Ann Oper Res (2013) 204:301–320
DOI 10.1007/s10479-012-1272-5

# Mixed integer nonlinear programming tools: an updated practical overview

**Claudia D'Ambrosio · Andrea Lodi**

Published online: 12 January 2013


**Abstract** We present a review of available tools for solving mixed integer nonlinear programming problems. Our aim is to give the reader a flavor of the difficulties one could face and to discuss the tools one could use to try to overcome such difficulties.

## 1 Introduction

Despite the fact that Jeroslow (1973) proved that mixed integer nonlinear programming (MINLP) is undecidable, in recent years there has been a renewed interest in practically solving MINLP problems. Indeed, under the often reasonable assumption of boundedness of integer variables, it is well-known that MINLP problems are $\mathcal{NP}$-hard because they are the generalization of mixed integer linear programming (MILP) problems, which are $\mathcal{NP}$-hard themselves. Such a renewed interest can be explained by different considerations: (i) the large improvement on the performance of solvers handling nonlinear constraints; (ii) the increased awareness that many real-world applications can be modeled as MINLP problems; and (iii) the challenging nature of this very general class of problems. Indeed, many real-world optimization problems can be cast in this interesting class of problems, such as the ones arising from portfolio optimization, design of water distribution networks, design of complex distillation systems, optimization of manufacturing, cutting-stock problems, and transportation logistics, just to mention a few.

The (general) mixed integer nonlinear programming problem, which we are interested in, has the following form:

This is an updated version of the paper that appeared in *4OR*, 9(4), 329–349 (2011).

C. D'Ambrosio
CNRS LIX, Ecole Polytechnique, 91128 Palaiseau, France
e-mail: dambrosio@lix.polytechnique.fr

A. Lodi (✉)
DEI, University of Bologna, viale Risorgimento 2, 40136 Bologna, Italy
e-mail: andrea.lodi@unibo.it

MINLP

$$\min f(x, y) \tag{1}$$
$$g(x, y) \leq 0 \tag{2}$$
$$x \in X \cap \mathbb{Z}^n \tag{3}$$
$$y \in Y, \tag{4}$$

where $f : \mathbb{R}^{n\times p} \to \mathbb{R}$, $g : \mathbb{R}^{n\times p} \to \mathbb{R}^m$, and $X$ and $Y$ are two polyhedra of appropriate dimension (including bounds on the variables). We assume that $f$ and $g$ are twice continuously differentiable, but we do not make any further assumptions on the characteristics of these functions or their convexity/concavity. In the following, we will call problems of this type nonconvex mixed integer nonlinear programming problems.

The most complex aspect one needs to have in mind when working with nonconvex MINLPs is that their continuous relaxation, i.e., the problem obtained by relaxing the integrality requirement on the $x$ variables, might have (and usually has) local optima, i.e., solutions that are optimal within a restricted part of the feasible region (neighborhood), but are not optimal with respect to the entire feasible region. This does not happen when $f$ and $g$ are convex (or linear): in such cases the local optima are also global optima.

*Contribution of the paper* The aim of this paper is to present tools for "practically" solving MINLPs, i.e., finding a high quality feasible solution, neglecting the difference between local and global solutions. In particular, the focus is not on globally solving a nonconvex MINLP, i.e., unless otherwise stated, when we use the term "solving" in the following we do not intend exactly (globally). Note that this is a "reasonable" setting for real-world applications, especially those involving nonconvex constraints. On the other hand, proving global optimality for nonconvex MINLPs is, with some extent, possible and we briefly overview the most commonly used methods, refereed to as *Global Optimization* algorithms. It is easy to see that global optimality comes at the price of (much) more computationally challenging and expensive algorithmic approaches that result to be impractical for large-scale problems.

In Sect. 2 we point out some difficulties arising from handling nonlinear functions within an optimization model and how modeling languages can help to overcome some of them. We then review the solvers for special classes of MINLP: mixed integer linear programming in Sect. 3 and nonlinear programming (NLP) in Sect. 4. In these two cases, only aspects closely related to MINLP problems are presented, but references to books and papers on MILP and NLP are provided to the interested reader. On the other hand, the main focus is on MINLP methods (Sect. 5) and solvers (Sect. 6). We devote Sect. 7 to the description of "NEOS, server for optimization" NEOS. Conclusions follow in Sect. 8.

The first version of this survey (D'Ambrosio and Lodi 2011) has been published in the international journal 4OR in 2011.

## 2 Modeling languages

Modeling languages are used to express optimization models (linear programming—LP—, NLP, MILP, MINLP) with an intuitive syntax avoiding the need for the user to implement software using programming languages. They translate a model from a form that is easy to read for the user to a form readable by the solvers. The most intuitive modeling languages are the Algebraic Modeling Languages (AML), widely used in (mixed integer) nonlinear

programming. Their syntax is similar to the mathematical notation, which makes the resulting models easy to read for the user. It is also flexible, allowing the user to implement complicated algorithms within the modeling language framework, and compact, thanks to the use of abstract entities like sets and indices, making also large instances easy to write. Another important feature of AMLs is the nonlinear interpreter: using symbolic and/or automatic differentiation, it provides the NLP solver the information about the evaluation of functions, and the first and second derivatives at a given point. This feature makes AMLs a fundamental tool for (mixed integer) nonlinear programming because the most widely used methods for solving such problems need this information.

The basic functionalities of AMLs are: (a) reading the model and the data provided by the user; (b) expanding this compact problem formulation into the problem instantiation (expand the indexes, substitute the parameters, etc.); (c) providing the solver an appropriate form of the problem instantiation; (d) providing, when needed, the value of the function, its first and second derivative at a point given by the solver; and (e) reading the solution provided by the solver and translate it in a form readable by the user.

Note that AMLs do not have solver capabilities and use external solvers as black-boxes, but provide standard interfaces to the most common solvers. Moreover, they usually provide effective preprocessing techniques to detect trivial infeasibility and simplify the models. The two most widely used algebraic modeling languages in MINLP are AMPL (Fourer et al. 2003) and GAMS (Brooke et al. 1992). They are commercial products, but an evaluation version of both modeling languages is available for students (allowing a maximum number of constraints and variables equal to 300).

## 3 Mixed integer linear programming technology

In this section we present considerations about MILP solvers, limited to aspects related to MINLP.[1]

Until recent years, the standard approach for handling MINLP problems (especially by scientists with a combinatorial optimization background) has basically been to solve an MILP "approximation" of it. Note that, solving an MILP approximation of an MINLP does not necessarily mean solving a relaxation of it. The optimal solution of the MILP might be neither optimal nor feasible for the original problem, if no assumption is made on the MINLP structure.

Standard techniques to approximate special classes of MINLP problems with an MILP model are, for example, piecewise linear approximation (when nonlinear functions are univariate) and, in this case, Special Ordered Sets (SOS) of type 2 can be defined to model the convex combinations that form the piecewise linear approximation. By defining a set of variables to be a *special ordered set of type 2* (SOS2), one imposes that at most 2 such variables can take a nonzero value, and that they must be adjacent (see, e.g., Beale and Tomlin 1970). In some of the available MILP solvers, the definition of special ordered sets of types 1 and 2 is recognized and these sets of variables are treated through special branching rules. However, it is possible to use any kind of MILP solver and to define a piecewise linear function by adding suitable binary variables.

In presence of special properties of the approximated functions, (even) a standard piecewise linear approximation might result in a relaxation of the original problem. However,

[1] Note that we do not consider MINLPs that can be exactly reformulated as MILP problems.

this is in general not the case. Alternatively, methods tailored for solving MINLPs can be employed, which consist in solving MILP relaxations of the MINLP problem. Algorithms in this category are, for example, outer approximation, generalized Benders decomposition and extended cutting plane (see Sect. 5 and Grossmann 2002). In these cases an MILP is obtained starting from the original MINLP, but the approximation is constructed on purpose to provide a relaxation, and it is tightened runtime.

From a software viewpoint, another important role MILP plays in the solution of MINLP problems is that some MINLP techniques use MILP solvers as black-boxes to address carefully defined MILP subproblems. These techniques are described as well in Sect. 5.

We conclude this section by noting that in the MILP case there are at least a couple of standard file formats widely used by the MILP solvers, namely, `MPS` and `LP` files. No fully accepted file format is available for models in which nonlinear functions are involved, although the AMPL `NL` file format has recently become quite standard. We will further discuss this issue in Sects. 4 and 6.

Finally, examples of MILP solvers are the commercial ones `GUROBI`, (GUROBI), `IBM-CPLEX`, (IBM-CPLEX), `XPRESSOptimizer`, (XPRESS), and the open-source (or partially open-source) solvers `CBC` (CBC) and `SCIP` (SCIP). For a survey of the development of methods and software for MILPs the reader is referred, e.g., to Lodi (2009), and, for detailed discussions, to Achterberg (2007) and Linderoth and Lodi (2011).

## 4 Nonlinear programming technology

Nonlinear programming subproblems play a fundamental role in the solution of MINLPs. Different approaches to the solution of NLPs are possible. Thus, different types of solvers are available, both commercial and open-source.

The most commonly used and effective NLP solvers need the first and usually second derivatives of the nonlinear functions involved in the model. Thus, the nonlinear functions are assumed to be smooth. The necessity of derivatives is also the cause of the lack of a fully accepted file format to express NLP/MINLP problems. Providing the first and, possibly, the second derivative at a given point is not straightforward for non-expert users and can easily be a source of errors. The use of modeling languages helps to partially overcome such difficulties. However, the information about derivatives sometimes cannot be computed. In this case, derivative free optimization methods can be applied. For details on this topic, the reader is referred to the book by Conn et al. (2008).

An additional aspect that NLP solvers have in common is that they only guarantee that the solution provided is a local optimum. Thus, when we use standard NLP solvers, we might not obtain the globally optimal solution. This is why some MINLP solvers are exact algorithms for convex MINLPs (where local optima are also global optima) but only heuristics for nonconvex MINLPs. We will discuss this issue in detail in Sect. 5.

An alternative would be solving NLP relaxations of (nonconvex) MINLPs to optimality by means of global optimization algorithms. Of course that would be at the cost of the solver's efficiency, and, for this reason, global optimization techniques are usually not used in MINLP solvers for handling the NLP subproblems.

Any serious discussion about NLP algorithms would lead us outside the scope of the present paper. We refer the reader to Nocedal and Wright (2006), where, at the end of each chapter describing an NLP method, a discussion about software is also provided. Examples of the most widely used solvers are `CONOPT`, `FILTERSQP`, `IPOPT`, `KNITRO`, `MINOS`, `MOSEK`, `NPSOL`, `SNOPT`. For an overview of the NLP software, the reader is referred to Leyffer and Mahajan (2011).

## 5 Mixed integer nonlinear programming

In this section we first describe the most commonly used algorithms for solving convex MINLP problems, and then introduce some specific considerations on nonconvex MINLPs. Although, as mentioned in the Introduction, the focus of the paper is on tools proving local solutions for MINLPs, nevertheless if a problem is recognized to have a convex NLP relaxation, then special methods can be exploited and, in addition, these methods provide local solutions for nonconvex MINLPs as well.

We do not describe special-purpose methods, such as, for example, those for Semi-Definite programming, or mixed integer Quadratic programming, but only general-purpose MINLP techniques. The interested reader is referred to (among many possible references) the recent MINLP Software survey in the Wiley Encyclopedia of Operations Research and Management Science (Bussieck and Vigerske 2011).

For a collection of some of the most exciting latest advances on MINLP, we refer the reader to the recent book (Lee and Leyffer 2012).

### 5.1 Convex MINLP

Algorithms studied for convex MINLPs basically differ on the way subproblems, namely MILPs and NLPs, are defined and used. Unless otherwise stated, we assume that those subproblems are solved to optimality, which is reasonable in this setting for NLPs because of the convexity assumption. In the following, we briefly present some of the most commonly used algorithms and refer the reader to the paper by Grossmann (2002) for an exhaustive discussion. At the end of the section, we summarize the different characteristics of the presented algorithms in Table 1.

- **Branch-and-Bound** (BB): Originally proposed for MILP problems (see Land and Doig 1960), it was first applied to convex MINLPs by Dakin (1965) and Gupta and Ravindran (1985). The first step solves the continuous relaxation of the MINLP, NLP$^0$. Then, given a fractional value $x_j^*$ in the solution of the continuous relaxation $(x^*, y^*)$, the problem is divided into two subproblems, by alternatively adding the constraint $x_j \leq \lfloor x_j^* \rfloor$ or the constraint $x_j \geq \lfloor x_j^* \rfloor + 1$. Each of these new constraints represents a "branching decision" (imposed by simply changing the bounds on the variables) because the partition of the problem into subproblems is represented with a tree structure, the BB tree. The process is iterated for each node, until the solution of the continuous relaxation of the subproblem is integer feasible, or is infeasible, or the lower bound value associated with the subproblem is not smaller than the value of the current incumbent solution, i.e., the best feasible solution encountered so far. In these three cases, the node is fathomed. The algorithm stops when no node is left to explore, returning the incumbent solution, which is proven to be optimal. The basic difference between the BB for MILPs and for MINLPs is that, at each node, the subproblem solved is an LP in the former case, and an NLP in the latter. We do not discuss specific approaches for branching variable selection, tree exploration strategy, etc. The reader is referred to Abhishek (2008), Bonami et al. (2008), Gupta and Ravindran (1985), Leyffer (1999) for details.
- **Outer-Approximation** (OA): Proposed by Duran and Grossman (1986), it exploits the outer approximation linearization technique. Namely, given a point $(x^*, y^*)$, we can derive the OA cut

$$g_j(x^*, y^*) + \nabla g_j(x^*, y^*)^T \begin{pmatrix} x - x^* \\ y - y^* \end{pmatrix} \leq 0.$$

Note that, by convexity, the OA cut is "safe", i.e., it does not cut off any solution of the MINLP. OA is an iterative method in which, at each iteration $k$, two subproblems are solved:

- $\underline{\text{NLP}}_x^k$, an NLP defined by fixing the integer variables to the values $x^{k-1}$[2]; and
- $\underline{\text{MILP}}^k$, an MILP relaxation of the original MINLP obtained by the OA cuts generated so far from the solutions of $\underline{\text{NLP}}_x^{k'}$ with $k' \leq k$.

The former subproblem, if feasible, gives an upper bound on the solution of the MINLP, while the latter always provides a lower bound. If the $\underline{\text{NLP}}_x^k$ subproblem is not feasible, a "feasibility NLP subproblem" is solved and its optimal solution $(x^*, y^*)$ is used to derive the new OA cuts for $\underline{\text{MILP}}^k$. The feasibility NLP subproblem is defined as $\min\{u \mid g(x^{k-1}, y) \leq u, y \in Y, u \geq 0\}$ where $u$ is an additional variable that represents the maximum constraints violation, which is minimized. At each iteration lower and upper bounds might be improved. In particular, the definition of $\underline{\text{NLP}}_x^k$ changes because the $x$ vector changes. Moreover, the definition of $\underline{\text{MILP}}^k$ changes because of the addition, at each iteration, of OA cuts, which make the solution of the previous iteration infeasible. The algorithm ends when the gap between the two bounds is within a fixed tolerance.

– **Generalized Benders Decomposition** (GBD): Proposed by Geoffrion (1972), this is the extension to MINLP of the famous work of Benders (1962) for MILP. The basic idea of this iterative method is to decompose, at each iteration, the MINLP into: (i) a simpler NLP subproblem obtained by fixing the $x$ variables; and (ii) a (MILP) master problem that is the relaxation of the projection of the MINLP on the $x$-space. In the spirit of OA cuts, if the NLP is feasible, a so-called optimality cut (obtained projecting on the $x$-space the weighted combination of the linearizations of objective function and active constraints) is added to the master problem. Otherwise, a so-called feasibility cut (obtained projecting on the $x$-space the weighted combination of the linearizations of active constraints) is added to it. This method is strongly related to the OA method, the unique difference being the form of the $\underline{\text{MILP}}^k$ subproblem. The $\underline{\text{MILP}}^k$ of the GBD method is a surrogate relaxation of the one of the OA method. Thus, the lower bound given by the OA is stronger than (i.e., greater than or equal to) the one given by the GBD (for details, see Duran and Grossmann 1986). Even if, on average, the GBD method requires more iterations than the OA method, the fact that $\underline{\text{MILP}}^k$ subproblems of the former are more "tractable" (smaller) can make sometimes convenient using GBD instead of OA.

– **Extended Cutting Plane** (ECP): Introduced by Westerlund and Pettersson (1995), the method is based on the iterative resolution of an $\underline{\text{MILP}}^k$ subproblem and, when its solution $(x^k, y^k)$ is infeasible for the $\underline{\text{MINLP}}$, on the determination of the most violated constraint (or constraints) $J^k = \{j^* \in \arg\{\max\{g_j(x^k, y^k), \forall j = 1, \ldots, m\}\}\}$. The linearization of such a constraint is added to the next $\underline{\text{MILP}}^k$. The produced lower bound increases at each iteration, but generally a large number of iterations is needed to reach the optimal solution. Obviously, whenever $(x^k, y^k)$ becomes feasible for $\underline{\text{MINLP}}$, such solution is optimal and the algorithm terminates. An extension to pseudo-convex MINLP problems[3] has been presented.

– **LP/NLP based Branch-and-Bound** (QG): Introduced by Quesada and Grossmann (1992), the method can be seen as the extension of the Branch-and-Cut approach to convex MINLPs. The idea is to use BB to solve the $\underline{\text{MILP}}^k$ obtained from the solution $(x^0, y^0)$

[2] If $k = 1$, no fixing is performed.

[3] A pseudo-convex MINLP is an MINLP involving pseudo-convex functions. Informally, a function is pseudo-convex if behaves like a convex function with respect to finding its local minima, but need not actually be convex, see, e.g., Mangasarian (1965).

**Table 1** Number of MILP and NLP subproblems solved by each algorithm of Sect. 5.1.

| | # MILP | # NLP | Note |
|---|---|---|---|
| BB | 0 | # nodes | |
| OA | # iterations | # iterations | |
| GBD | # iterations | # iterations | weaker lower bound w.r.t. OA |
| ECP | # iterations | 0 | |
| QG | 1 | 1 + # explored MILP solutions | |
| Hyb (Abhishek et al. 2010) | 1 | 1 + # explored MILP solutions | stronger lower bound w.r.t. QG |
| Hyb (Bonami et al. 2008) | 1 | [# explored MILP solutions,# nodes] | |

of the NLP relaxation. Whenever an integer feasible solution $x$ of $\underline{\text{MILP}}^k$ is found, an $\underline{\text{NLP}}^k_x$ subproblem is solved obtaining $(x^k, y^k)$. OA cuts generated from $(x^k, y^k)$ are added to all the open nodes of the current BB tree. QG algorithm differs from ECP because at some nodes NLP subproblems are solved, from BB because they are not solved at each node.

– **Hybrid algorithms** (Hyb): Proposed by Bonami et al. (2008) and Abhishek et al. (2010), they are enhanced versions of the QG algorithm. They are called hybrid because they combine BB and OA methods.

  The version proposed by Bonami et al. (2008) differs from the QG algorithm because: (i) at "some" nodes (not only when an integer solution is found like in QG) the $\underline{\text{NLP}}^k_x$ subproblem is solved to generate OA cuts (like in BB); and (ii) local search is performed at some nodes of the tree by partial enumeration of the associated MILP relaxations. This is done to improve the bounds and collect OA cuts early in the tree. If the local enumeration is not limited, the algorithm reduces to OA, while when the $\underline{\text{NLP}}^k_x$ is solved at each node, it reduces to BB.

  The version proposed by Abhishek et al. (2010) differs from the one proposed by Bonami et al. (2008) in the way linearizations are added and how these linearizations are managed. In particular, NLP subproblems are solved only when an integer MILP feasible solution is found (like in the original QG algorithm), but linearization cuts are added runtime also at other nodes (with fractional $x$'s). When no linearization cut is added at any other nodes, the algorithm reduces to QG, while when linearization cuts are added at each node, it reduces to ECP.

As anticipated, Table 1 summarizes the differences in the way the algorithms presented above address convex MINLPs. Specifically, the table gives an estimation of the number of MILP and NLP subproblems solved by each method, and summarizes some of the discussed relationships between pair of algorithms.

We end the section by briefly mentioning that an important role in the practical effectiveness of the discussed algorithms is played by (primal) heuristics, i.e., algorithms aimed at improving the incumbent solution (upper bound) within the various phases of the enumerative schemes. Part of the heuristic algorithms studied for MILPs have been adapted to convex MINLP problems. For details about primal heuristics the reader is referred, e.g., to Abhishek (2008), Bonami et al. (2009), Bonami and Gonçalves (2008).

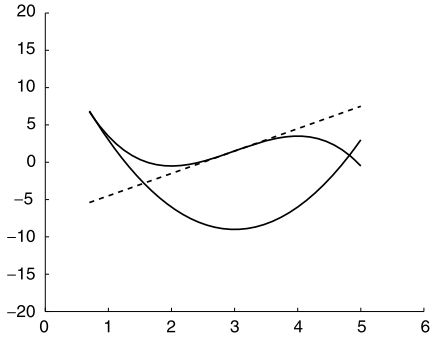


**Fig. 1** Example of "unsafe" linearization cut generated from a nonconvex constraint

### 5.2 Nonconvex MINLP

Coming back to the first model seen in this paper, MINLP, if we do not make any convexity assumption on the objective function and the constraints, then one of the main issues is that the NLP problems obtained by dropping the integrality requirements have, in general, local minima which are not global. This implies, that, if the NLP solver used to solve the NLP subproblems does not guarantee that the solution provided is a global optimum (and this is usually the case for the main NLP solvers, e.g., those discussed in Sect. 4), feasible and even optimal solutions might be fathomed, for example, by comparison with an invalid lower bound. As a result, methods like BB, QG, and Hyb that are exact for convex MINLPs work as heuristics for nonconvex MINLPs.

A second issue involves methods like OA, GBD, ECP, QG, and Hyb: the linearization cuts used by these algorithms are in general not valid for nonconvex constraints. It follows that the linearization cuts might cut off not only infeasible points, but also parts of the feasible region as depicted in Fig. 1. Adding the cut represented by the dashed line in the figure would result in cutting off the upper part of the nonconvex feasible region, i.e., that above the cut. For this reason, if nonconvex constraints are involved, one has to carefully use linearization cuts or accept that the final solution might be *not* the optimal one.

As for the methods for specifically handle nonconvexities, the first approach is to *fully* reformulate the problem. The exact reformulation is possible only in limited cases of nonconvex MINLPs and allows one to obtain an equivalent convex formulation of the nonconvex MINLP in the sense that: (i) all feasible pairs $(x, y) \in \mathbb{Z}^n \times \mathbb{R}^p$ are mapped to unique and feasible points of the feasible region of the convex formulation; and (ii) no other mixed integer (infeasible) solution $(x, y) \in \mathbb{Z}^n \times \mathbb{R}^p$ belongs to it. All the techniques described in Sect. 5.1 can then be (safely) applied to the reformulated MINLP. For a detailed description of exact reformulations to standard forms, see, for example, Liberti et al. (2009a).

The second approach, which applies to a larger subset of nonconvex MINLPs, is based on the use of convex envelopes or underestimators of the nonconvex feasible region. The relaxation of the original problem has the form:

rMINLP

$$\min z \tag{5}$$

$$\underline{f}(x, y) \leq z \tag{6}$$

$$\underline{g}(x, y) \leq 0 \tag{7}$$

$$x \in X \cap \mathbb{Z}^n \tag{8}$$

$$y \in Y, \tag{9}$$

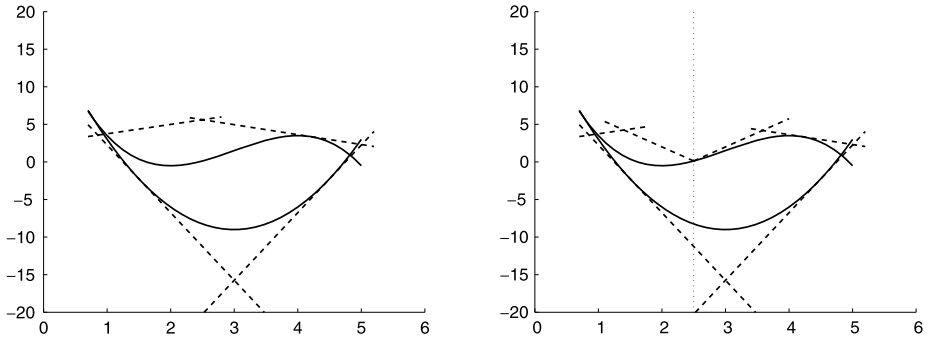


**Fig. 2** Linear underestimators before and after branching on continuous variables

where $z$ is an auxiliary variable introduced as a standard modeling trick to have a linear objective function, see (5) and (6). Moreover, $\underline{f} : \mathbb{R}^{n\times p} \to \mathbb{R}$ and $\underline{g} : \mathbb{R}^{n\times p} \to \mathbb{R}^m$ are convex (in some cases, linear) functions such that $\underline{f}(x, y) \le f(x, y)$ and $\underline{g}(x, y) \le g(x, y)$ within the $(x, y)$ feasible space.

Roughly speaking, the first step to reformulate the nonconvex problem MINLP to a "tractable" form like the mathematical programming problem (5)–(9) is to associate a convex relaxation with each *part* of the original model. This is obtained by adding auxiliary variables that "*linearize*" the nonlinearities of the original constraints. This way, complex nonlinearities are subdivided into simpler nonlinearities, at the cost of additional variables and constraints. The basic nonlinearities are then relaxed using simple envelopes/underestimators. This type of reformulation by pieces only applies to *factorable* functions, i.e., functions that can be expressed as summations and products of univariate functions after appropriate recursive splitting. More precisely, a function $f$ is factorable if it can be rewritten as:

$$f = \sum_h \prod_k f_{hk}(x),$$

where $f_{hk}(x)$ are univariate functions for all $h, k$. These functions can be reduced and reformulated into simpler nonlinear functions, adding auxiliary variables and constraints such as $s_{hk} = f_{hk}(x)$, $s'_h = \prod_k s_{hk}$, $s''_k = \sum_h s'_h$. This modeling trick allows reformulating the model through predetermined operators for which convex underestimators are known, such as, for example, bilinear, trilinear, or fractional terms (see, e.g., Liberti 2006; McCormick 1976; Smith and Pantelides 1999).

The use of underestimators makes the feasible region larger: If the optimal solution of rMINLP is feasible for the nonconvex MINLP, then it is also its global optimum. Otherwise, a refinement of the underestimation is needed. This is done by branching, not only on integer variables but also on continuous ones as depicted in Fig. 2. This technique allows one to have a lower bound on the nonconvex MINLP optimum that can be used within an algorithm, like branch-and-bound specialized versions for global optimization, namely *spatial* branch-and-bound (Smith and Pantelides 1999; Leyffer 2001; Liberti 2004a; Belotti et al. 2009), branch-and-reduce (Ryoo and Sahinidis 1996; Sahinidis 1996; Tawarmalani and Sahinidis 2004), $\alpha$-BB (Androulakis et al. 1995; Adjiman et al. 1997), Branch-and-Cut (Kesavan and Barto 2000; Nowak and Vigerske 2008). Such algorithms provide a solution that is guaranteed to be a global optimum. The above specialized branch-and-bound methods for global optimization mainly differ on the branching scheme adopted, namely,

they might either concentrate first on discrete variables (until an integer feasible solution is found), then branch on continuous variables, or branch on both variable types without a prefixed priority. From the algorithmic viewpoint, one could also branch on continuous variables appearing in nonconvex terms first (see, e.g., Adjiman et al. 2000 for details), but, this option is not available in the solvers, at least those we consider in Sect. 6.

It is clear that algorithms of this type are very time-consuming in general and can be employed only for small- to medium-size instances. This is the price one has to pay for the guarantee of the global optimality of the solution provided (of course, within a fixed tolerance).

From an implementation viewpoint, some complex structures are needed for implementing global optimization algorithms, such as the so-called symbolic expression trees (see, Smith and Pantelides 1999 for their use in (global) optimization). For a detailed description of the issues arising in developing and implementing a global optimization algorithm the reader is referred to Liberti (2006). The documentation of the specific solvers is also very useful. For an overview of the way to define functions $\underline{f}(x, y)$ and $\underline{g}(x, y)$ for nonconvex functions $f$ and $g$ with specific structure the reader is referred to Liberti (2004a, 2004b), McCormick (1976), Nowak (2005). Finally, it is worth mentioning that some of the primal heuristic algorithms developed for convex MINLPs have been generalized to the nonconvex setting by specifically taking into account the nonconvexity issues described above. Actually, this is a very active research topic on which several contributions are on the way, see, e.g., Liberti et al. (2009b), D'Ambrosio et al. (2010), D'Ambrosio (2010), Nannicini and Belotti (2011), to mention a few.

## 6 Mixed integer nonlinear programming solvers

This section is devoted to MINLP solvers. For each solver, we give a brief summary of its characteristics (often taken from manuals, web pages, and experience) and a list containing the following information:

- **Implemented algorithms**: the algorithm implemented within the solver. Examples of such algorithms are branch-and-bound (either standard or spatial), outer approximation, branch-and-cut (see Sect. 5). If present, special tools, aimed at speeding-up the solver, are mentioned, such as, e.g., bound reduction techniques.
- **Type of algorithm**: type of solution (exact/heuristic) and assumptions on the MINLP problem (for example, convexity of the MINLP). Recall that exact algorithms for convex MINLPs can be used as (generally very inaccurate) heuristics for the nonconvex case. We will point out if the solver has specific options to mitigate the inaccuracy due to the nonconvexities.
- **Available interface**: how the user can interface with the solver. The most common possibilities are: (i) the solver is available as stand-alone application, which receives the description of the model as a file; (ii) the user can provide information about the MINLP problem to be solved by implementing some routines linked to the solver libraries; and (iii) the user can write the model under a modeling language environment which is directly linked to the solver and call the solver from the environment.

  Clearly, the lack of a fully accepted file format for NLPs impacts MINLPs as well. Some of the difficulties in expressing nonlinear functions are partially overcome by modeling languages (see Sect. 2). Moreover, a useful tool for the conversion of files within the most common file formats can be found at http://www.gamsworld.org/performance/paver/convert_submit.htm.

– **Open-source software**: "yes" if the software is open-source, i.e., the source code is available and can be modified by the user.
– **Programming language**: the programming language used to implement the software. This information is only given if the solver is open-source.
– **Authors**: the authors of the software, and the main published reference, if any.
– **Web site**: the web site of the solver where information and user manual can be found.
– **Dependencies**: the external software which the solver depends on if it uses (parts of) other software to exploit capabilities, such as, e.g., interfacing to specific environments, solving special classes of MINLPs (like NLP/MILP subproblems) or generating cuts. A notation like `XX`/`YY` indicates that the considered software relies either on software `XX` or on software `YY`, to solve a special class of subproblems.

As extensively discussed in Sect. 5, methods for MINLP are usually based on decomposition/relaxation of the problem into simpler subproblems, which are typically NLPs and MILPs. For this reason, fundamental ingredients of MINLP solvers are often reliable and fast NLP and MILP solvers. The improvements on such solvers highly influence general-purpose software for MINLPs, and the recent success of MINLP solvers in solving real-world applications is also due to them. The kind of problems MINLP solvers are able to handle depends, in turn, on the NLP and MILP solvers integrated as part of the solving phase. From a development viewpoint, the use of available NLP and MILP solvers makes the work of a developer easier, and allows one to concentrate on aspects and issues typical of MINLP problems, exploiting more stable techniques for NLP and MILP problems (see, e.g., Bonami et al. 2007).

Finally, the order in which the solvers are presented is alphabetic.

### 6.1 `ALPHA-ECP`

`ALPHA-ECP` is an MINLP solver based on the extended cutting plane method. The ECP method is an extension of Kelley's (1960) cutting plane algorithm which was originally given for convex NLP problems. In Westerlund and Pettersson (1995) the method was extended to convex MINLP problems, and in Westerlund et al. (1998) to MINLP problems with $f$ and $g$ pseudo-convex functions. Pseudo-convex functions are those that satisfy the following property: $\nabla h(x^*)(x - x^*) < 0$ if $h(x) < h(x^*)$. The method was further extended, and its current version converges to the global optimal solution for nonconvex MINLP problems having a pseudo-convex objective function and pseudo-convex inequality constraints. The method requires the solution of an MILP subproblem at each iteration. To speed up the process, one can abort the MILP enumeration with a non-optimal solution, generate cuts and iterate the process. However, to prove global optimality for the MINLP, the last MILP relaxation must be solved exactly. In the implementation of `ALPHA-ECP`, a commercial MILP solver, namely `IBM-CPLEX`, is used to solve the MILP subproblems. In summary:

– Implemented algorithm: extended cutting plane method.
– Type of algorithm: exact for MINLPs with a pseudo-convex objective function and pseudo-convex inequality constraints. For general nonconvex problems one can tune the software to be less aggressive in the cutting plane generation.
– Available interface: `GAMS`.
– Open-source software: no.
– Authors: T. Westerlund and T. Lastusilta (2002).
– Web site: http://www.gams.com/dd/docs/solvers/alphaecp.pdf.
– Dependencies: `GAMS`, `IBM-CPLEX`, optionally an NLP solver.

### 6.2 BARON

BARON (Branch-and-Reduce Optimization Navigator) is a computational system for the solution of nonconvex optimization problems to global optimality. The solver combines branch-and-bound techniques specialized for nonconvex MINLPs with bound reduction based on both feasibility and optimality reasoning, which improve its performance. It is currently one of the most effective solvers in global optimization. In summary:

- Implemented algorithm: branch-and-reduce.
- Type of algorithm: exact.
- Available interface: AIMMS and GAMS.
- Open-source software: no.
- Authors: M. Tawarmalani and N.V. Sahinidis (Ref. Sahinidis 1996).
- Web site: http://archimedes.cheme.cmu.edu/baron/baron.html.
- Dependencies: IBM-CPLEX, either MINOS or SNOPT.

### 6.3 BONMIN

BONMIN (Basic Open-source Nonlinear Mixed INteger programming) is an open-source code for solving MINLP problems. It is distributed within COIN-OR (http://www.coin-or.org) under Common Public License (CPL). There are several algorithmic options that can be selected within BONMIN. B-BB is an NLP-based branch-and-bound algorithm, B-OA is an outer-approximation decomposition algorithm, B-QG is an implementation of Quesada and Grossmann's branch-and-cut algorithm, and B-Hyb is a hybrid outer-approximation based branch-and-cut algorithm. Some of the algorithmic choices require the ability to solve MILP and NLP problems. The default solvers for them are the COIN-OR codes CBC and IPOPT, respectively. The four algorithms above are exact for convex MINLPs.

To solve (heuristically) a problem with nonconvex constraints, one should only use the Branch-and-Bound algorithm B-BB. Some options have been specifically designed in BONMIN to treat problems whose continuous relaxation is not convex. Specifically, in the context of nonconvex problems, the NLP solver may find different local optima when started from different starting points. Two options allow for solving the root node or each node of the tree with a user-specified number of different randomly-chosen starting points, saving the best solution found. The function generating a starting point chooses a random point (uniformly) between the variable bounds. Because the solution given by the NLP solver does not give a valid lower bound, another option of BONMIN allows the user to change the fathoming rule to continue branching even when the solution value of the current node is worse than the best-known solution. In summary:

- Implemented algorithms: Branch-and-Bound, Outer-Approximation, LP/NLP based Branch-and-Bound, Hybrid.
- Type of algorithm: exact for convex MINLPs, options for nonconvex MINLPs, see above.
- Available interface: AMPL, GAMS, stand-alone application (reading NL files), invocation through C/C++ program, MATLAB interface through OPTIToolBox.
- Open-source software: yes.
- Programming language: C++.
- Authors: P. Bonami et al. (see the web page) (2008).
- Web site: https://projects.coin-or.org/Bonmin.
- Dependencies: CBC/IBM-CPLEX, IPOPT/FILTERSQP (beside Blas, HSL/MUMPS, optionally ASL, Lapack).

### 6.4 COUENNE

COUENNE (Convex Over and Under ENvelopes for Nonlinear Estimation) is a spatial branch-and-bound algorithm to solve MINLP problems. COUENNE aims at finding global optima of nonconvex MINLPs. It implements linearization, bound reduction, and branching methods within a branch-and-bound framework. Its main components are: (i) an "expression library", i.e., a library of predetermined operators that can be used to reformulate general nonlinear functions; (ii) separation of linearization cuts; (iii) branching rules; and (iv) bound tightening methods. It is distributed within COIN-OR under CPL. In summary:

- Implemented algorithms: spatial branch-and-bound, bound reduction.
- Type of algorithm: exact.
- Available interface: AMPL, GAMS, stand-alone application (reading NL files), invocation through C/C++ program.
- Open-source software: yes.
- Programming language: C++.
- Authors: P. Belotti et al. (see the web page) (2009).
- Web site: https://projects.coin-or.org/Couenne.
- Dependencies: CBC, IPOPT (beside Blas, HSL/MUMPS, optionally ASL, Lapack).

### 6.5 DICOPT

DICOPT (DIscrete and Continuous OPTimizer) is a software for solving MINLP problems where nonlinearities involve only continuous variables. Binary and integer variables appear only in linear functions. The implementation is based on outer approximation, equality relaxation and augmented penalty, i.e., equality constraints are relaxed obtaining inequalities and adding a penalization factor in the objective function. The MINLP algorithm inside DICOPT solves a series of NLP and MILP subproblems. These subproblems can be solved using any NLP or MILP solver that runs under GAMS. The solver can handle nonconvexities, but in such case it does not necessarily obtain the global optimum. According to the web-page, the GAMS/DICOPT software has been designed with two main goals in mind: (i) to build on existing modeling concepts and to introduce a minimum of extensions to the existing modeling language; and (ii) to use existing optimizers to solve the subproblems. This concept favors an effective exploitation of any new development and enhancement in the NLP and MILP solvers. In summary:

- Implemented algorithms: outer approximation, equality relaxation, augmented penalty.
- Type of algorithm: exact for convex MINLPs with binary and integer variables appearing only in linear constraints.
- Available interface: GAMS.
- Open-source software: no.
- Authors: J. Viswanathan and I.E. Grossmann (1989).
- Web site: http://www.gams.com/dd/docs/solvers/dicopt.pdf.
- Dependencies: GAMS, IBM-CPLEX/XPRESS/XA, CONOPT/MINOS/SNOPT.

### 6.6 FILMINT

FILMINT is a solver for convex MINLPs. It is based on the LP/NLP branch-and-bound algorithm of Quesada and Grossmann (1992). FILMINT combines the MINTO branch-and-cut framework (Nemhauser et al. 1994) for MILP with FILTERSQP used to solve the nonlinear programs that arise as subproblems in the algorithm. The MINTO framework allows

one to employ cutting planes, primal heuristics, and other well-known MILP enhancements to the MINLP context. FILMINT offers techniques for generating and managing linearizations for a wide range of MINLPs. In summary:

- Implemented algorithm: LP/NLP based branch-and-bound.
- Type of algorithm: exact for convex MINLPs.
- Available interface: AMPL, stand-alone application (reading NL files).
- Open-source software: no.
- Authors: K. Abhishek, S. Leyffer, and J. Linderoth (2010).
- Web site: http://www.mcs.anl.gov/~leyffer/papers/fm.pdf.
- Dependencies: MINTO, FILTERSQP (beside ASL).

### 6.7 LAGO

LAGO (LAgrangian Global Optimizer) is a Branch-and-Cut algorithm to solve MINLPs. A linear outer approximation is constructed from a convex relaxation of the problem. Because LAGO does not have an algebraic representation of the problem, reformulation techniques for the construction of the convex relaxation cannot be applied, and sampling techniques are used in case of non-quadratic nonconvex functions. The linear relaxation is further improved by general purpose cutting planes. Bound reduction techniques are applied to improve efficiency. The algorithm assumes the existence of procedures for evaluating function values, gradients, and Hessians of the functions. According to the webpage, by relying on such black-box functions LAGO has the advantage of handling very general functions, but has the disadvantage that advanced reformulations and bound reduction techniques cannot be used. Hence, when sampling methods are applied, no deterministic global optimization is guaranteed. LAGO is distributed within COIN-OR under CPL.[4] In summary:

- Implemented algorithms: branch-and-cut (reformulation, relaxation, linear cut generator).
- Type of algorithm: exact for mixed integer quadratically constrained quadratic programming problems and convex MINLPs.
- Available interface: AMPL, GAMS, stand-alone application (reading NL files), invocation through C/C++ program.
- Open-source software: yes.
- Programming language: C++.
- Authors: I. Nowak and S. Vigerske (2008).
- Web site: https://projects.coin-or.org/LaGO.
- Dependencies: IPOPT, CGL, CLP, optionally IBM-CPLEX (beside ASL/GAMSIO, METIS, ranlib, TNT, optionally Lapack).

### 6.8 LINGO and LINDOGlobal

LINGO is an optimization suite that handles virtually any mathematical programming problem including MINLP. It implements branch-and-cut methods for global optimization but, in case a global solution is computationally too costly (or not 'needed'), LINGO solution method can be turned into a heuristic that uses special features like multistart techniques for solving NLPs. The LINGO package also provides an internal modeling language and Spreadsheet/Database capabilities. LINDOGlobal is the GAMS version of the suite, which comes with a model size limitation of 3,000 variables and 2,000 constraints. In summary:

[4]The development of LAGO is currently ceased but the software is open-source, so we prefer to keep it into the list because future development is possible.

– Implemented algorithm: branch-and-cut.
– Type of algorithm: exact.
– Available interface: GAMS.
– Open-source software: no.
– Authors: LINDO Systems.
– Web site: http://www.lindo.com/, and http://www.gams.com/solvers/solvers.htm#LINDOGLOBAL.
– Dependencies (for LINDOGlobal): GAMS, CONOPT, optionally MOSEK.

### 6.9 MINLPBB

MINLPBB is a package of FORTRAN 77 subroutines for finding solutions to MINLP problems. The package implements the branch-and-bound method in a nonlinear setting. The package must be used in conjunction with FILTERSQP and BQPD (Nocedal and Wright 2006) for solving NLPs. Problems are specified in the same way as for FILTERSQP (i.e., either via subroutines or via AMPL or CUTE Bongartz et al. 1995). The additional integer structure is specified using a vector to identify the indices of integer variables. The user can influence the choice of the branching variable by providing priorities. In summary:

– Implemented algorithm: branch-and-bound.
– Type of algorithm: exact for convex MINLPs.
– Available interface: AMPL, CUTE, stand-alone application (reading NL files), MATLAB through TOMLAB.
– Open-source software: no.
– Authors: S. Leyffer.
– Web site: http://www-unix.mcs.anl.gov/~leyffer/solvers.html.
– Dependencies: FILTERSQP, BQPD (beside CUTE/ASL).

### 6.10 MINOPT

MINOPT is a package that comprehends solvers for virtually any mathematical programming problem including MINLP. Specifically, MINOPT provides implementations of GBD and variants of OA. In addition, the MINOPT algorithmic framework provides a modeling language and has connections to a number of solvers. In summary:

– Implemented algorithms: generalized Benders decomposition, outer approximation, generalized cross decomposition.
– Type of algorithm: exact.
– Available interface: stand-alone application, MINOPT modeling language.
– Open-source software: no.
– Authors: C.A. Schweiger and C.A. Floudas (1998a, 1998b).
– Web site: http://titan.princeton.edu/MINOPT.
– Dependencies: IBM-CPLEX/LPSOLVE, MINOS/NPSOL/SNOPT.

### 6.11 MINOTAUR

MINOTAUR (Mixed Integer Nonlinear Optimization Toolkit: Algorithms, Underestimators, Relaxations) is an open-source toolkit for solving Mixed Integer Nonlinear Optimization Problems. It provides different solvers that implement state-of-the-art algorithms for MINLP. The Minotaur library can also be used to customize algorithms to exploit on specific problem structures. In summary:

- Implemented algorithms: NLP based Branch-and-Bound for convex MINLPs, Branch-and-Bound for global optimization of mixed integer quadratically constrained quadratic programming problems.
- Type of algorithm: exact.
- Available interface: stand-alone application, AMPL.
- Open-source software: yes.
- Programming language: C++.
- Authors: A. Mahajan, S. Leyffer, J. Linderoth, J. Luedtke, and T. Munson.
- Web site: http://wiki.mcs.anl.gov/minotaur/index.php/Main_Page.
- Dependencies: Lapack (and optionally CLP, IBM-CPLEX, FILTERSQP, IPOPT with MUMPS/HSL).

### 6.12 SBB

SBB is a GAMS solver for MINLP models. It is based on a combination of the standard branch-and-bound method for MILP and some of the standard NLP solvers already supported by GAMS. Currently, SBB can use CONOPT, MINOS, and SNOPT as solvers for subproblems. SBB supports all types of discrete variables supported by GAMS, including binary, integer, semicontinuous, semiinteger, SOS1, SOS2. In summary:

- Implemented algorithm: branch-and-bound.
- Type of algorithm: exact for convex MINLPs. In the nonconvex case one can limit the fathoming of infeasible nodes.
- Available interface: GAMS.
- Open-source software: no.
- Authors: ARKI Consulting and Development (Bussieck and Drud).
- Web site: http://www.gams.com/solvers/solvers.htm#SBB.
- Dependencies: GAMS, CONOPT/MINOS/SNOPT.

### 6.13 SCIP

SCIP (Solving Constraint Integer Programs) is a solver developed by the Optimization Department at the Zuse Institute Berlin and its collaborators. For academic institutions, it is available in source code and as standalone binary and is distributed within GAMS. SCIP ensures global optimal solutions for convex and nonconvex MINLPs. It implements a spatial branch-and-bound algorithm that utilizes LPs for bounding. Similar to BARON, the outer-approximation is generated from a reformulation of the MINLP. Note, that the support for non-quadratic nonlinear constraints is still in a BETA-stadium and not yet as robust as the rest of SCIP. In summary:

- Implemented algorithms: spatial Branch-and-Bound, bound tightening.
- Type of algorithm: exact.
- Available interface: GAMS, invocation through C program.
- Open-source software: yes.
- Programming language: C.
- Authors: S. Vigerske for the MINLP extension (Vigerske 2012; Achterberg 2007; Berthold et al. 2012).
- Web site: http://scip.zib.de/.
- Dependencies: SoPlex/IBM-CPLEX, CppAD (optionally IPOPT).

## 7 NEOS, server for optimization

Most of the solvers described in the previous sections can be freely accessed on the Internet through "NEOS (Network-Enabled Optimization System): Server for Optimization" (NEOS), maintained by the Optimization Technology Center of Northwestern University, the Argonne National Laboratory and the University of Wisconsin-Madison. A user can submit an optimization problem to the server and obtain the solution and running time statistics using the preferred solver through different interfaces. In particular, three ways to interact with the server are available:

1. Web Interface: from the web site the user can submit a problem, typically written with the `AMPL` or `GAMS` modeling languages, and obtain results and statistics via email.
2. Client Interface: the user can communicate with the server using a client implementation with different programming languages such as Python, Perl, PHP, C, C++, Java, Ruby (see NEOS and XML-RPC for details).
3. Kestrel Interface: the user can submit a problem to the server and use the results within a modeling environment. In the client machine, an executable file is called in order to directly communicate with the server within the `AMPL` or `GAMS` environment (see http://www-neos.mcs.anl.gov/neos/kestrel.html for details).

The chance to access different solvers is a great opportunity for the optimization community, both because a user can test the performance of different solvers and decide which one is most suitable for a specific model and because some of the available solvers are commercial solvers that cannot be used in other ways by users who do not own a license. The uniformity of the interfaces available for the different solvers makes switching from one solver to another easy, without any additional work. Moreover, it is intuitive to use, not requiring a deep expertise in optimization.

## 8 Conclusions

In this paper we have presented an overview of software for solving mixed integer nonlinear programming problems. We have introduced such problems and their theoretical and practical difficulties. Because the solution of MINLPs often requires the solution of NLP and/or MILP subproblems, we have briefly discussed the relationship among these problems and mentioned their specific methods. We have introduced the most important algorithms and the most commonly used solvers for MINLPs, pointing out their crucial characteristics.

For a reader interested in testing the developed algorithms on literature test instances, among the library of MINLP problems we cite the most widely used: CMU/IBM Library (http://egon.cheme.cmu.edu/ibm/page.htm), the new CMU-IBM Cyber-Infrastructure for MINLP collaborative site (http://www.minlp.org), the MacMINLP library maintained by Leyffer (http://www.mcs.anl.gov/~leyffer/macminlp/index.html), and the MINLPlib (http://www.gamsworld.org/minlp/minlplib.htm).

Finally, we point out that benchmark results on optimization solvers, which are beyond the scope of this paper, can be found at the web sites http://plato.asu.edu/bench.html and http://www.gamsworld.org.

**Acknowledgements** We are grateful to Silvano Martello for precious suggestions. Thanks are also due to Stefan Vigerske for useful comments and discussions.

## References


Abhishek, K. (2008). *Topics in mixed integer nonlinear programming*. Ph.D. thesis, Lehigh University.

Abhishek, K., Leyffer, S., & Linderoth, J. (2010). FilMINT: an outer-approximation-based solver for nonlinear mixed integer programs. *INFORMS Journal on Computing*, *22*, 555–567.

Achterberg, T. (2007). *Constraint integer programming*. Ph.D. thesis, Technische Universität Berlin.

Adjiman, C., Androulakis, I., & Floudas, C. (1997). Global optimization of MINLP problems in process synthesis and design. *Computers & Chemical Engineering*, *21*, 445–450.

Adjiman, C., Androulakis, I., & Floudas, C. (2000). Global optimization of mixed-integer nonlinear problems. *AIChE Journal*, *46*, 1769–1797.

Androulakis, I., Maranas, C., & Floudas, C. (1995). $\alpha$BB: a global optimization method for general constrained nonconvex problems. *Journal of Global Optimization*, *7*, 337–363.

Beale, E., & Tomlin, J. (1970). Special facilities in a general mathematical programming system for nonconvex problems using ordered sets of variables. In J. Lawrence (Ed.), *Proceedings of the Fifth International Conference on Operational Research: OR 69* (pp. 447–454). London: Tavistock.

Belotti, P., Lee, J., Liberti, L., Margot, F., & Wächter, A. (2009). Branching and bounds tightening techniques for non-convex MINLP. *Optimization Methods & Software*, *24*, 597–634.

Benders, J. (1962). Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, *4*, 267–299.

Berthold, T., Heinz, S., & Vigerske, S. (2012). Extending a CIP framework to solve MIQCPs. In J. Lee & S. Leyffer (Eds.), *IMA volumes in mathematics and its applications: Vol. 154. Mixed-integer nonlinear optimization: algorithmic advances and applications* (pp. 427–444). Berlin: Springer.

Bonami, P., & Gonçalves, J. (2008). *Primal heuristics for mixed integer nonlinear programs* (Tech. Rep.). IBM Research Report RC24639.

Bonami, P., Forrest, J., Lee, J., & Wächter, A. (2007). Rapid development of an MINLP solver with COIN-OR. *Optima*, *75*, 1–5.

Bonami, P., Biegler, L., Conn, A., Cornuéjols, G., Grossmann, I., Laird, C., Lee, J., Lodi, A., Margot, F., Sawaya, N., & Wächter, A. (2008). An algorithmic framework for convex mixed integer nonlinear programs. *Discrete Optimization*, *5*, 186–204.

Bonami, P., Cornuéjols, G., Lodi, A., & Margot, F. (2009). A feasibility pump for mixed integer nonlinear programs. *Mathematical Programming*, *119*, 331–352.

Bongartz, I., Conn, A. R., Gould, N., & Toint, P. L. (1995). CUTE: constrained and unconstrained testing environment. *ACM Transactions on Mathematical Software*, *21*, 123–160. doi:10.1145/200979.201043.

Brooke, A., Kendrick, D., & Meeraus, A. (1992). *GAMS: a user's guide*. URL citeseer.ist.psu.edu/brooke92gams.html.

Bussieck, M., & Drud, A. SSB: a new solver for mixed integer nonlinear programming. In *Recent advances in nonlinear mixed integer optimization*, INFORMS Fall, Invited talk.

Bussieck, M., & Vigerske, S. (2011). MINLP solver software. In J. Cochran (Ed.), *Wiley encyclopedia of operations research and management science*. New York: Wiley.

CBC. URL https://projects.coin-or.org/Cbc.

Conn, A., Scheinberg, K., & Vicente, L. (2008). *MPS/SIAM book series on optimization. Introduction to derivative free optimization*. Philadelphia: SIAM.

Dakin, R. (1965). A tree-search algorithm for mixed integer programming problems. *Computer Journal*, *8*(3), 250–255. doi:10.1093/comjnl/8.3.250. URL http://comjnl.oxfordjournals.org/content/8/3/250.abstract.

D'Ambrosio, C. (2010). Application-oriented mixed integer non-linear programming. *4OR*, *8*, 319–322.

D'Ambrosio, C., Frangioni, A., Liberti, L., & Lodi, A. (2010). *A storm of feasibility pumps for nonconvex MINLP* (Tech. Rep. OR/10/13). Università di Bologna. To appear in *Mathematical Programming*.

D'Ambrosio, C., & Lodi, A. (2011). Mixed integer non-linear programming tools: a practical overview. 4OR: A. *4OR*, *9*, 329–349.

Duran, M., & Grossmann, I. (1986). An outer-approximation algorithm for a class of mixed-integer nonlinear programs. *Mathematical Programming*, *36*, 307–339.

Fourer, R., Gay, D., & Kernighan, B. (2003). *AMPL: a modeling language for mathematical programming* (2nd ed.). Monterey: Duxbury Press/Brooks/Cole Publishing Co.

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

Grossmann, I. (2002). Review of nonlinear mixed-integer and disjunctive programming techniques. *Optimization and Engineering*, *3*, 227–252.

Gupta, O., & Ravindran, V. (1985). Branch and bound experiments in convex nonlinear integer programming. *Management Science*, *31*, 1533–1546.

GUROBI. URL http://www.gurobi.com/.

IBM-CPLEX. URL http://www-01.ibm.com/software/integration/optimization/cplex/. (v. 12.0).
Jeroslow, R. (1973). There cannot be any algorithm for integer programming with quadratic constraints. *Operations Research*, *21*, 221–224.
Kelley, J. E. Jr. (1960). The cutting-plane method for solving convex programs. *Journal of the Society for Industrial and Applied Mathematics*, *8*, 703–712.
Kesavan, P., & Barto, P. (2000). Generalized branch-and-cut framework for mixed-integer nonlinear optimization problems. *Computers & Chemical Engineering*, *24*, 1361–1366.
Kocis, G., & Grossmann, I. (1989). Computational experience with DICOPT solving MINLP problems in process systems engineering. *Computers & Chemical Engineering*, *13*, 307–315.
Land, A., & Doig, A. (1960). An automatic method of solving discrete programming problems. *Econometrica*, *28*(3), 497–520. URL http://www.jstor.org/stable/1910129.
Lee, J., & Leyffer, S. (Eds.) (2012). *IMA volumes in mathematics and its applications: Vol. 154. Mixed integer nonlinear programming*. Berlin: Springer.
Leyffer, S. (1999). *User manual for MINLP_BB* (Tech. Rep.). University of Dundee.
Leyffer, S. (2001). Integrating SQP and branch-and-bound for mixed integer nonlinear programming. *Computational Optimization and Applications*, *18*, 295–309.
Leyffer, S., & Mahajan, A. (2011). *Software for nonlinearly constrained optimization*. New York: Wiley.
Liberti, L. (2004a). *Reformulation and convex relaxation techniques for global optimization*. Ph.D. thesis, Imperial College, London, UK.
Liberti, L. (2004b). Reformulation and convex relaxation techniques for global optimization. *4OR*, *2*, 255–258.
Liberti, L. (2006). Writing global optimization software. In L. Liberti & N. Maculan (Eds.), *Global optimization: from theory to implementation* (pp. 211–262). Berlin: Springer.
Liberti, L., Cafieri, S., & Tarissan, F. (2009a). Reformulations in mathematical programming: a computational approach. In A. Abraham, A. Hassanien, & P. Siarry (Eds.), *Studies in computational intelligence: Vol. 203. Foundations on computational intelligence, vol. 3* (pp. 153–234). New York: Springer.
Liberti, L., Nannicini, G., & Mladenovic, N. (2009b). A good recipe for solving MINLPs. In V. Maniezzo, T. Stützle, & S. Voss (Eds.), *Annals of information systems: Vol. 10. MATHEURISTICS: hybridizing metaheuristics and mathematical programming* (pp. 231–244). Berlin: Springer.
Linderoth, J., & Lodi, A. (2011). MILP software. In J. Cochran (Ed.), *Wiley encyclopedia of operations research and management science* (Vol. 5, pp. 3239–3248). New York: Wiley.
Lodi, A. (2009). Mixed integer programming computation. In M. Jünger, T. Liebling, D. Naddef, G. Nemhauser, W. Pulleyblank, G. Reinelt, G. Rinaldi, & L. Wolsey (Eds.), *50 Years of integer programming 1958–2008: from the early years to the state-of-the-art* (pp. 619–645). Berlin: Springer.
Mangasarian, O. (1965). Pseudo-convex functions. *Journal of the Society for Industrial and Applied Mathematics*, *3*, 281–290.
McCormick, G. (1976). Computability of global solutions to factorable nonconvex programs: Part I—convex underestimating problems. *Mathematical Programming*, *10*, 147–175.
Nannicini, G., & Belotti, P. (2011). *Rounding based heuristics for nonconvex MINLPs* (Tech. Rep.). Tepper, School of Business, Carnegie Mellon University. March.
Nemhauser, G., Savelsbergh, M., & Sigismondi, G. (1994). MINTO, a mixed INTeger optimizer. *Operations Research Letters*, *15*, 47–585.
NEOS. URL www-neos.mcs.anl.gov/neos (v. 5.0).
Nocedal, J., & Wright, S. (2006). *Springer series in operations research. Numerical optimization*.
Nowak, I. (2005). *International series of numerical mathematics. Relaxation and decomposition methods for mixed integer nonlinear programming*. Berlin: Birkhäuser.
Nowak, I., & Vigerske, S. (2008). LaGO—a (heuristic) branch and cut algorithm for nonconvex MINLPs. *Central European Journal of Operations Research*, *16*, 127–138.
Quesada, I., & Grossmann, I. (1992). An LP/NLP based branch and bound algorithm for convex MINLP optimization problems. *Computers & Chemical Engineering*, *16*, 937–947.
Ryoo, H., & Sahinidis, N. (1996). A branch-and-reduce approach to global optimization. *Journal of Global Optimization*, *8*, 107–138.
Sahinidis, N. (1996). BARON: a general purpose global optimization software package. *Journal of Global Optimization*, *8*, 201–205.
Schweiger, C., & Floudas, C. (1998a). *MINOPT: a modeling language and algorithmic framework for linear, mixed-integer, nonlinear, dynamic, and mixed-integer nonlinear optimization*. Princeton: Princeton University Press.
Schweiger, C., & Floudas, C. (1998b). MINOPT: a software package for mixed-integer nonlinear optimization (3rd ed.).
SCIP. URL http://scip.zib.de/scip.shtml.

Smith, E., & Pantelides, C. (1999). A symbolic reformulation/spatial branch and bound algorithm for the global optimization of nonconvex MINLPs. *Computers & Chemical Engineering*, *23*, 457–478.

Tawarmalani, M., & Sahinidis, N. (2004). Global optimization of mixed-integer nonlinear programs: a theoretical and computational study. *Mathematical Programming*, *99*, 563–591.

Vigerske, S. (2012). *Decomposition in multistage stochastic programming and a constraint integer programming approach to mixed-integer nonlinear programming*. PhD Thesis, Humboldt-Universität zu Berlin.

Westerlund, T., & Pettersson, F. (1995). A cutting plane method for solving convex MINLP problems. *Computers & Chemical Engineering*, *19*, S131–S136.

Westerlund, T., & Pörn, R. (2002). Solving pseudo-convex mixed integer problems by cutting plane techniques. *Optimization and Engineering*, *3*, 253–280.

Westerlund, T., Skrifvars, H., Harjunkoski, I., & Pörn, R. (1998). An extended cutting plane method for solving a class of non-convex MINLP problems. *Computers & Chemical Engineering*, *22*, 357–365.

XML-RPC. URL http://www.xmlrpc.com.

XPRESS. URL http://www.fico.com/en/Products/DMTools/Pages/FICO-Xpress-Optimization-Suite.aspx.