ALGORITHMS AND SOFTWARE FOR CONVEX MIXED INTEGER NONLINEAR PROGRAMS

PIERRE BONAMI[*], MUSTAFA KILINÇ[†], AND JEFF LINDEROTH[‡]

Abstract. This paper provides a survey of recent progress and software for solving convex Mixed Integer Nonlinear Programs (MINLP)s, where the objective and constraints are defined by convex functions and integrality restrictions are imposed on a subset of the decision variables. *Convex MINLP*s have received sustained attention in recent years. By exploiting analogies to well-known techniques for solving Mixed Integer Linear Programs and incorporating these techniques into software, significant improvements have been made in the ability to solve these problems.

Key words. Mixed Integer Nonlinear Programming; Branch and Bound.

1. Introduction. Mixed-Integer Nonlinear Programs (MINLP)s are optimization problems where some of the variables are constrained to take integer values and the objective function and feasible region of the problem are described by nonlinear functions. Such optimization problems arise in many real world applications. Integer variables are often required to model logical relationships, fixed charges, piecewise linear functions, disjunctive constraints and the non-divisibility of resources. Nonlinear functions are required to accurately reflect physical properties, covariance, and economies of scale.

In full generality, MINLPs form a particularly broad class of challenging optimization problems, as they combine the difficulty of optimizing over integer variables with the handling of nonlinear functions. Even if we restrict our model to contain only linear functions, MINLP reduces to a Mixed-Integer Linear Program (MILP), which is an NP-Hard problem [55]. On the other hand, if we restrict our model to have no integer variable but allow for general nonlinear functions in the objective or the constraints, then MINLP reduces to a Nonlinear Program (NLP) which is also known to be NP-Hard [90]. Combining both integrality and nonlinearity can lead to examples of MINLP that are undecidable [67].

---

[*]Laboratoire d'Informatique Fondamentale de Marseille, CNRS, Aix-Marseille Universités, Parc Scientifique et Technologique de Luminy, 163 avenue de Luminy - Case 901, F-13288 Marseille Cedex 9, France (`pierre.bonami@lif.univ-mrs.fr`). Supported by ANR grand BLAN06-1-138894.

[†]Department of Industrial and Systems Engineering, University of Wisconsin-Madison, 1513 University Ave., Madison, WI, 53706 (`kilinc@wisc.edu`).

[‡]Department of Industrial and Systems Engineering, University of Wisconsin-Madison, 1513 University Ave., Madison, WI 53706 (`linderoth@wisc.edu`). The work of the second and third authors is supported by the US Department of Energy under grants DE-FG02-08ER25861 and DE-FG02-09ER25869, and the National Science Foundation under grant CCF-0830153.

In this paper, we restrict ourselves to the subclass of MINLP where the objective function to minimize is convex, and the constraint functions are all convex and upper bounded. In these instances, when integrality is relaxed, the feasible set is convex. Convex MINLP is still NP-hard since it contains MILP as a special case. Nevertheless, it can be solved much more efficiently than general MINLP since the problem obtained by dropping the integrity requirements is a convex NLP for which there exist efficient algorithms. Further, the convexity of the objective function and feasible region can be used to design specialized algorithms.

There are many diverse and important applications of MINLPs. A small subset of these applications includes portfolio optimization [21, 68], block layout design in the manufacturing and service sectors [33, 98], network design with queuing delay constraints [27], integrated design and control of chemical processes [53], drinking water distribution systems security [73], minimizing the environmental impact of utility plants [46], and multi-period supply chain problems subject to probabilistic constraints [75].

Even though convex MINLP is NP-Hard, there are exact methods for its solution—methods that terminate with a guaranteed optimal solution or prove that no such solution exists. In this survey, our main focus is on such exact methods and their implementation.

In the last 40 years, at least five different algorithms have been proposed for solving convex MINLP to optimality. In 1965, Dakin remarked that the branch-and-bound method did not require linearity and could be applied to convex MINLP. In the early 70's, Geoffrion [56] generalized Benders decomposition to make an exact algorithm for convex MINLP. In the 80's, Gupta and Ravindran studied the application of branch and bound [62]. At the same time, Duran and Grossmann [43] introduced the Outer Approximation decomposition algorithm. This latter algorithm was subsequently improved in the 90's by Fletcher and Leyffer [51] and also adapted to the branch-and-cut framework by Quesada and Grossmann [96]. In the same period, a related method called the Extended Cutting Plane method was proposed by Westerlund and Pettersson [111]. Section 3 of this paper will be devoted to reviewing in more detail all of these methods.

Two main ingredients of the above mentioned algorithms are solving MILP and solving NLP. In the last decades, there have been enormous advances in our ability to solve these two important subproblems of convex MINLP.

We refer the reader to [100, 92] and [113] for in-depth analysis of the theory of MILP. The advances in the theory of solving MILP have led to the implementation of solvers both commercial and open-source which are now routinely used to solve many industrial problems of large size. Bixby and Rothberg [22] demonstrate that advances in algorithmic technology alone have resulted in MILP instances solving more than 300 times faster than a decade ago. There are effective, robust commercial MILP solvers

such as CPLEX [66], XPRESS-MP [47], and Gurobi [63]. Linderoth and Ralphs [82] give a survey of noncommercial software for MILP.

There has also been steady progress over the past 30 years in the development and successful implementation of algorithms for NLPs. We refer the reader to [12] and [94] for a detailed recital of nonlinear programming techniques. Theoretical developments have led to successful implementations in software such as SNOPT [57], filterSQP [52], CONOPT [42], IPOPT [107], LOQO [103], and KNITRO [32]. Waltz [108] states that the size of instance solvable by NLP is growing by nearly an order of magnitude a decade.

Of course, solution algorithms for convex MINLP have benefit from the technological progress made in solving MILP and NLP. However, in the realm of MINLP, the progress has been far more modest, and the dimension of solvable convex MINLP by current solvers is small when compared to MILPs and NLPs. In this work, our goal is to give a brief introduction to the techniques which are in state-of-the-art solvers for convex MINLPs. We survey basic theory as well as recent advances that have made their way into software. We also attempt to make a fair comparison of all algorithmic approaches and their implementations.

The remainder of the paper can be outlined as follows. A precise description of a MINLP and algorithmic building blocks for solving MINLPs are given in Section 2. Section 3 outlines five different solution techniques. In Section 4, we describe in more detail some advanced techniques implemented in the latest generation of solvers. Section 5 contains descriptions of several state-of-the-art solvers that implement the different solution techniques presented. Finally, in Section 6 we present a short computational comparison of those software packages.

2. MINLP. The focus of this section is to mathematically define a MINLP and to describe important special cases. Basic elements of algorithms and subproblems related to MINLP are also introduced.

2.1. MINLP problem classes. A Mixed Integer Nonlinear Program may be expressed in algebraic form as follows:

$$
\begin{aligned}
z_{\text{MINLP}} = \text{minimize} \quad & f(x) \\
\text{subject to} \quad & g_j(x) \leq 0 \quad \forall j \in J, \qquad \text{(MINLP)} \\
& x \in X, \quad x_I \in \mathbb{Z}^{|I|},
\end{aligned}
$$

where X is a polyhedral subset of $\mathbb{R}^n$ (e.g. $X = \{x \mid x \in \mathbb{R}^n_+, Ax \leq b\}$). The functions $f : X \to \mathbb{R}$ and $g_j : X \to \mathbb{R}$ are sufficiently smooth functions. The algorithms presented here only require continuously differentiable functions, but in general algorithms for solving continuous relaxations converge much faster if functions are twice-continuously differentiable. The set J is the index set of nonlinear constraints, I is the index set of discrete variables and C is the index set of continuous variables, so $I \cup C = \{1, \ldots, n\}$.

For convenience, we assume that the set X is bounded; in particular some finite lower bounds L_I and upper bounds U_I on the values of the integer variables are known. In most applications, discrete variables are restricted to 0-1 values, i.e., $x_i \in \{0, 1\} \ \forall i \in I$. In this survey, we focus on the case where the functions f and g_j are convex. Thus, by relaxing the integrality constraint on x, a convex program, minimization of a convex function over a convex set, is formed. We will call such problems *convex MINLPs*. From now on, unless stated, we will refer convex MINLPs as MINLPs.

There are a number of important special cases of MINLP. If $f(x) = x^T Q x + d^T x + h$, is a (convex) quadratic function of x, and there are only linear constraints on the problem ($J = \emptyset$), the problem is known as a mixed integer quadratic program (MIQP). If both $f(x)$ and $g_j(x)$ are quadratic functions of x for each $j \in J$, the problem is known as a mixed integer quadratically constrained program (MIQCP). Significant work was been devoted to these important special cases [87, 29, 21].

If the objective function is linear, and all nonlinear constraints have the form $g_j(x) = \|Ax + b\|_2 - c^T x - d$, then the problem is a Mixed Integer Second-Order Cone Program (MISOCP). Through a well-known transformation, MIQCP can be transformed into a MISOCP. In fact, many different types of sets defined by nonlinear constraints are representable via second-order cone inequalities. Discussion of these transformations is out of the scope of this work, but the interested reader may consult [15]. Relatively recently, commercial software packages such as CPLEX [66], XPRESS-MP [47], and Mosek [88] have all been augmented to include specialized algorithms for solving these important special cases of convex MINLPs. In what follows, we focus on general convex MINLP and software available for its solution.

2.2. Basic elements of MINLP methods. The basic concept underlying algorithms for solving (MINLP) is to generate and refine bounds on its optimal solution value. Lower bounds are generated by solving a relaxation of (MINLP), and upper bounds are provided by the value of a feasible solution to (MINLP). Algorithms differ in the manner in which these bounds are generated and the sequence of subproblems that are solved to generate these bounds. However, algorithms share many basic common elements, which are described next.

Linearizations: Since the objective function of (MINLP) may be nonlinear, its optimal solution may occur at a point that is interior to the convex hull of its set of feasible solutions. It is simple to transform the instance to have a linear objective function by introducing an auxiliary variable η and moving the original objective function into the constraints. Specifically, (MINLP) may be equivalently stated as

$$z_{\text{MINLP}} = \text{minimize} \quad \eta$$
$$\text{subject to} \quad f(x) \leq \eta$$
$$g_j(x) \leq 0 \quad \forall j \in J, \qquad \qquad \text{(MINLP-1)}$$
$$x \in X, \quad x_I \in \mathbb{Z}^{|I|}.$$

Many algorithms rely on linear relaxations of (MINLP), obtained by linearizing the objective and constraint functions at a given point $\hat{x}$. Since f and g_j are convex and differentiable, the inequalities

$$f(\hat{x}) + \nabla f(\hat{x})^T (x - \hat{x}) \leq f(x),$$
$$g_j(\hat{x}) + \nabla g_j(\hat{x})^T (x - \hat{x}) \leq g_j(x),$$

are valid for all $j \in J$ and $\hat{x} \in \mathbb{R}^n$. Since $f(x) \leq \eta$ and $g_j(x) \leq 0$, then the linear inequalities

$$f(\hat{x}) + \nabla f(\hat{x})^T (x - \hat{x}) \leq \eta, \qquad \qquad (2.1)$$
$$g_j(\hat{x}) + \nabla g_j(\hat{x})^T (x - \hat{x}) \leq 0 \qquad \qquad (2.2)$$

are valid for (MINLP-1). Linearizations of $g_j(x)$ outer approximate the feasible region, and linearizations of $f(x)$ underestimate the objective function. We often refer to (2.1)–(2.2) as outer approximation constraints.

Subproblems: One important subproblem used by a variety of algorithms for (MINLP) is formed by relaxing the integrality requirements and restricting the bounds on the integer variables. Given bounds $(l_I, u_I) = \{(\ell_i, u_i) \mid \forall i \in I\}$, the *NLP relaxation* of (MINLP) is

$$z_{\text{NLPR}(l,u)} = \text{minimize} \quad f(x)$$
$$\text{subject to} \quad g_j(x) \leq 0 \quad \forall j \in J, \qquad \qquad \text{(NLPR}(l_I, u_I))$$
$$x \in X; \quad l_I \leq x_I \leq u_I.$$

The value $z_{\text{NLPR}(l,u)}$ is a lower bound on the value of z_{MINLP} that can be obtained in the subset of the feasible region of (MINLP) where the bounds $\ell_I \leq x_I \leq u_I$ are imposed. Specifically, if (l_I, u_I) are the lower and upper bounds (L_I, U_I) for the original instance, then $z_{\text{NLPR}(L_I, U_I)}$ provides a lower bound on z_{MINLP}.

In the special case that all of the integer variables are fixed ($l_I = u_I = \hat{x}_I$), the *fixed NLP subproblem* is formed:

$$z_{\text{NLP}(\hat{x}_I)} = \text{minimize} \quad f(x)$$
$$\text{subject to} \quad g_j(x) \leq 0, \ \forall j \in J \qquad \qquad \text{(NLP}(\hat{x}_I))$$
$$x \in X; \quad x_I = \hat{x}_I.$$

If $\hat{x}_I \in \mathbb{Z}^{|I|}$ and (NLP($\hat{x}_I$)) has a feasible solution, the value $z_{\text{NLP}(\hat{x}_I)}$ provides an upper bound to the problem (MINLP). If (NLP($\hat{x}_I$)) is infeasible,

NLP software typically will deduce infeasibility by solving an associated feasibility subproblem. One choice of feasibility subproblem employed by NLP solvers is

$$z_{\text{NLPF}(\hat{x}_I)} = \text{minimize} \quad \sum_{j=1}^{m} w_j g_j(x)^+$$
$$\text{s.t.} \ x \in X, \quad x_I = \hat{x}_I, \qquad (\text{NLPF}(\hat{x}_I))$$

where $g_j(x)^+ = \max\{0, g_j(x)\}$ measures the violation of the nonlinear constraints and $w_j \geq 0$. Since when $\text{NLP}(\hat{x}_I)$ is infeasible NLP solvers will return the solution to $\text{NLPF}(\hat{x}_I)$, we will often say, by abuse of terminology, that $\text{NLP}(\hat{x}_I)$ is solved and its solution $\bar{x}$ is optimal or minimally infeasible, meaning that it is the optimal solution to $\text{NLPF}(\hat{x}_I)$.

3. Algorithms for convex MINLP. With elements of algorithms defined, attention can be turned to describing common algorithms for solving MINLPs. The algorithms share many general characteristics with the well-known branch-and-bound or branch-and-cut methods for solving MILPs.

3.1. NLP-Based Branch and Bound. Branch and bound is a divide-and-conquer method. The dividing (branching) is done by partitioning the set of feasible solutions into smaller and smaller subsets. The conquering (fathoming) is done by bounding the value of the best feasible solution in the subset and discarding the subset if its bound indicates that it cannot contain an optimal solution.

Branch and bound was first applied to MILP by Land and Doig [74]. The method (and its enhancements such as branch and cut) remain the workhorse for all of the most successful MILP software. Dakin [38] realized that this method does not require linearity of the problem. Gupta and Ravindran [62] suggested an implementation of the branch-and-bound method for convex MINLPs and investigated different search strategies. Other early works related to NLP-Based Branch and Bound (NLP-BB for short) for convex MINLP include [91], [28], and [78].

In NLP-BB, the lower bounds come from solving the subproblems $(\text{NLPR}(l_I, u_I))$. Initially, the bounds (L_I, U_I) (the lower and upper bounds on the integer variables in (MINLP)) are used, so the algorithm is initialized with a continuous relaxation whose solution value provides a lower bound on z_{MINLP}. The variable bounds are successively refined until the subregion can be fathomed. Continuing in this manner yields a tree $\mathcal{L}$ of subproblems. A node N of the search tree is characterized by the bounds enforced on its integer variables: $N \stackrel{\text{def}}{=} (l_I, u_I)$. Lower and upper bounds on the optimal solution value $z_L \leq z_{\text{MINLP}} \leq z_U$ are updated through the course of the algorithm. Algorithm 1 gives pseudocode for the NLP-BB algorithm for solving (MINLP).

Algorithm 1 The NLP-Based Branch-and-Bound algorithm

0. **Initialize.**
    $\mathcal{L} \leftarrow \{(L_I, U_I)\}$. $z_U = \infty$. $x^* \leftarrow$ NONE.
1. **Terminate?**
    Is $\mathcal{L} = \emptyset$? If so, the solution x^* is optimal.
2. **Select.**
    Choose and delete a problem $N^i = (l_I^i, u_I^i)$ from $\mathcal{L}$.
3. **Evaluate.**
    Solve NLPR(l_I^i, u_I^i). If the problem is infeasible, go to step 1, else let $z_{\text{NLPR}(l_I^i, u_I^i)}$ be its optimal objective function value and $\hat{x}^i$ be its optimal solution.
4. **Prune.**
    If $z_{\text{NLPR}(l_I^i, u_I^i)} \geq z_U$, go to step 1. If $\hat{x}^i$ is fractional, go to step 5, else let $z_U \leftarrow z_{\text{NLPR}(l_I^i, u_I^i)}$, $x^* \leftarrow \hat{x}^i$, and delete from $\mathcal{L}$ all problems with $z_L^j \geq z_U$. Go to step 1.
5. **Divide.**
    Divide the feasible region of N^i into a number of smaller feasible subregions, creating nodes $N^{i_1}, N^{i_2}, \ldots, N^{i_k}$. For each $j = 1, 2, \ldots, k$, let $z_L^{i_j} \leftarrow z_{\text{NLPR}(l_I^i, u_I^i)}$ and add the problem N^{i_j} to $\mathcal{L}$. Go to 1.

As described in step 4 of Algorithm 1, if NLPR(l_I^i, u_I^i) yields an integral solution (a solution where all discrete variables take integer values), then $z_{\text{NLPR}(l_I^i, u_I^i)}$ gives an upper bound for MINLP. Fathoming of nodes occurs when the lower bound for a subregion obtained by solving NLPR(l_I^i, u_I^i) exceeds the current upper bound z_U, when the subproblem is infeasible, or when the subproblem provides a feasible integral solution. If none of these conditions is met, the node cannot be pruned and the subregion is divided to create new nodes. This **Divide** step of Algorithm 1 may be performed in many ways. In most successful implementations, the subregion is divided by *dichotomy branching*. Specifically, the feasible region of N^i is divided into subsets by changing bounds on one integer variable based on the solution $\hat{x}^i$ to NLPR(l_I^i, u_I^i). An index $j \in I$ such that $\hat{x}_j \notin \mathbb{Z}$ is chosen and two new children nodes are created by adding the bound $x_j \leq \lfloor \hat{x}_j \rfloor$ to one child and $x_j \geq \lceil \hat{x}_j \rceil$ to the other child. The tree search continues until all nodes are fathomed, at which point x^* is the optimal solution.

The description makes it clear that there are various choices to be made during the course of the algorithm. Namely, how do we select which subproblem to evaluate, and how do we divide the feasible region? A partial answer to these two questions will be provided in Sections 4.2 and 4.3. The NLP-Based Branch-and-Bound algorithm is implemented in solvers MINLP-BB [77], SBB [30], and Bonmin [24].

3.2. Outer Approximation. The Outer Approximation (OA) method for solving (MINLP) was first proposed by Duran and Grossmann [43]. The fundamental insight behind the algorithm is that (MINLP) is equivalent to a Mixed Integer *Linear* Program (MILP) of finite size. The MILP is constructed by taking linearizations of the objective and constraint functions about the solution to the subproblem NLP($\hat{x}_I$) or NLPF($\hat{x}_I$) for various choices of $\hat{x}_I$. Specifically, for each integer assignment $\hat{x}_I \in \text{Proj}_{x_I}(X) \cap \mathbb{Z}^{|I|}$ (where $\text{Proj}_{x_I}(X)$ denotes the projection of X onto the space of integer constrained variables), let $\overline{x} \in \arg\min \text{NLP}(\hat{x}_I)$ be an optimal solution to the NLP subproblem with integer variables fixed according to $\hat{x}_I$. If NLP($\hat{x}_I$) is not feasible, then let $\overline{x} \in \arg\min \text{NLPF}(\hat{x}_I)$ be an optimal solution to its corresponding feasibility problem. Since $\text{Proj}_{x_I}(X)$ is bounded by assumption, there are a finite number of subproblems NLP($\hat{x}_I$). For each of these subproblems, we choose one optimal solution, and let K be the (finite) set of these optimal solutions. Using these definitions, an outer-approximating MILP can be specified as

$$
\begin{aligned}
z_{\text{OA}} = \min \quad & \eta \\
\text{s.t.} \quad & \eta \geq f(\overline{x}) + \nabla f(\overline{x})^T(x - \overline{x}) \quad \overline{x} \in K, \quad \text{(MILP-OA)} \\
& g_j(\overline{x}) + \nabla g_j(\overline{x})^T(x - \overline{x}) \leq 0 \quad j \in J, \overline{x} \in K, \\
& x \in X, \quad x_I \in \mathbb{Z}^I.
\end{aligned}
$$

The equivalence between (MINLP) and (MILP-OA) is specified in the following theorem:

THEOREM 3.1. *[43, 51, 24] If $X \neq \emptyset$, f and g are convex, continuously differentiable, and a constraint qualification holds for each $x^k \in K$ then $z_{\text{MINLP}} = z_{\text{OA}}$. All optimal solutions of* (MINLP) *are optimal solutions of* (MILP-OA).

From a practical point of view it is not relevant to try and formulate explicitly (MILP-OA) to solve (MINLP)—to explicitly build it, one would have first to enumerate all feasible assignments for the integer variables in X and solve the corresponding nonlinear programs NLP($\hat{x}_I$). The OA method uses an MILP relaxation (MP($\mathcal{K}$)) of (MINLP) that is built in a manner similar to (MILP-OA) but where linearizations are only taken at a subset $\mathcal{K}$ of K:

$$
\begin{aligned}
z_{\text{MP}(\mathcal{K})} = \min \quad & \eta \\
\text{s.t.} \quad & \eta \geq f(\bar{x}) + \nabla f(\bar{x})^T(x - \bar{x}) \quad \bar{x} \in \mathcal{K}, \quad \text{(MP($\mathcal{K}$))} \\
& g_j(\bar{x}) + \nabla g_j(\bar{x})^T(x - \bar{x}) \leq 0 \quad j \in J, \bar{x} \in \mathcal{K}, \\
& x \in X, \quad x_I \in \mathbb{Z}^I.
\end{aligned}
$$

We call this problem the *OA-based reduced master problem*. The solution value of the reduced master problem (MP($\mathcal{K}$)), $z_{\text{MP}(\mathcal{K})}$, gives a lower bound to (MINLP), since $\mathcal{K} \subseteq K$. The OA method proceeds by iteratively

adding points to the set $\mathcal{K}$. Since function linearizations are accumulated as iterations proceed, the reduced master problem (MP($\mathcal{K}$)) yields a non-decreasing sequence of lower bounds.

OA typically starts by solving (NLPR(L_I,U_I)). Linearizations about the optimal solution to (NLPR(l_I, u_I)) are used to construct the first reduced master problem (MP($\mathcal{K}$)). Then, (MP($\mathcal{K}$)) is solved to optimality to give an integer solution, $\hat{x}$. This integer solution is then used to construct the NLP subproblem (NLP($\hat{x}_I$)). If (NLP($\hat{x}_I$)) is feasible, linearizations about the optimal solution of (NLP($\hat{x}_I$)) are added to the reduced master problem. These linearizations eliminate the current solution $\hat{x}$ from the feasible region of (MP($\mathcal{K}$)) unless $\hat{x}$ is optimal for (MINLP). Also, the optimal solution value $z_{\text{NLP}(\hat{x}_I)}$ yields an upper bound to MINLP. If (NLP($\hat{x}_I$)) is infeasible, the feasibility subproblem (NLPF($\hat{x}_I$)) is solved and linearizations about the optimal solution of (NLPF($\hat{x}_I$)) are added to the reduced master problem (MP($\mathcal{K}$)). The algorithm iterates until the lower and upper bounds are within a specified tolerance ϵ. Algorithm 2 gives pseudocode for the method. Theorem 3.1 guarantees that this algorithm cannot cycle and terminates in a finite number of steps.

Note that the reduced master problem need not be solved to optimality. In fact, given the upper bound UB and a tolerance ϵ, it is sufficient to generate any new $(\hat{\eta}, \hat{x})$ that is feasible to (MP($\mathcal{K}$)), satisfies the integrality requirements, and for which $\eta \leq UB - \epsilon$. This can usually be achieved by setting a *cutoff value* in the MILP software to enforce the constraint $\eta \leq UB - \epsilon$. If a cutoff value is not used, then the infeasibility of (MP($\mathcal{K}$)) implies the infeasibility of (MINLP). If a cutoff value is used, the OA iterations are terminated (Step 1 of Algorithm 2) when the OA master problem has no feasible solution. OA is implemented in the software packages DICOPT [60] and Bonmin [24].

3.3. Generalized Benders Decomposition. Benders Decomposition was introduced by Benders [16] for the problems that are linear in the "easy" variables, and nonlinear in the "complicating" variables. Geoffrion [56] introduced the Generalized Benders Decomposition (GBD) method for MINLP. The GBD method is very similar to the OA method, differing only in the definition of the MILP master problem. Specifically, instead of using linearizations for each nonlinear constraint, GBD uses duality theory to derive one single constraint that combines the linearizations derived from all the original problem constraints.

In particular, let $\overline{x}$ be the optimal solution to (NLP($\hat{x}_I$)) for a given integer assignment $\hat{x}_I$ and $\overline{\mu} \geq 0$ be the corresponding optimal Lagrange multipliers. The following generalized Benders cut is valid for (MINLP)

$$\eta \geq f(\overline{x}) + (\nabla_I f(\overline{x}) + \nabla_I g(\overline{x})\overline{\mu})^T (x_I - \hat{x}_I). \qquad (\text{BC}(\hat{x}))$$

Note that $\overline{x}_I = \hat{x}_I$, since the integer variables are fixed. In (BC($\hat{x}$)), ∇_I refers to the gradients of functions f (or g) with respect to discrete vari-

---

Algorithm 2 The Outer Approximation algorithm.

---

0. **Initialize.**
    $z_U \leftarrow +\infty$. $z_L \leftarrow -\infty$. $x^* \leftarrow$ NONE. Let $\overline{x}^0$ be the optimal solution of $(\text{NLPR}(L_I, U_I))$
    $\mathcal{K} \leftarrow \{\overline{x}^0\}$. Choose a convergence tolerance ϵ.
1. **Terminate?**
    Is $z_U - z_L < \epsilon$ or $(\text{MP}(\mathcal{K}))$ infeasible? If so, x^* is $\epsilon-$optimal.
2. **Lower Bound**
    Let $z_{\text{MP}(\mathcal{K})}$ be the optimal value of $\text{MP}(\mathcal{K})$ and $(\hat{\eta}, \hat{x})$ its optimal solution.
    $z_L \leftarrow z_{\text{MP}(\mathcal{K})}$
3. **NLP Solve**
    Solve $(\text{NLP}(\hat{x}_I))$.
    Let $\overline{x}^i$ be the optimal (or minimally infeasible) solution.
4. **Upper Bound?**
    Is $\overline{x}^i$ feasible for (MINLP) and $f(\overline{x}^i) < z_U$? If so, $x^* \leftarrow \overline{x}^i$ and $z_U \leftarrow f(\overline{x}^i)$.
5. **Refine**
    $\mathcal{K} \leftarrow \mathcal{K} \cup \{\overline{x}^i\}$ and $i \leftarrow i + 1$.
    Go to 1.

---

ables. The inequality $(\text{BC}(\hat{x}))$ is derived by building a surrogate of the OA constraints using the multipliers $\overline{\mu}$ and simplifying the result using the Karush-Kuhn-Tucker conditions satisfied by $\overline{x}$ (which in particular eliminates the continuous variables from the inequality).

If there is no feasible solution to $(\text{NLP}(\hat{x}_I))$, a feasibility cut can be obtained similarly by using the solution $\overline{x}$ to $(\text{NLPF}(\hat{x}_I))$ and corresponding multipliers $\overline{\lambda} \geq 0$:

$$\overline{\lambda}^T[g(\overline{x}) + \nabla_I g(\overline{x})^T(x_I - \hat{x}_I)] \leq 0. \qquad (\text{FCY}(\hat{x}))$$

In this way, a relaxed master problem similar to (MILP-OA) can be defined as:

$$
\begin{aligned}
z_{\text{GBD(KFS,KIS)}} = \min \quad & \eta \\
\text{s.t. } \quad & \eta \geq f(\overline{x}) + (\nabla_I f(\overline{x}) + \nabla_I g(\overline{x})\overline{\mu})^T(x_I - \overline{x}_I) \quad \forall \overline{x} \in \text{KFS}, \\
& \overline{\lambda}^T[g(\overline{x}) + \nabla_I g(\overline{x})^T(x_I - \overline{x}_I)] \leq 0 \quad \forall \overline{x} \in \text{KIS}, \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (\text{RM-GBD}) \\
& x \in X, \quad x_I \in \mathbb{Z}^I,
\end{aligned}
$$

where KFS is the set of solutions to feasible subproblems $(\text{NLP}(\hat{x}_I))$ and KIS is the set solutions to infeasible subproblems $(\text{NLPF}(\hat{x}_I))$. Convergence results for the GBD method are similar to those for OA.

THEOREM 3.2. *[56] If $X \neq \emptyset$, f and g are convex, and a constraint qualification holds for each $x^k \in K$, then $z_{\text{MINLP}} = z_{\text{GBD(KFS,KIS)}}$. The algorithm terminates in a finite number of steps.*

The inequalities used to create the master problem (RM-GBD) are aggregations of the inequalities used for (MILP-OA). As such, the lower bound obtained by solving a reduced version of (RM-GBD) (where only a subset of the constraints is considered) can be significantly weaker than for (MP($\mathcal{K}$)). This may explain why there is no available solver that uses solely the GBD method for solving convex MINLP. Abhishek, Leyffer and Linderoth [2] suggest to use the Benders cuts to aggregate inequalities in an LP/NLP-BB algorithm (see Section 3.5).

3.4. Extended Cutting Plane. Westerlund and Pettersson [111] proposed the Extended Cutting Plane (ECP) method for convex MINLPs, which is an extension of Kelley's cutting plane method [70] for solving convex NLPs. The ECP method was further extended to handle pseudo-convex function in the constraints [109] and in the objective [112] in the α-ECP method. Since this is beyond our definition of (MINLP), we give only here a description of the ECP method when all functions are convex. The reader is invited to refer to [110] for an up-to-date description of this enhanced method. The main feature of the ECP method is that it does not require the use of an NLP solver. The algorithm is based on the iterative solution of a reduced master problem (RM-ECP($\mathcal{K}$)). Linearizations of the most violated constraint at the optimal solution of (RM-ECP($\mathcal{K}$)) are added at every iteration. The MILP reduced master problem (RM-ECP($\mathcal{K}$)) is defined as:

$$
\begin{aligned}
z_{\text{ECP}(\mathcal{K})} = \min \quad & \eta \\
\text{s.t.} \quad & \eta \geq f(\bar{x}) + \nabla f(\bar{x})^T(x - \bar{x}) \quad \bar{x} \in \mathcal{K} \qquad \text{(RM-ECP($\mathcal{K}$))} \\
& g_j(\bar{x}) + \nabla g_j(\bar{x})^T(x - \bar{x}) \leq 0 \quad \bar{x} \in \mathcal{K} \quad j \in J(\bar{x}) \\
& x \in X, \quad x_I \in \mathbb{Z}^I
\end{aligned}
$$

where $J(\bar{x}) \overset{\text{def}}{=} \{j \in \arg\max_{j \in J} g_j(\bar{x})\}$ is the index set of most violated constraints for each solution $\bar{x} \in \mathcal{K}$, the set of solutions to (RM-ECP($\mathcal{K}$)). It is also possible to add linearizations of all violated constraints to (RM-ECP($\mathcal{K}$)). In that case, $J(\bar{x}) = \{j \mid g_j(\bar{x}) > 0\}$. Algorithm 3 gives the pseudocode for the ECP algorithm.

The optimal values $z_{\text{ECP}(\mathcal{K})}$ of (RM-ECP($\mathcal{K}$)) generate a non-decreasing sequence of lower bounds. Finite convergence of the algorithm is achieved when the maximum constraint violation is smaller than a specified tolerance ϵ. Theorem 3.3 states that the sequence of objective values obtained from the solutions to (RM-ECP($\mathcal{K}$)) converge to the optimal solution value.

THEOREM 3.3. *[111] If $X \neq \emptyset$ is compact and f and g are convex and continuously differentiable, then $z_{\text{ECP}(\mathcal{K})}$ converges to z_{MINLP}.*

The ECP method may require a large number of iterations, since the linearizations added at Step 3 are not coming from solutions to NLP subproblems. Convergence can often be accelerated by solving NLP subproblems (NLP($\hat{x}_I$)) and adding the corresponding linearizations, as in the OA method. The Extended Cutting Plane algorithm is implemented in the α-ECP software [110].

Algorithm 3 The Extended Cutting Plane algorithm

---

0. **Initialize.**
    Choose convergence tolerance ϵ. $\mathcal{K} \leftarrow \emptyset$.
1. **Lower Bound**
    Let $(\overline{\eta}^i, \overline{x}^i)$ be the optimal solution to (RM-ECP($\mathcal{K}$)).
2. **Terminate?**
    Is $g_j(\overline{x}^i) < \epsilon \ \forall j \in J$ and $f(\overline{x}^i) - \overline{\eta}^i < \epsilon$? If so, $\overline{x}^i$ is optimal with ϵ−feasibility.
3. **Refine**
    $\mathcal{K} \leftarrow \mathcal{K} \cup \{\overline{x}^i\}, t \in \arg\max_j g_j(\overline{x}^i)$, and $J(\overline{x}^i) = \{t\}$
    $i \leftarrow i + 1$. Go to 1.

3.5. LP/NLP-Based Branch-and-Bound. The LP/NLP-Based Branch-and-Bound algorithm (LP/NLP-BB) was first proposed by Quesada and Grossmann [96]. The method is an extension of the OA method outlined in Section 3.2, but instead of solving a sequence of master problems (MP($\mathcal{K}$)), the master problem is dynamically updated in a single branch-and-bound tree that closely resembles the branch-and-cut method for MILP.

We denote by LP($\mathcal{K}, \ell_I^i, u_I^i$) the LP relaxation of (MP($\mathcal{K}$)) obtained by dropping the integrality requirements and setting the lower and upper bounds on the x_I variables to l_I and u_I respectively. The LP/NLP-BB method starts by solving the NLP relaxation (NLPR(L_I, U_I)), and sets up the reduced master problem (MP($\mathcal{K}$)). A branch-and-bound enumeration is then started for (MP($\mathcal{K}$)) using its LP relaxation. The branch-and-bound enumeration generates linear programs LP($\mathcal{K}, \ell_I^i, u_I^i$) at each node $N^i = (\ell_I^i, u_I^i)$ of the tree. Whenever an integer solution is found at a node, the standard branch and bound is interrupted and (NLP($\hat{x}_I^i$)) (and (NLPF($\hat{x}_I^i$)) if NLP($\hat{x}_I^i$) is infeasible) is solved by fixing integer variables to solution values at that node. The linearizations from the solution of (NLP($\hat{x}_I^i$)) are then used to update the reduced master problem (MP($\mathcal{K}$)). The branch-and-bound tree is then continued with the updated reduced master problem. The main advantage of LP/NLP-BB over OA is that the need of restarting the tree search is avoided and only a single tree is required. Algorithm 4 gives the pseudo-code for LP/NLP-BB.

Adding linearizations dynamically to the reduced master problem (MP($\mathcal{K}$)) is a key feature of LP/NLP-BB. Note, however that the same

idea could potentially be applied to both the GBD and ECP methods. The LP/NLP-BB method commonly significantly reduces the total number of nodes to be enumerated when compared to the OA method. However, the trade-off is that the number of NLP subproblems might increase. As part of his Ph.D. thesis, Leyffer implemented the LP/NLP-BB method and reported substantial computational savings [76]. The LP/NLP-Based Branch-and-Bound algorithm is implemented in solvers Bonmin [24] and FilMINT [2].

Algorithm 4 The LP/NLP-Based Branch-and-Bound algorithm.

---

0. **Initialize.**
   $\mathcal{L} \leftarrow \{(L_I, U_I)\}$. $z_U \leftarrow +\infty$. $x^* \leftarrow$ NONE.
   Let $\overline{x}$ be the optimal solution of $(\text{NLPR}(l_I, u_I))$.
   $\mathcal{K} \leftarrow \{\overline{x}\}$.
1. **Terminate?**
   Is $\mathcal{L} = \emptyset$? If so, the solution x^* is optimal.
2. **Select.**
   Choose and delete a problem $N^i = (l_I^i, u_I^i)$ from $\mathcal{L}$.
3. **Evaluate.**
   Solve $\text{LP}(\mathcal{K}, l_I^i, u_I^i)$. If the problem is infeasible, go to step 1, else let $z_{\text{MPR}(\mathcal{K}, l_I^i, u_I^i)}$ be its optimal objective function value and $(\hat{\eta}^i, \hat{x}^i)$ be its optimal solution.
4. **Prune.**
   If $z_{\text{MPR}(\mathcal{K}, l_I^i, u_I^i)} \geq z_U$, go to step 1.
5. **NLP Solve?**
   Is $\hat{x}_I^i$ integer? If so, solve $(\text{NLP}(\hat{x}_I^i))$, otherwise go to step 8.
   Let $\overline{x}^i$ be the optimal (or minimally infeasible) solution.
6. **Upper bound?**
   Is $\overline{x}^i$ feasible for (MINLP) and $f(\overline{x}^i) < z_U$? If so, $x^* \leftarrow \overline{x}^i$, $z_U \leftarrow f(\overline{x}^i)$.
7. **Refine.**
   Let $\mathcal{K} \leftarrow \mathcal{K} \cup (\overline{x}^i)$. Go to step 3.
8. **Divide.**
   Divide the feasible region of N^i into a number of smaller feasible subregions, creating nodes $N^{i_1}, N^{i_2}, \ldots, N^{i_k}$. For each $j = 1, 2, \ldots, k$, let $z_L^{i_j} \leftarrow z_{\text{MPR}(\mathcal{K}, l_I^i, u_I^i)}$ and add the problem N^{i_j} to $\mathcal{L}$. Go to step 1.

---

4. Implementation techniques for convex MINLP. Seasoned algorithmic developers know that proper engineering and implementation can have a large positive impact on the final performance of software. In this section, we present techniques that have proven useful in efficiently implementing the convex MINLP algorithms of Section 3.

The algorithms for solving MINLP we presented share a great deal in common with algorithms for solving MILP. NLP-BB is similar to a branch and bound for MILP, simply solving a different relaxation at each node. The LP/NLP-BB algorithm can be viewed as a branch-and-cut algorithm, similar to those employed to solve MILP, where the refining linearizations are an additional class of cuts used to approximate the feasible region. An MILP solver is used as a subproblem solver in the iterative algorithms (OA, GBD, ECP). In practice, all the methods spend most of their computing time doing variants of the branch-and-bound algorithm. As such, it stands to reason that advances in techniques for the implementation of branch and bound for MILP should be applicable and have a positive impact for solving MINLP. The reader is referred to the recent survey paper [84] for details about modern enhancements in MILP software.

First we discuss improvements to the **Refine** step of LP/NLP-BB, which may also be applicable to the GBD or ECP methods. We then proceed to the discussion of the **Select** and **Divide** steps which are important in any branch-and-bound implementation. The section contains an introduction to classes of cutting planes that may be useful for MINLP and reviews recent developments in heuristics for MINLP.

We note that in the case of iterative methods OA, GBD and ECP, some of these aspects are automatically taken care of by using a "black-box" MILP solver to solve $(\text{MP}(\mathcal{K}))$ as a component of the algorithm. In the case of NLP-BB and LP/NLP-BB, one has to more carefully take these aspects into account, in particular if one wants to be competitive in practice with methods employing MILP solvers as components.

4.1. Linearization generation. In the OA Algorithm 2, the ECP Algorithm 3, or the LP/NLP-BB Algorithm 4, a key step is to **Refine** the approximation of the nonlinear feasible region by adding linearizations of the objective and constraint functions (2.1) and (2.2). For convex MINLPs, linearizations may be generated at *any* point and still give a valid outer approximation of the feasible region, so for all of these algorithms, there is a mechanism for enhancing them by adding many linear inequalities. The situation is similar to the case of a branch-and-cut solver for MILP, where cutting planes such as Gomory cuts [59], mixed-integer-rounding cuts [85], and disjunctive (lift and project) cuts [9] can be added to approximate the convex hull of integer solutions, but care must be taken in a proper implementation to not overwhelm the software used for solving the relaxations by adding too many cuts. Thus, key to an effective refinement strategy in many algorithms for convex MINLP is a policy for deciding when inequalities should be added and removed from the master problem and at which points the linearizations should be taken.

Cut Addition: In the branch-and-cut algorithm for solving MILP, there is a fundamental implementation choice that must be made when confronted with an infeasible (fractional) solution: should the solution be

eliminated by cutting or branching? Based on standard ideas employed for answering this question in the context of MILP, we offer three rules-of-thumb that are likely to be effective in the context of linearization-based algorithms for solving MINLP. First, linearizations should be generated early in the procedure, especially at the very top of the branch-and-bound tree. Second, the incremental effectiveness of adding additional linearizations should be measured in terms of the improvement in the lower bound obtained. When the rate of lower bound change becomes too low, the refinement process should be stopped and the feasible region divided instead. Finally, care must be taken to not overwhelm the solver used for the relaxations of the master problem with too many linearizations.

Cut Removal: One simple strategy for limiting the number of linear inequalities in the continuous relaxation of the master problem $(\mathrm{MP}(\mathcal{K}))$ is to only add inequalities that are violated by the current solution to the linear program. Another simple strategy for controlling the size of $(\mathrm{MP}(\mathcal{K}))$ is to remove inactive constraints from the formulation. One technique is to monitor the dual variable for the row associated with the linearization. If the value of the dual variable is zero, implying that removal of the inequality would not change the optimal solution value, for many consecutive solutions, then the linearization is a good candidate to be removed from the master problem. To avoid cycling, the removed cuts are usually stored in a pool. Whenever a cut of the pool is found to be violated by the current solution it is put back into the formulation.

Linearization Point Selection. A fundamental question in any linearization-based algorithm (like OA, ECP, or LP/NLP-BB) is *at which points* should the linearizations be taken. Each algorithm specifies a minimal set of points at which linearizations must be taken in order to ensure convergence to the optimal solution. However, the algorithm performance may be improved by additional linearizations. Abhishek, Leyffer, and Linderoth [2] offer three suggestions for choosing points about which to take linearizations.

The first method simply linearizes the functions f and g about the fractional point $\hat{x}$ obtained as a solution to an LP relaxation of the master problem. This method does not require the solution of an additional (nonlinear) subproblem, merely the evaluation of the gradients of objective and constraint functions at the (already specified) point. (The reader will note the similarity to the ECP method).

A second alternative is to obtain linearizations about a point that is feasible with respect to the nonlinear constraints. Specifically, given a (possibly fractional) solution $\hat{x}$, the nonlinear program $(\mathrm{NLP}(\hat{x}_I))$ is solved to obtain the point about which to linearize. This method has the advantage of generating linearization about points that are closer to the feasible region than the previous method, at the expense of solving the nonlinear program $(\mathrm{NLP}(\hat{x}_I))$.

In the third point-selection method, no variables are fixed (save those that are fixed by the nodal subproblem), and the NLP relaxation (NLPR(l_I, u_I)) is solved to obtain a point about which to generate linearizations. These linearizations are likely to improve the lower bound by the largest amount when added to the master problem since the bound obtained after adding the inequalities is equal to $z_{\text{NLPR}(l_i,u_i)}$, but it can be time-consuming to compute the linearizations.

These three classes of linearizations span the trade-off spectrum of time required to generate the linearization versus the quality/strength of the resulting linearization. There are obviously additional methodologies that may be employed, giving the algorithm developer significant freedom to engineer linearization-based methods.

4.2. Branching rules. We now turn to the discussion of how to split a subproblem in the **Divide** step of the algorithms. As explained in Section 2.1, we consider here only branching by dichotomy on the variables. Suppose that we are at node N^i of the branch-and-bound tree with current solution $\hat{x}^i$. The goal is to select an integer-constrained variable $x_j \in I$ that is not currently integer feasible ($\hat{x}_j^i \notin \mathbb{Z}$) to create two subproblems by imposing the constraint $x_j \leq \lfloor \hat{x}_j^i \rfloor$ (*branching down*) and $x_j \geq \lceil \hat{x}_j^i \rceil$ (*branching up*) respectively. Ideally, one would want to select the variable that leads to the smallest enumeration tree. This of course cannot be performed exactly, since the variable which leads to the smallest subtree cannot be know *a priori*.

A common heuristic reasoning to choose the branching variable is to try to estimate how much one can improve the lower bound by branching on each variable. Because a node of the branch-and-bound tree is fathomed whenever the lower bound for the node is above the current upper bound, one should want to increase the lower bound as much as possible. Suppose that, for each variable x_j, we have estimates D_{j-}^i and D_{j+}^i on the increase in the lower bound value obtained by branching respectively down and up. A reasonable choice would be to select the variable for which both D_{j-}^i and D_{j+}^i are large. Usually, D_{j-}^i and D_{j+}^i are combined in order to compute a score for each variable and the variable of highest score is selected. A common formula for computing this score is:

$$\mu \min(D_{j-}^i, D_{j+}^i) + (1 - \mu) \max(D_{j-}^i, D_{j+}^i)$$

(where $\mu \in [0, 1]$ is a prescribed parameter typipcally larger than $\frac{1}{2}$).

As for the evaluation or estimation of D_{j-}^i and D_{j+}^i, two main methods have been proposed: pseudo-costs [17] and strong-branching [66, 6]. Next, we will present these two methods and how they can be combined.

4.2.1. Strong-branching. Strong-branching consists in computing the values D_{j-}^i and D_{j+}^i by performing the branching on variable x_j and solving the two associated sub-problems. For each variable x_j currently

fractional in $\hat{x}^i_j$, we solve the two subproblems N^i_{j-} and N^i_{j+} obtained by branching down and up, respectively, on variable j. Because N^i_{j-} and/or N^i_{j+} may be proven infeasible, depending on their status, different decision may be taken.

- If both sub-problems are infeasible: the node N^i is infeasible and is fathomed.
- If one of the subproblems is infeasible: the bound on variable x_j can be strengthened. Usually after the bound is modified, the node is reprocessed from the beginning (going back to the **Evaluate** step).
- If both subproblems are feasible, their values are used to compute D^i_{j-} and D^i_{j+}.

Strong-branching can very significantly reduce the number of nodes in a branch-and-bound tree, but is often slow overall due to the added computing cost of solving two subproblems for each fractional variable. To reduce the computational cost of strong-branching, it is often efficient to solve the subproblems only approximately. If the relaxation at hand is an LP (for instance in LP/NLP-BB) it can be done by limiting the number of dual simplex iterations when solving the subproblems. If the relaxation at hand is an NLP, it can be done by solving an approximation of the problem to solve. Two possible relaxations that have been recently suggested [23, 106, 80] are the LP relaxation obtained by constructing an Outer Approximation or the Quadratic Programming approximation given by the last Quadratic Programming sub-problem in a Sequential Quadratic Programming (SQP) solver for nonlinear programming (for background on SQP solvers see [94]).

4.2.2. Pseudo-costs. The pseudo-costs method consists in keeping the history of the effect of branching on each variable and utilizing this historical information to select good branching variables. For each variable x_j, we keep track of the number of times the variable has been branched on (τ_j) and the total per-unit degradation of the objective value by branching down and up, respectively, P_{j-} and P_{j+}. Each time variable j is branched on, P_{j-} and P_{j+} are updated by taking into account the change of bound at that node:

$$P_{j-} = \frac{z^{i-}_L - z^i_L}{f^i_j} + P_{j-}, \text{ and } P_{j+} = \frac{z^{i+}_L - z^i_L}{1 - f^i_j} + P_{j+},$$

where x_j is the branching variable, N^i_- and N^i_+ denote the nodes from the down and up branch, z^i_L (resp. z^{i-}_L and z^{i+}_L) denote the lower bounds computed at node N^i (resp. N^i_- and N^i_+), and $f^i_j = \hat{x}^i_j - \lfloor \hat{x}^i_j \rfloor$ denotes the fractional part of $\hat{x}^i_j$. Whenever a branching decision has to be made, estimates of D^i_{j-}, D^i_{j+} are computed by multiplying the average of observed degradations with the current fractionality:

$$D^i_{j-} = f^i_j \frac{P_{j-}}{\tau_j}, \text{ and } D^i_{j+} = (1 - f^i_j) \frac{P_{j+}}{\tau_j}.$$

Note that contrary to strong-branching, pseudo-costs require very little computation since the two values P_{j-}^i and P_{j+}^i are only updated once the values z_L^{i-} and z_L^{i+} have been computed (by the normal process of branch and bound). Thus pseudo-costs have a negligible computational cost. Furthermore, statistical experiments have shown that pseudo-costs often provide reasonable estimates of the objective degradations caused by branching [83] when solving MILPs.

Two difficulties arise with pseudo-costs. The first one, is how to update the historical data when a node is infeasible. This matter is not settled. Typically, the pseudo-costs update is simply ignored if a node is infeasible.

The second question is how the estimates should be initialized. For this, it seems that the agreed upon state-of-the art is to combine pseudo-costs with strong-branching, a method that may address each of the two methods' drawbacks— strong-branching is too slow to be performed at every node of the tree, and pseudo-costs need to be initialized. The idea is to use strong-branching at the beginning of the tree search, and once all pseudo-costs have been initialized, to revert to using pseudo-costs. Several variants of this scheme have been proposed. A popular one is reliability branching [4]. This rule depends on a reliability parameter κ (usually a natural number between 1 and 8), pseudo-costs are trusted for a particular variable only after strong-branching has been performed κ times on this variable.

Finally, we note that while we have restricted ourselves in this discussion to dichotomy branching, one can branch in many different ways. Most state-of-the-art solvers allow branching on SOS constraints [14]. More generally, one could branch on *split disjunctions* of the form $(\pi^T x_I \leq \pi_0) \vee (\pi^T x_I \geq \pi_0 + 1)$ (where $(\pi, \pi_0) \in \mathbb{Z}^{n+1}$). Although promising results have been obtained in the context of MILP [69, 37], as far as we know, these methods have not been used yet in the context of MINLPs. Finally, methods have been proposed to branch efficiently in the presence of symmetries [86, 95]. Again, although they would certainly be useful, these methods have not yet been adapted into software for solving MINLPs, though some preliminary work is being done in this direction [81].

4.3. Node selection rules. The other important strategic decision left unspecified in Algorithms 1 and 4 is which node to choose in the **Select** step. Here two goals needs to be considered: decreasing the global upper bound z^U by finding good feasible solutions, and proving the optimality of the current incumbent x^* by increasing the lower bound as fast as possible. Two classical node selection strategies are *depth-first-search* and *best-first* (or *best-bound*). As its name suggest, depth first search selects at each iteration the deepest node of the enumeration tree (or the last node put in $\mathcal{L}$). Best-first follows an opposite strategy of picking the open node N^i with the smallest z_L^i (the best lower bound).

Both these strategies have their inherent strengths and weaknesses. Depth-first has the advantage of keeping the size of the list of open-nodes as small as possible. Furthermore, the changes from one subproblem to the next are minimal, which can be very advantageous for subproblem solvers that can effective exploit "warm-start" information. Also, depth-first search is usually able to find feasible solutions early in the tree search. On the other hand, depth-first can exhibit extremely poor performance if no good upper bound is known or found: it may explore many nodes with lower bound higher than the actual optimal solution. Best-bound has the opposite strengths and weaknesses. Its strength is that, for a fixed branching, it minimizes the number of nodes explored (because all nodes explored by it would be explored independently of the upper bound). Its weaknesses are that it may require significant memory to store the list $\mathcal{L}$ of active nodes, and that it usually does not find integer feasible solutions before the end of the search. This last property may not be a shortcoming if the goal is to prove optimality but, as many applications are too large to be solved to optimality, it is particularly undesirable that a solver based only on best-first aborts after several hours of computing time without producing one feasible solution.

It should seem natural that good strategies are trying to combine both best-first and depth first. Two main approaches are *two-phase methods* [54, 13, 44, 83] and *diving methods* [83, 22].

Two-phase methods start by doing depth-first to find one (or a small number of) feasible solution. The algorithm then switches to best-first in order to try to prove optimality (if the tree grows very large, the method may switch back to depth-first to try to keep the size of the list of active nodes under control).

Diving methods are also two-phase methods in a sense. The first phase called *diving* does depth-first search until a leaf of the tree (either an integer feasible or an infeasible one) is found. When a leaf is found, the next node is selected by *backtracking* in the tree for example to the node with best lower bound, and another diving is performed from that node. The search continues by iterating diving and backtracking.

Many variants of these two methods have been proposed in context of solving MILP. Sometimes, they are combined with estimations of the quality of integer feasible solutions that may be found in a subtree computed using pseudo-costs (see for example [83]). Computationally, it is not clear which of these variants performs better. A variant of diving called probed diving that performs reasonably well was described by Bixby and Rothberg [22]. Instead of conducting a pure depth-first search in the diving phase, the probed diving method explores both children of the last node, continuing the dive from the best one of the two (in terms of bounds).

4.4. Cutting planes. Adding inequalities to the formulation so that its relaxation will more closely approximate the convex hull of integer fea-

sible solutions was a major reason for the vast improvement in MILP solution technology [22]. To our knowledge, very few, if any MINLP solvers add inequalities that are specific to the nonlinear structure of the problem. Nevertheless, a number of cutting plane techniques that could be implemented have been developed in the literature. Here we outline a few of these techniques. Most of them have been adapted from known methods in the MILP case. We refer the reader to [36] for a recent survey on cutting planes for MILP.

4.4.1. Gomory cuts. The earliest cutting planes for Mixed Integer Linear Programs were Gomory Cuts [58, 59]. For simplicity of exposition, we assume a pure Integer Linear Program (ILP): $I = \{1, \ldots, n\}$, with linear constraints given in matrix form as $Ax \leq b$ and $x \geq 0$. The idea underlying the inequalities is to choose a set of non-negative multipliers $u \in \mathbb{R}_+^m$ and form the surrogate constraint $u^T Ax \leq u^T b$. Since $x \geq 0$, the inequality $\sum_{j \in N} \lfloor u^T a_j \rfloor x_j \leq u^T b$ is valid, and since $\lfloor u^T a_j \rfloor x_j$ is an integer, the right-hand side may also be rounded down to form the *Gomory cut* $\sum_{j \in N} \lfloor u^T a_j \rfloor x_j \leq \lfloor u^T b \rfloor$. This simple procedure suffices to generate *all* valid inequalities for an ILP [35]. Gomory cuts can be generalized to *Mixed Integer Gomory* (MIG) cuts which are valid for MILPs. After a period of not being used in practice to solve MILPs, Gomory cuts made a resurgence following the work of Balas *et al.* [10], which demonstrated that when used in combination with branch and bound, MIG cuts were quite effective in practice.

For MINLP, Cezik and Iyengar [34] demonstrate that if the nonlinear constraint set $g_j(x) \leq 0 \ \forall j \in J$ can be described using conic constraints $Tx \succeq_{\mathcal{K}} b$, then the Gomory procedure is still applicable. Here $\mathcal{K}$, is a homogeneous, self-dual, proper, convex cone, and the notation $x \succeq_{\mathcal{K}} y$ denotes that $(x - y) \in \mathcal{K}$. Each cone $\mathcal{K}$ has a dual cone $\mathcal{K}^*$ with the property that $\mathcal{K}^* \stackrel{\text{def}}{=} \{u \mid u^T z \geq 0 \ \forall z \in \mathcal{K}\}$. The extension of the Gomory procedure to the case of conic integer programming is clear from the following equivalence:

$$Ax \succeq_{\mathcal{K}} b \quad \Leftrightarrow \quad u^T Ax \geq u^T b \ \forall u \succeq_{\mathcal{K}^*} 0.$$

Specifically, elements from the dual cone $u \in \mathcal{K}^*$ can be used to perform the aggregation, and the regular Gomory procedure applied. To the authors' knowledge, no current MINLP software employs conic Gomory cuts. However, most solvers generate Gomory cuts from the existing linear inequalities in the model. Further, as pointed out by Akrotirianakis, Maros, and Rustem [5], Gomory cuts may be generated from the linearizations (2.1) and (2.2) used in the OA, ECP, or LP/NLP-BB methods. Most linearization-based software will by default generate Gomory cuts on these linearizations.

4.4.2. Mixed integer rounding. Consider the simple two-variable set $X = \{(x_1, x_2) \in \mathbb{Z} \times \mathbb{R}_+ \mid x_1 \leq b + x_2\}$. It is easy to see that the

mixed integer rounding inequality $x_1 \leq \lfloor b \rfloor + \frac{1}{1-f} x_2$, where $f = b - \lfloor b \rfloor$ represents the fractional part of b, is a valid inequality for X. Studying the convex hull of this simple set and some related counterparts have generated a rich classes of inequalities that may significantly improve the ability to solve MILPs [85]. Key to generating useful inequalities for computation is to combine rows of the problem in a clever manner and to use variable substitution techniques.

Atamtürk and Narayan [7] have extended the concept of mixed integer rounding to the case of Mixed Integer Second-Order Cone Programming (MISOCP). For the conic mixed integer set

$$T = \left\{ (x_1, x_2, x_3) \in \mathbb{Z} \times \mathbb{R}^2 \mid \sqrt{(x_1 - b)^2 + x_2^2} \leq x_3 \right\}$$

the following *simple conic mixed integer rounding* inequality

$$\sqrt{[(1 - 2f)(x_1 - \lfloor b \rfloor) + f]^2 + x_2^2} \leq x_3$$

helps to describe the convex hull of T. They go on to show that employing these inequalities in a cut-and-branch procedure for solving MISOCPs is significantly beneficial. To the authors' knowledge, no available software employs this technology, so this may be a fruitful line of computational research.

4.4.3. Disjunctive inequalities. Stubbs and Mehrotra [101], building on the earlier seminal work of Balas [8] on disjunctive programming and its application to MILP (via lift and project cuts) of Balas, Ceria and Cornuéjols [9], derive a lift and project cutting plane framework for convex (0-1) MINLPs. Consider the feasible region of the continuous relaxation of (MINLP-1) $R = \{(x, \eta) \mid f(x) \leq \eta, \ g_j(x) \leq 0 \ \forall j \in J, \ x \in X\}$. The procedure begins by choosing a (branching) dichotomy $x_i = 0 \ \vee \ x_i = 1$ for some $i \in I$. The convex hull of the union of the two (convex) sets $R_i^- \stackrel{\text{def}}{=} \{(x, \eta) \in R \mid x_i = 0\}, R_i^+ = \{(x, \eta) \in R \mid x_i = 1\}$ can be represented in a space of dimension $3n + 5$ as

$$\mathcal{M}_i(R) = \left\{ \begin{array}{c} (x, \eta, x^-, \eta^-, \\ x^+, \eta^+, \lambda^-, \lambda^+) \end{array} \left| \begin{array}{l} x = \lambda^- x^- + \lambda^+ x^+, \\ \eta = \lambda^- \eta^- + \lambda^+ \eta^+, \\ \lambda^- + \lambda^+ = 1, \ \lambda^- \geq 0, \ \lambda^+ \geq 0 \\ (x^-, \eta^-) \in R_i^-, \ (x^+, \eta^+) \in R_i^+ \end{array} \right. \right\}.$$

One possible complication with the convex hull description $\mathcal{M}_i(R)$ is caused by the nonlinear, nonconvex relationships $x = \lambda^- x^- + \lambda^+ x^+$ and $\eta = \lambda^- \eta^- + \lambda^+ \eta^+$. However, this description can be transformed to an equivalent description $\tilde{\mathcal{M}}_i(R)$ with only convex functional relationships between variables using the perspective function [101, 65].

Given some solution $(\bar{x}, \bar{\eta}) \notin \operatorname{conv}(R_i^- \cup R_i^+)$, the lift and project procedure operates by solving a convex separation problem

$$\min_{(x, \eta, \tilde{x}^-, \tilde{\eta}^-, \tilde{x}^+, \tilde{\eta}^+, \lambda^-, \lambda^+) \in \tilde{\mathcal{M}}_i(R)} d(x, \eta) \tag{4.1}$$

(where $d(x, \eta)$ is the distance to the point $(\bar{x}, \bar{\eta})$ in any norm). The lift-and-project inequality

$$\xi_x^T (x - \bar{x}) + \xi_\eta^T (\eta - \bar{\eta}) \geq 0 \tag{4.2}$$

separates $(\bar{x}, \bar{\eta})$ from $\operatorname{conv}(R_i^- \cup R_i^+)$, where ξ is a subgradient of $d(x, \eta)$ at the optimal solution of (4.1).

An implementation and evaluation of some of these ideas in the context of MISOCP has been done by Drewes [41]. Cezik and Iyengar [34] had also stated that the application of disjunctive cuts to conic-IP should be possible.

A limitation of disjunctive inequalities is that in order to generate a valid cut, one must solve an auxiliary (separation) problem that is three times the size of the original relaxation. In the case of MILP, clever intuition of Balas and Perregaard [11] allow to solve this separation problem while staying in the original space. No such extension is known in the case of MINLP. Zhu and Kuno [114] have suggested to replace the true nonlinear convex hull by a linear approximation taken about the solution to a linearized master problem like MP($\mathcal{K}$).

Kılınç *et al.* [71] have recently made the observation that a weaker form of the lift and project inequality (4.2) can be obtained from branching dichotomy information. Specifically, given values $\hat{\eta}_i^- = \min\{\eta | (x, \eta) \in R_i^-\}$ and $\hat{\eta}_i^+ \min\{\eta | (x, \eta) \in R_i^+\}$, the *strong-branching cut*

$$\eta \geq \hat{\eta}_i^- + (\hat{\eta}_i^+ - \hat{\eta}_i^-) x_i$$

is valid for MINLP, and is a special case of (4.2). Note that if strong-branching is used to determine the branching variable, then the values $\hat{\eta}_i^-$ and $\hat{\eta}_i^+$ are produced as a byproduct.

4.5. Heuristics. Here we discuss heuristic methods that are aimed at finding integer feasible solutions to MINLP with no guarantee of optimality or success. Heuristics are usually fast algorithms. In a branch-and-bound algorithm they are typically run right after the **Evaluate** step. Depending on the actual running time of the heuristic, it may be called at every node, every n^{th} node, or only at the root node. In linearization-based methods like OA, GBD, ECP, or LP/NLP-BB, heuristics may be run in the **Upper Bound** and **Refine** step, especially in the case when NLP($\hat{x}_I$) is infeasible. Heuristics are very important because by improving the upper bound z_U, they help in the **Prune** step of the branch-and-bound algorithm or in the convergence criterion of the other algorithms. From a practical point of

view, heuristics are extremely important when the algorithm cannot be carried out to completion, so that a feasible solution may be returned to the user.

Many heuristic methods have been devised for MILP, we refer the reader to [19] for a recent and fairly complete review. For convex MINLP, two heuristic principles that have been used are *diving heuristics* and the *feasibility pump*.

We note that several other heuristic principles could be used such as RINS [39] or Local Branching [49] but as far as we know, these have not been applied yet to (convex) MINLPs and we will not cover them here.

4.5.1. Diving heuristics. Diving heuristics are very related to the diving strategies for node selection presented in Section 4.3. The basic principle is to simulate a dive from the current node to a leaf of the tree by fixing variables (either one at a time or several variables at a time).

The most basic scheme is, after the NLP relaxation has been solved, to fix the variable which is the least integer infeasible in the current solution to the closest integer and resolve. This process is iterated until either the current solution is integer feasible or the NLP relaxation becomes infeasible. Many variants of this scheme have been proposed for MILP (see [19] for a good review). These differ mainly in the the number of variables fixed, the way to select variables to fix, and in the possibility of doing a certain amount of *backtracking* (unfixing previously fixed variables). The main difficulty when one tries to adapt these scheme to MINLP is that instead of having to resolve an LP with a modified bound at each iteration (an operation which is typically done extremely efficiently by state-of-the-art LP solvers) one has to solve an NLP (where warm-starting methods are usually much less efficient).

Bonami and Gonçalves [26] have adapted the basic scheme to MINLPs in two different manners. First in a straightforward way, but trying to limit the number of NLPs solved by fixing many variables at each iteration and backtracking if the fixings induce infeasibility. The second adaptation tries to reduce the problem to a MILP by fixing all the variables that appear in a nonlinear term in the objective or the constraints. This MILP problem may be given to a MILP solver in order to find a feasible solution. A similar MINLP heuristic idea of fixing variables to create MILPs can be found in the work [20].

4.5.2. Feasibility pumps. The feasibility pump is another heuristic principle for quickly finding feasible solutions. It was initially proposed by Fischetti, Glover and Lodi [48] for MILP, and can be extended to convex MINLP in several manners.

First we present the feasibility pump in its most trivial extension to MINLP. The basic principle of the Feasibility Pump consists of generating a sequence of points $\overline{x}^0, \ldots, \overline{x}^k$ that satisfy the constraints of the continuous relaxation NLPR(L_I,U_I). Associated with the sequence $\overline{x}^0, \ldots, \overline{x}^k$

of integer infeasible points is a sequence $\hat{x}^1, \ldots, \hat{x}^{k+1}$ of points that are integer feasible but do not necessarily satisfy the other constraints of the problem. Specifically, $\overline{x}^0$ is the optimal solution of NLPR(L_I, U_I). Each $\hat{x}^{i+1}$ is obtained by rounding $\overline{x}_j^i$ to the nearest integer for each $j \in I$ and keeping the other components equal to $\overline{x}_j^i$. The sequence $\overline{x}^i$ is generated by solving a nonlinear program whose objective function is to minimize the distance of x to $\hat{x}^i$ on the integer variables according to the ℓ_1-norm:

$$z_{\text{FP-NLP}(\hat{x}_I)} = \text{minimize} \quad \sum_{k \in I} |x_k - \hat{x}_k^i|$$

$$\text{subject to} \quad g_j(x) \leq 0 \ \forall j \in J, \quad \quad (\text{FP-NLP}(\hat{x}_I))$$

$$x \in X; L_I \leq \hat{x}_I \leq U_I.$$

The two sequences have the property that at each iteration the distance between $\overline{x}^i$ and $\hat{x}^{i+1}$ is non-increasing. The procedure stops whenever an integer feasible solution is found (or $\hat{x}^k = \overline{x}^k$). This basic procedure may cycle or stall without finding an integer feasible solution and randomization has been suggested to restart the procedure [48]. Several variants of this basic procedure have been proposed in the context of MILP [18, 3, 50]. The authors of [1, 26] have shown that the basic principle of the Feasibility Pump can also find good solutions in short computing time also in the context of MINLP.

Another variant of the Feasibility Pump for convex MINLPs was proposed by Bonami *et al.* [25]. Like in the basic FP scheme two sequences are constructed with the same properties: $\overline{x}^0, \ldots, \overline{x}^k$ are points in X that satisfy $g(\overline{x}^i) \leq 0$ but not $\overline{x}_I^i \in \mathbb{Z}^{|I|}$ and $\hat{x}^1, \ldots, \hat{x}^{k+1}$ are points that do not necessarily satisfy $g(\hat{x}^i) \leq 0$ but satisfy $\hat{x}_I^i \in \mathbb{Z}^{|I|}$. The sequence $\overline{x}^i$ is generated in the same way as before but the sequence $\hat{x}^i$ is now generated by solving MILPs. The MILP to solve for finding $\hat{x}^{i+1}$ is constructed by building an outer approximation of the constraints of the problem with linearizations taken in all the points of the sequence $\overline{x}^0, \ldots, \overline{x}^i$. Then, $\hat{x}^{i+1}$ is found as the point in the current outer approximation of the constraints that is closest to $\overline{x}^i$ in ℓ_1-norm in the space of integer constrained variables:

$$z_{\text{FP-M}^i} = \text{minimize} \sum_{i \in I} |x_j - \overline{x}_j^i|$$

$$\text{s.t.} \ g(\overline{x}^l) + \nabla g(\overline{x}^l)^T (x - \overline{x}^l) \leq 0 \quad l = 1, \ldots, i, \quad \quad (\text{M-FP}^i)$$

$$x \in X, \quad x_I \in \mathbb{Z}^I.$$

Unlike the procedure of Fischetti, Glover and Lodi, the *Feasibility Pump for MINLP* cannot cycle and it is therefore an exact algorithm: either it finds a feasible solution or it proves that none exists. This variant of the FP principle can also be seen as a variant of the Outer Approximation decomposition scheme presented in Section 3.2. In [25], it was also proposed

to iterate the FP scheme by integrating the linearization of the objective function in the constraint system of (M-FP^i) turning the feasibility pump into an exact iterative algorithm which finds solutions of increasingly better cost until eventually proving optimality. Abhishek *et al.* [1] have also proposed to integrate this Feasibility Pump into a single tree search (in the same way as Outer Approximation decomposition can be integrated in a single tree search when doing the LP/NLP-BB).

5. Software. There are many modern software packages implementing the algorithms of Section 3 that employ the modern enhancements described in Section 5. In this section, we describe the features of six different packages. The focus is on solvers for *general* convex MINLPs, not only special cases such as MIQP, MIQCP, or MISOCP. All of these packages may be freely used via a web interface on NEOS (`http://www-neos.mcs.anl.gov`).

5.1. α-ECP. α-ECP [110] is a solver based on the ECP method described in Section 3.4. Problems to be solved may be specified in a text-based format, as user-supplied subroutines, or via the GAMS algebraic modeling language. The software is designed to solve convex MINLP, but problems with a pseudo-convex objective function and pseudo-convex constraints can also be solved to global optimality with α-ECP. A significant feature of the software is that *no* nonlinear subproblems are required to be solved. (Though recent versions of the code have included an option to occasionally solve NLP subproblems, which may improve performance, especially on pseudo-convex instances.) Recent versions of the software also include enhancements so that each MILP subproblem need not be solved to global optimality. α-ECP requires a (commercial) MILP software to solve the reduced master problem (RM-ECP($\mathcal{K}$)), and CPLEX, XPRESS-MP, or Mosek may be used for this purpose.

In the computational experiment of Section 6, α-ECP (v1.75.03) is used with CPLEX (v12.1) as MILP solver, CONOPT (v3.24T) as NLP solver and α-ECP is run via GAMS. Since all instances are convex, setting the *ECPstrategy* option to 1 instructed α-ECP to not perform algorithmic steps relating to the solution of pseudo-convex instances.

5.2. Bonmin. Bonmin is an open-source MINLP solver and framework with implementations of algorithms NLP-BB, OA, and two different LP/NLP-BB algorithms with different default parameters. Source code and binaries of Bonmin are available from COIN-OR (`http://www.coin-or.org`). Bonmin may be called as a solver from both the AMPL and GAMS modeling languages.

Bonmin interacts with the COIN-OR software Cbc to manage the branch-and-bound trees of its various algorithms. To solve NLP subproblems, Bonmin may be instrumented to use either Ipopt [107] or FilterSQP [52]. Bonmin uses the COIN-OR software Clp to solve linear programs,

and may use Cbc or Cplex to solve MILP subproblems arising in its various algorithms.

The Bonmin NLP-BB algorithm features a range of different heuristics, advanced branching techniques such as strong-branching or pseudo-costs branching, and five different choices for node selection strategy. The Bonmin LP/NLP-BB methods use row management, cutting planes, and branching strategies from Cbc. A distinguishing feature of Bonmin is that one may instruct Bonmin to use a (time-limited) OA or feasibility pump heuristic at the beginning of the optimization.

In the computational experiments, Bonmin (v1.1) is used with Cbc (v2.3) as the MILP solver, Ipopt (v2.7) as NLP solver, and Clp (v1.10) is used as LP solver. For Bonmin, the algorithms, NLP-BB (denoted as B-BB) and LP/NLP-BB (denoted as B-Hyb) are tested. The default search strategies of dynamic node selection (mixture of depth-first-search and best-bound) and strong-branching were employed.

5.3. DICOPT. DICOPT is a software implementation of the OA method described in Section 3.2. DICOPT may be used as a solver from the GAMS modeling language. Although OA has been designed to solve convex MINLP, DICOPT may often be used successfully as a heuristic approach for nonconvex MINLP, as it contains features such as equality relaxation [72] and augmented penalty methods [105] for dealing with nonconvexities. DICOPT requires solvers for both NLP subproblems and MILP subproblems, and it uses available software as a "black-box" in each case. For NLP subproblems, possible NLP solvers include CONOPT [42], MINOS [89] and SNOPT [57]. For MILP subproblems, possible MILP solvers include CPLEX [66] and XPRESS [47]. DICOPT contains a number of heuristic (inexact) stopping rules for the OA method that may be especially effective for nonconvex instances.

In our computational experiment, the DICOPT that comes with GAMS v23.2.1 is used with CONOPT (v3.24T) as the NLP solver and Cplex (v12.1) as the MILP solver. In order to ensure that instances are solved to provable optimality, the GAMS/DICOPT option `stop` was set to value 1.

5.4. FilMINT. FilMINT [2] is a non-commercial solver for convex MINLPs based on the LP/NLP-BB algorithm. FilMINT may be used through the AMPL language.

FilMINT uses MINTO [93] a branch-and-cut framework for MILP to solve the reduced master problem $(MP(\mathcal{K}))$ and filterSQP [52] to solve nonlinear subproblems. FilMINT uses the COIN-OR LP solver Clp or CPLEX to solve linear programs.

FilMINT by default employs nearly all of MINTO's enhanced MILP features, such as cutting planes, primal heuristics, row management, and enhanced branching and node selection rules. By default, pseudo-costs branching is used as branching strategy and best estimate is used as node

selection strategy. An NLP-based Feasibility Pump can be run at the beginning of the optimization as a heuristic procedure. The newest version of FilMINT has been augmented with the simple strong-branching disjunctive cuts described in Section 4.4.3.

In the computational experiments of Section 6, FilMINT v0.1 is used with Clp as LP solver. Two versions of FilMINT are tested—the default version and a version including the strong-branching cuts (Filmint-SBC).

5.5. MINLP_BB. MINLP_BB [77] is an implementation of the NLP-BB algorithm equipped with different node selection and variable selection rules. Instances can be specified to MINLP_BB through an AMPL interface.

MINLP_BB contains its own tree-manager implementation, and NLP subproblems are solved by FilterSQP [52]. Node selection strategies available in MINLP_BB include depth-first-search, depth-first-search with backtrack to best-bound, best-bound, and best-estimated solution. For branching strategies, MINLP_BB contains implementations of most fractional branching, strong-branching, approximate strong-branching using second-order information, pseudo-costs branching and reliability branching. MINLP_BB is written in FORTRAN. Thus, there is no dynamic memory allocation, and the user must specify a maximum memory (stack) size at the beginning of algorithm to store the list of open nodes.

For the computational experiments with MINLP_BB, different levels of stack size were tried in an attempt to use the entire available memory for each instance. The default search strategies of depth-first-search with backtrack to best-bound and pseudo-costs branching were employed in MINLP_BB (v20090811).

5.6. SBB. SBB [30] is a NLP-based branch-and-bound solver that is available through the GAMS modeling language. The NLP subproblems can be solved by CONOPT [42], MINOS [89] and SNOPT [57]. Pseudo-costs branching is an option as a branching rule. As a node selection strategy, depth-first-search, best-bound, best-estimate or combination of these three can be employed. Communication of subproblems between the NLP solver and tree manager is done via files, so SBB may incur some extra overhead when compared to other solvers.

In our computational experiments, we use the version of SBB shipped with GAMS v23.2.1. CONOPT is used as NLP solver, and the SBB default branching variable and node selection strategies are used.

6. Computational study.

6.1. Problems. The test instances used in the computational experiments were gathered from the MacMINLP collection of test problems [79], the GAMS collection of MINLP problems [31], the collection on the website of IBM-CMU research group [99], and instances created by the authors. Characteristics of the instances are given in Table 1, which lists whether

or not the instance has a nonlinear objective function, the total number of variables, the number of integer variables, the number of constraints, and how many of the constraints are nonlinear.

BatchS: The *BatchS* problems [97, 104] are multi-product batch plant design problems where the objective is to determine the volume of the equipment, the number of units to operate in parallel, and the locations of intermediate storage tanks.

CLay: The *CLay* problems [98] are constrained layout problems where non-overlapping rectangular units must be placed within the confines of certain designated areas such that the cost of connecting these units is minimized.

FLay: The *FLay* problems [98] are farmland layout problems where the objective is to determine the optimal length and width of a number of rectangular patches of land with fixed area, such that the perimeter of the set of patches is minimized.

fo-m-o: These are block layout design problems [33], where an orthogonal arrangement of rectangular departments within a given rectangular facility is required. A distance-based objective function is to be minimized, and the length and width of each department should satisfy given size and area requirements.

RSyn: The *RSyn* problems [98] concern retrofit planning, where one would like to redesign existing plants to increase throughput, reduce energy consumption, improve yields, and reduce waste generation. Given limited capital investments to make process improvements and cost estimations over a given time horizon, the problem is to identify the modifications that yield the highest income from product sales minus the cost of raw materials, energy, and process modifications.

SLay: The *SLay* problems [98] are safety layout problems where optimal placement of a set of units with fixed width and length is determined such that the Euclidean distance between their center point and a predefined "safety point" is minimized.

sssd: The *sssd* instances [45] are stochastic service system design problems. Servers are modeled as M/M/1 queues, and a set of customers must be assigned to the servers which can be operated at different service levels. The objective is to minimize assignment and operating costs.

Syn: The *Syn* instances [43, 102] are synthesis design problems dealing with the selection of optimal configuration and parameters for a processing system selected from a superstructure containing alternative processing units and interconnections.

trimloss: The *trimloss* (*tls*) problems [64] are cutting stock problems where one would like to determine how to cut out a set of product paper rolls from raw paper rolls such that the trim loss as well as the overall production is minimized.

uflquad: The *uflquad* problems [61] are (separable) quadratic uncapacitated facility location problems where a set of customer demands must be

TABLE 1
Test set statistics.

Problem	NonL Obj	Vars	Ints	Cons	NonL Cons
BatchS121208M	√	407	203	1510	1
BatchS151208M	√	446	203	1780	1
BatchS201210M	√	559	251	2326	1
CLay0303H		100	21	114	36
CLay0304H		177	36	210	48
CLay0304M		57	36	58	48
CLay0305H		276	55	335	60
CLay0305M		86	55	95	60
FLay04H		235	24	278	4
FLay05H		383	40	460	5
FLay05M		63	40	60	5
FLay06M		87	60	87	6
fo7_2		115	42	197	14
fo7		115	42	197	14
fo8		147	56	257	16
m6		87	30	145	12
m7		115	42	197	14
o7_2		115	42	197	14
RSyn0805H		309	37	426	3
RSyn0805M02M		361	148	763	6
RSyn0805M03M		541	222	1275	9
RSyn0805M04M		721	296	1874	12
RSyn0810M02M		411	168	854	12
RSyn0810M03M		616	252	1434	18
RSyn0820M		216	84	357	14
RSyn0830M04H		2345	496	4156	80
RSyn0830M		251	94	405	20
RSyn0840M		281	104	456	28
SLay06H	√	343	60	435	0
SLay07H	√	477	84	609	0
SLay08H	√	633	112	812	0
SLay09H	√	811	144	1044	0
SLay09M	√	235	144	324	0
SLay10M	√	291	180	405	0
sssd-10-4-3		69	52	30	12
sssd-12-5-3		96	75	37	15
sssd-15-6-3		133	108	45	18
Syn15M04M		341	120	762	44
Syn20M03M		316	120	657	42
Syn20M04M		421	160	996	56
Syn30M02M		321	120	564	40
Syn40M03H		1147	240	1914	84
Syn40M		131	40	198	28
tls4		106	89	60	4
tls5		162	136	85	5
uflquad-20-150	√	3021	20	3150	0
uflquad-30-100	√	3031	30	3100	0
uflquad-40-80	√	3241	40	3280	0

satisfied by open facilities. The objective is to minimize the sum of the fixed cost for operating facilities and the shipping cost which is proportional to the square of the quantity delivered to each customer.

All test problems are available in AMPL and GAMS formats and are available from the authors upon request. In our experiments, $\alpha-$ECP, DICOPT, and SBB are tested through the GAMS interface, while Bonmin, FilMINT and MINLP_BB are tested through AMPL.

6.2. Computational results. The computational experiments have been run on a cluster of identical 64-bit Intel Xeon microprocessors clocked at 2.67 GHz, each with 3 GB RAM. All machines run the Red Hat Enterprise Linux Server 5.3 operating system. A three hour time limit is enforced. The computing times used in our comparisons are the wall-clock times (including system time). All runs were made on processors dedicated to the computation. Wall-clock times were used to accurately account for system overhead incurred by file I/O operations required by the SBB solver. For example, on the problem FLay05M, SBB reports a solution time of 0.0 seconds for 92241 nodes, but the wall-clock time spent is more than 17 minutes.

Table 3 summarizes the performance of the solvers on the 48 problems of the test set. The table lists for each solver the number of times the optimal solution was found, the number of times the time limit was exceeded, the number of times the node limit exceeded, the number of times an error occurred (other than time limit or memory limit), the number of times the solver is fastest, and the arithmetic and geometric means of solution times in seconds. When reporting aggregated solution times, unsolved or failed instances are accounted for with the time limit of three hours. A performance profile [40] of solution time is given in Figure 1. The detailed performance of each solver on each test instance is listed in Table 4.

There are a number of interesting observations that can be made from this experiment. First, for the instances that they can solve, the solvers DICOPT and α-ECP tend to be very fast. Also, loosely speaking, for each class of instances, there seems to be one or two solvers whose performance dominates the others, and we have listed these in Table 2.

TABLE 2
Subjective Rating of Best Solver on Specific Instance Families.

Instance Family	Best Solvers
Batch	DICOPT
CLay, FLay, sssd	FilMINT, MINLP_BB
Fo, RSyn, Syn	DICOPT, α-ECP
SLay	MINLP_BB
uflquad	Bonmin (B-BB)

TABLE 3
Solver statistics on the test set.

Solver	Opt.	Time Limit	Mem. Limit	Error	Fastest	Arith. Mean	Geom. Mean
α-ECP	37	9	0	2	4	2891.06	105.15
Bonmin-BB	35	5	8	0	4	4139.60	602.80
Bonmin-Hyb	32	0	15	1	1	3869.08	244.41
Dicopt	30	16	0	2	21	4282.77	**90.79**
Filmint	41	7	0	0	4	2588.79	343.47
Filmint-SBC	43	5	0	0	3	**2230.11**	274.61
MinlpBB	35	3	7	3	12	3605.45	310.09
Sbb	18	23	6	1	0	7097.49	2883.75

In general, the variation in solver performance on different instance families indicates that a "portfolio" approach to solving convex MINLPs is still required. Specifically, if the performance of a specific solver is not satisfactory, one should try other software on the instance as well.

7. Conclusions. Convex Mixed-Integer Nonlinear Programs (MINLPs) can be used to model many decision problems involving both nonlinear and discrete components. Given their generality and flexibility, MINLPs have been proposed for many diverse and important scientific applications. Algorithms and software are evolving so that instances of these important models can often be solved in practice. The main advances are being made along two fronts. First, new theory is being developed. Second, theory and implementation techniques are being translated from the more-developed arena of Mixed Integer Linear Programming into MINLP. We hope this survey has provided readers the necessary background to delve deeper into this rapidly evolving field.

TABLE 4

*Comparison of running times (in seconds) for the solvers α-ECP(α**ECP**), Bonmin-BB(**B-BB**), Bonmin-LP/NLP-BB(**B-Hyb**), DICOPT, FilMINT(**Fil**), FilMINT with strong-branching cuts(**Fil-SBC**), MINLP_BB(**M-BB**) and SBB (bold face for best running time). If the solver could not provide the optimal solution, we state the reason with following letters: "t" states that the 3 hour time limit is hit, "m" states that the 3 GB memory limit is passed over and "f" states that the solver has failed to find optimal solution without hitting time limit or memory limit.*

| Problem	αECP	B-BB	B-Hyb	Dicopt	Fil	Fil-SBC	M-BB	Sbb
BatchS121208M	384.8	47.8	43.9	**6.9**	31.1	14.7	128.7	690.1
BatchS151208M	297.4	139.2	71.7	**9.3**	124.5	42.8	1433.5	138.3
BatchS201210M	137.3	188.1	148.8	**9.5**	192.4	101.9	751.2	463.8
CLay0303H	7.0	21.1	13.0	33.2	2.0	1.4	**0.5**	14.0
CLay0304H	22.1	76.0	68.9	t	21.7	8.7	**7.3**	456.6
CLay0304M	16.2	54.1	50.4	t	**5.0**	**5.0**	6.2	192.4
CLay0305H	88.8	2605.5	125.2	1024.3	162.0	**58.4**	87.6	1285.6
CLay0305M	22.7	775.0	46.1	2211.0	**10.3**	32.9	31.9	582.6
FLay04H	106.0	48.1	42.2	452.3	**8.3**	8.6	12.5	41.9
FLay05H	t	2714.0	m	t	1301.2	1363.4	**1237.8**	4437.0
FLay05M	5522.3	596.8	m	t	698.4	775.4	**57.5**	1049.2
FLay06M	t	m	m	t	t	t	**2933.6**	t
fo7_2	**16.5**	m	104.5	t	271.9	200.0	964.1	t
fo7	**98.9**	t	285.2	4179.6	1280.1	1487.9	f	t
fo8	**447.4**	t	m	t	3882.7	7455.5	m	t
m6	0.9	79.4	31.1	**0.2**	89.0	29.8	7.1	2589.9
m7	4.6	3198.3	75.4	**0.5**	498.6	620.0	215.9	t
o7_2	**728.4**	m	m	t	3781.3	7283.8	m	t
RSyn0805H	0.6	4.0	**0.1**	0.1	0.5	3.4	2.0	4.3
RSyn0805M02M	9.8	2608.2	m	**3.4**	485.0	123.8	806.1	t
RSyn0805M03M	16.6	6684.6	10629.9	**7.2**	828.2	668.3	4791.9	t
RSyn0805M04M	10.5	10680.0	m	**8.1**	1179.2	983.6	4878.8	m
RSyn0810M02M	10.1	t	m	**4.6**	7782.2	3078.1	m	m
RSyn0810M03M	43.3	m	m	**10.8**	t	8969.5	m	m
RSyn0820M	1.4	5351.4	11.8	**0.3**	232.2	231.6	1005.1	t
RSyn0830M04H	19.9	320.4	30.5	**5.6**	78.7	193.7	f	f
RSyn0830M	2.2	m	7.4	**0.5**	375.7	301.1	1062.3	m
RSyn0840M	1.6	m	13.7	**0.4**	3426.2	2814.7	m	m
SLay06H	316.8	3.4	34.4	7.5	186.6	5.0	**1.3**	731.9
SLay07H	2936.0	7.6	46.6	39.8	385.0	81.5	**5.6**	t
SLay08H	8298.0	12.5	178.1	t	1015.0	156.0	**9.7**	t
SLay09H	t	**25.6**	344.0	3152.5	7461.5	1491.9	55.0	t
SLay09M	t	8.6	63.0	t	991.7	41.8	**2.1**	1660.7
SLay10M	t	108.4	353.7	t	t	3289.6	**43.3**	2043.7
sssd-10-4-3	2.4	16.8	31.2	2.8	3.7	3.0	**1.0**	t
sssd-12-5-3	f	56.1	m	f	9.6	13.3	**5.7**	t
sssd-15-6-3	f	163.4	m	f	**36.9**	1086.4	41.6	t
Syn15M04M	2.9	379.5	8.5	**0.2**	39.5	47.5	60.4	278.0
Syn20M03M	2.9	9441.4	16.2	**0.1**	1010.3	901.9	1735.0	t
Syn20M04M	3.9	m	22.6	**0.2**	7340.9	5160.4	m	t
Syn30M02M	3.0	t	13.6	**0.4**	2935.5	3373.8	8896.1	t
Syn40M03H	8.3	18.9	3.9	**1.1**	6.5	87.9	f	19.7
Syn40M	1.8	877.7	0.6	**0.1**	108.1	110.7	101.4	t
tls4	377.7	t	f	t	383.0	**336.7**	1281.8	t
tls5	m	m	m	t	t	t	m	m
uflquad-20-150	t	**422.2**	m	t	t	t	t	t
uflquad-30-100	t	**614.5**	m	t	t	t	t	t
uflquad-40-80	t	**9952.3**	m	t	t	t	t	t
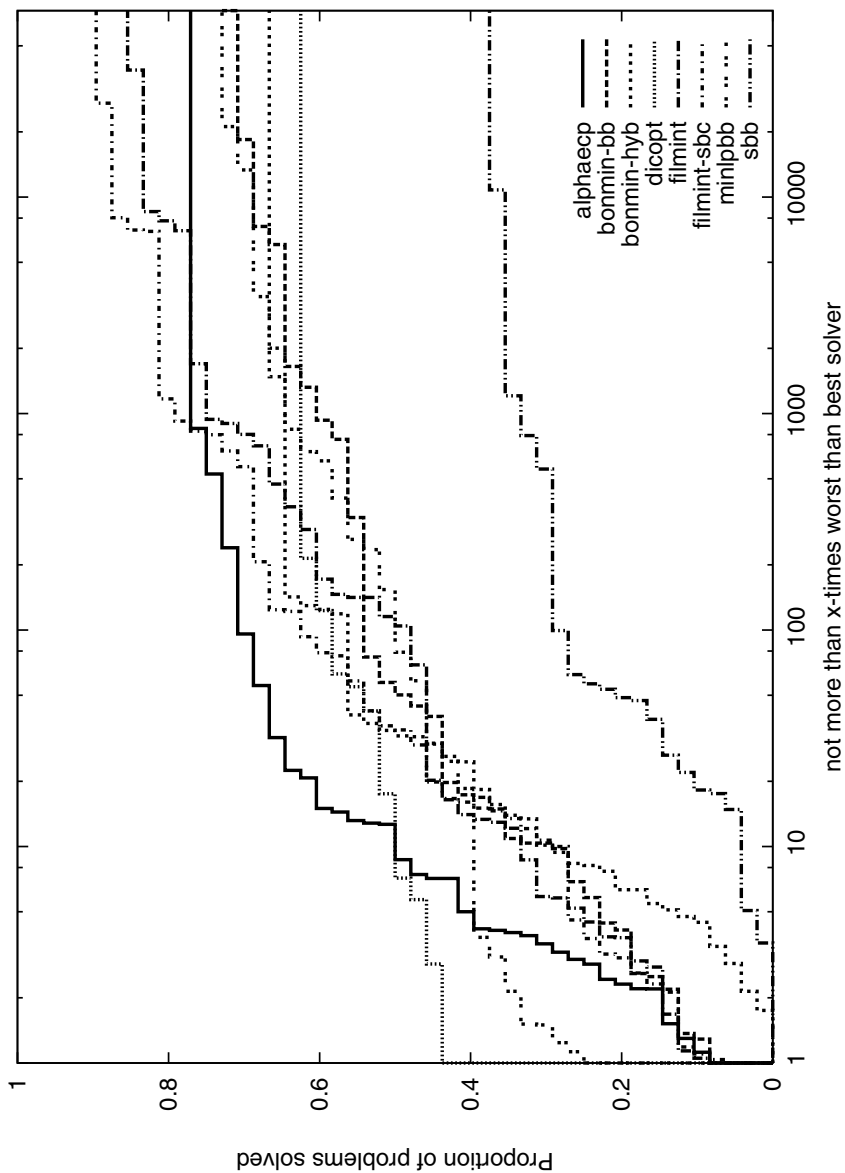
Fig. 1. *Performance Profile Comparing Convex MINLP Solvers.*

REFERENCES

[1] K. ABHISHEK, S. LEYFFER, AND J.T. LINDEROTH, *Feasibility pump heuristics for Mixed Integer Nonlinear Programs.* Unpublished working paper, 2008.

[2] ———, *FilMINT: An outer-approximation-based solver for convex Mixed-Integer Nonlinear Programs*, INFORMS Journal on Computing, **22**, No. 4 (2010), pp. 555–567.

[3] T. ACHTERBERG AND T. BERTHOLD, *Improving the feasibility pump*, Technical Report ZIB-Report 05-42, Zuse Institute Berlin, September 2005.

[4] T. ACHTERBERG, T. KOCH, AND A. MARTIN, *Branching rules revisited*, Operations Research Letters, **33** (2004), pp. 42–54.

[5] I. AKROTIRIANAKIS, I. MAROS, AND B. RUSTEM, *An outer approximation based branch-and-cut algorithm for convex 0-1 MINLP problems*, Optimization Methods and Software, **16** (2001), pp. 21–47.

[6] D. APPLEGATE, R. BIXBY, V. CHVÁTAL, AND W. COOK, *On the solution of traveling salesman problems*, in Documenta Mathematica Journal der Deutschen Mathematiker-Vereinigung, International Congress of Mathematicians, 1998, pp. 645–656.

[7] A. ATAMTÜRK AND V. NARAYANAN, *Conic mixed integer rounding cuts*, Mathematical Programming, **122** (2010), pp. 1–20.

[8] E. BALAS, *Disjunctive programming*, in Annals of Discrete Mathematics 5: Discrete Optimization, North Holland, 1979, pp. 3–51.

[9] E. BALAS, S. CERIA, AND G. CORNEUJOLS, *A lift-and-project cutting plane algorithm for mixed 0-1 programs*, Mathematical Programming, **58** (1993), pp. 295–324.

[10] E. BALAS, S. CERIA, G. CORNUÉJOLS, AND N. R. NATRAJ, *Gomory cuts revisited*, Operations Research Letters, **19** (1999), pp. 1–9.

[11] E. BALAS AND M. PERREGAARD, *Lift-and-project for mixed 0-1 programming: recent progress*, Discrete Applied Mathematics, **123** (2002), pp. 129–154.

[12] M.S. BAZARAA, H.D. SHERALI, AND C.M. SHETTY, *Nonlinear Programming: Theory and Algorithms*, John Wiley and Sons, New York, second ed., 1993.

[13] E.M.L. BEALE, *Branch and bound methods for mathematical programming systems*, in Discrete Optimization II, P.L. Hammer, E.L. Johnson, and B.H. Korte, eds., North Holland Publishing Co., 1979, pp. 201–219.

[14] E.W.L. BEALE AND J.A. TOMLIN, *Special facilities in a general mathematical programming system for non-convex problems using ordered sets of variables*, in Proceedings of the 5th International Conference on Operations Research, J. Lawrence, ed., 1969, pp. 447–454.

[15] A. BEN-TAL AND A. NEMIROVSKI, *Lectures on Modern Convex Optimization*, SIAM, 2001. MPS/SIAM Series on Optimization.

[16] J.F. BENDERS, *Partitioning procedures for solving mixed variable programming problems*, Numerische Mathematik, **4** (1962), pp. 238–252.

[17] M. BÉNICHOU, J.M. GAUTHIER, P. GIRODET, G. HENTGES, G. RIBIÈRE, AND O. VINCENT, *Experiments in Mixed-Integer Linear Programming*, Mathematical Programming, **1** (1971), pp. 76–94.

[18] L. BERTACCO, M. FISCHETTI, AND A. LODI, *A feasibility pump heuristic for general mixed-integer problems*, Discrete Optimization, **4** (2007), pp. 63–76.

[19] T. BERTHOLD, *Primal Heuristics for Mixed Integer Programs*, Master's thesis, Technische Universität Berlin, 2006.

[20] T. BERTHOLD AND A. GLEIXNER, *Undercover - a primal heuristic for MINLP based on sub-mips generated by set covering*, Tech. Rep. ZIB-Report 09-40, Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB), 2009.

[21] D. BIENSTOCK, *Computational study of a family of mixed-integer quadratic programming problems*, Mathematical Programming, **74** (1996), pp. 121–140.

[22] R. Bixby and E. Rothberg, *Progress in computational mixed integer programming. A look back from the other side of the tipping point*, Annals of Operations Research, **149** (2007), pp. 37–41.

[23] P. Bonami, *Branching strategies and heuristics in branch-and-bound for convex MINLPs*, November 2008. Presentation at IMA Hot Topics Workshop: Mixed-Integer Nonlinear Optimization: Algorithmic Advances and Applications.

[24] P. Bonami, L.T. Biegler, A.R. Conn, G. Cornuéjols, I.E. Grossmann, C.D. Laird, J. Lee, A. Lodi, F. Margot, N. Sawaya, and A. Wächter, *An algorithmic framework for convex Mixed Integer Nonlinear Programs*, Discrete Optimization, **5** (2008), pp. 186–204.

[25] P. Bonami, G. Cornuéjols, A. Lodi, and F. Margot, *A feasibility pump for Mixed Integer Nonlinear Programs*, Mathematical Programming, **119** (2009), pp. 331–352.

[26] P. Bonami and J. Gonçalves, *Heuristics for convex mixed integer nonlinear programs*, Computational Optimization and Applications. To appear DOI: 10.1007/s10589-010-9350-6.

[27] R. Boorstyn and H. Frank, *Large-scale network topological optimization*, IEEE Transactions on Communications, **25** (1977), pp. 29–47.

[28] B. Borchers and J.E. Mitchell, *An improved branch and bound algorithm for Mixed Integer Nonlinear Programs*, Computers & Operations Research, **21** (1994), pp. 359–368.

[29] ———, *A computational comparison of branch and bound and outer approximation algorithms for 0-1 Mixed Integer Nonlinear Programs*, Computers & Operations Research, **24** (1997), pp. 699–701.

[30] M.R. Bussieck and A. Drud, *Sbb: A new solver for Mixed Integer Nonlinear Programming*, talk, OR 2001, Section "Continuous Optimization", 2001.

[31] M.R. Bussieck, A.S. Drud, and A. Meeraus, *MINLPLib – a collection of test models for Mixed-Integer Nonlinear Programming*, INFORMS Journal on Computing, **15** (2003).

[32] R.H. Byrd, J. Nocedal, and R.A. Waltz, *KNITRO: An integrated package for nonlinear optimization*, in Large Scale Nonlinear Optimization, Springer Verlag, 2006, pp. 35–59.

[33] I. Castillo, J. Westerlund, S. Emet, and T. Westerlund, *Optimization of block layout deisgn problems with unequal areas: A comparison of milp and minlp optimization methods*, Computers and Chemical Engineering, **30** (2005), pp. 54–69.

[34] M.T. Cezik and G. Iyengar, *Cuts for mixed 0-1 conic programming*, Mathematical Programming, **104** (2005), pp. 179–202.

[35] V. Chvátal, *Edmonds polytopes and a heirarchy of combinatorial problems*, Discrete Mathematics, **4** (1973), pp. 305–337.

[36] M. Conforti, G. Cornuéjols, and G. Zambelli, *Polyhedral approaches to Mixed Integer Linear Programming*, in 50 Years of Integer Programming 1958—2008, M. Jünger, T. Liebling, D. Naddef, W. Pulleyblank, G. Reinelt, G. Rinaldi, and L. Wolsey, eds., Springer, 2009.

[37] G. Cornuéjols, L. Liberti, and G. Nannicini, *Improved strategies for branching on general disjunctions*. To appear in Mathematical Programming A. DOI: 10.1007/s10107-009-0333-2.

[38] R.J. Dakin, *A tree search algorithm for mixed programming problems*, Computer Journal, **8** (1965), pp. 250–255.

[39] E. Danna, E. Rothberg, and C. LePape, *Exploring relaxation induced neighborhoods to improve MIP solutions*, Mathematical Programming, **102** (2005), pp. 71–90.

[40] E. Dolan and J. Moré, *Benchmarking optimization software with performance profiles*, Mathematical Programming, **91** (2002), pp. 201–213.

[41] S. Drewes, *Mixed Integer Second Order Cone Programming*, PhD thesis, Technische Universität Darmstadt, 2009.

[42] A. S. Drud, *CONOPT – a large-scale GRG code*, ORSA Journal on Computing, **6** (1994), pp. 207–216.

[43] M.A. Duran and I. Grossmann, *An outer-approximation algorithm for a class of Mixed-Integer Nonlinear Programs*, Mathematical Programming, **36** (1986), pp. 307–339.

[44] J. Eckstein, *Parallel branch-and-bound algorithms for general mixed integer programming on the CM-5*, SIAM Journal on Optimization, **4** (1994), pp. 794–814.

[45] S. Elhedhli, *Service System Design with Immobile Servers, Stochastic Demand, and Congestion*, Manufacturing & Service Operations Management, **8** (2006), pp. 92–97.

[46] A. M. Eliceche, S. M. Corvalán, and P. Martínez, *Environmental life cycle impact as a tool for process optimisation of a utility plant*, Computers and Chemical Engineering, **31** (2007), pp. 648–656.

[47] Fair Isaac Corporation, *XPRESS-MP Reference Manual*, 2009. Release 2009.

[48] M. Fischetti, F. Glover, and A. Lodi, *The feasibility pump*, Mathematical Programming, **104** (2005), pp. 91–104.

[49] M. Fischetti and A. Lodi, *Local branching*, Mathematical Programming, **98** (2003), pp. 23–47.

[50] M. Fischetti and D. Salvagnin, *Feasibility pump 2.0*, Tech. Rep., University of Padova, 2008.

[51] R. Fletcher and S. Leyffer, *Solving Mixed Integer Nonlinear Programs by outer approximation*, Mathematical Programming, **66** (1994), pp. 327–349.

[52] ———, *User manual for filterSQP*, 1998. University of Dundee Numerical Analysis Report NA-181.

[53] A. Flores-Tlacuahuac and L.T. Biegler, *Simultaneous mixed-integer dynamic optimization for integrated design and control*, Computers and Chemical Engineering, **31** (2007), pp. 648–656.

[54] J.J.H. Forrest, J.P.H. Hirst, and J.A. Tomlin, *Practical solution of large scale mixed integer programming problems with UMPIRE*, Management Science, 20 (1974), pp. 736–773.

[55] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, New York, 1979.

[56] A. Geoffrion, *Generalized Benders decomposition*, Journal of Optimization Theory and Applications, **10** (1972), pp. 237–260.

[57] P.E. Gill, W. Murray, and M.A. Saunders, *SNOPT: An SQP algorithm for large–scale constrained optimization*, SIAM Journal on Optimization, **12** (2002), pp. 979–1006.

[58] R.E. Gomory, *Outline of an algorithm for integer solutions to linear programs*, Bulletin of the American Mathematical Monthly, **64** (1958), pp. 275–278.

[59] ———, *An algorithm for the mixed integer problem*, Tech. Rep. RM-2597, The RAND Corporation, 1960.

[60] I. Grossmann, J. Viswanathan, A.V.R. Raman, and E. Kalvelagen, *GAMS/DICOPT: A discrete continuous optimization package*, Math. Methods Appl. Sci, **11** (2001), pp. 649–664.

[61] O. Günlük, J. Lee, and R. Weismantel, *MINLP strengthening for separaable convex quadratic transportation-cost ufl*, Tech. Rep. RC24213 (W0703-042), IBM Research Division, March 2007.

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

[63] Gurobi Optimization, *Gurobi Optimizer Reference Manual*, 2009. Version 2.

[64] I. Harjunkoski, R.Pörn, and T. Westerlund, *MINLP: Trim-loss problem*, in Encyclopedia of Optimization, C.A. Floudas and P.M. Pardalos, eds., Springer, 2009, pp. 2190–2198.

[65] J.-B. Hiriart-Urruty and C. Lemarechal, *Convex Analysis and Minimization Algorithms I: Fundamentals (Grundlehren Der Mathematischen Wissenschaften)*, Springer, October 1993.

[66] IBM, *Using the CPLEX Callable Library, Version 12*, 2009.

[67] R. Jeroslow, *There cannot be any algorithm for integer programming with quadratic constraints*, Operations Research, **21** (1973), pp. 221–224.

[68] N.J. Jobst, M.D. Horniman, C.A. Lucas, and G. Mitra, *Computational aspects of alternative portfolio selection models in the presence of discrete asset choice constraints*, Quantitative Finance, **1** (2001), pp. 489–501.

[69] M. Karamanov and G. Cornuéjols, *Branching on general disjunctions*, tech. rep., Carnegie Mellon University, July 2005. Revised August 2009. Available at http://integer.tepper.cmu.edu.

[70] J.E. Kelley, *The cutting plane method for solving convex programs*, Journal of SIAM, **8** (1960), pp. 703–712.

[71] M. Kılınç, J. Linderoth, J. Luedtke, and A. Miller, *Disjunctive strong branching inequalities for Mixed Integer Nonlinear Programs*.

[72] G.R. Kocis and I.E. Grossmann, *Relaxation strategy for the structural optimization of process flowheets*, Industrial Engineering Chemical Research, **26** (1987), pp. 1869–1880.

[73] C.D. Laird, L.T. Biegler, and B. van Bloemen Waanders, *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** (2006), pp. 242–251.

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

[75] M. Lejeune, *A unified approach for cycle service levels*, Tech. Rep., George Washington University, 2009. Available on Optimization Online http://www.optimization-online.org/DB_HTML/2008/11/2144.html.

[76] S. Leyffer, *Deterministic Methods for Mixed Integer Nonlinear Programming*, PhD thesis, University of Dundee, Dundee, Scotland, UK, 1993.

[77] ———, *User manual for MINLP-BB*, 1998. University of Dundee.

[78] ———, *Integrating SQP and branch-and-bound for Mixed Integer Nonlinear Programming*, Computational Optimization & Applications, **18** (2001), pp. 295–309.

[79] ———, *MacMINLP: Test problems for Mixed Integer Nonlinear Programming*, 2003. http://www.mcs.anl.gov/~leyffer/macminlp.

[80] ———, *Nonlinear branch-and-bound revisited*, August 2009. Presentation at 20th International Symposium on Mathematical Programming.

[81] L. Liberti, *Reformulations in mathematical programming: Symmetry*, Mathematical Programming (2010). To appear.

[82] J. Linderoth and T. Ralphs, *Noncommercial software for Mixed-Integer Linear Programming*, in Integer Programming: Theory and Practice, CRC Press Operations Research Series, 2005, pp. 253–303.

[83] J.T. Linderoth and M.W.P. Savelsbergh, *A computational study of search strategies in mixed integer programming*, INFORMS Journal on Computing, **11** (1999), pp. 173–187.

[84] A. Lodi, *MIP computation and beyond*, in 50 Years of Integer Programming 1958—2008, M. Jünger, T. Liebling, D. Naddef, W. Pulleyblank, G. Reinelt, G. Rinaldi, and L. Wolsey, eds., Springer, 2009.

[85] H. Marchand and L.A. Wolsey, *Aggregation and mixed integer rounding to solve MIPs*, Operations Research, **49** (2001), pp. 363–371.

[86] F. Margot, *Exploiting orbits in symmetric ILP*, Mathematical Programming, Series B, **98** (2003), pp. 3–21.

[87] R.D. McBride and J.S. Yormark, *An implicit enumeration algorithm for quadratic integer programming*, Management Science, **26** (1980), pp. 282–296.

[88] *Mosek ApS*, 2009. www.mosek.com.

[89] B. Murtagh and M. Saunders, *MINOS 5.4 user's guide*, Report SOL 83-20R, Department of Operations Research, Stanford University, 1993.

[90] K.G. Murty and S.N. Kabadi, *Some NP-complete problems in quadratic and nonlinear programming*, Mathematical Programming, **39** (1987), pp. 117–129.

[91] S. Nabal and L. Schrage, *Modeling and solving nonlinear integer programming problems.* Presented at Annual AIChE Meeting, Chicago, 1990.

[92] G. Nemhauser and L.A. Wolsey, *Integer and Combinatorial Optimization*, John Wiley and Sons, New York, 1988.

[93] G.L. Nemhauser, M.W.P. Savelsbergh, and G.C. Sigismondi, *MINTO, a Mixed INTeger Optimizer*, Operations Research Letters, **15** (1994), pp. 47–58.

[94] J. Nocedal and S.J. Wright, *Numerical Optimization*, Springer-Verlag, New York, second ed., 2006.

[95] J. Ostrowski, J. Linderoth, F. Rossi, and S. Smriglio, *Orbital branching*, Mathematical Programming, **126** (2011), pp. 147–178.

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

[97] D.E. Ravemark and D.W.T. Rippin, *Optimal design of a multi-product batch plant*, Computers & Chemical Engineering, **22** (1998), pp. 177 – 183.

[98] N. Sawaya, *Reformulations, relaxations and cutting planes for generalized disjunctive programming*, PhD thesis, Chemical Engineering Department, Carnegie Mellon University, 2006.

[99] N. Sawaya, C.D. Laird, L.T. Biegler, P. Bonami, A.R. Conn, G. Cornuéjols, I. E. Grossmann, J. Lee, A. Lodi, F. Margot, and A. Wächter, *CMU-IBM open source MINLP project test set*, 2006. http://egon.cheme.cmu.edu/ibm/page.htm.

[100] A. Schrijver, *Theory of Linear and Integer Programming*, Wiley, Chichester, 1986.

[101] R. Stubbs and S. Mehrotra, *A branch-and-cut method for 0-1 mixed convex programming*, Mathematical Programming, **86** (1999), pp. 515–532.

[102] M. Türkay and I.E. Grossmann, *Logic-based minlp algorithms for the optimal synthesis of process networks*, Computers & Chemical Engineering, **20** (1996), pp. 959 – 978.

[103] R.J. Vanderbei, *LOQO: An interior point code for quadratic programming*, Optimization Methods and Software (1998).

[104] A. Vecchietti and I.E. Grossmann, *LOGMIP: a disjunctive 0-1 non-linear optimizer for process system models*, Computers and Chemical Engineering, **23** (1999), pp. 555 – 565.

[105] J. Viswanathan and I.E. Grossmann, *A combined penalty function and outer–approximation method for MINLP optimization*, Computers and Chemical Engineering, **14** (1990), pp. 769–782.

[106] A. Wächter, *Some recent advanced in Mixed-Integer Nonlinear Programming*, May 2008. Presentation at the SIAM Conference on Optimization.

[107] A. Wächter and L.T. Biegler, *On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming*, Mathematical Programming, **106** (2006), pp. 25–57.

[108] R. Waltz, *Current challenges in nonlinear optimization*, 2007. Presentation at San Diego Supercomputer Center: CIEG Spring Orientation Workshop, available at www.sdsc.edu/us/training/workshops/2007sac_studentworkshop/docs/SDSC07.ppt.

[109] T. WESTERLUND, H.I. HARJUNKOSKI, AND R. PÖRN, *An extended cutting plane method for solving a class of non-convex minlp problems*, Computers and Chemical Engineering, **22** (1998), pp. 357–365.

[110] T. WESTERLUND AND K. LUNDQVIST, *Alpha-ECP, version 5.101. an interactive minlp-solver based on the extended cutting plane method*, in Updated version of Report 01-178-A, Process Design Laboratory, Abo Akademi Univeristy, 2005.

[111] T. WESTERLUND AND F. PETTERSSON, *A cutting plane method for solving convex MINLP problems*, Computers and Chemical Engineering, **19** (1995), pp. s131–s136.

[112] T. WESTERLUND AND R. PÖRN, *. a cutting plane method for minimizing pseudo-convex functions in the mixed integer case*, Computers and Chemical Engineering, **24** (2000), pp. 2655–2665.

[113] L.A. WOLSEY, *Integer Programming*, John Wiley and Sons, New York, 1998.

[114] Y. ZHU AND T. KUNO, *A disjunctive cutting-plane-based branch-and-cut algorithm for 0-1 mixed-integer convex nonlinear programs*, Industrial and Engineering Chemistry Research, **45** (2006), pp. 187–196.