BFOUR: A Fortran Subroutine for Solving Integer Optimization Problems by Branch and Bound

- User's Guide -

T. Lehmann, K. Schittkowski

Address:	Prof. K. Schittkowski Siedlerstr. 3 D - 95488 Eckersdorf Germany
Phone:	(+49) 921 32887
E-mail:	klaus@schittkowski.de
Web:	http://www.klaus-schittkowski.de
Date:	January 2015

Abstract

The Fortran subroutine BFOUR is an implementation of a branch and bound algorithm for mixed-integer optimization. It is applied to solve mixed-integer quadratic programs as part of the code MIQL, and to solve mixed integer nonlinear optimization problems in combination with the code MISQP. The usage is documented and numerical results are presented for MIQL.

Keywords: branch and bound; mixed integer programming; mixed integer quadratic programming; MIQP; mixed integer nonlinear programming; MINLP; numerical algorithms

1 Introduction

The code BFOUR is an implementation of a branch and bound algorithm, and can only be used as part of a more general framework, e.g., mixed integer quadratic or mixed integer nonlinear programming. Proceeding from a scalar function $F(y)$ defined for $y \in \mathbb{R}^{n_i}$ with $y_l \le y \le y_u$, the code is designed to solve general integer optimization problems of the form

$$y \in \mathbb{Z}^{n_i} : \begin{array}{l} \min F(y) \\ y_l \le y \le y_u \end{array} . \tag{1}$$

The basic idea is to generate and solve a sequence of continuous subproblems

$$y \in \mathbb{R}^{n_i} : \begin{array}{l} \min F(y) \\ y_l' \le y \le y_u' \end{array} , \tag{2}$$

where $y_l \le y_l' \le y_u' \le y_u$ are lower and upper bounds set by the branch and bound algorithm. It is assumed that F is relaxable, i.e., that $F(y)$ can be evaluated also for non-integral parameter values of $y \in \mathbb{R}^{n_i}$.

A typical application is mixed integer quadratic programming (MIQP)

$$x \in \mathbb{R}^{n_c}, y \in \mathbb{Z}^{n_i} : \begin{array}{l} \min \ \frac{1}{2}(x^T, y^T) C \begin{pmatrix} x \\ y \end{pmatrix} + (x^T, y^T) d \\ (x^T, y^T) a_j + b_j = 0 \ , \ \ j = 1, \ldots, m_e \ , \\ (x^T, y^T) a_j + b_j \ge 0 \ , \ \ j = m_e + 1, \ldots, m \ , \\ x_l \le x \le x_u \ , \\ y_l \le y \le y_u \ . \end{array} \tag{3}$$

The matrix C is supposed to be positive definite, i.e., problem (4) is strictly convex. Thus, the resulting subproblem (2) becomes

$$x \in \mathbb{R}^{n_c}, y \in \mathbb{R}^{n_i} : \begin{array}{l} \min \ \frac{1}{2}(x^T, y^T) C \begin{pmatrix} x \\ y \end{pmatrix} + (x^T, y^T) d \\ (x^T, y^T) a_j + b_j = 0 \ , \ \ j = 1, \ldots, m_e \ , \\ (x^T, y^T) a_j + b_j \ge 0 \ , \ \ j = m_e + 1, \ldots, m \ , \\ x_l \le x \le x_u \ , \\ y_l' \le y \le y_u' \ , \end{array} \tag{4}$$

$y_l \le y'_l \le y'_u \le y_u$. The generated subproblems differ from the original formulation (3) in restricted lower and upper bounds for integer variables. Convex continuous quadratic programming problems can be solved, e.g., by the code QL, see Schittkowski [11], Goldfarb and Idnani [5], or Powell [9].

In case of nonlinear mixed integer programming (MINLP),

$$x \in I\!R^{n_c}, y \in \mathbb{Z}^{n_i}: \begin{array}{l} \min\ f(x,y) \\ g_j(x,y) = 0\ ,\quad j = 1,\ldots,m_e\ , \\ g_j(x,y) \ge 0\ ,\quad j = m_e+1,\ldots,m\ , \\ x_l \le x \le x_u\ , \\ y_l \le y \le y_u\ . \end{array} \tag{5}$$

one has to solve continuous nonlinear subproblems of the form

$$x \in I\!R^{n_c}, y \in I\!R^{n_i}: \begin{array}{l} \min\ f(x,y) \\ g_j(x,y) = 0\ ,\quad j = 1,\ldots,m_e\ , \\ g_j(x,y) \ge 0\ ,\quad j = m_e+1,\ldots,m\ , \\ x_l \le x \le x_u\ , \\ y'_l \le y \le y'_u\ . \end{array} \tag{6}$$

In these situations, we define $F(y) = f(x_y, y)$, where x_y is the continuous solution of (5) or (6) for a fixed y. $F(y)$ may become undefined if the feasible domain is empty, i.e., if the bounds generated by the branch and bound procedure, are too stringent.

The functions $f(x,y)$ and $g_j(x,y), j = 1,\ldots,m$, are supposed to be twice continuously differentiable in order to apply efficient solution methods to (6). Here m_e is the number of equality constraint, m denotes the total number of constraints, x_l and x_u define box constraints for the continuous variables $x \in I\!R^{n_c}$ and y_l and y_u are box constraints for the integer variables $y \in \mathbb{Z}^{n_i}$. Possible solution methods for the continuous relaxation are NLPQLP of Schittkowski [10, 12] or MISQP of Exler and Schittkowski [3, 4]. In the latter case, it is possible to branch only subject to a subset of the given integer variables.

This manual is organized as follows. The subsequent section introduces the basic concept of a branch and bound method and outlines the various solution strategies of BFOUR. Implementation details and program documentation are found in section 3, which also includes an illustrative example of BFOUR executed in combination with the solver QL for quadratic programming. Section 4 summarizes some numerical results obtained by the solver MIQL of Lehmann and Schittkowski [6].

2 The Branch and Bound Procedure

A branch and bound method is an iterative interaction of a branching and bounding process and an appropriate subproblem solver for evaluating $F(y)$, see (2). Starting point is the relaxation of the original mixed integer program, i.e., the corresponding continuous problem obtained by replacing the condition $y \in \mathbb{Z}^{n_i}$ by $y \in \mathbb{R}^{n_i}$. The relaxed problem must be solved by a suitable subproblem solver and is interpreted as the root node of a binary search tree. New nodes of the enumeration tree are successively generated by changing bounds for integer variables. Again, continuous problems corresponding to the created node are to be solved in the same way. See also an early paper about a branch and bound algorithm for linear programming published by Dakin [2].

After exploring all possible nodes of the tree, i.e., if all corresponding subproblems are solved, the integer feasible subproblem solution possessing the lowest objective value is the optimal solution of the original problem (1). There are certain situations that allow to speed up the solution process, such that not all nodes have to be explored. This will be explained now in more detail.

In the following, we assume that the continuous relaxation problems are convex, i.e., the quadratic programs (4) or the nonlinear programs (6) for all possible bounds. Thus, if the optimal solution of the relaxed root problem is integer, the branch and bound process can be stopped and the optimal solution is found. Otherwise, a fractional integer variable is selected and two new continuous subproblems are generated. They are obtained by rounding the fractional value of a selected integer variable, say y_k, to get two separate subproblems, one with upper bound $\lfloor y_k \rfloor$, another one with lower bound $\lfloor y_k \rfloor + 1$. Each subproblem determines a new child node of the binary search tree.

An integer feasible solution of a subproblem gives an upper bound for the optimal objective function value of (1). The minimal objective value at all nodes that are not completely explored, is a lower bound for the optimal objective value. Thus, whenever an feasible integer solution is found, there exists an upper and a lower bound for the optimal solution.

Furthermore, certain subtrees of the binary search tree can be eliminated or *fathomed* at an early stage:

- The corresponding subproblem of a node is infeasible: All subproblems obtained by branching from this node would also be infeasible.
- The subproblem has a feasible integer solution: The corresponding optimal objective function value, if less than the known upper bound, provides a better upper bound for the optimal solution.
- The continuous solution of a subproblem has a minimal solution value greater than the actual upper bound: Further branching from the node would only increase function values, and there is no chance to find a better integer solution.

Fathoming is critical for nonconvex problems, since it might cut off the optimal solution. In general, the procedure is continued until all free nodes are either explored or fathomed. The integer feasible node with minimal value represents the solution of the mixed integer problem or we get the information that no feasible mixed integer solution exists.

At each node of the enumeration tree, one has to check whether the actual values of all integer variables are integral and whether the subproblem is feasible. Furthermore, the current objective value has to be compared with the best known upper bound. Whenever possible, fathoming is initialized and the memory required for this node can be saved. Otherwise, some node data, i.e., the objective value, the position within the search tree and the corresponding bounds, must be stored as part of the tree. Next, a branching variable is selected and a new node is created by adjusting bounds of integer variables.

Thus, there remain two important steps by which the performance of a branch and bound algorithm is heavily influenced, the selection of a branching variable and the choice of the subsequent node of the enumeration tree to be explored next. The following options are implemented in the current version of BFOUR:

1. Selection of an integer variable with a fractional solution value for branching:
   (a) *maximal fractional branching*: Select a fractional integer variable value from the solution of the relaxed subproblem with largest distance from nearest integer value.
   (b) *minimal fractional branching*: As above, with smallest distance from nearest integer value.
   (c) *branching by priorities*: Select a fractional integer variable from the solution of the relaxed subproblem with highest user-defined priority.
2. Search for a free node from where branching, i.e., the generation of two new subproblems, is started:
   (a) *best of two*: The optimal objective function values of the two child nodes are compared and the node with a lower value is chosen. If both are leafs, i.e., if the corresponding solution is integral, or if the corresponding problem is infeasible or if there is already a better integral solution, strategy *best of all* is started.
   (b) *best of all*: All free nodes are explored and the node with lowest objective function value is selected.
   (c) *depth first*: Select a child node whenever possible. If the node is a leaf, the *best of all* strategy is applied.

BFOUR also supports the input of an integer feasible point when starting the branch and bound process. This integral solution might either be known in advance or might be found by some primal heuristics. Depending on the distance of the corresponding objective function value from the optimal value, a solution is usually found much faster.

There are additional options which allow the user to influence the branch and bound process especially in connection with the mixed integer quadratic programming code MIQP. The user is allowed to interact with BFOUR after a current node is evaluated and a branching variable is selected. In other words, the standard branch and bound process can be interrupted and the user can provide additional information that might be useful for the enumeration. For example, it is possible to provide an external branching direction, especially in combination with *depth first* search, see Lehmann and Schittkowski [6]. Furthermore, one could supply sharper lower bounds for the current node to allow earlier fathoming by exploiting dual information according to Leyffer and Fletcher [7].

To exploit warm starts, one should know how two successive subproblems are related. Thus, BFOUR passes information whether a current node is a child node or possesses the same parent node or is a nephew, i.e., a child node of a node possessing the same parent node or that no relation exists. BFOUR also reports the index of the selected branching variable at the current node, which can be beneficial for determining an external branching direction, e.g., according to the value of a Lagrangian function. Furthermore, BFOUR informs the user about the following events:

- new best solution is found
- current node is integral
- current node is marked, i.e., a leaf of the search tree

In addition, BFOUR prints the current lower bound, i.e., the lowest function value of an unexplored fractional node. This allows the user to monitor progress and to estimate the quality of the best known integral solution.

In the nonconvex case, the fathoming procedure is adapted to reduce the probability of a cut-off of the optimal solution. Usually, much more subproblems need to be solved.

3 Implementation and Program Documentation

The latest version of BFOUR is enhanced as outlined above and especially tuned to cooperate with the convex continuous quadratic programming solver QL for solving mixed integer quadratic programs.

The user has to provide function values for F(Y) in the same program, which executes also BFOUR, according to the following rules:

1. Choose starting values for the variables to be optimized, and store them in Y.
2. Compute a solution of the underlying mixed-integer optimization problem subject to the given initial bounds and store the solution in Y and the optimum objective function value in F. Treat Y as continuous variable vector.
3. Set IFAIL=0 and execute BFOUR.
4. If BFOUR terminates with IFAIL=0, the internal stopping criteria are satisfied.
5. In case of IFAIL<0, an error occurred.
6. If BFOUR returns IFAIL=1, compute an optimal solution of the underlying mixed-integer optimization problem subject to the lower and upper bounds found in YL and YU, respectively, and store the solution in Y and the optimum objective function value in F. If a feasible solution exists, set IFEAS=.TRUE. and set IFEAS=.FALSE. otherwise. Then call MISQP again.

Note that the applied Fortran dialect is as close as possible to F77 to allow straightforward transition from FORTRAN to C by f2c. The calling sequence and the meaning of the parameters are described below.

Usage:

```
      CALL  BFOUR(    NINT,      Y,       F,   IFEAS,      YL,
    /                   YU,  YBEST,     ACC,  PRIORS,  MAXNDS,
    /                IFAIL,   IOUT,  IPRINT,  B4IOPT,  B4LOPT,
    /               B4ROPT,     RW,     LRW,      IW,     LIW,
    /                   LW,    LLW                            )
```

Definition of the parameters:

NINT:	Input parameter for the number of integer variables
Y(NINT):	Double precision input array containing the optimal relaxed integer variable values of the solution of the subproblem
F:	Double precision input parameter for the optimal objective function value at the current node
IFEAS:	Logical input parameter to indicate whether current node possesses a feasible solution or not
YL(NINT):	Integer input and output array for lower bounds of the integer variables Y, replaced subsequently by the actual node bounds
YU(NINT):	As above for upper bounds
YBEST(NINT):	Double precision input and output array for the integer values of a feasible solution, if known at the first call, subsequently replaced the best integer feasible solution found so far
ACC:	Double precision input parameter for a tolerance to identify integer values
PRIORS(NINT):	Integer input array for priority values of integer variables to determine the branching variable in combination with branching strategy 3, i.e., B4IOPT(1)=3
MAXNDS:	Integer input parameter for the maximum number of branch and bound nodes
IFAIL:	Initially, IFAIL must be set to zero. Subsequently, IFAIL might get one of the subsequent values:

-3 :	original relaxed problem not solvable
-2 :	no integral solution found within maximum number of nodes
-1 :	integral solution does not exist
0 :	optimal solution found
1 :	solve new branch and bound problem, corresponding bounds are found in YL and YU
2 :	integral solution found, but optimality not yet proved because of reaching maximum number of nodes
3 :	wrong input parameter, either ACC, MAXNDS, IOUT, IPRINT, B4IOPT(1) or B4IOPT(2)
4 :	number of integer variables changed
5 :	working array to small

IOUT:	Integer input parameter for the output unit number, i.e., all write-statements start with WRITE(IOUT,...
IPRINT:	Integer input parameter for the output level:
	0 : no output
	1 : error messages, root node and solution output
	2 : in addition, output of improved solutions
	3 : one line for each node with status, i.e.,
	* - fractional feasible node
	B - new best integer feasible solution
	I - infeasible node
	M - marked node
	4 : output of index and bounds for all nodes
	5 : full output of all node data
B4IOPT(20):	Integer option array
B4IOPT(1):	Input parameter for selecting a branching variable,
	1 : maximum fractional branching
	2 : minimum fractional branching
	3 : branching according to priorities
B4IOPT(2):	Input parameter for selecting the next node,
	1 : best of all
	2 : best of two
	3 : depth first
B4IOPT(3):	Output parameter defining the relation of two successive nodes, required for warm starts:
	0 : arbitrary node, i.e., no specific relation
	1 : child of previous node
	2 : same parent node as previous node
	3 : nephew of previous node
B4IOPT(4):	Input parameter controlling a BFOUR run, to be initialized by zero for standard execution,
	0 : standard BFOUR run
	1 : interrupt after node evaluation is completed, BFOUR continued with B4IOPT(4)=2 or 3
	2 : continuing an interrupted run with improved

	bounds 3 : continuing an interrupted run with external branching direction given by B4IOPT(6)
B4IOPT(5):	Output parameter for the index of the selected branching variable at current node
B4IOPT(6):	Input parameter for an external branching direction, 0 : branch left at current node 1 : branch right at current node
B4LOPT(20):	Logical option array
B4LOPT(1):	Output parameter indicating that a new best integer feasible solution found (.TRUE.)
B4LOPT(2):	Output parameter indicating that current node is integral (.TRUE.)
B4LOPT(3):	Output parameter indicating that current node is marked (.TRUE.)
B4LOPT(4):	Input parameter indicating that the underlying problem is convex (.TRUE.)
B4LOPT(5):	Input and output parameter to indicate a known integer feasible solution (.TRUE.); if set to true when calling BFOUR, B4ROPT(3) and YBEST must be set correctly
B4LOPT(6):	Input parameter indicating first call (.TRUE.)
B4ROPT(20):	Double precision option array
B4ROPT(1):	Input parameter for a worse bound, to be used in connection with B4IOPT(4)=2
B4ROPT(2):	Output parameter for the actual lower bound
B4ROPT(3):	Input and output parameter for the function value of the best known integer feasible solution, used to pass an integer feasible point to BFOUR at first call if B4LOPT(5) and YBEST are correctly set
RW(LRW):	Double precision working array
LRW:	Integer input parameter for the length of RW, must be at least 3*MAXNDS+2*NINT+5
IW(LIW):	Integer working array
LIW:	Integer input parameter for length of IW, must be at least 6*MAXNDS+7

LW:	Logical working array
LLW:	Integer input parameter for length of LW, must be at least 3*NINT+2

For a standard execution of BFOUR not exploiting the special features designed for MIQL, only the parameter options of Table 1 are relevant. All other options can be set to arbitrary values, e.g., zero. Initialization of working arrays is not required.

Option	Value
B4IOPT(1)	1 or 2
B4IOPT(2)	1, 2 or 3
B4IOPT(4)	0
B4LOPT(4)	.TRUE. or .FALSE.
B4LOPT(5)	.FALSE.
B4LOPT(6)	.TRUE.

Table 1: Option Setting for a Standard BFOUR Run

The subsequent Fortran program illustrates the standard usage of BFOUR in combination with the continuous quadratic solver QL, see Schittkowski [11] to solve the following strictly convex integer quadratic program.

$$\begin{aligned} \min_{y_1,y_2\in\mathbb{Z}} \quad & \frac{1}{2}\left((y_1-3.4)^2+(y_2-12.6)^2\right) \\ & y_1+y_2 \geq 15 \\ & 1 \leq y_1 \leq 10; \quad 3 \leq y_2 \leq 20 \end{aligned} \tag{7}$$

```
      IMPLICIT NONE
      INTEGER          N, M, MAXNDS, LRWB4, LIWB4, LLWB4, LRW, LIW, LLW
      PARAMETER       (N      = 2,
     /                 M      = 1,
     /                 MAXNDS = 100,
     /                 LRWB4  = 3*MAXNDS + 2*N + 5,
     /                 LIWB4  = 6*MAXNDS + 7,
     /                 LLWB4  = 3*N + 2,
     /                 LRW    = LRWB4 + 3*N*N/2 + 10*N + 2*M + 1,
     /                 LIW    = LIWB4 + N,
     /                 LLW    = LLWB4)
      INTEGER          IFAIL, IOUT, IPRINT, PRIORS(N), B4IOPT(20),
     /                 IW(LIW), IFAILQL, I
      DOUBLE PRECISION Y(N), F, YL(N), YU(N),
     /                 C(N,N), D(N), A(M,N), B(M), U(N+N+M),
     /                 YBEST(N), ACC, B4ROPT(20), RW(LRW)
```

```
      LOGICAL          LFEAS, B4LOPT(20), LW(LLW)
C
C   Problem data
C
      YL(1)  = 1.0D0
      YU(1)  = 10.0D0
      YL(2)  = 3.0D0
      YU(2)  = 20.0D0
      C(1,1) = 1.0D0
      C(1,2) = 0.0D0
      C(2,1) = 0.0D0
      C(2,2) = 1.0D0
      D(1)   = -3.4D0
      D(2)   = -12.6D0
      A(1,1) = 1.0D0
      A(1,2) = 1.0D0
      B(1)   = -15.0D0
C
C   Initialization
C
      IFAIL     = 0
      IFAILQL   = 0
      IOUT      = 6
      IPRINT    = 3
      ACC       = 1.0D-14
      B4IOPT(1) = 1 !max frac branching
      B4IOPT(2) = 3
      B4IOPT(4) = 0
      B4LOPT(4) = .TRUE.
      B4LOPT(5) = .FALSE.
      B4LOPT(6) = .TRUE.
C
C   Main loop
C
    1 CONTINUE
      CALL QL( M, 0, M, N, N, M+N+N, C, D, A, B, YL, YU, Y, U,
     /         ACC, 1, IOUT, IFAILQL, 1, RW(LRWB4+1), LRW,
     /         IW(LIWB4+1), LIW)
      F = 0.5D0*(Y(1)**2 + Y(2)**2) + D(1)*Y(1) + D(2)*Y(2)
      IF (IFAILQL.GT.0) THEN
         LFEAS = .FALSE.
      ELSE
         LFEAS = .TRUE.
      END IF
      CALL BFOUR( N, Y, F, LFEAS, YL, YU, YBEST, ACC, PRIORS,
     /            MAXNDS, IFAIL, IOUT, IPRINT, B4IOPT, B4LOPT,
     /            B4ROPT, RW, LRWB4, IW, LIWB4, LW, LLWB4)
      IF (IFAIL.GE.1) GOTO 1
C
C   Done
C
      STOP
      END
```

The following output should appear on screen:

```
    --------------------------------------------------------------------
           Start of the Mixed-Integer Branch-and-Bound Code BFOUR
                       Version 3.0 (Mar 2013)
    --------------------------------------------------------------------

  Parameters:
       Number of integer variables                   2
       Maximal number of nodes                     100
       Integer tolerance                      0.10D-13
       Branching strategy                            1
       Node selection                                3
       Convex problem
       No initial integer feasible solution known

    Output in the following order:
       S     - status of current node
               B ... branched node
               I ... infeasible node
               M ... marked node
               * ... other node
       IT    - iteration count
       ND    - index of current node
       F     - objective function value
       P     - index of parent node
       LB    - lower bound
       UB    - upper bound

  S      IT       ND         F            P        LB           UB
----------------------------------------------------------------------
  *        1        1  0.000000D+00        0  0.000000D+00  0.100000D+31
  *        2        2  0.180000D+00        1  0.000000D+00  0.100000D+31
```

```
B        3        3  0.260000D+00        2  0.000000D+00  0.260000D+00
*        4        3  0.800000D-01        1  0.000000D+00  0.260000D+00
B        5        4  0.160000D+00        3  0.000000D+00  0.160000D+00
M        6        4  0.260000D+00        3  0.800000D-01  0.160000D+00

  Optimal solution:
  Number of explored subproblems:     6
  Highest index:      3
  F(Y)    =    0.16000000
  Y(   1) =     3.0000000
  Y(   2) =     13.000000
```

4 Numerical Results

We show some numerical results obtained by BFOUR in combination with the quadratic solver QL as implemented in a Fortran subroutine called MIQL for mixed integer quadratic programming (MIQP). The first set of test problems are MIQP subproblems as established by the nonlinear mixed integer solver MISQP, see Exler and Schittkowski [3, 4]. Out of a set of 100 test examples, nine problems are selected where MIQL needs at least 0.1 seconds. Some characteristic data of the MINLP test problems are presented in Table 2. Six test problems

problem	continuous variables	integer variables	inequality constraints	equality constraints
MISQP57	2	4	0	1
MISQP60	0	24	35	0
MISQP62	0	35	44	0
MISQP63	0	48	53	0
MISQP66	16	19	1	17
MISQP83	0	20	20	0
MISQP115	9	27	12	9
MISQP116	9	27	12	9
MISQP117	9	27	12	9

Table 2: Problem Characteristics

belong to the GAMS MINLP-Library, see Bussieck, Drud, and Meeraus [1], the remaining three are test problems for oil and gas production. These problems, MISQP115, MISQP116 and MISQP117, possess multiple global optima leading to non-comparable performance. Note that MISQP requires the solution of many MIQP subproblems.

Table 3 shows the influence of the different node selection strategies on the performance of MIQL. Also some of the more advanced options of BFOUR are evaluated. The strategy *best of all* selects a node with lowest function value, whereas *best of two* always selects a child node with better function value. *Depth first* dives into the search tree by selecting a child problem without evaluating the other child node. There are special tuning possibilities for all node selection strategies including warm starts. We selected *maximal fractional branching* as branching variable selection strategy for this evaluation, because it is clearly superior as can be seen in Table 4.

strategy	nodes	time (sec)
best of all standard	115,588	36.3
best of all improved bounds	99,717	40.3
best of two standard	122,516	33.8
best of two warm start	121,996	26.5
depth first standard	336,783	136.9
depth first warm start	302,145	44.5
depth first external directions	121,739	33.6
depth first warm start, external directions	122,055	26.4

Table 3: Node Selection Strategies and BFOUR Options

Obviously, warm starts and the calculation of the Lagrangian function value to determine the search direction for *depth first* are very beneficial. But Table 3 indicates also how important it is to adapt the search process to the outer optimization algorithm.

The subsequent Table 4 contains numerical results to compare *maximal fractional branching* and *minimal fractional branching*. We report only the fastest MIQL run for each node selection strategy. Strategy *minimal fractional branching* exhibits worse performance compared to *maximal fractional branching*.

strategy	nodes	time (sec)
best of all, *maximal fractional branching*	115,588	36.3
best of all, *minimal fractional branching*	184,794	78.0
best of two, *maximal fractional branching*	121,996	26.5
best of two, *minimal fractional branching*	191,406	47.3
depth first, *maximal fractional branching*	122,055	26.4
depth first, *minimal fractional branching*	206,482	49.3

Table 4: Branching Variable Selection Strategies

The following results are obtained by some benchmark test examples of Mittelmann [8]. Since MIQL is a dense solver and cannot exploit sparsity patterns, some of these test problems cannot be solved because of their dimensions. For other test cases, the continuous relaxation is not solvable without special adjustments. Therefore, we only consider the test cases ibell3a and imas284, see Table 5 for some characteristic data. Results for different node selection strategies, including tuning, are presented in Table 6.

problem	constraints	number of variables	number of integer variables	non-zero density of constraint matrix	non-zero density of objective function matrix
ibell3a	104	122	60	3.1%	0.4%
imas284	68	151	150	95.3%	0.6%

Table 5: Characteristics of Mittelmann Test Examples

problem	branching strategy	node selection	nodes	time (sec)
ibell3a	*maximal fraction*	*best of all*	61,588	1,201
ibell3a	*maximal fraction*	*best of two*	63,777	1,125
ibell3a	*maximal fraction*	*depth first*	74,720	952
imas284	*maximal fraction*	*best of all*	142,241	3,610
imas284	*maximal fraction*	*best of two*	186,412	3,576
imas284	*maximal fraction*	*depth first*	142,434	1,940

Table 6: Results for Mittelmann Test Examples

5 Conclusions

We present a generally applicable branch and bound algorithm for integer optimization. Special features of the underlying mathematical model allow warm starts and user interaction. They are exploited as part of the mixed integer quadratic solver MIQL in combination with the continuous quadratic solver QL. We evaluate different options of BFOUR on mixed integer quadratic and mixed integer nonlinear programs and were able to suggest standard settings depending on the abilities of the continuous solver called by BFOUR.

The *depth first* strategy is especially profitable if warm starts are possible, e.g., when solving MIQP problems in combination with the primal-dual quadratic programming solver QL, see Schittkowski [11], Goldfarb and Idnani [5], or Powell [9]. In addition, the search tree usually gets smaller, i.e., there are fewer unexplored nodes. The disadvantage is, that the child node is arbitrarily selected leading to a larger number of subproblems to be solved. But BFOUR provides the possibility to use a branching direction given by the user, which might overcome this disadvantage of *blind diving*, see MIQL of Lehmann and Schittkowski [6].

The *best of all* method usually needs the lowest number of subproblems until termination. Furthermore, it increases the lower bound more quickly. On the other hand, a first integer feasible solution is usually obtained quite late, so that fathoming is not very efficient at the beginning. Warm starts are more or less impossible, since in general no relation between subsequent subproblems exists.

Best of two strikes the balance between the other two node selection strategies. The number of subproblems required in the solution process is usually moderate compared to *depth first*, but larger than *best of all*. Warm starts are also possible since most of the subproblems are closely related.

In general, the quality of the node selection strategy depends heavily on the abilities of the continuous solver interacting with the branch and bound framework. Furthermore, the implementation plays a crucial role, e.g., one can overcome the disadvantage of *depth first* by imposing sensible branching directions, such that the number of nodes is comparable to *best of two*. Moreover, it could be very beneficial for *best of all* if an integer feasible starting point is known, enhancing the efficiency of fathoming.

In general, we conclude that *maximal fractional branching* is superior to *minimal fractional branching*. Only if the best integer feasible solution is located very close to the relaxed solution, *minimal fractional branching* is a better choice. *Branching by priorities* might lead to good results, if the user is able to supply information on beneficial branching variables.

References

[1] Bussieck M.R., Drud A.S., Meeraus A. (2007): *MINLPLib - A collection of test models for mixed integer nonlinear programming*, GAMS Development Corp., Washington D.C., USA

[2] Dakin R.J. (1965): *A tree search algorithm for mixed integer programming problems*, Computer Journal, Vol. 8, 250-255

[3] Exler O., Schittkowksi K. (2006): *A trust region SQP algorithm for mixed integer nonlinear programming*, Optimization Letters, DOI 10.1007/s11590-006-0026-1

[4] Exler O., Schittkowksi K. (2006): *MISQP: A Fortran implementation of a trust region SQP algorithm for mixed integer nonlinear programming - User's guide, version 2.0*, Report, Department of Computer Science, University of Bayreuth, Germany

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

[6] Lehmann T., Schittkowski K. (2007): *MIQL : A Fortran code for convex mixed integer quadratic programming by branch and bound - User's guide, version 2.0 -* Report, Department of Computer Science, University of Bayreuth, Germany

[7] Leyffer S., Fletcher R. (1998): *Numerical experiments with lower bounds for MIQP branch and bound*, SIAM Journal on Optimization, Vol. 8, 604-616

[8] Mittelmann H. (2007): *Mixed Integer (QC)QP Benchmark*, http://plato.asu.edu/ftp/miqp.html

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

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

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

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