moreint	Number of extra INTEGER workspace locations. Set to <0 for problem dependent default value.

Description of Outputs

Result, The following fields are used:

Result	The structure with results (see ResultDef.m).
f_k	Function value at optimum.
x_k	Solution vector.
x_0	Initial solution vector.
g_k	Gradient of the function.
xState	State of variables. Free == 0; On lower == 1; On upper == 2; Fixed == 3;
bState	State of linear constraints. Free == 0; Lower == 1; Upper == 2; Equality == 3;
v_k	Lagrangian multipliers (for bounds + dual solution vector).
ExitFlag	Exit status from miqpBB.m (similar to TOMLAB).
Inform	miqpBB information parameter. 0 - Optimal solution obtained 1 - Error in parameters for BQPD 2 - Unbounded QP encountered 3 - Stack overflow NO ifs found 4 - Stack overflow some ifs obtained 5 - Integer infeasible

Result, The following fields are used:, continued

	6 - (on I/O) only search for first ifs and stop
	7 - Infeasible root problem
rc	Reduced costs. NOT SET.
Iter	Number of iterations.
FuncEv	Number of function evaluations. Set to Iter.
GradEv	Number of gradient evaluations. Set to Iter.
ConstrEv	Number of constraint evaluations. Set to 0.
QP.B	Basis vector in TOMLAB QP standard.
MinorIter	Number of minor iterations. NOT SET.
Solver	Name of the solver (miqpBB)
SolverAlgorithm	Description of the solver.
DUNDEE.kmax	Max dimension of reduced space (k), default n, set as 0 if LP.
DUNDEE.mlp	Maximum number of levels of recursion.
DUNDEE.stackmax	Maximum size of the LIFO stack storing info about B&B tree.
DUNDEE.mode	Mode of operation, default set as 2*Prob.WarmStart.
DUNDEE.x	Solution (Warm Start).
DUNDEE.k	Dimension of the reduced space (Warm Start).
DUNDEE.e	Steepest-edge normalization coefficients (Warm Start).
DUNDEE.ls	Indices of active constraints, first n-k used for warm start.
DUNDEE.lp	List of pointers to recursion information in ls (Warm Start).
DUNDEE.peq	Pointer to the end of equality constraint indices in ls (Warm Start).

4 The DUNDEE structure

Table 19: Information stored in the structure *Prob.DUNDEE*

Field	Description
callback	If 1, use a callback to Matlab to compute $QP.F \cdot x$ in *BQPD* and *miqpBB* . Faster when F is large and nearly dense. Avoids copying the matrix to the MEX solvers.
kmax	Maximum dimension of the reduced space (k), default equal to dimension of problem. Set to 0 if solving an LP problem.
mlp	Maximum number of levels of recursion.
mode	Mode of operation, default set as $2 * Prob.WarmStart$.
x	Solution (warmstart).
k	Dimension of reduced space (warmstart).
e	Steepest-edge normalization coefficient (warmstart).
ls	Indices of active constraints, first $n - k$. (warmstart).
lp	List of pointers to recursion information in *ls* (warmstart).
peq	Pointer to end of equality constraint indices in *ls* (warmstart).
PrintFile	Name of print file. Amount/print type is determined by *Prob.DUNDEE.optPar(1)*.
optPar	Vector with optimization parameters. Described in Table 20.

Table 20: *Prob.DUNDEE.optPar* values used by TOMLAB /MINLP solvers . -999 in any element gives default value.

				Used by:			
Index	**Name**	**Default**	**Description**	BQPD	miqpBB	filterSQP	minlpBB
1	iprint	0	Print level in DUNDEE solvers.	O	O	O	O
2	tol	10^{-10}	Relative accuracy in BQPD solution.	O	O	O	O
3	emin	1	1/0: Use/do not use constraint scaling in BQPD	O	O	O	O
4	sgnf	$5 \cdot 10^{-4}$	Maximum relative error in two numbers equal in exact arithmetic.	O	O	O	O
5	nrep	2	Maximum number of refinement steps.	O	O	O	O
6	npiv	3	No repeat of more than *npiv* steps were taken.	O	O	O	O
7	nres	2	Maximum number of restarts if unsuccessful.	O	O	O	O
8	nfreq	500	Maximum interval between refactorizations.	O	O	O	O
9	ubd	100	Constraint violation parameter I	-	-	O	O
10	tt	0.125	Constraint violation parameter II	-	-	O	O
11	NLP_eps	10^{-6}	Relative tolerance for NLP solutions	-	-	O	O
12	epsilon	10^{-6}	Accuracy for x tests	O	O	O	O
13	MIopttol	10^{-4}	Accuracy for f tests	-	O	-	O
14	fIP	10^{20}	Upper bound on the IP value wanted.	-	O	-	O
15	timing	0	1/0: Use/do not use timing	-	O	-	-
16	max_time	4000	Maximum time (sec's) allowed for the run	-	O	-	-
17	branchtype	1	Branching strategy $(1, 2, 3)$	-	O	-	-
18	ifsFirst	0	Exit when first IP solution is found	-	O	-	-
19	infty	10^{20}	Large value used to represent infinity	O	O	O	O
20	Nonlin	0	Treat all constraints as nonlinear if 1	-	-	O	O

5 Algorithmic description

5.1 Overview

Classical methods for the solution of MINLP problems decompose the problem by separating the nonlinear part from the integer part. This approach is largely due to the existence of packaged software for solving Nonlinear Programming (NLP) and Mixed Integer Linear Programming problems.

In contrast, an integrated approach to solving MINLP problems is considered here. This new algorithm is based on branch-and-bound, but does not require the NLP problem at each node to be solved to optimality. Instead, branching is allowed after each iteration of the NLP solver. In this way, the nonlinear part of the MINLP problem is solved whilst searching the tree. The nonlinear solver that is considered in this manual is a Sequential Quadratic Programming solver.

A numerical comparison of the new method with nonlinear branch-and-bound is presented and a factor of $\boxed{\text{up to}}$ 3 improvement over branch-and-bound is observed.

5.2 Introduction

This manual considers the solution of Mixed Integer Nonlinear Programming (MINLP) problems. Problems of this type arise when some of the variables are required to take integer values. Applications of MINLP problems include the design of batch plants (e.g. [17] and [22]), the synthesis of processes (e.g. [7]), the design of distillation sequences (e.g. [29]), the optimal positioning of products in a multi-attribute space (e.g. [7]), the minimization of waste in paper cutting [28] and the optimization of core reload patterns for nuclear reactors [1]. For a comprehensive survey of applications of MINLP applications see Grossmann and Kravanja [16].

MINLP problems can be modelled in the following form

$$(P)\left\{\begin{array}{ll} \underset{x,y}{\text{minimize}} & f(x,y) \\ \text{subject to} & g(x,y) \le 0 \\ & x \in X,\ y \in Y \text{ integer,} \end{array}\right.$$

where y are the integer variables modelling for instance decisions ($y \in \{0,1\}$), numbers of equipment ($y \in \{0,1,\ldots,N\}$) or discrete sizes and x are the continuous variables.

Classical methods for the solution of (P) "decompose" the problem by separating the nonlinear part from the integer part. This approach is $\boxed{\text{made possible by}}$ the existence of packaged software for solving Nonlinear Programming (NLP) and Mixed Integer Linear Programming (MILP) problems. The decomposition is even more apparent in Outer Approximation ([7] and [10]) and Benders Decomposition ([15] and [13]) where an alternating sequence of MILP master problems and NLP subproblems obtained by fixing the integer variables is solved (see also the recent monograph of Floudas [14]). The recent branch-and-cut method for 0-1 convex MINLP of Stubbs and Mehrotra [27] also separates the nonlinear (lower bounding and cut generation) and integer part of the problem.

Branch-and-bound [6] can also be viewed as a "decomposition" in which the tree-search (integer part) is largely separated from solving the NLP problems at each node (nonlinear part).

In contrast to these approaches, we considers an integrated approach to MINLP problems. The algorithm developed here is based on branch-and-bound, but instead of solving an NLP problem at each node of the tree, the tree-search and the iterative solution of the NLP are interlaced. Thus the nonlinear part of (P) is solved whilst searching the tree. The nonlinear solver that is considered in this manual is a Sequential Quadratic Programming (SQP) solver. SQP solves an NLP problem via a sequence of quadratic programming (QP) problem approximations.

The basic idea underlying the new approach is to branch early – possibly after a single QP iteration of the SQP solver. This idea was first presented by Borchers and Mitchell [4]. The present algorithm improves on their idea in a number of ways:

1. In order to derive lower bounds needed in branch-and-bound Borchers and Mitchell propose to evaluate the Lagrangian dual. This is effectively an unconstrained nonlinear optimization problem. In Section 5.4 it is shown that there is no need to compute a lower bound if the linearizations in the SQP solver are interpreted as cutting planes.
2. In [4] *two* QP problems are solved before branching. This restriction is due to the fact that a packaged SQP solver is used. By using our own solver we are able to remove this unnecessary restriction, widening the scope for early branching.
3. A new heuristic for deciding when to branch early is introduced which does not rely on a *fixed* parameter, but takes the second order rate of convergence of the SQP solver into account when deciding whether to branch or continue with the SQP iteration.

The ideas presented here have a similar motivation as a recent algorithm by Quesada and Grossmann [26]. Their algorithm is related to outer approximation but avoids the re-solution of related MILP master problems by interrupting the MILP branch-and-bound solver each time an integer node is encountered. At this node an NLP problem is solved (obtained by fixing all integer variables) and new outer approximations are added to all problems on the MILP branch-and-bound tree. Thus the MILP is updated and the tree-search resumes. The difference to the approach considered here is that Quesada and Grossmann still solve NLP problems at some nodes, whereas the new solver presented here usually solves one quadratic programming (QP) problem at each node.

The algorithmic description is organized as follows. Section 5.3 briefly reviews the SQP method and branch-and-bound. In Section 5.4 the new algorithm is developed for the special case of convex MINLP problems. Section 5.4.4 introduces a heuristic for handling nonconvex MINLP problems. Finally, Section 5.5 presents a comparison of the new method with nonlinear branch-and-bound. These results are very encouraging, often improving on branch-and-bound by a factor of up to 3.

5.3 Background

5.3.1 Sequential Quadratic Programming

A popular iterative method for solving NLP problems is Sequential Quadratic Programming (SQP). The basic form of SQP methods date back to Wilson [30] and were popularized by Han [19] and Powell [25], see Fletcher [9, Chapter 12.4] and Conn, Gould and Toint [3] for a selection of other references. In its basic form, SQP solves an NLP problem via a sequence of quadratic programming (QP) approximations obtained by replacing the nonlinear constraints by a linear first order Taylor series approximation and the nonlinear objective by a second order Taylor series approximation augmented by second order information from the constraints. It can be shown under certain conditions that the SQP method converges quadratically near a solution.

It is well known that the SQP method may fail to converge if it is started far from a local solution. In order to induce convergence, many popular methods use a *penalty function* which is a linear combination of the objective function f and some measure of the constraint violation. A related idea is an *augmented Lagrangian function* in which a weighted penalty term is added to a Lagrangian function. A step in an SQP method is then accepted when it produces a sufficient decrease in the penalty function.

Two frameworks exist which enforce sufficient decrease, namely line-search in the direction of the QP solution or a trust-region that limits the QP step that is computed (see e.g. [9, Chapter 12.4] and references therein). In our implementation global convergence is promoted through the use of a trust-region and the new concept of a "filter" [11] which accepts a trial point whenever the objective or the constraint violation is improved compared to all previous iterates. The size of the trust-region is reduced if the step is rejected and increased if it is accepted.

5.3.2 Branch-and-bound

Branch-and-bound dates back to Land and Doig [23]. The first reference to nonlinear branch-and-bound can be found in Dakin [6]. It is most conveniently explained in terms of a tree-search.

Initially, all integer restrictions are relaxed and the resulting NLP relaxation is solved. If all integer variables take an integer value at the solution then this solution also solves the MINLP. Usually, some integer variables take a non-integer value. The algorithm then selects one of those integer variables which take a non-integer value, say y_i with value $\hat{y}_i$, and branches on it. Branching generates two new NLP problems by adding simple bounds $y_i \leq [\hat{y}_i]$ and $y_i \geq [\hat{y}_i] + 1$ respectively to the NLP relaxation (where $[a]$ is the largest integer not greater than a).

One of the two new NLP problems is selected and solved next. If the integer variables take non-integer values then branching is repeated, thus generating a branch-and-bound tree whose nodes correspond to NLP problems and where an edge indicates the addition of a branching bound. If one of the following fathoming rules is satisfied, then no branching is required, the corresponding node has been fully explored (fathomed) and can be abandoned. The fathoming rules are

FR1 An infeasible node is detected. In this case the whole subtree starting at this node is infeasible and the node has been fathomed.

FR2 An integer feasible node is detected. This provides an upper bound on the optimum of the MINLP; no branching is possible and the node has been fathomed.

FR3 A lower bound on the NLP solution of a node is greater or equal than the current upper bound. In this case the node is fathomed, since this NLP solution provides a lower bound for all problems in the corresponding sub-tree.

Once a node has been fathomed the algorithm backtracks to another node which has not been fathomed until all nodes are fathomed.

Many heuristics exist for selecting a branching variable and for choosing the next problem to be solved after a node has been fathomed (see surveys by Gupta and Ravindran [18] and $\boxed{\text{Volkovich}}$ et.al. [24]).

Branch-and-bound can be inefficient in practice since it requires the solution of one NLP problem per node which is usually solved iteratively through a sequence of quadratic programming problems. Moreover, a large number of NLP problems are solved which have often no physical meaning if the integer variables do not take integer values.

5.4 Integrating SQP and branch-and-bound

An alternative to nonlinear branch-and-bound for *convex* MINLP problems is due to Borchers and Mitchell [4]. They observe that it is not necessary to solve the NLP at each node to optimality before branching and propose an *early branching rule*, which branches on an integer variable before the NLP has converged. Borchers and Mitchell implement this approach and report encouraging numerical results compared to Outer Approximation [5]. In their implementation the SQP solver is interrupted after two QP solves [1] and branching is done on an integer variable which is more than a given tolerance away from integrality.

[1]Ideally one would prefer to interrupt the SQP method after each QP solve. Borchers and Mitchell use an SQP method from the NAG library which does not allow this.

The drawback of the *early branching rule* is that since the NLP problems are *not* solved to optimality, there is no guarantee that the current value of $f(x,y)$ is a lower bound. As a consequence, bounding (fathoming rule **FR3**) is no longer applicable. Borchers and Mitchell seek to overcome this difficulty by evaluating the Lagrangian dual for a given set of multiplier estimates, λ. The evaluation of the Lagrangian dual amounts to solving

$$\left\{\begin{array}{ll} \underset{x,y}{\text{minimize}} & f(x,y) + \lambda^T g(x,y) \\ \text{subject to} & x \in X,\ y \in \hat{Y} \end{array}\right.$$

where the $\hat{Y} \subset Y$ now also includes bounds that have been added during the branching.

Note that this evaluation requires the solution of a bound constrained nonlinear optimization problem. In order to diminish the costs of these additional calculations, Borchers and Mitchell only evaluate the dual every sixth QP solve.

Clearly, it would be desirable to avoid the solution of this problem and at the same time obtain a lower bounding property at each QP solve. In the remainder of this section it is shown how this can be achieved by integrating the iterative NLP solver with the tree search.

Throughout this section it is assumed that both f and g are smooth, *convex* functions. The nonconvex case is discussed in Section 5.4.4 where a simple heuristic is proposed. The ideas are first presented in the context of a basic SQP method *without* globalization strategy. Next this basic algorithm is modified to allow for the use of a line-search or a trust-region. It is then shown how the early branching rule can take account of the quadratic rate of convergence of the SQP method. Finally, a simple heuristic for nonconvex MINLP problems is suggested.

5.4.1 The basic SQP algorithm

This section shows how the solution of the Lagrangian dual can be avoided by interpreting the constraints of the QP approximation as supporting hyperplanes. If f and g are convex, then the linear approximations of the QP are outer approximations and this property is exploited to replace the lower bounding.

Consider an NLP problem at a given node of the branch-and-bound tree,

$$(\hat{P}) \left\{\begin{array}{ll} \underset{x,y}{\text{minimize}} & f(x,y) \\ \text{subject to} & g(x,y) \le 0 \\ & x \in X,\ y \in \hat{Y}, \end{array}\right.$$

where the integer restrictions have been relaxed and $\hat{Y} \subset Y$ contains bounds that have been added by the branching. Let $\hat{f}$ denote the solution of $(\hat{P})$.

Applying the SQP method to $(\hat{P})$ results in solving a sequence of QP problems of the form

$$(QP^k) \left\{\begin{array}{ll} \underset{d}{\text{minimize}} & f^{(k)} + \nabla f^{(k)^T} d + \frac{1}{2} d^T W^{(k)} d \\ \text{subject to} & g^{(k)} + \nabla g^{(k)^T} d \le 0 \\ & x^k + d_x \in X,\ y^k + d_y \in \hat{Y}. \end{array}\right.$$

where $f^{(k)} = f(x^{(k)}, y^{(k)})$ etc. and

$$W^{(k)} \simeq \nabla^2 \mathcal{L}^{(k)} = \nabla^2 f^{(k)} + \sum \lambda_i \nabla^2 g_i^{(k)}$$

approximates the Hessian of the Lagrangian,
$\boxed{\text{where } \lambda_i \text{ is the Lagrange multiplier of } g_i(x,y) \le 0}$.

Unfortunately, the solution of (QP^k) does not underestimate $\hat{f}$, even if all functions are assumed to be convex. This implies that fathoming rule **FR3** cannot be used if early branching is employed. However, it turns out that it is not necessary to compute an underestimator explicitly. Instead, a mechanism is needed of terminating the sequence of (QP^k) problems once such an underestimator would become greater than the current upper bound, U, of the branch-and-bound process. This can be achieved by adding the $\boxed{\textit{objective cut}}$

$$f^{(k)} + \nabla f^{(k)^T} d \leq U - \epsilon$$

to (QP^k), where $\epsilon > 0$ is the optimality tolerance of branch-and-bound. Denote the new QP with this objective cut by (QP_c^k).

$$(QP_c^k) \left\{ \begin{array}{ll} \underset{d}{\text{minimize}} & f^{(k)} + \nabla f^{(k)^T} d + \frac{1}{2} d^T W^{(k)} d \\ \text{subject to} & g^{(k)} + \nabla g^{(k)^T} d \leq 0 \\ & f^{(k)} + \nabla f^{(k)^T} d \leq U - \epsilon \\ & x^k + d_x \in X, \; y^k + d_y \in \hat{Y}. \end{array} \right.$$

Note that (QP_c^k) is the QP that is generated by the SQP method when applied to the following NLP problem

$$\left\{ \begin{array}{ll} \underset{x,y}{\text{minimize}} & f(x,y) \\ \text{subject to} & g(x,y) \leq 0 \\ & f(x,y) \leq U - \epsilon \\ & x \in X, \; y \in \hat{Y}, \end{array} \right.$$

where a nonlinear objective cut has been added to $(\hat{P})$. This NLP problem is infeasible, if fathoming rule **FR3** holds at the solution to $(\hat{P})$.

It is now possible to replace fathoming rule **FR3** applied to $(\hat{P})$ by a test for feasibility of (QP_c^k) (replacing the bounding in branch-and-bound). This results in the following lemma.

Lemma 1 *Let f and g be smooth, convex functions. A sufficient condition for fathoming rule* **FR3** *applied to $(\hat{P})$ to be satisfied is that any QP problem (QP_c^k) generated by the SQP method in solving $(\hat{P})$ is infeasible.*

Proof: If (QP_c^k) is infeasible, then it follows that there exists no step d such that

$$f^{(k)} + \nabla f^{(k)^T} d \leq U - \epsilon \tag{11}$$

$$g^{(k)} + \nabla g^{(k)^T} d \leq 0. \tag{12}$$

Since (12) is an outer approximation of the nonlinear constraints of $(\hat{P})$ and (11) underestimates f, it follows that there exists no point $(\hat{x}, \hat{y}) \in X \times \hat{Y}$ such that

$$f(\hat{x}, \hat{y}) \leq U - \epsilon.$$

Thus any lower bound on f at the present node has to be larger than $U - \epsilon$. Thus to within the tolerance ϵ, $f \geq U$ and fathoming rule **FR3** holds. *q.e.d.*

In practice, an SQP method would use a primal active set QP solver which establishes feasibility first and then optimizes. Thus the test of Lemma 1 comes at no extra cost. The basic integrated SQP and branch-and-bound algorithm can now be stated.

Algorithm 1: Basic SQP and branch-and-bound

Initialization: Place the continuous relaxation of (P) on the tree and set $U = \infty$.

```
while (there are pending nodes in the tree) do
   Select an unexplored node.
   repeat (SQP iteration)
      1. Solve $(QP_c^k)$ for a step $d^{(k)}$ of the SQP method.
      2. if $((QP_c^k)$ infeasible) then fathom node, exit
      3. Set $(x^{(k+1)}, y^{(k+1)}) = (x^{(k)}, y^{(k)}) + (d_x^{(k)}, d_y^{(k)})$.
      4. if $((x^{(k+1)}, y^{(k+1)})$ NLP optimal) then
            if $(y^{(k+1)}$ integral) then
               Update current best point by setting
               $(x^*, y^*) = (x^{(k+1)}, y^{(k+1)})$, $f^* = f^{(k+1)}$ and $U = f^*$.
            else
               Choose a non-integral $y_i^{(k+1)}$ and branch.
            endif
            exit
         endif
      5. Compute the integrality gap $\theta = \max_i |y_i^{(k+1)} - \text{anint}(y_i^{(k+1)})|$.
      6. if $(\theta > \tau)$ then
            Choose a non-integral $y_i^{(k+1)}$ and branch, exit
         endif
end while
```

Here anint(y) is the nearest integer to y. Step 2 implements the fathoming rule of Lemma 1 and Step 6 is the early branching rule. In [4] a value of $\tau = 0.1$ is suggested for the early branching rule and this value has also been chosen here.

A convergence proof for this algorithm is given in the next section where it is shown how the algorithm has to be modified if a line-search or a trust-region is used to enforce global convergence for SQP. The key idea in the convergence proof is that (as in nonlinear branch-and-bound), the union of the child problems that are generated by branching is equivalent to the parent problem. The fact, that only a single QP has been solved in the parent problem does not alter this equivalence. Finally, the new fathoming rule implies **FR3** and hence, any node that has been fathomed by Algorithm 1 can also be considered as having been fathomed by nonlinear branch-and-bound. Thus convergence follows from the convergence of branch-and-bound.

Algorithm 1 has two important advantages over the work in [4]. Firstly, the lower bounding can be implemented at no additional cost (compared to the need to solve the Lagrangian dual in [4]). Secondly, the lower bounding is available at *all* nodes of the branch-and-bound tree (rather than at every sixth node).

Note that if no further branching is possible at a node, then the algorithm resorts to normal SQP at that node.

5.4.2 The globalized SQP algorithm

It is well known that the basic SQP method may fail to converge if started far from a solution. In order to obtain a globally convergent SQP method, descent in some merit function has to be enforced. There are two concepts which enforce descent: a line-search and a trust-region. This section describes how the basic algorithm of the previous section has to be modified to accommodate these global features.

The use of a line-search requires only to replace $d^{(k)}$ by $\alpha^{(k)} d^{(k)}$, where $\alpha^{(k)}$ is the step-length chosen in the line-search. Similarly a Levenberg-Marquardt approach would require minimal change to the algorithm.

An alternative to using a line-search is to use a trust region. Trust-region SQP methods usually possess stronger convergence properties and this leads us to explore this approach. In this type of approach the step that is computed in (QP_c^k) is restricted to lie within a trust-region. Thus the QP that is being solved at each iteration becomes

$$(QP_{c,\infty}^k)\left\{\begin{array}{ll} \underset{d}{\text{minimize}} & f^{(k)} + \nabla f^{(k)^T} d + \frac{1}{2} d^T W^{(k)} d \\ \text{subject to} & g^{(k)} + \nabla g^{(k)^T} d \leq 0 \\ & f^{(k)} + \nabla f^{(k)^T} d \leq U - \epsilon \\ & \|d\|_\infty \leq \rho^k \\ & x^k + d_x \in X,\ y^k + d_y \in \hat{Y} \end{array}\right.$$

where ρ^k is the trust-region radius which is adjusted according to how well the quadratic model of $(QP_{c,\infty}^k)$ agrees with the true function (see [9, Chapter 14.5] for details).

The drawback of using a trust-region in the present context is that the trust-region may cause $(QP_{c,\infty}^k)$ to be infeasible even though the problem without a trust-region (QP_c^k) has a non-empty feasible region. Thus Lemma 1 is no longer valid. However, if $(QP_{c,\infty}^k)$ is infeasible and the trust-region is inactive, then the argument of Lemma 1 still holds and the node can be fathomed.

Otherwise, the *feasibility restoration phase* has to be entered and this part of the SQP method is briefly described next. The aim of the restoration phase is to get closer to the feasible region by minimizing the violation of the constraints in some norm, subject to the linear constraints $x \in X,\ y \in \hat{Y}$.

$$(F)\left\{\begin{array}{ll} \underset{x,y}{\text{minimize}} & \|h^+(x,y)\| \\ \text{subject to} & x \in X,\ y \in \hat{Y}, \end{array}\right.$$

where

$$h(x,y) = \begin{pmatrix} f(x,y) - (U-\epsilon) \\ g(x,y) \end{pmatrix}$$

and the operation $a^+ = \max(0,a)$ is taken componentwise.

In this phase, an SQP algorithm is applied to minimize (F). Methods for solving this problem include Yuan [31] for the l_∞ norm, El-Hallabi and Tapia [8] for the l_2 norm and Dennis, El-Alem and Williamson [21] for the l_1 norm. These methods converge under mild assumptions similar to those needed for the convergence of SQP methods. The details of the restoration phase are beyond the scope of this manual and we concentrate instead on the interaction between this phase and branch-and-bound. In the remainder it will be assumed that this phase converges.

In order to obtain an efficient MINLP solver it is important to also integrate the restoration phase with the tree-search. After each QP in the restoration phase four possible outcomes arise.

1. $(QP^k_{c,\infty})$ is feasible. In this case the solver exits the restoration phase and resumes SQP and early branching.
2. $(QP^k_{c,\infty})$ is not feasible and a step $d^{(k)}$ is computed for which the trust-region is inactive ($\|d^{(k)}\| < \rho^k$). This indicates that the current node can be fathomed by Lemma 1.
3. The feasibility restoration converges to a minimum of the constraint violation (x^*, y^*) with $\|h^+(x^*, y^*)\| > 0$. This is taken as an indication that $(\hat{P})$ has no feasible point with an objective that is lower than the current upper bound and the corresponding node can be fathomed. Note that convergence of the feasibility SQP ensures that the trust-region will be inactive at (x^*, y^*).
4. The feasibility restoration SQP continues if none of the above occur.

Thus the feasibility restoration can be terminated before convergence, if 1 or 2 hold above. In the first case early branching resumes and in the second case, the node can be fathomed.

The modifications required to make Algorithm 1 work for trust-region SQP methods are given below.

Algorithm 2: Trust-region SQP and branch-and-bound

Only steps 1. and 2. need to be changed to:

```
1'. Compute an acceptable step $d^{(k)}$ of the trust-region SQP method.
2'. **if** ($(QP^k_{c,\infty})$ infeasible and $\|d^{(k)}\|_\infty < \rho^k$) **then**
        fathom node **exit**
    **elseif** ($(QP^k_{c,\infty})$ infeasible and $\|d^{(k)}\|_\infty = \rho^k$) **then**
        Enter feasibility restoration phase:
        **repeat** (SQP iteration for feasibility restoration)
            *(a)* Compute a step of the feasibility restoration.
            *(b)* **if** ($\|d^{(k)}\|_\infty < \rho^k$) **then** fathom node, **exit**
            *(c)* **if** ($(QP^k_{c,\infty})$ feasible) **then** return to normal SQP, **exit**
            *(d)* **if** ($\min \|g^+(x, y)\| > 0$) **then** fathom node, **exit**
    **endif**
```

The following theorem shows that Algorithm 2 converges under standard assumptions (e.g.[7] or [10]).

Theorem 1 *If f and g are smooth and convex functions, if X and Y are convex sets, if Y integer is finite and if the underlying trust-region SQP method is globally convergent, then Algorithm 2 terminates after visiting a finite number of nodes at the solution (x^*, y^*) of (P).*

Proof: The finiteness of Y integer implies that the branch-and-bound tree is finite. Thus the algorithm must terminate after visiting a finite number of nodes. Nodes are only fathomed if they are infeasible, integer feasible or if the lower bounding of Lemma 1 holds. Thus an optimal node must eventually be solved and convergence follows from the convergence of the trust-region SQP method. *q.e.d.*

5.4.3 Quadratic convergence and early branching

When implementing Algorithm 2 the following adverse behaviour has been observed on some problems.

1. Some integer variables converge to a non-integral solution, namely $y_i^{(k)} \to \hat{y}_i$ but $\theta \simeq 0.02 < \tau$, i.e. the integrality gap remains bounded away from zero.
2. The SQP solver converges at second order rate during this time and $\theta \gg \|d^{(k)}\|_\infty \to 0$.

In other words the early branching heuristic is not activated during the second order convergence of the SQP method so that no early branching takes place in the final stages of the SQP solver (as $\tau = 0.1$ is too large).

One way to avoid this would be to reduce τ, but this would simply repeat the problem at a smaller threshold. Choosing τ to be of the order of the tolerance of the SQP solver on the other hand would trigger early branching more often which could duplicate the work to be done if integer variables are converging to an integral solution.

These observations lead to a new early branching heuristic which takes the quadratic rate of convergence of the SQP method into account. Steps 5. and 6. of Algorithm 2 are replaced by

```
5'. Compute the integrality gap $\theta = \max_i |y_i^{(k+1)} - \text{anint}(y_i^{(k+1)})|$
    and the experimental order of convergence $p := \ln(\|d^{(k)}\|_\infty)/\ln(\|d^{(k-1)}\|_\infty)$.
6'. if ($\theta > \tau$ or ($p > 1.5$ and $\|d^{(k)}\|_\infty < \theta$ )) then
        Choose a non-integral $y_i^{(k+1)}$ and branch, exit
    endif
```

For a quadratically convergent method one expects that $p \simeq 2$ and once quadratic convergence sets in, that $\|d^{(k+1)}\|_\infty \simeq \|d^{(k)}\|_\infty^p$. Thus if $\|d^{(k)}\|_\infty < \theta$, one cannot expect to reduce the integrality gap to zero in the remaining SQP iterations.

The new early branching rule resulted in a small but consistent reduction in the number of QP problems solved (roughly a 5% reduction) compared to the original heuristic.

5.4.4 Nonconvex MINLP problems

In practice many MINLP problems are nonconvex. In this case, the underestimating property of Lemma 1 no longer holds, as linearizations of nonconvex functions are not necessarily underestimators. Thus it may be possible that a node is wrongly fathomed on account of Lemma 1 and this could prevent the algorithm from finding the global minimum.

A rigorous way of extending Algorithm 2 to classes of nonconvex MINLP problems would be to replace the linearizations by linear underestimators (e.g. Horst and Tuy [20, Chapter VI]). However, it is not clear what effect this would have on the convergence properties of the SQP method. Instead the following heuristic is proposed, which consists of replacing 2'. by 2":

```
2". if (($QP^k_{c,\infty}$) infeasible) then
        Enter feasibility restoration phase:
        repeat (SQP iteration)
            (a) Compute a step of the feasibility restoration.
            (c) if (($QP^k_{c,\infty}$) feasible) then return to normal SQP, exit
            (d) if ($\min \|g^+(x,y)\| > 0$) then fathom node, exit
     endif
```

The main difference between 2'. and 2". is that a node is no longer fathomed if the QP problem is infeasible and the step lies strictly inside the trust-region ($\|d\| < \rho$). Instead a feasibility problem has to be solved to optimality (Step *(d)* above) or until a feasible QP problem is encountered (Step *(b)* above). This heuristic is motivated by the success of nonlinear branch-and-bound in solving some nonconvex MINLP problems. It ensures that a node is only fathomed if it would have been fathomed by nonlinear branch-and-bound.

Clearly, there is no guarantee that the feasibility problem is solved to *global* optimality if the constraints or the objective are nonconvex.

5.5 Numerical Experience

Nonlinear branch-and-bound and Algorithm 2 have been implemented in Fortran 77 using filterSQP [12] as the underlying NLP solver. The implementation of Algorithm 2 uses a callback routine that is invoked after each step of the SQP method. This routine manages the branch-and-bound tree stored on a stack, decides on when to branch and which problem to solve next. The use of this routine has kept the necessary changes to filterSQP to a minimum.

Header	Description
problem	The SIF name of the problem being solved.
n	Total number of variables of the problem.
n_i	Number of integer variables of the problem.
m	The total number of constraints (excl. simple bounds).
m_n	Number of nonlinear constraints.
k	The dimension of the null–space at the solution.
# NLPs	Number of NLP problems solved (branch-and-bound).
# nodes	Number of nodes visited by Algorithm 2.
# QPs	Total number of QP (and LP) problems solved.
# f-QPs	Number of QPs in feasibility restoration.
CPU	Seconds of CPU time needed for the solve.

Table 21: Description of headers of the result tables

The test problems are from a wide range of backgrounds. Their characteristics are displayed in Table 22. The SYNTHES* problems are small process synthesis problems [7]. OPTPRLOC is a problem arising from the optimal positioning of a product in a multi-attribute space [7]. BATCH is a batch plant design problem that has been transformed to render it convex [22]. FEEDLOC is a problem arising from the optimal sizing and feed tray location of a distillation column [29], TRIMLOSS is a trim loss optimization problem that arises in the paper industry [28] and TRIMLON4 is an equivalent nonconvex formulation of the same problem. All problems are input as SIF files [2] plus an additional file to identify the integer variables. The test-problems are available at `http://www.mcs.dundee.ac.uk:8080/~sleyffer/MINLP_TP/`.

Table 21 lists a description of the headers of the result tables. Table 22 list the problem characteristics. The performance of branch-and-bound and Algorithm 2 is compared in Table 23. All runs were performed on a SUN SPARC station 4 with 128 Mb memory, using the compiler options `-r8 -O` and SUN's `f77` Fortran compiler. In both algorithms a depth first search with branching on the most fractional variable was implemented.

Table 23 shows that branch-and-bound only requires about 2-6 QP solves per node that is solved making it a very efficient solver. By comparison, Algorithm 2 often visits more nodes in the branch-and-bound tree. That is not surprising, as the early termination rule is based on a weaker lower bound than **FR2**. However, the overall number

problem	n	n_i	m	m_n	Description
SYNTHES1	6	3	6	2	Process synthesis
SYNTHES2	11	5	14	3	Process synthesis
SYNTHES3	17	8	23	4	Process synthesis
OPTPRLOC	30	25	30	25	Optimal product location
BATCH	49	24	73	1	Batch plant design
FEEDLOC	90	37	259	89	Distillation column design (nonconvex)
TRIMLOSS	142	122	75	4	Trimloss optimization
TRIMLON4	24	24	27	4	Trimloss optimization (nonconvex)

Table 22: Problem characteristics of test problems

	Branch-and-bound				Algorithm 2			
problem	# NLPs	# QPs	(# f-QPs)	CPU	# nodes	# QPs	(# f-QPs)	CPU
SYNTHES1	5	18	(0)	0.1	5	13	(0)	0.1
SYNTHES2	17	53	(0)	0.4	17	40	(0)	0.3
SYNTHES3	25	80	(0)	0.8	25	64	(3)	0.8
OPTPRLOC	109	491	(119)	8.5	112	232	(28)	5.3
BATCH	143	371	(10)	8.9	181	391	(11)	14.8
FEEDLOC	47	340	(35)	70.8	55	103	(4)	29.0
TRIMLOSS	1336	3820	(160)	254.3	944	1624	(1)	90.3
TRIMLON4	887	1775	(66)	19.0	566	900	(112)	13.3

Table 23: Results for branch-and-bound and Algorithm 2

of QP problems that are being solved is reduced by 20% – 80%. This results in a $\boxed{\text{similar}}$ reduction in terms of CPU time compared to branch-and-bound. For the largest problem, Algorithm 2 is 3 times faster than nonlinear branch-and-bound.

These results are roughly in line with what one might expect taking into account that branch-and-bound solves between 3 and 5 QPs per NLP node. They are somewhat better than the results in [5].

Even though problem TRIMLON4 is nonconvex, Algorithm 2 terminated with the global minimum. For the nonconvex problem FEEDLOC, Algorithm 2 did not find an integer feasible point. This behaviour is due to the fact that the linearization are *not* outer approximations in the nonconvex case and shows that the early termination rule does not hold for nonconvex MINLP problems. Running Algorithm 2 in nonconvex mode (see Section 5.4.4) produced the same minimum as branch-and-bound. As one would expect, the performance of Algorithm 2 in nonconvex mode is not as competitive as in convex mode (see Table 24) but still competitive with branch-and-bound. Better heuristics for nonconvex MINLP problems remain an open problem.

	Branch-and-bound				Algorithm 2			
problem	# NLPs	# QPs	(# f-QPs)	CPU	# nodes	# QPs	(# f-QPs)	CPU
FEEDLOC	47	340	(35)	70.8	65	159	(75)	55.9
TRIMLON4	887	1775	(66)	19.0	984	1606	(259)	21.9

Table 24: Results for Algorithm 2 in nonconvex mode

5.6 Conclusions

We have presented an algorithm for MINLP problems which interlaces the iterative solution of each NLP node with the branch-and-bound tree search.
$\boxed{\text{The algorithm presented here can also be interpreted as an improved lower bounding scheme for branch-and-bound.}}$
An implementation of this idea gave a factor of $\boxed{\text{up to}}$ 3 improvement in terms of CPU time compared to branch-and-bound.

References

[1] J.E. Hoogenboom T. Illés E. de Klerk C. Roos A.J. Quist, R. van Geemert and T. Terlaky. Optimization of a nuclear reactor core reload pattern using nonlinear optimization and search heuristics. 1997.

[2] N.I.M. Gould A.R. Conn and Ph.L. Toint. Lancelot: a fortran package for large-scale nonlinear optimization (release a). 1992.

[3] N.I.M. Gould A.R. Conn and Ph.L. Toint. Methods for nonlinear constraints in optimization calculations. *To appear in "The State of the Art in Numerical Analysis", I. Duff and G.A. Watson (eds.)*, 1996.

[4] B. Borchers and J.E. Mitchell. An improved branch and bound algorithm for Mixed Integer Nonlinear Programming. *Computers and Operations Research*, 21, 1994.

[5] B. Borchers and J.E. Mitchell. A computational comparison of branch and bound and outer approximation methods for 0-1 mixed integer nonlinear programs. *Computers and Operations Research*, 24, 1997.

[6] R.J. Dakin. A tree search algorithm for mixed integer programming problems. *Computer Journal*, 8:250–255, 1965.

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

[8] M. El-Hallabi and R.A. Tapia. An inexact trust-region feasible-point algorithm for nonlinear systems of equalities and inequalities. 1995.

[9] R. Fletcher. *Practical Methods of Optimization, 2^{nd} edition.* 1987.

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

[11] R. Fletcher and S. Leyffer. Nonlinear programming without a penalty function. 1997.

[12] R. Fletcher and S. Leyffer. User manual for filterSQP. 1998.

[13] O.E. Flippo and A.H.G. Rinnoy Kan. Decomposition in general mathematical programming. *Mathematical Programming*, 60:361–382, 1993.

[14] C.A. Floudas. Nonlinear and mixed–integer optimization. 1995.

[15] A.M. Geoffrion. Generalized benders decomposition. *Journal of Optimization Theory and Applications*, 10:237–260, 1972.

[16] I.E. Grossmann and Z. Kravanja. Mixed–integer nonlinear programming: A survey of algorithms and applications. 1997.

[17] I.E. Grossmann and R.W.H. Sargent. Optimal design of multipurpose batch plants. *Ind. Engng. Chem. Process Des. Dev.*, 18:343–348, 1979.

[18] O.K. Gupta and A. Ravindran. Branch and bound experiments in convex nonlinear integer programming. *Management Science*, 31:1533–1546, 1985.

[19] S.P. Han. A globally convergent method for nonlinear programming. *Journal of Optimization Theory and Application*, 22, 1977.

[20] R. Horst and H. Tuy. A radial basis function method for global optimization. *Global Optimization: Deterministic Approaches, 2^{nd} edition*, 1993.

[21] M. El-Alem J.E. Dennis and K.A. Williamson. A trust-region approach to nonlinear systems of equalities and inequalities. *SIAM Journal on Optimization*, 9:291–315, 1999.

[22] G.R. Kocis and I.E. Grossmann. Global optimization of nonconvex mixed–integer nonlinear programming (MINLP) problems in process synthesis. *Industrial Engineering Chemistry Research*, 27:1407–1421, 1988.

[23] A.H. Land and A.G Doig. An automatic method for solving discrete programming problems. *Econometrica*, 28:497–520, 1960.

[24] V.A. Roshchin O.V. Volkovich and I.V. Sergienko. Models and methods of solution of quadratic integer programming problems. *Cybernetics*, 23:289–305, 1987.

[25] M.J.D. Powell. A fast algorithm for nonlinearly constrained optimization calculations. *In G.A. Watson, editor,* Numerical Analysis, 1977, pages 144–157, 1978.

[26] I. Quesada and I.E. Grossmann. An LP/NLP based branch–and–bound algorithm for convex MINLP optimization problems. *Computers and Chemical Engineering*, 16:937–947, 1992.

[27] R.A. Stubbs and S. Mehrotra. A branch–and–cut method for 0–1 mixed convex programming. *Technical report 96-01, Department of IE/MS, Northwestern University, Evanston, IL 60208, USA*, 1996.

[28] J. Isaksson T. Westerlund and I. Harjunkoski. Solving a production optimization problem in the paper industry. *Report 95–146–A, Department of Chemical Engineering, Abo Akademi, Abo, Finland*, 1995.

[29] J. Viswanathan and I.E. Grossmann. Optimal feed location and number of trays for distillation columns with multiple feeds. 32:2942–2949, 1993.

[30] R.B. Wilson. A simplicial algorithm for concave programming. 2001.

[31] Y. Yuan. Trust region algorithms for nonlinear equations. 1994.