An Improved Search Approach for Solving Non-Convex Mixed-Integer Non Linear Programming Problems

Joni Wilson Sitopu¹, Herman Mawengkang², Riri Syafitri Lubis¹
¹Graduate School of Mathematics, University of Sumatera Utara, Medan, Indonesia
²Department of Mathematics, University of Sumatera Utara, Medan, Indonesia

Email: hmawengkang@yahoo.com

Abstract. The nonlinear mathematical programming problem addressed in this paper has a structure characterized by a subset of variables restricted to assume discrete values, which are linear and separable from the continuous variables. The strategy of releasing nonbasic variables from their bounds, combined with the “active constraint” method, has been developed. This strategy is used to force the appropriate non-integer basic variables to move to their neighbourhood integer points. Successful implementation of these algorithms was achieved on various test problems.

Keywords: Nonlinear programming, active constraints, direct search, global optimization

1. Introduction

There are a lot of optimization problem involved integer or discrete variables which occur naturally in a nonlinear function. Mixed Integer Nonlinear Programming (MINLP) refers to these kind of optimization problems with continuous and discrete variables and nonlinearities in the objective function and constraints. This paper concerns with a special class of MINLP problem in which the discrete variables are linear and separable from the continuous variables.

This problem is defined by the following model.

\[ \begin{align*}
\text{Minimize} & \quad f(x, y) \\
\text{s.t.} & \quad g_i(x, y) \leq b_i, \quad i = 1, \ldots, m \\
& \quad x \in X, \quad y \in Y
\end{align*} \]

where \( f \) and \( g_i, i = 1, \ldots, m \), are real-valued function on \( \mathbb{R}^{n+q} \), \( X \subseteq \mathbb{R}^n \), and \( Y \) is a continuous subset in \( \mathbb{R}^q \).

There are many diverse and important applications of MINLPs. A small subset of these applications includes portfolio optimization [1], [2], block layout design in the manufacturing and service sectors [3], network design with queuing delay constraints [4], integrated design and control of chemical processes [5], drinking water distribution systems security [6]. Energy system [7], [8], and multiperiod supply chain problems [9].
The needs in such diverse areas have motivated research and development in MINLP solver technology, particularly in algorithms for handling large-scale, highly combinatorial and highly nonlinear problems.

Methods for solving MINLPs include innovative approaches and related techniques taken and extended from MIP, such as, Outer Approximation (OA) methods [10, 11], Branch-and-Bound (B&B) [12, 13, 24], Integrating Successive Quadratic Programming (SQP) and B&B [15], Generalized Bender’s Decomposition (GBD) [16], and Branch and Cut [17] for solving MINLPs have been discussed in the literature since the early 1980’s. These approaches generally rely on the successive solutions of closely related NLP problems. For example, B&B starts out forming a pure continuous NLP problem by dropping the integrality requirements of the discrete variables (often called the relaxed MINLP or RMINLP). Moreover, each node of the emerging B&B tree represents a solution of the RMINLP with adjusted bounds on the discrete variables. A survey regarding to solving a nonconvex MINLP can be found in [18].

Whenever the MINLP problem become too large to be solved by exact method, most of researchers move to heuristic method. Heuristic approaches to solving MINLPs include Variable Neighbourhood Search [19], automatically tuned variable fixing strategies [20], feasible neighbourhood search [21], Feasibility Pump [23, 27], heuristics based on Iterative Rounding [25]. [19] propose a MINLP heuristic called the Relaxed-Exact-Continuous-Integer Problem Exploration (RECIPE) algorithm. The algorithm puts together a global search phase based on Variable Neighbourhood Search [19] and a local search phase based on a MINLP heuristic. Masmoudi et al. (017) propose a hybrid of genetic algorithm for solving MINLP. In heuristic approaches, however, one of the main algorithmic difficulties connected to MINLPs is to find a feasible solution.

Due to the fact that the functions in MINLPs are not smooth, therefore in this paper we use a direct search method, known as unconstrained optimization techniques that do not explicitly use derivatives. More information regarding to direct search method in optimization can be found in (Vayasis, 1991).

In this paper we address a strategy of releasing nonbasic variables from their bounds, combined with the “active constrained” method and the notion of superbasic for efficiently tackling a particular class of MINLP problems.

The rest of this paper is organized as follows. In Section 2 we give a brief notion of neighbourhood search. The basic approach of the proposed method is presented in Section 3. How to derive the proposed method is given in Section 4. The algorithm is presented in Section 5. Section 6 addresses a computational experience. The conclusions can be found in Section 7.

2. Neighbourhood Search

It should be noted that, generally, in integer programming the reduced gradient vector, which is normally used to detect an optimality condition, is not available, even though the problems are convex. Thus we need to impose a certain condition for the local testing search procedure in order to assure that we have obtained the “best” suboptimal integer feasible solution.

Scarf [18] has proposed a quantity test to replace the pricing test for optimality in the integer programming problem. The test is conducted by a search through the neighbors of a proposed feasible point to see whether a nearby point is also feasible and yields an improvement to the objective function.

Let \( [\beta]_k \) be an integer point belongs to a finite set of neighborhood \( N([\beta]_k) \). We define a neighbourhood system associated with \( [\beta]_k \), that is, if such an integer point satisfies the following two requirements

1. \( \forall j \in N([\beta]_k) \), then \( [\beta]_k \neq [\beta]_j \)
2. \( N([\beta]_k) = \{[\beta]_k + N(0) \}

With respect to the neighborhood system mentioned above, the proposed integerizing strategy can be described as follows.

Given a non-integer component, \( x_{k} \), of an optimal vector, \( x_{k} \). The adjacent points of \( x_{k} \), being considered are \( [x_{k}] \) and \( [x_{k}] + 1 \). If one of these points satisfies the constraints and yields a minimum
deterioration of the optimal objective value we move to another component, if not we have integer-feasible solution.

Let \([x_k]\) be the integer feasible point which satisfies the above conditions. We could then say if
\([x_k] + 1 \in N([x_k])\) implies the point \([x_k] + 1\) is either infeasible or yields an inferior value to the objective function obtained with respect to \([x_k]\). In this case \([x_k]\) is said to be an “optimal” integer feasible solution to the integer programming problem. Obviously, in our case, a neighbourhood search is conducted through proposed feasible points such that the integer feasible solution would be at the least distance from the optimal continuous solution.

3. Solving Nonlinear Programming Problem
It has been our experience that many nonlinear programming problems are inordinately large because they are attempting to approximate, by piecewise linearization, what is essentially a nonlinear problem. It also appears that many real-life problems are such that only a small percentage of the variables are involved nonlinearly in the objective function. Therefore we should be able to write a class of nonlinear programming problems with the following form:

\[
\begin{align*}
\text{minimize} & \quad F(x) = f(x^N) + c^T x, \\
\text{subject to} & \quad Ax = b, \\
& \quad l \leq x \leq u
\end{align*}
\]

(4) where \(A\) is \(m \times n\), \(m \leq n\). Due to the separable property, \(w\) could partition \(x\) into a linear portion \(x^L\) and a nonlinear portion \(x^N\). The partition can be expressed as

\[
x = \begin{bmatrix}
x^N \\
x^L
\end{bmatrix}
\]

where \(x^N\) are called the nonlinear variables. Note that \(A\) and \(c\) operate on all variables \(x\). In some cases the part of \(c^T x\) involving \(x^N\) may be incorporated into \(f(x^N)\); in other cases \(c\) may be zero. We assume that the function \(f(x^N)\) is continuously differentiable in the feasible region, with gradient

\[
\nabla f(x^N) = g(x^N),
\]

and we assume that both \(f\) and \(g\) can be computed at any feasible point \(x^N\).

In essence the algorithm is an extension of the revised simplex method. To use some of the associated terminology, it might be described as an extension which permits more than \(m\) variables to be basic.

Partitioning \(x\) and \(F(x)\) into linear and nonlinear terms is of considerable practical importance; for descriptive purposes, however, it is convenient to denote \(F(x)\) and \(\nabla F(x)\) simply by \(f(x)\) and \(g(x)\).

With a few conventional exceptions, we use upper-case letters for matrices, lower-case for vectors and Greek lower-case for scalars. The quantity \(\varepsilon > 0\) represents the precision of floating-point arithmetic.

4. Basic of The Method
Basic solutions; justification for standard form

Before proceeding to the nonlinear problem, we need to introduce some linear programming background. In particular, equations (1) – (3) with \(f(x^N) = 0\) are the standard form for stating linear programs and for solving them in practical implementations of the simplex method. A basic solution is characterized by having at most \(m\) "basic" variables lying between their bounds while the remaining \(n - m\) "nonbasic" variables are equal to one bound or other. An associated square basis matrix \(B\) is drawn from the columns of the constraint matrix \(A\), and as the simplex method proceeds the columns of \(B\) are replaced one at a time.

Note that it is assumed in the standard form that \(A\) and \(x\) contain a full identity matrix and a full set of slack variables, respectively. (General equality and inequality constraints are accommodated by placing suitable upper and lower bounds on the slacks.) There are many practical reasons for retaining the standard form here. Full justification would require much implementation background, but very briefly, a sparse triangular factorization of \(B\) can be maintained more easily if columns (but not rows)
of \( B \) are altered. Further, although the total number of variables is apparently greater, it is very easy to take advantage of the unit vectors associated with slacks, whenever \( B \) is re-factorized.

### A. Superbasic variables

One virtue of the concept of basic solutions is the emphasis thereby given to the upper and lower bounds, \( l \leq x \leq u \). It is misleading to regard these as "sparse constraints"; more importantly, they serve directly to eliminate a large percentage of the variables. The simplex method is therefore free to focus its attention on transforming (factorizing) just \( B \), rather than the whole of \( A \).

With nonlinear problems we cannot expect an optimal point to be a basic solution. However, if the number of nonlinear variables is small, it seems reasonable to suppose that an optimal solution will be "nearly" basic. Thus, as a simple generalization we introduce the notion of superbasic variables and partition the set of general constraints (2) as follows:

\[
Ax = \begin{bmatrix} B & S & N \end{bmatrix} \begin{bmatrix} x_B \\ x_S \\ x_N \end{bmatrix} = \begin{bmatrix} b \end{bmatrix}
\]

The matrix \( B \) is square and nonsingular as in the simplex method, \( S \) is \( m \times s \) with \( 0 \leq s \leq n-m \), and \( N \) is the remaining columns of \( A \). The associated variables \( x_B, x_S, x_N \) are called the basics, superbasics and nonbasics respectively. Both basics and superbasics are free to vary between their bounds. The name is chosen to highlight the superbasics' role as "driving force"; they may be moved in any direction at all (preferably one that improves the objective value), and the basics are then obliged to change in a definite way to maintain feasibility with respect to the constraints \( Ax = b \).

Our expectation that solutions will often be "nearly" basic is confirmed by the following theorem:

**Theorem 1.** Suppose a nonlinear program has \( t \) variables occurring nonlinearly (in either the objective or the constraints). An optimal solution exists in which the number of superbasic variables \( s \) satisfies \( s \leq t \).

### Derivation of method

We will assume that \( f(x) \) can be expanded in a Taylor's series with remainder of second order:

\[
f(x + \Delta x) = f(x) + g(x)\Delta x + \frac{1}{2} \Delta x^T G(x + \gamma \Delta x) \Delta x
\]

where \( 0 \leq \gamma \leq 1 \), and \( G(x + \gamma \Delta x) \) is the Hessian matrix of second partial derivatives evaluated at some point between \( x \) and \( x + \Delta x \). Note that \( G \) is a constant matrix if \( f(x) \) is a quadratic function.

Let us partition \( \Delta x \) and \( g(x) \) corresponding to the partitioning of \( A \).

If \( f(x) \) were truly quadratic, we could obtain a constrained stationary point at \( x + \Delta x \) by requiring two properties of the step \( \Delta x \):

**Property 1.**

\[
\begin{bmatrix} B & S & N \end{bmatrix} \begin{bmatrix} \Delta x_B \\ \Delta x_S \\ \Delta x_N \end{bmatrix} = 0
\]

i.e., the step remains on the surface given by the intersection of the active constraints.
Property 2.

\[
\begin{bmatrix}
    g_B \\
g_S \\
g_N
\end{bmatrix} + G
\begin{bmatrix}
    \Delta x_B \\
    \Delta x_S \\
    \Delta x_N
\end{bmatrix} =
\begin{bmatrix}
    B^T & 0 \\
    S^T & 0 \\
    N^T & I
\end{bmatrix}
\begin{bmatrix}
    \mu \\
    \lambda
\end{bmatrix}
\]  \tag{10}

i.e., the gradient at \( x + \Delta x \) (given by the left hand-side of (10)) is orthogonal to the surface of active constraints and is therefore expressible as a linear combination of the active constraint normals.

For a more general function \( f(x) \), the step \( \Delta x \) may not lead directly to a stationary point, but we shall use Properties 1 and 2 to determine a feasible descent direction.

From (9) we have:

\[
\Delta x^N = 0,
\]  \tag{11}

and

\[
\Delta x_B = -W\Delta x_s,
\]  \tag{12}

where

\[
W = B^{-1}s.
\]  \tag{13}

Thus,

\[
\Delta x = \begin{bmatrix}
-\ W \\
 I \\
0
\end{bmatrix} \Delta x_s
\]

The following matrix can be used to simplify Eq. (10)

\[
\begin{bmatrix}
    I & 0 & 0 \\
    -W^T & I & 0 \\
    0 & 0 & I
\end{bmatrix}
\]  \tag{14}

First it provides an expression for estimates of the Lagrange multipliers for the general constraints:

\[
B^T \mu = g_B + \begin{bmatrix}
    I & 0 & 0 \\
\end{bmatrix} G \begin{bmatrix}
    -W \\
    I \\
0
\end{bmatrix} \Delta x_s
\]  \tag{15}

Note that when \( \|\Delta x_s\| = 0 \) (which will mean \( x \) is stationary) we have

\[
B^T \mu = g_B
\]  \tag{16}

As we observed that \( \mu \) is analogous to the pricing vector \( \pi \) in the revised simplex method. (From now on we shall denote the solution of (16) by \( \pi \).) Next we have from (10) that

\[
\lambda = g_N - N^T \mu + \begin{bmatrix}
    0 & 0 & I
\end{bmatrix} G \begin{bmatrix}
    -W \\
    I \\
0
\end{bmatrix} \Delta x_s
\]  \tag{17}

and again when \( \|\Delta x_s\| = 0 \) this equation reduces to

\[
\lambda = g_N - N^T \pi
\]  \tag{18}

which is analogous to the vector of reduced costs in linear programming.

The third result from equation (10), following pre-multiplication by the matrix (11), is an expression for the appropriate step:
\[
\begin{bmatrix}
-W^T & 0 \\
-I & 0
\end{bmatrix} \begin{bmatrix}
-W \\
-I \\
0
\end{bmatrix} \Delta x = -h
\]  

(19)

where

\[h = \begin{bmatrix}
-W^T & 0
\end{bmatrix} g = g_s - W^T g_B = g_s - S^T \pi\]  

(20)

The form of equation (16) suggests that

\[
\begin{bmatrix}
-W^T & 0 \\
-I & 0
\end{bmatrix} \begin{bmatrix}
-W \\
-I \\
0
\end{bmatrix}
\]

(21)

can be regarded as a "reduced" Hessian and \(h = [-W^T \ I \ 0]g\) a reduced gradient, with (19) giving a Newton step in the independent variables \(\Delta x\). Note that \(\|\mu\| = 0\) becomes a necessary condition for a stationary point on the current set of active constraints, which, if the reduced Hessian is nonsingular, implies that \(\|\Delta x\| = 0\).

Summary

We considered a class of algorithms in which the search direction along the surface of active constraints is characterized as being in the range of a matrix \(Z\) which is orthogonal to the matrix of constraint normals. Thus, if \(\tilde{A}x = \tilde{b}\) is the current set of \(n - s\) active constraints, \(Z\) is an \(n \times s\) matrix such that \(AZ = 0\).

\(\tilde{A}Z = 0\).  

(22)

The main steps to be performed at each iteration are as follows.

(A) Compute the reduced gradient \(g_A = Z^T g\).

(B) Form some approximation to the reduced Hessian, viz.

\[G_A = Z^T G Z\]

(C) Obtain an approximate solution to the system of equations

\[Z^T G Z p_A = -Z^T g\]

(23)

by solving the system

\[G_A p_A = -g_A.\]

(D) Compute the search direction \(p = Z p_A\).

(E) Perform a linesearch to find an approximation to \(a^*\), where

\[f(x + \alpha p) = \min_{\{x + \alpha \mu \text{ feasible}\}} f(x + \alpha p)\]

Apart from having full column rank, eq. (22) is (algebraically) the only constraint on \(Z\) and thus \(Z\) may take several forms. The particular \(Z\) corresponding to our own procedure is of the form

\[
Z = \begin{bmatrix}
-W \\
-I \\
0
\end{bmatrix} = \begin{bmatrix}
m \\
-s \\
-n-m-s
\end{bmatrix}
\]  

(24)

This is a convenient representation which we will refer to for exposition purposes in later sections, but we emphasize that computationally we work only with \(S\) and a triangular (LU) factorization of \(B\). The matrix \(Z\) itself is never computed.

5. The Algorithm

After solving the relaxed problem, the procedure for searching a suboptimal but integer-feasible solution from an optimal continuous solution can be described as follows.

Let...
be the (continuous) solution of the relaxed problem, \([x]\) is the integer component of non-integer variable \(x\) and \(f\) is the fractional component.

Stage 1.

Step 1. Get row \(i^*\) the smallest integer infeasibility, such that
\[
\delta_i^* = \min\{f_i, 1 - f_i\}
\]
Step 2. Calculate
\[
v_i^T = e_i^T B^{-1}
\]
this is a pricing operation
Step 3. Calculate \(\sigma_{ij} = v_i^T a_j\)
With \(j\) corresponds to
\[
\min_j \left\{\frac{d_j}{\sigma_{ij}}\right\}
\]
I. For nonbasic \(j\) at lower bound
- If \(\sigma_{ij} < 0\) and \(\delta_i^* = f_i\) calculate \(\Delta = \frac{(1-\delta_i^*)}{-\sigma_{ij}}\)
- If \(\sigma_{ij} > 0\) and \(\delta_i^* = 1 - f_i\) calculate \(\Delta = \frac{(1-\delta_i^*)}{\sigma_{ij}}\)
- If \(\sigma_{ij} < 0\) and \(\delta_i^* = 1 - f_i\) calculate \(\Delta = \frac{\delta_i^*}{-\sigma_{ij}}\)
- If \(\sigma_{ij} > 0\) and \(\delta_i^* = f_i\) calculate \(\Delta = \frac{\delta_i^*}{\sigma_{ij}}\)
II. For nonbasic \(j\) at upper bound
- If \(\sigma_{ij} < 0\) and \(\delta_i^* = 1 - f_i\) calculate \(\Delta = \frac{(1-\delta_i^*)}{-\sigma_{ij}}\)
- If \(\sigma_{ij} > 0\) and \(\delta_i^* = f_i\) calculate \(\Delta = \frac{(1-\delta_i^*)}{\sigma_{ij}}\)
- If \(\sigma_{ij} > 0\) and \(\delta_i^* = 1 - f_i\) calculate \(\Delta = \frac{\delta_i^*}{\sigma_{ij}}\)

Otherwise go to next non-integer nonbasic or superbasic \(j\) (if available). Eventually the column \(j^*\) is to be increased from LB or decreased from UB. If none go to next \(i^*\).

Step 4. Calculate
\[
a_j^* = B^{-1} a_j^*
\]

i.e. solve \(B a_j^* = a_j^*\) for \(a_j^*\)
Step 5. Ratio test; there would be three possibilities for the basic variables in order to stay feasible due to the releasing of nonbasic \(j^*\) from its bounds.

If \(j^*\) lower bound
Let
\[
A = \min_{i^* \neq i^* | \sigma_{ij} > 0} \left\{\frac{x_{B_i} - l_i^*}{\sigma_{ij}}\right\}
\]
\[
B = \min_{i^* \neq i^* | \sigma_{ij} > 0} \left\{\frac{u_i^* - x_{B_i}}{-\sigma_{ij}}\right\}
\]
\[
C = \Delta
\]
the maximum movement of \(j^*\) depends on: \(\theta^* = \min(A, B, C)\)

If \(j^*\) upper bound
Let
\[ A' = \min_{i' \neq i \mid a_{ij} < 0} \left( \frac{x_{B_i} - l_i}{a_{ij}} \right) \]

\[ B' = \min_{i' \neq i \mid a_{ij} > 0} \left( \frac{u_i - X_{B_i}}{-a_{ij}} \right) \]

\[ C' = \Delta \]

The maximum movement of \( j^* \) depends on: \( \theta^* = \min(A', B', C') \)

Step 6. Exchanging basis for the three possibilities
1. If \( A \) or \( A' \)
   - \( x_{B_i} \) becomes nonbasic at lower bound \( l_i \)
   - \( x_{j^*} \) becomes basic (replaces \( x_{B_i} \))
   - \( x_{i^*} \) stays basic (non-integer)
2. If \( B \) or \( B' \)
   - \( x_{B_i} \) becomes nonbasic at upper bound \( u_i \)
   - \( x_{j^*} \) becomes basic (replaces \( x_{B_i} \))
   - \( x_{i^*} \) stays basic (non-integer)
3. If \( C \) or \( C' \)
   - \( x_{j^*} \) becomes basic (replaces \( x_{i^*} \))
   - \( x_{i^*} \) becomes superbasic at integer-valued

Step 7. If row \( i^* = \{ \} \) go to Stage 2, otherwise
Repeat from step 1.

Stage 2. Do integer lines search to improve the integer feasible solution

6. Computational Experience: A Plant Upgrading Problem

A plant upgrading problem, cited in Daellenbach et al. (1983), describes a decision problem faced by a manufacturer of an electrical wire in order to upgrade its machinery. At present some of the machines are nearly worn out or obsolete, so that running costs are becoming unacceptably high. A shift in technology has caused an expansion in demand for the type of wire this manufacturer produces. However, the present plant cannot meet the demand projected for the near future.

There are two types of wire produced by this manufacturer: bare wire and covered wire. Each of these two products has two different sizes. In terms of the machines availability, the manufacturer has various options, including retaining and modifying some existing machines and purchasing new machines. Therefore it is desirable to draw a recommended configuration of the plant to meet the projected demand as well as to schedule which machine should produce each type of product such that the total running cost is minimized.

7. Mathematical Statement of the Problem

Let \( y_i \) be the number of machines of options \( i \), and \( x_{ij} \) be the proportion of annual machines time on machines of options \( i \) to produce wire size \( j \), \( (i = 1, \ldots, 5 \text{ and } j = 1, 2) \).

The problem can be formulated mathematically as follows.

The objective of the problem is to minimize the total running cost (in $1000’s)

\[
\text{Minimize } P = 50y_1 + 33.6x_{11} + 32.88x_{12} + 50y_2 + 47.4x_{21} \\
+47.04x_{22} + 80y_3 + 58.8x_{31} + 57x_{32} + 100y_4 \\
+62.4x_{41} + 54.7x_{42} + 140y_5 + 86.4x_{51} + 82.8x_{52}
\]

There are several constraints have to be satisfied. Firstly is to meet the projected sales demand for each type and size of wire.

For bare wire of size 1 and 2

\[ 5880x_{11} + 8820x_{21} - 7200x_{11} - 9600x_{41} \geq 3000 \text{ (BARE1)} \]
4th International Conference on Operational Research (InteriOR) IOP Publishing
IOP Conf. Series: Materials Science and Engineering 300 (2018) 012022 doi:10.1088/1757-899X/300/1/012022

4704x_{12} + 8232x_{22} - 6000x_{32} - 7800x_{42} \geq 2000 \text{ (BARE2)}

For covered wire of size 1 and 2

\[ 6984x_{11} + 9312x_{11} + 9312x_{11} \geq 14000 \text{ (COVRD1)} \]

\[ 5820x_{22} + 7566x_{22} + 6984x_{32} \geq 10000 \text{ (COVRD2)} \]

Next is to make certain that the number of machines available for each option provides a production capacity that covers the production needs. Hence, the number of machine of type \( i \) is required for the production of all wire sizes.

\[ y_i \geq x_{i1} + x_{i2}, \quad i = 1, \ldots, 5 \text{ (MATCH}\ i\text{)} \]

The restrictions of the machine options

\[ y_i \leq 2 \text{ (REST1)} \]

\[ y_3 + y_4 \leq 1 \text{ (REST2)} \]

8. Solution to the Problem

The plant upgrading problem mentioned in the previous section is a MILP problem. It has 11 constraints, 10 continuous variables and 5 integer variables.

The relaxed problem is solved on PC optimization software. We then integerized the continuous result using our proposed algorithm. The continuous results as well as the integer results of the plant upgrading problem are presented in Table 1. The total computational time for both continuous and integer results is 2.5 seconds.

**Table 1. Results for Plant Upgrading Problem**

| Variable | Activity in Cont. Soln. | Activity After Integ. Process |
|----------|-------------------------|------------------------------|
| \( x_{11} \) | 0.5102 | 0.51020 |
| \( x_{12} \) | 0.0 | 0.42517 |
| \( x_{21} \) | 0.0 | 0.0 |
| \( x_{22} \) | 0.24295 | 0.0 |
| \( x_{31} \) | 0.0 | 0.0 |
| \( x_{32} \) | 0.0 | 0.0 |
| \( x_{41} \) | 0.0 | 0.0 |
| \( x_{42} \) | 0.0 | 0.0 |
| \( x_{51} \) | 1.50344 | 1.50344 |
| \( x_{52} \) | 1.43184 | 1.43184 |
| \( y_1 \) | 0.5102 | 1.0 |
| \( y_2 \) | 0.24295 | 0.0 |
| \( y_3 \) | 0.0 | 0.0 |
| \( y_4 \) | 0.0 | 0.0 |
| \( y_5 \) | 2.93528 | 3.0 |

**Obj. Value (P)**

| 715.41817 | 729.57606 |

9. Conclusions

This paper has presented a direct search method for achieving integer-feasibility for a class of mixed-integer nonlinear programming problems in a relatively short time. The direct search approach used the
strategy of releasing nonbasic variable from their bounds, combined with the “active constraint” method and the notion of superbasic. After solving a problem by ignoring the integrality requirements, this strategy is used to force the appropriate non-integral basic variables to move to their neighborhoods integer points.

A study of the criteria for choosing a nonbasic variable to work with in the integerizing strategy has also been made. The number of integerizing steps would be finite if the number of integer variables contained in the problem is finite. However, it should be noted that the computational time for the integerizing process does not necessarily depend on the number of integer variables, since many of the integer variables may have an integer value at the continuous optimal solution.

The new direct search method has been shown to be successful on a range of problems, while not always able to achieve global optimality. In a number of cases to obtain the suboptimal point is acceptable, since the exponential complexity of the combinatorial problems in general precludes branch-and-bound, except on small to medium problems.

Computational testing of the procedure presented this paper has demonstrated that it is a promised approach for large problems.

10. References
[1] Bienstock, D., 1996. Computational study of a family of mixed-integer quadraticprogramming problems, Mathematical Programming. 74, pp. 121-140.
[2] Jobst, N. J., Horniman, M. D., Lucas, C. A., & Mitra, G., 2001. Computational aspects of alternative portfolio selection models in the presence of discrete asset choice constraints, Quantitative Finance 1, pp. 489-501.
[3] Danna, E., Rothberg, E., & LePape, C., 2005. Exploring relaxation induced neighborhoods to improve MIP solutions, Mathematical Programming. 102, pp. 71-90.
[4] Sawaya, N., 2006. Reformulations, relaxations and cutting planes for generalized disjunctive programming. PhD thesis, Chemical Engineering Department, Carnegie Mellon University.
[5] Flores-Tlacuahuac, A., & Biegler, L. T., 2007. Simultaneous mixed-integer dynamic optimization for integrated design and control, Computers and Chemical Engineering. 31, pp. 648–656.
[6] Laird, C. D., Biegler, L. T., & Van Bloemen Waanders, B., 2006. A mixed integer approach for obtaining unique solutions in source inversion of drinking water networks, Journal of Water Resources Planning and Management, Special Issue on Drinking Water Distribution Systems Security. 132, pp. 242-251.
[7] Kim, J. S., & Edgar, T. F., 2014. Optimal scheduling of combined heat and powerplants using mixed-integer nonlinear programming. Energy, 77, pp. 675-690.
[8] Yang, Y., Zhang, S., & Xiao, Y., 2015. An MILP (mixed integer linear programming) model for optimal design of district scale distributed energy resource system. Energy, 90, pp. 1901-1915.
[9] Kaya, O., Urek, B., 2016. A mixed integer nonlinear model and heuristic solutions for location, inventory and pricing decisions in a closed loop supply chain. Computers & Operations Research, 65, pp. 93-103.
[10] Duran, M. A., & Grossmann, I. E., 1986. An Outer-Approximation Algorithm For a class of Mixed-Integer Nonlinear Programs. Mathematical Programming 36, pp. 307.
[11] Roger Fletcher & Sven Leyffer, 1994. Solving Mixed Integer Nonlinear Programming by Outer Approximation. Mathematical Programming, 66, pp. 327.
[12] Belotti, P., Lee, J., Liberti, L., & Margot, F., 2009. A. Wachter, Branching and Bounds Tightening Techniques for non-convex MINLP. Optimization Methods and Software, 24(4), pp. 597-634.
[13] Gupta, O. K., & Ravindran, A., 1985. Branch and Bound Experiments in Convex Nonlinear Integer Programming. Manage Sci., 31 (12), pp. 1533-1546 (1985).
[14] Nannicini, G., & Belotti, P., 2009. Local Branching for MINPs. Technical Report workingpaper, CMU.
[15] Leyffer, S., 2001. Integrating SQP and Branch and Bound for mixed integer nonlinear programming. Computational Optimization and Applications, 18, pp. 295-309.
[16] Geoffrion, A. M., 1972. *A Generalized Benders Decomposition*. J. Optim. Theory Appl., 10 (4), pp. 237-260.

[17] Vigerske, S., & Gleixner, A., 2017. *SCIP: Global optimization of mixed integer nonlinear programs in a branch and cut framework*. ZIB Report, pp. 16-24 (Revised version May 2017).

[18] Burer, S., & Letchford, A., 2012. *Non-convex mixed-integer nonlinear programming: A survey*, Surv. Oper. Res. Manag. Sci., 17, pp. 97-106.

[19] Liberti, L., Nannicini, G., & Mladenovic, N., 2009. *A good recipe for solving MINLPs*. In: V. Maniezzo, T. Stutzle, S. Voss (eds.) Metaheuristics: Hybridizing metaheuristics and Mathematical Programming. Annals of Information Systems, 10, pp. 231-245. Springer.

[20] Berthold, T., & Gleixner, A., 2010. *Undercover-Primal MINLP Heuristic*. In P. Bonami, L. Liberti, A. Miller and A. Sartenaer (eds.) Proceedings of the European Workshop on MINLP, pp 103-113, Marseille.

[21] Mawengkang, H., & Murtagh, B. A., 1986. *Solving Nonlinear Integer Programs with Large-Scale Optimization Software*. Annals of Operations Research, pp. 5425-437.

[22] D’Ambrosio, C., Frangioni, A., Liberti, L., & Lodi, A., 2010. *A storm of Feasibility Pump for non-convex MINLP*. Tech. Rep. OR-10-13, DEIS, Universita di Bologna.

[23] Fischetti, M., Glover, F., & Lodi, A., 2005. *The Feasibility Pump*. Mathematical Programming A 104(1), 91-104.

[24] Nannicini, G., Belotti, P., 2009. *Rounding-based heuristics for non-convex MINLPs*. In: P. Bonami, L. Liberti, A. Miller, A. Sartenaer (eds). Proceedings of the European Workshop on MINLP, CIRM, Marseille, France.

[25] Liberti, L., Miladenoiev, N., & Nannicini, G., 2011. *A Recipe for Finding Good Solutions to MINLPs*. Mathematical Programming Computation, August 2011.

[26] Vavasis, S. A., 1991. *Nonlinear Optimization: Complexity Issues*. Oxford University Press, Oxford.

[27] D’Ambrosio, C., Frangioni, A., Liberti, L., & Lodi, A., 2010. *Experiments with a Feasibility Pump approach for non-convex MINLPs*. In: P. Festa (ed) Proceedings of the 9th Symposium on Experimental Algorithms (SEA 2010), Lecture Notes in Computer Science, 6049. Springer, Berlin.

[28] Scarf, H. E., 1986. *Testing for Optimality in the Absence of Convexity*. In: W. P. Heller, R. M Starr and D. A. Starett (Eds) Cambridge University Press, pp. 117-134

[29] Tamara G. Kolda., Robert M. Lewis, Virginia Torezon, 2003. *Optimization by Direct Search: New Perspective on Some Classical and Modern Methods*. SIAM Review, 45 (1), pp. 385-482.

[30] Westerlund, T., & Petersson, F., 1995. *A Cutting Plane Method for Solving Convex MINLP Problems*. Computers Chem. Eng., 19, pp. 131-136.