# An Outer Approximation based Branch and Cut Algorithm for convex 0-1 MINLP problems

Ioannis Akrotirianakis[1], Istvan Maros[2] and Berç Rustem[3]

**Abstract**

A branch and cut algorithm is developed for solving 0-1 MINLP problems. The algorithm integrates Branch and Bound, Outer Approximation and Gomory Cutting Planes. Only the initial Mixed Integer Linear Programming (MILP) master problem is considered. At integer solutions Nonlinear Programming (NLP) problems are solved, using a primal-dual interior point algorithm. The objective and constraints are linearized at the optimum solution of those NLP problems and the linearizations are added to all the unsolved nodes of the enumerations tree. Also, Gomory cutting planes, which are valid throughout the tree, are generated at selected nodes. These cuts help the algorithm to locate integer solutions quickly and consequently improve the linear approximation of the objective and constraints, held at the unsolved nodes of the tree. Numerical results show that the addition of Gomory cuts can reduce the number of nodes in the enumeration tree.

**Key Words:** Outer Approximation, Branch and Cut, Gomory Cuts.

June, 2000,
Technical Report 2000-06,
Department of Computing,
Imperial College of Science, Technology and Medicine,
180 Queen's Gate, London SW7 2BZ, U.K.

1. e-mail: ia4@doc.ic.ac.uk
2. e-mail: im@doc.ic.ac.uk
3. e-mail: br@doc.ic.ac.uk

# 1 Introduction

In this paper, we develop an algorithm for solving 0-1 convex MINLP problems. The general structure of convex 0-1 MINLP problems, considered in this paper, has the form

$$P \begin{cases} \min & f(x,\psi) \\ ST & g(x,\psi) \leq 0 \\ & x \in X, \psi \in Y, \end{cases} \tag{1}$$

where $f(x,\psi)$ and $g(x,\psi) = (g^1(x,\psi), ..., g^m(x,\psi))^T$ are convex functions, $X \subseteq \Re^n$, $Y \subseteq \{0,1\}^p$, and $x$ and $\psi$ are the vectors of the continuous and 0-1 variables, respectively. Also, $X$ is assumed to be a convex compact polyhedron, defined as $X = \{x \in \Re^n : \ A_1 x \leq a_1\}$ and $Y$ is a finite discrete set, defined as $Y = \{\psi \in \{0,1\}^p : \ A_2\psi \leq a_2\}$. If the objective and the constraint functions are linear then problem $P$ forms a 0-1 Mixed Integer Linear Programming (MILP) problem.

There is a large number of engineering problems that can be modeled and solved as MINLP problems. These include the synthesis of processes [7], the design of batch plants [15], the minimization of waste in paper cutting [24], the optimization of core reload patterns for nuclear reactors [20], the facility location in a multi attribute space [7], to mention but a few. An extensive survey of application areas of MINLP can be found in Grossmann and Kravanja [12].

The methods for solving MINLP problems use mainly two approaches. The first approach fixes all the integer variables at a feasible integer vector. As a result, the MINLP problem becomes a continuous NLP problem. The optimum solution of the NLP problem together with the corresponding integer vector provide a feasible solution of the original MINLP problem. Also, the corresponding objective value provides an upper bound to the optimal objective value. Outer Approximation [4] and Generalized Benders Decomposition [9] use this approach. The second approach relaxes all the integrality restrictions, that is the integer variables are allowed to take real values. The resulting problem, usually called the *continuous relaxation* of the MINLP problem, has some attractive properties. The first is that the optimum objective value of the continuous relaxation provides a lower bound to the optimum objective value of the original MINLP problem. If the optimum solution of the continuous relaxation satisfies all the integrality restrictions, then it is a feasible solution of the MINLP problem. Similarly, if the continuous relaxation is infeasible, the MINLP problem is also infeasible. Branch and Bound uses these properties and a tree search to find the optimum solution of problem P.

The basic idea underlying the algorithm presented in this paper is to include cutting plane methods, an area that is mainly used to solve MILP problems, within an algorithm designed to solve 0-1 convex MINLP problems. Recently, Balas *et al* [2] has shown that the incorporation of classical Gomory mixed integer cuts within a Branch and Cut algorithm can solve not only faster but also larger MILP problems than simple Branch and Bound. In the last decade, many Branch and Cut algorithms have been developed enabling the solution of very large MILP problems previously considered to be unsolvable. Jünger *et al.* [13] provide an extensive and recent survey of Branch and Cut algorithms for MILP.

Our algorithm uses the LP/NLP based Branch and Bound method developed by Quesada and Grossmann as the platform for introducing Gomory mixed integer cuts in the solution process of a convex MINLP problem. LP/NLP based Branch and Bound is a combination of Outer Approximation [4], [6] and Branch and Bound [16], [3] methods. Its advantage over Outer Approximation is that it avoids the explicit solution of all the MILP master problems. Instead, it solves only the initial MILP master problem by using Branch and Bound. At integer nodes of the tree, NLP problems are solved, resulting by fixing the integer variables of the original MINLP problem. New linearizations are generated around the optimum solution of those NLP problems and then added to all the unsolved nodes, thereby dynamically updating the linear approximations of the original MINLP problem. The objective of the proposed algorithm is to reduce both the number of nodes and the number of NLP problems, required by the LP/NLP based Branch and Bound method, by incorporating Gomory mixed integer cuts within it.

The assumptions used throughout this paper are the following:

**Assumptions**:

A1: $X$ is a nonempty compact convex set defined by a system of linear inequality constraints and the objective $f(x, \psi)$ and constraints $g(x, \psi)$ are convex functions.

A2: $f(x, \psi)$ and $g(x, \psi)$ are once continuously differentiable.

A3: The set of feasible directions at the solution of a primal problem (defined in section 2.1 below) can be identified with the set of feasible directions for the constraint linearized at the solution (see for example Fletcher [5], page 202), for which a sufficient condition is that the gradients of the active constraints are linear independent. This is known as the constraint qualification condition.

The convexity assumption ensures that problem (1) has only one minimum. The method discussed in this paper need this assumption in order to guarantee convergence to the minimizer. Although it can be used to solve non convex MINLP problems (i.e., the objective or at least one of the constraints are non convex functions) there is no guarantee that it will find the global minimum.

The paper is organized as follows. Section 2 briefly reviews Outeer Approximation, Branch and Bound and Gomory mixed integer cuts. In section 3 the motivation of the proposed algorithm is presented. In section 4 the new algorithm is developed. In section 5 various implementation issues are discussed. Finally, in section 6 numerical results obtained by solving several MINLP problems are presented.

# 2 Background

In this section, we discuss Branch and Bound, and Outer Approximation methods. They constitute two of the most commonly used methods for solving 0-1 convex MINLP problems. The algorithm developed in this paper uses ideas from both of these algorithms. We also discuss Cutting Plane methods for 0-1 MILP problems, with particular emphasis on

Gomory cutting planes. Our purpose is to make the reader familiar with the basic ideas and terminology used in section 4 where we present our algorithm for solving 0-1 convex MINLP problems.

## 2.1 Outer Approximation

Outer Approximation was initially proposed by Duran and Grossmann [4] for the class of convex 0-1 MINLP problems where only inequalities are present, the objective and constraints are assumed to be separable in the continuous and 0-1 variables as well as linear in the 0-1 variables. Fletcher and Leyffer [6] generalize the method of Duran and Grossmann [4] to allow nonlinear integer variables at the objective and constraints in a non-separable way.

The algorithm alternates between NLP and MILP problems. The NLP problems are formed by fixing the integer variables to an integer vector, say $\psi_j$. The resulting problem is usually called the primal problem and has the form

$$NLP(\psi_j) \begin{cases} \min_x & f(x, \psi_j) \\ ST & g(x, \psi_j) \leq 0 \\ & x \in X \end{cases} \tag{2}$$

If (2) is feasible and $x_j$ is its optimal solution, then $f(x_j, \psi_j)$ provides an upper bound to the solution of the original MINLP problem. Also, $x_j$ provides the point at which the objective and constraints are linearised in order to form the master problem.

If the primal problem (2) is infeasible, the point used to linearise the objective and constraints is the solution of a *feasibility* problem. Two straightforward and widely used feasibility problems are the minimization of the $\ell_1$ or the $\ell_\infty$ sum of the constraint violations of the infeasible primal problem. That is, if the integer vector $\psi_j$ gives rise to an infeasible primal problem, then the $\ell_1$ feasibility problem is defined by

$$F_{\ell_1}(\psi_j) \begin{cases} \min\limits_{x \in X} & \sum\limits_{l=1}^{q} g_+^l(x, \psi_j) \end{cases}$$

and the $\ell_\infty$ feasibility problem by

$$F_{\ell_\infty}(\psi_j) \begin{cases} \min\limits_{x \in X} & \max\limits_{l=1,\ldots,q} g_+^l(x, \psi_j) \end{cases}$$

where $g_+^l(x, \psi_j) = \max\left\{0, g^l(x, \psi_j)\right\}$.

At the $i$-th iteration of the algorithm the MILP master problem has the form

$$M_i \begin{cases} \min_{x,\psi,\eta} & \eta \\ ST & \eta < UBD_i \\ & f(x_j,\psi_j) + \nabla f(x_j,\psi_j)^T \begin{pmatrix} x - x_j \\ \psi - \psi_j \end{pmatrix} \leq \eta \\ & g(x_j,\psi_j) + \nabla g(x_j,\psi_j)^T \begin{pmatrix} x - x_j \\ \psi - \psi_j \end{pmatrix} \leq 0, \quad \forall j \in T_i \\ & g(x_k,\psi_k) + \nabla g(x_k,\psi_k)^T \begin{pmatrix} x - x_k \\ \psi - \psi_k \end{pmatrix} \leq 0, \quad \forall k \in S_i \\ & x \in X, \psi \in Y, \eta \in \Re \end{cases} \tag{3}$$

where $UBD_i$ is the objective value of the best integer solution found so far, the set $T_i$ contains all the iterations $j \leq i$ in which the corresponding primal problems $NLP(\psi_j)$ are feasible, whereas the set $S_i$ contains the iterations $k \leq i$ in which feasibility problems need to be solved. The presence of the constraint $\eta < UBD_i$ in the master problem, ensures that each integer assignment $\psi_j$, with $j \in T_i$, is generated only once. Since there is a finite number of integer vectors in the feasible region of the original MINLP problem, the algorithm terminates after a finite number of steps [6]. The optimum solution $(\eta_i^*, x_i^*, \psi_i^*)$ of the relaxed master problem $M_i$ can be found by any standard MILP technique such as Branch and Bound. The vector $\psi_i^*$ provides the assignment at which the integer variables will be fixed at the next iteration, that is $\psi_{i+1} = \psi_i^*$. The algorithm proceeds by solving primal, feasibility and master problems until an infeasible master problem is encountered.

## 2.2 Branch and Bound

Branch and bound is a classical algorithm for solving mixed integer linear or nonlinear programming problems. In this section, we describe the application of Branch and Bound on 0-1 convex MINLP problems. The basic concept of Branch and Bound is to split the initial 0-1 MINLP problem $P$ into smaller subproblems, which are then solved in some specified order. It was initially developed by Land and Doig [16] for linear integer programming problems and later was generalized by Dakin [3] to be able to handle nonlinear integer problems as well.

Before describing how a general iteration of the Branch and Bound algorithm works, we introduce some basic notions used by the algorithm throughout its execution. The algorithm maintains a list which keeps all the subproblems that need to be solved. Each subproblem is based on the initial problem $P$ and has some of its binary variables fixed at 0 or 1. Hence, the $i$-th subproblem in the list can be identified by the pair $(R_0^i, R_1^i)$, where the sets $R_0^i, R_1^i \subseteq \{1, 2, ...p\}$ contain the indices of those binary variables which have been fixed at 0 and 1 respectively. Any 0-1 variable $\psi^{(j)}$, with $j \notin R_0^i \cup R_1^i$, is called

*free binary variable* and can take any value in the interval $[0, 1]$. If the solution of the continuous relaxation of a subproblem $(R_0^i, R_1^i)$ satisfies the integrality restrictions, then this solution provides an *upper bound* on the optimum solution of the original problem $P$. The algorithm keeps a record of the integer solution that provides the lowest upper bound found so far. That solution is called *incumbent*.

At the beginning of a general iteration $k$, the algorithm removes a subproblem $(R_0^i, R_1^i)$ from the list and solves its continuous relaxation. Four possible cases can be identified. The first case arises when the optimum solution contains integer restricted variables with fractional values and its optimal objective value is less than the objective value at the incumbent solution. In this case two new subproblems are generated, by branching on a violated free binary variable, say $\psi^{(j)}$. The two new subproblems are defined by the pairs $(R_0^i \cup \{j\}, R_1^i)$ and $(R_0^i, R_1^i \cup \{j\})$. Both of these new subproblems are added to the list of unsolved problems. This is the only case where two new subproblems are added to the list for one deleted. The remaining three cases do not add new subproblems to the list.

The second case arises when the current subproblem $(R_0^i, R_1^i)$ is infeasible. This means that any further restriction of it will also be infeasible. The third case results when the optimal objective value of the current subproblem is greater than the objective value at the incumbent solution. In this case no further restriction of $(R_0^i, R_1^i)$ will give a better integer solution. The fourth and final case appears when the optimum solution of the current subproblem satisfies all the integrality constraints and the objective value is less than the one at the incumbent solution. In this case the solution of the current subproblem becomes the new incumbent solution. The Branch and Bound algorithm terminates when the list of unsolved problems becomes empty. The current incumbent solution becomes the optimum solution of the original problem $P$.

This process of branching, solving and fathoming subproblems, based on the original problem $P$, can be viewed as an iterative generation of a binary tree. The $i$-th node of the binary tree corresponds to a subproblem $(R_0^i, R_1^i)$. The root of the binary tree then corresponds to the pair $(\emptyset, \emptyset)$, since none of the integer variables has yet been fixed. The two new subproblems $(R_0^i \cup \{j\}, R_1^i)$ and $(R_0^i, R_1^i \cup \{j\})$ which are generated by branching on the violated free binary variable $\psi^{(j)}$ are called *child nodes* whereas the subproblem $(R_0^i, R_1^i)$ is called the *parent node*. Finally, we define the *level* of a node $(R_0, R_1)$ in the binary tree as the number of edges on the path connecting the root of the tree with that node.

## 2.3 Gomory Mixed Integer Cuts

Cutting Plane methods provide deterministic procedures for solving Mixed Integer Linear Programming (MILP) problems. They attempt to solve an MILP problem by solving a finite sequence of LP problems. Each of these LP problems is based on the continuous relaxation of the initial MILP problem and contains a number of additional constraints called *cuts*. A cut is a fractional constraint that reduces the feasible region of the LP relaxation problem without eliminating any feasible solution of the initial 0-1 MILP problem. Thus the main aim of Cutting Plane algorithms is to approximate the convex hull of

an MILP problem, at least in a neighborhood of its optimum solution, by generating and adding cuts to its continuous relaxation.

The first cutting plane method was developed by Gomory [10], [11]. Gomory mixed integer cuts can be obtained from every row corresponding to a fractional basic variable in the optimal tableau of an LP relaxation. Let the optimal tableau be

$$\begin{aligned} x_i &= g_{i0} + \sum_{j \in J_1} g_{ij}(-x_j) + \sum_{j \in J_2} g_{ij}(-\psi_j),\ i \in I_1 \\ \psi_i &= g_{i0} + \sum_{j \in J_1} g_{ij}(-x_j) + \sum_{j \in J_2} g_{ij}(-\psi_j),\ i \in I_2 \\ & x_k, \psi_k \geq 0,\ \forall k \in I \cup J \end{aligned} \tag{4}$$

where $I_1$ and $I_2$ denote the index sets of the basic continuous and 0-1 variables respectively, $J_1$ and $J_2$ denote the index sets of the non-basic continuous and 0-1 variables respectively, $I = I_1 \cup I_2$, $J = J_1 \cup J_2$ and $I_1 \cap I_2 = J_1 \cap J_2 = \emptyset$. Assume that the $i$-th 0-1 variable $\psi_i$ has a fractional value and define the following partitions of the sets $J_1$ and $J_2$:

$$J_1^+ = \{j \in J_1 :\ g_{ij} > 0\},\ J_1^- = J - J_1^+$$

and

$$J_2^+ = \{j \in J_2 :\ f_{ij} < f_{i0}\},\ J_2^- = J - J_2^+$$

The Gomory mixed integer cut is defined by the inequality

$$\sum_{j \in J_1^+} \frac{g_{ij}}{f_{i0}} x_j + \sum_{i \in J_2^+} \frac{f_{ij}}{f_{i0}} \psi_j + \sum_{j \in J_1^-} \frac{g_{ij}}{1 - f_{i0}} x_j + \sum_{i \in J_2^-} \frac{1 - f_{ij}}{1 - f_{i0}} \psi_j \geq 1 \tag{5}$$

which can be written in the following compact form

$$\sum_{j \in J_1} \max\left\{\frac{g_{ij}}{f_{i0}}, \frac{g_{ij}}{1 - f_{i0}}\right\} x_j + \sum_{j \in J_2} \min\left\{\frac{f_{ij}}{f_{i0}}, \frac{1 - f_{ij}}{1 - f_{i0}}\right\} \psi_j \geq 1 \tag{6}$$

The derivation of the above inequality can be found for example in [2], [21].

Gomory mixed integer cuts have fallen out of favour for more than thirty years due to their slow convergence to the integer solution. Recently, however, Balas *et al* [2] have shown that Gomory mixed integer cuts can be very useful when they are incorporated within a Branch and Bound algorithm, since they can improve the bounds at different nodes of the entire search tree resulting in substantial reductions of its size.

# 3 Motivation for the algorithm

In this section we initially discuss the advantages and disadvantages of nonlinear Branch and Bound, Outer Approximation and LP/NLP based Branch and Bound which is a combination of the first two. Next we present our motivation for developing the new algorithm.

Although Branch and Bound is a classical, easy to implement and widely used method, it has two important disadvantages that may damage its performance in medium or large convex MINLP problems. The first is that an NLP problem has to be solved at every node of the enumeration tree. Thus, the computational effort that the algorithm needs to spend at every node is high. The second disadvantage, which is related to the first, arises in problems where integer feasible solutions are not available in the early stages of the algorithm. Due to poor fathoming of unsolved nodes, the enumeration tree becomes excessively large. As a result the algorithm may need prohibitively long time to solve the problem or even fail to find the optimum solution because there is not enough memory to accommodate the large number of unsolved nodes.

On the other hand Outer Approximation involves the complete solution of an NLP primal problem and an MILP master problem at every major iteration. Although it is generally expected to need more computational effort at every iteration than Branch and Bound, it requires fewer iterations to reach the integer optimum solution. As a result, Outer Approximation solves less NLP problems than Branch and Bound. In addition those problems are smaller than the problems solved at every node of the Branch and Bound tree, since the integer variables have been fixed. However, Outer Approximation requires the complete solution of an MILP master problem at each iteration which is usually a heavy computational task.

A major limitation of Outer Approximation is that the size of the master problems increases very fast as the iterations proceed. This is mainly due to two factors. The first is that at every major iteration of the algorithm a new linear approximation is added to the current master problem for each nonlinear constraint of the original MINLP problem. The second is that the master problem of the current iteration inherits all the constraints of the master problem of the previous iteration. For MINLP problems with many nonlinear constraints and integer variables, few iterations are enough to create MILP master problems that have many linear constraints. Taking into account that Outer Approximation requires the complete solution of the master problems, it can be deduced that it needs to spend increasingly large amounts of computing time to solve the master problems.

To avoid the heavy computational task of successively solving the master problems, LP/NLP based Branch and Bound, proposed by Quesada and Grossmann [19] and generalised by Leyffer [17], considers only the first MILP master problem of the Outer Approximation method and uses Branch and Bound to solve it. When a node with an integer feasible solution is encountered the corresponding NLP primal problem is solved. The objective and constraints of the initial MINLP problem are linearised around the optimum solution of the primal problem. The new linearizations are then added to all the unsolved nodes of the Branch and Bound tree including the current one which is not fathomed.

This process can be thought of as a dynamic update of the sets of linear approximations of the objective and constraints, held at the unsolved nodes of the tree. As a result the algorithm does not need to solve independently and completely all the MILP master problems. Instead the solution of the master problems is implicitly done within the Branch and Bound tree used to solve the first MILP master problem through the dynamic update of the unsolved nodes with new linearizations. An important advantage of this method is that the total number of nodes that need to be solved is less than that required by

Outer Approximation when all the master problems are separately solved to optimality by Branch and Bound.

However the number of NLP primal problems may be larger than that required by Outer Approximation. This is because the objective and constraints are linearised at sub-optimal integer solutions of the master problems as opposed to Outer Approximation where the master problems are completely solved and therefore the objective and constraints are linearised at the optimal integer solution of the corresponding master problem. Due to the increase of the NLP problems, the number of the new linearizations which are dynamically added to the tree of LP/NLP based Branch and Bound method increases too. Consequently, the size of the unsolved nodes may become prohibitively large and damage the overall performance of the algorithm.

Our main motivation is to improve the LP/NLP based Branch and Bound method by reducing both the number of NLP problems and the number of nodes needed to be solved. To achieve this, instead of using the classical Branch and Bound method, we use the Branch and Cut framework proposed recently by Balas *et al.* [2] to solve the initial MILP master problem. In the above framework, Gomory mixed integer cuts are generated at specific nodes of the enumeration tree. These cuts are incorporated into the constraint set of the current node and tighten it. Those cuts are also valid for the entire tree and therefore may be used to tighten the constraint sets of other nodes. Since tighter representations of the feasible region provide better upper bounds, the solution process can be considerably sped up. The resulting algorithm provides an efficient way to incorporate Gomory cuts in an algorithm designed to solve MINLP problems.

# 4 Description of the proposed algorithm

In this section we present our approach to the solution of the convex 0-1 MINLP problems (1). It is an improvement of the LP/NLP based Branch and Bound method proposed by Quesada and Grossman [19] and generalised by Leyffer [17]. The new feature is the use of the Gomory mixed integer cuts in order to reduce the number of both the 0-1 MILP and the NLP subproblems that need to be solved before the optimum solution of the original 0-1 MINLP problem is found.

Initially the algorithm is given a 0-1 vector $\psi_0$. The 0-1 variables of the original 0-1 MINLP problem (1) are fixed at $\psi_0$ and the resulting primal problem

$$NLP(\psi_0) \begin{cases} \min_x & f(x, \psi_0) \\ ST & g(x, \psi_0) \leq 0 \\ & x \in X \end{cases} \tag{7}$$

is solved, provided that it is feasible. If it is infeasible a feasibility problem

$$F(\psi_0) \begin{cases} \min_x & u \\ ST & g(x,\psi_0) \leq u \\ & x \in X, u \in \Re \end{cases} \tag{8}$$

is solved, where $u$ is the variable that represents the largest violation of the constraints. Let $x_0$ be the optimum solution of the primal problem $NLP(\psi_0)$ or the feasibility problem $F(\psi_0)$, depending on which one is solved.

The initial master problem $M_0$ is obtained by linearising the objective and constraints of the original 0-1 MINLP problem around the point $(x_0, \psi_0)$ yielding the following 0-1 MILP problem

$$\hat{M}_0 \begin{cases} \min_{x,\psi,\eta} & \eta \\ ST & \eta < UBD_0 \\ & f(x_0,\psi_0) + \nabla f(x_0,\psi_0)^T \begin{pmatrix} x - x_0 \\ \psi - \psi_0 \end{pmatrix} \leq \eta \\ & g(x_0,\psi_0) + \nabla g(x_0,\psi_0)^T \begin{pmatrix} x - x_0 \\ \psi - \psi_0 \end{pmatrix} \leq 0, \\ & x \in X, \eta \in \Re \\ & \psi \in \{0,1\}^p \end{cases} \tag{9}$$

Problem $\hat{M}_0$ is solved by a Branch and Cut procedure with the additional feature that each time a node that yields a 0-1 feasible solution, say $(\tilde{x}, \tilde{\psi})$ with $\tilde{\psi} \in \{0,1\}^p$, is encountered, a new set of linearizations of the objective and constraints is generated and added to that node and every other unsolved node of the search tree.

More specifically, at the beginning of a general iteration $i$, the algorithm selects a node, say $(R_0^i, R_1^i)$, from the list $\Lambda$ that contains all the unsolved nodes generated so far. The

corresponding 0-1 MILP problem has the form

$$\hat{M}_i \begin{cases} \min_{x,\psi,\eta} & \eta \\ ST & \eta < UBD_i \\ & f(x_j,\psi_j) + \nabla f(x_j,\psi_j)^T \begin{pmatrix} x - x_j \\ \psi - \psi_j \end{pmatrix} \leq \eta \\ & g(x_j,\psi_j) + \nabla g(x_j,\psi_j)^T \begin{pmatrix} x - x_j \\ \psi - \psi_j \end{pmatrix} \leq 0, \quad \forall j \in \hat{T}_i \\ & g(x_k,\psi_k) + \nabla g(x_k,\psi_k)^T \begin{pmatrix} x - x_k \\ \psi - \psi_k \end{pmatrix} \leq 0, \quad \forall k \in \hat{S}_i \\ & \Gamma_1 x + \Gamma_2 \psi \leq \xi \\ & x \in X,\ \eta \in \Re,\ \psi^l \leq 0, l \in R_0^i \ \text{ and } \ \psi^l \geq 1, l \in R_1^i \\ & \psi \in \{0,1\}^p \end{cases} \tag{10}$$

The set $\hat{T}_i$ contains all the indices of those nodes $(R_0^j, R_1^j)$ of the Branch and Cut tree with $j \leq i$ in which the corresponding $NLP(\psi_j)$ is feasible. On the other hand, the set $\hat{S}_i$ contains all the indices of those nodes $(R_0^k, R_1^k)$ with $k \leq i$ in which the problems $NLP(\psi_k)$ are infeasible and the feasibility problems $F(\psi_k)$, defined by (8), are solved. $UBD_i$ is the value of the objective function of the initial 0-1 MINLP problem at the best integer solution that has been found so far, that is $UBD_i = \min\left\{f(x_j,\psi_j):\ \forall j \leq i, j \in \hat{T}_i\right\}$. The linear constraints $\Gamma_1 x + \Gamma_2 \psi \leq \xi$ represent those Gomory mixed integer cuts that have been generated so far and are tight to or violated by the optimum solution of the parent node of $(R_0^i, R_1^i)$. Their presence in $\hat{M}_i$ tightens the outer approximation of the feasible region of the original 0-1 MINLP problem. Furthermore the constraint set of the 0-1 MILP problem $\hat{M}_i$ is tighter than that of the corresponding 0-1 MILP problem solved by the LP/NLP based Branch and Bound method [19]. The fixing of the 0-1 variables is performed by including the constraints

$$\psi^l \leq 0, l \in R_0^i \text{ and } \psi^l \geq 1, l \in R_1^i$$

and not by eliminating those 0-1 variables that have already been fixed. As discussed in section 2.3, this technique allows the Gomory cuts generated using (6) to be valid throughout the enumeration tree.

The algorithm solves the LP relaxation of $\hat{M}_i$. Let $(\hat{x}, \hat{\psi}, \hat{\eta})$ be its optimum solution. $\hat{\eta}$ provides a lower bound to the optimum solution of the original 0-1 MINLP problem since $\hat{M}_i$ is an outer approximation of it. Depending on the type of the solution of $\hat{M}_i$, three different cases can be identified.

The first case arises when $\hat{\psi} \in \{0,1\}^p$. The algorithm solves the corresponding primal problem $NLP(\hat{\psi})$, provided that it is feasible. Otherwise it forms and solves the feasibility problem $F(\hat{\psi})$. Let $\bar{x}$ be the optimum solution of either $NLP(\hat{\psi})$ or $F(\hat{\psi})$, depending on

which one is eventually solved. The objective and constraints are linearised around the point $(\bar{x}, \hat{\psi})$ and the new linearizations are added to the current node and all the unsolved nodes of the search tree. The current node is not fathomed, since the new linearizations added to it cut off the solution $(\hat{x}, \hat{\psi}, \hat{\eta})$. Thus the updated version of $\hat{M}_i$ is placed back in the list of unsolved nodes. Furthermore, if $NLP(\hat{\psi})$ is feasible and $f(\bar{x}, \hat{\psi}) < UBD_i$ then $(\bar{x}, \hat{\psi})$ becomes the new incumbent solution and $f(\bar{x}, \hat{\psi})$ becomes the new upper bound.

The second case arises when $\hat{M}_i$ is infeasible. The current node is fathomed, since any further restriction of it will also be infeasible. Furthermore, every node of the search tree can also be fathomed if its lower bound is greater than or equal to the current upper bound, because the constraint $\eta < UBD_i$ becomes infeasible. The use of that constraint, first introduced by Fletcher and Leyffer [6], ensures that no 0-1 assignment is generated more than once. Recalling that there is only a finite number of 0-1 assignments the algorithm is guaranteed to terminate after a finite number of steps.

The third case arises when there is at least one violated 0-1 variable, i.e., $\hat{\psi} \notin \{0,1\}^p$. In this case the algorithm has two options as opposed to only one in the original algorithm of Quesada and Grossmann [19]. The first option is to select a violated 0-1 variable and branch on it. Two new subproblems are generated and added to the list of unsolved problems. The second option is to generate a round of Gomory's mixed integer cuts from the violated 0-1 variables using (6). These cuts are then added to the constraint set of $\hat{M}_i$ and the resulting problem is solved again. These cuts are also stored in the pool for future use. In the next section we describe the rule that decides how often to branch and how often to generate Gomory cuts.

Our algorithm differs from a classical Branch and Cut mainly in three points. The first is that the search tree it generates is dynamically updated with new constraints resulting from linearizations of the objective and constraints. The second is that the incumbent solution $(\bar{x}, \hat{\psi})$ is constructed by combining the integer solution $(\hat{x}, \hat{\psi}, \hat{\eta})$ found at a node of the tree and the optimum solution $\bar{x}$ of the corresponding primal problem $NLP(\hat{\psi})$. Consequently the upper bounds used to fathom unsolved nodes of the tree are the values of the objective function $f(x, \psi)$ of the original 0-1 MINLP problem at the incumbent solutions. The third difference is that when an integer node is encountered it is not fathomed but re-solved after adding the new linearizations.

The differences mentioned above reflect the main aim of the algorithm which is to solve the original 0-1 MINLP problem by solving a sequence of dynamically updated 0-1 MILP problems interrupted by the solution of NLP problems. Despite these differences, the Gomory mixed integer cuts generated at a node and defined by (6) remain valid for the entire search tree of our algorithm. They can therefore be stored in a pool and used at every unsolved node of the entire tree.

A formal description of the algorithm is given below

**Algorithm: OA-BC**

STEP 1. *Initialisation:* $\psi_0 \in \{0,1\}^p$ is given; set $i = 1$, $\hat{T}_{-1} = \hat{S}_{-1} = \emptyset$.

STEP 2. *Set up initial master problem:*

2.1 If $NLP(\psi_0)$ is feasible, solve it and set $\hat{T}_0 = \{0\}$. Otherwise solve $F(\psi_0)$ and set $\hat{S}_0 = \{0\}$. Let $x_0$ be the optimum of $NLP(\psi_0)$ or $F(\psi_0)$.

2.2 If $x_0$ is the optimum of $NLP(\psi_0)$ then set $UBD_0 = f(x_0, \psi_0)$. Otherwise set $UBD_0 = \infty$.

2.3 Linearise objective and constraints about $(x_0, \psi_0)$ and form the initial 0-1 MILP master problem $\hat{M}_0$.

2.4 Define $\hat{M}_0$ as the root of the search tree. Let $\Lambda$ be the list which contains the unsolved nodes and set $\Lambda = \{(R_0^0, R_1^0)\} = \{(\emptyset, \emptyset)\}$.

STEP 3. *Node selection:* If $\Lambda = \emptyset$, then Stop. Otherwise select a pair $(R_0^i, R_1^i)$ and remove it from the list $\Lambda$

STEP 4. Solve the LP relaxation of the 0-1 MILP problem $\hat{M}_i$ and let $(\hat{x}, \hat{\psi}, \hat{\eta})$ be its optimum solution

STEP 5. If $\hat{\psi} \in \{0,1\}^p$ then

5.1 Set $\psi_i = \hat{\psi}$ and solve $NLP(\psi_i)$ if it is feasible or $F(\psi_i)$ otherwise. Let $x_i$ be the optimum of $NLP(\psi_i)$ or $F(\psi_i)$.

5.2 Linearise objective and constraints around $(x_i, \psi_i)$ and set $\hat{T}_i = \hat{T}_{i-1} \cup \{i\}$ or $\hat{S}_i = \hat{S}_{i-1} \cup \{i\}$ as appropriate.

5.3 Add the linearizations to $\hat{M}_i$ and to all the nodes in $\Lambda$. Place $\hat{M}_i$ back in $\Lambda$.

5.4 Update incumbent solution and upper bound:

If $NLP(\psi_i)$ is feasible and $f(x_i, \psi_i) < UBD_i$ then

$(x_*, \psi_*) = (x_i, \psi_i)$ and $UBD_{i+1} = f(x_i, \psi_i)$

Otherwise $UBD_{i+1} = UBD_i$.

5.5 *Pruning:* Delete all nodes from $\Lambda$ with $\eta > UBD_{i+1}$.

Go to Step 3

STEP 6. If $\hat{\psi} \notin \{0,1\}^p$ then

6.1 *Cutting versus Branching Decision:* If cutting planes should be generated then go to Step 6.2. Otherwise go to Step 6.3

6.2 *Cut generation:* Generate a round of Gomory mixed integer cuts using (6). Add all those cuts to $\hat{M}_i$ and store them in the pool. Go to step 4.

6.3 *Branching:* Select a violated 0-1 variable in $\hat{\psi}$, say $\hat{\psi}^{(r)} \in (0,1)$. Create two new nodes $(R_0^{i+1}, R_1^{i+1}) = (R_0^i \cup \{r\}, R_1^i)$ and $(R_0^{i+1}, R_1^{i+1}) = (R_0^i, R_1^i \cup \{r\})$. Add both nodes to the list $\Lambda$. Go to Step 4.

If $UBD_i < \infty$ upon the termination of the algorithm, then $(x_*, \psi_*)$ is the optimal solution of the original 0-1 MINLP problem. Otherwise the problem is infeasible.

Algorithm OA-BC requires an initial 0-1 vector to be given by the user. If such a vector is not available then the algorithm can start by solving the NLP relaxation of the initial 0-1 MINLP problem (1). If the solution of the NLP relaxation satisfies all the integrality constraints then that solution also solves the initial 0-1 MINLP problem and the algorithm can stop. If the NLP relaxation is infeasible then the initial 0-1 MINLP problem is also infeasible and the algorithm can stop. Finally if the NLP relaxation is feasible and has a non-integer optimum solution, then the initial 0-1 MILP master problem can be formulated by linearising the objective and constraints around that solution. This technique was initially suggested by Viswanathan and Grossmann [23], and has been used successfully by many others (e.g., [17], [19]).

# 5 Implementation issues

This section is devoted to discussing issues of practical importance considered during the implementation and testing of Algorithm OABC regarding the solution of the NLP and MILP problems.

## 5.1 The solution of NLP problems

Algorithm OABC needs to solve NLP problems throughout its execution. These problems are the primal and the feasibility problems, defined by (7) and (8), and are solved by the primal-dual interior point algorithm developed by Akrotirianakis and Rustem [1]. The algorithm solves the perturbed optimality conditions of the corresponding NLP problems using Newton's method. In order to induce convergence, a penalty-barrier merit function is used. The merit function incorporates the inequality constraints by means of the logarithmic barrier function and the equality constraints by means of the quadratic penalty function. Global convergence of the algorithm is achieved using a linesearch procedure which ensures the monotonic Al decrease of the merit function. Numerical results demonstrate the efficient performance of the algorithms for a variety of test problems. The current implementation requires the user to define the objective, the constraints and their gradients in separate subroutines. These are then used by the interior point algorithms to find the solution of the corresponding NLP problem.

An issue of significant practical importance is the selection of starting points for the interior point algorithms. If those points are carefully selected their efficiency is enhanced. The primal-dual interior point algorithm requires initial values for the continuous variables $x$. It also requires initial values for the slack variables $v$, added to the inequalities of problems (7) and (8) to convert them to equalities (i.e., $g(x,\psi) + v = 0$). All of these values must be positive. In addition, the performance of any primal-dual interior point algorithm improves dramatically if the initial values are not close to the boundary of the feasible region.

A separate subroutine has been developed to generate the initial values. For those

variables that have simple bounds $l^{(i)} \leq x^{(i)} \leq u^{(i)}$ with $l^{(i)} > 0$ we select

$$x_0^{(i)} = \frac{l^{(i)} + u^{(i)}}{2} \tag{11}$$

as the initial value. However, for those variables that have a negative lower bound i.e., $l^{(i)} \leq x^i \leq u^{(i)}$ with $l^{(i)} < 0$ we define the transformation $X^{(i)} = x^{(i)} - l^{(i)}$. We then substitute $X^{(i)}$ for $x^{(i)}$ throughout the NLP model. Thus the simple bounds of the new variable $X^{(i)}$ are $0 \leq X^{(i)} \leq u^{(i)} - l^{(i)}$ and its initial value is

$$X_0^{(i)} = \frac{u^{(i)} - l^{(i)}}{2} \tag{12}$$

which is always a positive number. For those variables that have no simple bounds (i.e., they are free) we take $x_0^{(i)} = 0$ as their initial values.

For the slack variables $w$ an obvious way of defining their initial values would be to compute the values of the constraints at $x_0$ and then set $v_0 = -g(x_0, \psi)$. There are two difficulties with this approach when it is used in an interior point based algorithm. The first is that, if $x_0$ violates one or more inequality constraints (i.e., if $g^i(x_0, \psi) > 0$ for some $i \in \{1, 2, ..., m\}$), then the corresponding slack variable $v^{(i)}$ gets a negative initial value, i.e., $v_0^{(i)} = -g^i(x_0, \psi) < 0$. This is not allowed in interior point methods, since the corresponding logarithmic term cannot be defined. The second difficulty arises when $x_0$ is feasible (i.e., $g^i(x_0, \psi) \leq 0$ for all $i$) but it may lie on the boundary of the feasible region (i.e., $g^i(x_0, \psi) = 0$ for some $i$) or very close to it (i.e., $g^i(x_0, \psi) = \epsilon$ for some $i$ and $\epsilon > 0$ very small). In both of the above cases primal-dual interior point algorithms have a very poor performance or even fail to converge.

To overcome the above mentioned difficulties we define the initial values of the slack variables using the formula

$$v_0^{(i)} = \max\left\{-g^i(x_0, \psi), \kappa\right\}, \text{ for all } i = 1, 2, ..., q \tag{13}$$

where $\kappa$ is a positive constant. In our implementation we used $\kappa = 1$. Vanderbei and Shanno [22] proposed and tested (13) in the context of NLP but it has also been used in deriving starting points for interior point algorithms in linear programming (e.g., [18]).

In the context of MINLP, (13) has the additional advantage of taking into account the changes of the continuous feasible region caused by different integer vectors. Thus it guarantees that the initial values of the slack variables are sufficiently positive and away from the boundary for every integer assignment. The initial values, defined by (11), (12) and (13), provided good starting points for the primal-dual interior point algorithm, resulting in an efficient and robust behaviour of the nonlinear solver.

## 5.2 The solution of the initial 0-1 MILP master problem

In our implementation we used many of the strategies proposed by Balas *et al.* [2] regarding cut generation and management in a Branch and Cut framework for solving 0-1 MILP problems. In all the tests we tried we used the best bound strategy for selecting the next

node to be solved. Hence the list where all the unsolved nodes are stored is always kept in an increasing order of the objective function value of the problems $\hat{M}_i$. The algorithm selects the first element of the list and solves it. If the algorithm needs to branch on a 0-1 variable, it selects the variable whose fractional value is closer to 0.5. This is a simple yet very effective branching strategy [17]. All the cuts are stored in a common pool. The maximum size of the pool is 500. When that limit is exceeded, those cuts which are not active at any node of the tree are deleted to make room for the new ones.

The number of cuts generated at a node of the tree plays an important role in the efficiency of the algorithm. Algorithm OABC generates Gomory mixed integer cuts for all the violated 0-1 variables present at the optimum solution of the current node, whenever it decides to generate cuts instead of branching. This decision is based on recent computational experience of Balas *et al.* [2], where it is reported that in a fixed period of computing time the best improvements in the objective function value of a 0-1 MILP problem are achieved when cuts are generated for all the violated integer variables.

Although the incorporation of cutting planes in the enumeration tree is a crucial factor for the efficiency of Branch and Cut algorithms it is not necessary to generate cuts at every node, since the time required to generate them may be large and damage the overall performance of the algorithm. Thus the skip factor which defines the frequency by which cutting planes are generated in the tree plays a critical role in the overall performance of our algorithm. The skip factor used by Balas *et al.* [2] in their Branch and Cut algorithm for 0-1 MILP problems, is calculated at the root node of the tree and retains that value throughout the enumeration tree. It is a function of several parameters and is defined by

$$S = \min \left\{ S_{max}, \lceil \frac{f}{c \ d \ \log_{10} p} \rceil \right\} \tag{14}$$

The most important parameter in (14) is $d$ which represents the average distance cut off by all the Gomory cuts generated at the root node. Also $d$ is used as an estimate of the average distance cut off by all the cuts generated throughout the tree. $f$ represents the number of violated 0-1 variables at the optimum solution of the root node whereas $p$ is the total number of 0-1 variables. $S_{max}$ and $c$ are positive constant parameters. Using (14) to determine the skip factor, Balas *et al.* report very good computational results in their Branch and Cut algorithm for 0-1 MILP problems [2].

Our choice of the skip factor is based on (14) but adapted to the environment of the dynamically updated tree generated by Algorithm OABC. Its main difference from (14) is that its value does not remain constant throughout the enumeration tree. Every time a new node with an integer solution is encountered and new linearizations are generated, its value is calculated again using the formula

$$s = \min \left\{ S_{max}, \lceil \frac{t}{t + w} \frac{f}{c \ d \ \log_{10} p} \rceil \right\} \tag{15}$$

where $t$ represents the number of integer nodes that have been met so far and $w$ is a positive constant parameter. Its new feature is that it generates more cuts at early stages of the algorithm where the number of linearizations is not adequate to provide a good approximation of the feasible region of the initial 0-1 MINLP problem. This is due to the

introduction of the ratio $t/(t+w) \in (0,1)$ which makes our $s$ less than the skip factor $S$ of Balas *et al.* However as the number of integer nodes encountered increases and more linearizations are added to the unsolved nodes, $s$ approaches $S$ and thus cuts are generated less frequently.

# 6 Numerical results

The algorithm has been implemented using standard $C$ on a Dual processor Sun UltraSparc-2, 167 MHz, with 256 megabytes of RAM, running Solaris (release 5.5.1). Several convex 0-1 MINLP problems have been tried. The size and other characteristics of the tests are depicted in Table 1.

| Problem name | Variables | | Constraints | |
|---|---|---|---|---|
| | Continuous | 0-1 | Linear | Nonlinear |
| HW74 | 7 | 3 | 5 | 3 |
| Synthesis1 | 3 | 3 | 4 | 2 |
| Synthesis2 | 5 | 6 | 11 | 3 |
| Synthesis3 | 8 | 9 | 19 | 4 |
| Yuan | 3 | 4 | 5 | 4 |
| OptProLoc | 5 | 25 | 5 | 25 |

Table 1: Characteristics of test problems

Tests $HW74$, $Synthesis1$, $Synthesis2$ and $Synthesis3$, are process synthesis problems taken from [14] and [4]. These problems are used to derive the optimal configuration of a number of processes used to produce a specific product from different raw materials. The 0-1 variables represent the existence/non-existence of those process. The fifth test is taken from Yuan *et al.* [25]. Test $OptProLoc$ is a problem arising from the optimal positioning of a product in a multi-attribute space [8], [4]. The first five tests have a moderate degree of complexity, i.e., the number of 0-1 variables and the constraints is not large. However in the sixth test the number of 0-1 variables is five times the number of the continuous variables. Also the number of nonlinear constraints is again five times the number of the linear ones. Hence that test can be considered as more difficult to optimise.

The first set of numerical results investigates what effect the incorporation of Gomory cuts has in the LP/NLP based Branch and Bound algorithm [19], [17]. Table 2 shows that when Gomory cuts are added, using either $S$ or $s$ as skip factors, both fewer nodes and fewer NLP problems need to be solved. It can also be seen that $s$ becomes more effective than $S$ as the size and complexity of the original 0-1 MINLP problem increases.

| Problem name | LP/NLP based BB | | OABC (S) | | OABC (s) | |
|---|---|---|---|---|---|---|
| | nodes | NLP | nodes | NLP | nodes | NLP |
| HW74 | 7 | 3 | 7 | 3 | 7 | 3 |
| Synthesis1 | 7 | 3 | 7 | 3 | 7 | 3 |
| Synthesis2 | 18 | 4 | 14 | 4 | 14 | 4 |
| Synthesis3 | 35 | 7 | 29 | 5 | 24 | 4 |
| Yuan | 27 | 6 | 24 | 4 | 21 | 3 |
| OptProLoc | 167 | 4 | 140 | 4 | 108 | 3 |

Table 2: Computational comparison of skip factors

For the first two tests the addition of cuts has no effect, because their size is small. The first reduction on the number of nodes occurs in the third example. Both skip factors, $S$ and $s$, produce the same reductions in the number of nodes. Larger reductions in both the number of nodes and the number of NLP problems occur at the last three tests. Those problems require to spend a very large portion of their time for solving MILP problems. It is clear therefore that in such tests the addition of Gomory mixed integer cuts can be very effective.

Finally, Table 3 lists the CPU time needed by LP/NLP based Branch and Bound [19] and Algorithm OABC. For the first two problems the addition of Gomory cuts increases the CPU time. This is justified by the fact that the size of the corresponding enumeration trees is small. As the size of the problems increases, the addition of Gomory cuts results in substantial reductions of the overall CPU time.

| Problem name | CPU Time | | |
|---|---|---|---|
| | LP/NLP based BB | OABC (S) | OABC (s) |
| HW74 | 0.03 | 0.03 | 0.04 |
| Synthesis1 | 0.03 | 0.03 | 0.05 |
| Synthesis2 | 0.22 | 0.19 | 0.15 |
| Synthesis3 | 1.07 | 0.71 | 0.37 |
| Yuan | 0.19 | 0.11 | 0.08 |
| OptProLoc | 4.93 | 4.02 | 3.16 |

Table 3: Comparison of the CPU time

# References


[1] I. Akrotirianakis and B. Rustem. A globally convergent interior point algorithm for general non-linear programming problems. Technical Report 97-14, Department of Computing, Imperial College, London, UK, November 1997.

[2] E. Balas, S. Ceria, G. Cornuejols, and N. Natraj. Gomory cuts revisited. *Operations Research Letters*, 19:1–9, 1996.

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

[4] M. Duran and I. Grossmann. An outer approximation algorithm for a class of minlp problems. *Computers and Chemical Engineering*, 1986.

[5] R. Fletcher. *Practical methods of optimization*. John Wiley and Sons, UK, 1987.

[6] R. Fletcher and S. Leyffer. Solving mixed integer nonlinear programs by outer approximation. *Mathematical Programming*, 1994.

[7] C. Floudas. *Nonlinear and Mixed Integer Optimization*. Oxford University Press, 1995.

[8] B. Gavish, D. Horsky, and K. Srikanth. An approach to the optimal positioning of a new product. *Management Science*, 29(11):1277–1297, 1983.

[9] A. Geofrion. Generalized benders decomposition. *Journal of Optimization Theory and Applications*, 10(4), 1972.

[10] R. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64:275–278, 1958.

[11] R. E. Gomory. An algorithm for integer solutions to linear programs. In L. Graves R. and P. Wolfe, editors, *Recent Advances in Mathematical Programming*, pages 269–302. McGraw-Hill, New York, 1963.

[12] I. Grossmann and Z. Kravanja. Mixed integer nonlinear programming: A survey of algorithms and applications. In A. R. Conn, L. T. Biegler, T. F. Coleman, and F. N.Santosa, editors, *Large Scale Optimization with Applications, Part II: Optimization Design and Control*. Springer, New York, 1997.

[13] M. Jünger, G. Reinelt, and S. Thienel. Practical problem solving with cutting plane algorithms in combinatorial optimization. In *Combinatorial Optimization*.

[14] G. Kocis and I. Grossmann. Relaxation strategy for the structural optimization of process flow sheets. *I. and E.C. Res.*, 26:1869–1880, 1987.

[15] G. Kocis and I. Grossmann. Global optimization of nonconvex minlp problems in process synthesis. *I. and E.C. Res.*, 27(8):1407, 1988.

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

[17] S. Leyffer. *Deterministic methods for mixed integer nonlinear programming*. PhD thesis, Department of Mathematics and Computer Science, University of Dundee, Scotland, UK, 1993.

[18] I. J. Lusting, R. E. Marsten, and D. F. Shanno. Computational experience with a primal-dual interior point method for linear programming. *Linear Algebra and Applications*, 152:191–222, 1991.

[19] I. Quessada and I. E. Grossmann. An lp/nlp based branch and bound algorithm for convex minlp optimization problems. *Computers and Chemical Engineering*, 16(10/11):937–947, 1992.

[20] A. J. Quist, R. van Geemert, R. Hoogenboom, J. E. Illes, T. de Klerk, C. Roos, and T. Terlaky. Optimization of a nuclear reactor core reload pattern using nonlinear optimization and search heuristics. Technical report, Department of Operations Research, Delft University of Technology, Delft, The Netherlands, 1997.

[21] H. Taha. *Integer Programming: Theory, applications and Computations*. Academic Press, 1975.

[22] R. J. Vanderbei and D. F. Shanno. An interior point algorithm for nonconvex nonlinear programming. Technical Report SOR-97-21, CEOR, Princeton University, Princeton, NJ, 1997.

[23] C. Viswanathan and I. E. Grossmann. A combined penalty function and outer approximation method for minlp optimization. *Computers and Chemical Engineering*, 14:769, 1990.

[24] T. Westerlund, J. Isaksoson, and I. Harjunkowski. Solving a production optimization problem in the paper industry. Technical Report 95-146-A, Department of Chemical Engineering, Abo Akademi, Abo, Finland, 1995.

[25] X. Yuan, S. Zhang, A. Pibouleau, and S. Domenech. Une methode d' optimasation non lineare en variables mixtes pour la conception de procedes. *Recherch Operationnelle*, 22(4):331–346, 1988.