Math. Prog. Comp. (2012) 4:1–31
DOI 10.1007/s12532-011-0032-x

FULL LENGTH PAPER

Rounding-based heuristics for nonconvex MINLPs

Giacomo Nannicini · Pietro Belotti

Received: 15 August 2010 / Accepted: 12 August 2011 / Published online: 1 September 2011

Abstract We propose two primal heuristics for nonconvex mixed-integer nonlinear programs. Both are based on the idea of rounding the solution of a continuous nonlinear program subject to linear constraints. Each rounding step is accomplished through the solution of a mixed-integer linear program. Our heuristics use the same algorithmic scheme, but they differ in the choice of the point to be rounded (which is feasible for nonlinear constraints but possibly fractional) and in the linear constraints. We propose a feasibility heuristic, that aims at finding an initial feasible solution, and an improvement heuristic, whose purpose is to search for an improved solution within the neighborhood of a given point. The neighborhood is defined through *local branching* cuts or box constraints. Computational results show the effectiveness in practice of these simple ideas, implemented within an open-source solver for nonconvex mixed-integer nonlinear programs.

Mathematics Subject Classification (2000) 90C11 · 90C57 · 90C59

1 Introduction

Mixed-integer nonlinear programming (MINLP) problems are a class of optimization problems whose objective and constraints are, in general, nonlinear, and such that a

G. Nannicini (✉)
Singapore University of Technology and Design, Singapore, Singapore
e-mail: nannicini@sutd.edu.sg

G. Nannicini
Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA, USA

P. Belotti
Department of Mathematical Sciences, Clemson University, Clemson, SC, USA
e-mail: pbelott@clemson.edu

subset of variables is constrained to be integer. These problems find application in several fields [4,14] and are very difficult to solve in practice. In this paper we focus on nonconvex MINLPs, that is, mathematical programs which involve both integer and continuous variables, and where the objective function and constraints can be nonconvex. Note that when we write nonconvex MINLPs, we simply indicate that the constraints and objective function need not be convex. This is the most expressive class of single-objective mathematical programs, in that all single-objective deterministic optimization problems can be cast in this form. MINLPs generalize both nonlinear programs (NLPs) and mixed-integer linear programs (MILPs), and hence inherit difficulties from both classes. In recent years, some attempts have been made in developing exact solvers for nonconvex MINLPs [2,29]. These solvers are based on the branch-and-bound (BB) algorithm, where lower bounds (we assume a minimization problem) at each node of the BB tree are typically obtained by solving a linear program (LP) that defines a relaxation of the corresponding MINLP. Examples of LP-based BB solvers for nonconvex MINLPs are `Baron` [25], `Couenne` [2], and `Lindoglobal` [20].

For most MINLP solvers, obtaining good upper bounds as quickly as possible is of great practical importance. In this paper we study two heuristics for this purpose. Our primal heuristics are based on the same algorithmic scheme: first, we obtain a feasible point for the continuous relaxation by solving an NLP. Then we round the solution to this NLP, subject to linear constraints, to obtain an integral feasible point. This rounding is accomplished through the solution of an MILP. Finally, we fix the integer variables and act on the continuous variables to generate an MINLP feasible point.

We design two heuristics of this type in such a way that they are fast in practice: the first one has the purpose of obtaining an initial feasible solution, the second one acts on a feasible solution and tries to compute a better one, searching within a neighborhood of the starting point. The difference between the two lies in the definition of the NLPs that are solved to obtain the point to be rounded, and in the linear constraints to which the rounding phase is subject.

The rest of this paper is organized as follows. In Sect. 2 we provide the necessary notation and preliminaries. In Sect. 3 we introduce the basic scheme for our heuristics. Section 4 describes the linear inequalities called *no-good cuts* that are employed by our heuristics. Then, the basic scheme is specialized to define a feasibility heuristic that finds a first solution (Sect. 5), and an improvement heuristic (Sect. 6). Some issues related to dealing with a nonconvex optimization problem are discussed in Sect. 7. In Sect. 8 we provide computational experiments on a set of benchmark instances to show the practical usefulness of the heuristics that we propose. Section 9 concludes the paper. The appendix contains more details on the computational experiments.

2 Notation and Preliminaries

Consider the following mathematical program:

$$\left.\begin{array}{lrl} \min & & f(x) \\ \forall j \in M & & g_j(x) \leq 0 \\ \forall i \in N & x_i^L \leq & x_i \leq x_i^U \\ \forall i \in N_I & & x_i \in \mathbb{Z}, \end{array}\right\} \mathcal{P} \tag{1}$$

where f and all g_i's are possibly nonconvex functions, $N = \{1, \ldots, n\}$ is the set of variables, $M = \{1, \ldots, m\}$ the set of constraints, and $x = (x_i)_{i \in N}$ is the vector of variables with finite lower/upper bounds $x^L = (x_i^L)_{i \in N}$, $x^U = (x_i^U)_{i \in N}$. The variables with indices in $N_I \subset N$ are constrained to take on integer values in the solution. Difficulties arise from the integrality of the variables in N_I, as well as nonconvex nonlinear objective and constraints (if present). Exact solution methods typically require that the functions f and g_j's be *factorable*, that is, they can be computed in a finite number of simple steps, starting with model variables and real constants, using unary and binary operators. When the function f is linear and the g_j's are affine, $\mathcal{P}$ is an MILP, for which efficient branch-and-bound methods have been developed [24,31]. State-of-the-art MILP solvers (e.g. Cplex, Gurobi, or Xpress) are capable to solve, in reasonable time, problems with thousands of variables. Branch-and-bound methods for MINLPs attempt to closely mimick their MILP counterparts, but many difficulties have to be overcome. In particular, obtaining lower bounds is not straightforward. The continuous relaxation of each subproblem may be a nonconvex NLP, which is NP-hard. One possibility is to compute a convex relaxation (simply *convexification* from now on) of the feasible region of the problem, so that lower bounds can be easily computed.

In the following, we assume that a linear convexification of the original problem is available; that is, the objective function f and all constraints g_j are replaced by suitable linear terms which underestimate the original functions over all the feasible region. The computation of this linear convexification is a key step for LP-based branch-and-bound solvers for nonconvex MINLPs, and has been investigated extensively in the past [21,28]. The accuracy of the convexification greatly depends on the variable bounds. If the interval over which a variable is defined is large, the convexification of functions which contain that variable may be a very loose estimate of the original functions, leading to a poor lower bound. Furthermore, bounds can be tightened by branching or by applying various techniques, such as Feasibility Based Bound Tightening [26,27] and Optimality Based Bound Tightening [3,26]. A good incumbent solution not only provides a better upper bound to be used as a cutoff value within the branch-and-bound, but also allows for the propagation of tighter bounds through bound tightening techniques based on expression trees. Reduced cost-based bound tightening also benefits from a better upper bound. Therefore, finding good feasible solutions is doubly important for BB algorithms for MINLPs.

Much work has been done on primal heuristics for MILPs and convex MINLPs: see e.g. local branching [13], RINS [11], Feasibility Pump [5,12]. Bonami and Gonçalves [6] describe how to modify several heuristics for linear integer programs in order to deal with the nonlinear convex case. On the other hand, not much work has been carried out on heuristics for nonconvex MINLPs. A VNS-based heuristic is given in [19]. In a technical report [23], we described a way to perform local branching on nonconvex MINLPs in a computationally efficient way; in this paper, we extend those ideas to a more general framework. An earlier version of this work was published in [22]. Very recently, a Feasibility Pump for nonconvex MINLPs was also proposed [8, Chapter 2; 9].

3 Main algorithmic ideas

The heuristics presented in the paper follow a common scheme, whose main component is the rounding of a feasible solution to the continuous relaxation of $\mathcal{P}$, subject to linear constraints. These linear constraints always include the convexification of the original feasible region, but are not limited to it. We introduce some notation.

Let $\mathcal{Q}$ be the continuous relaxation of $\mathcal{P}$, that is, $\mathcal{P}$ without integrality requirements on the variables. In the following, we denote a solution $x \in \mathcal{Q}$ a *constraint feasible* or *NLP feasible* point. An *MINLP feasible* point is a feasible solution to the original problem $\mathcal{P}$. We denote by F the convexification of the feasible region of $\mathcal{P}$ amended with integrality constraints on the variables x_i with $i \in N_I$. Since we assume that the convexification involves linear constraints only, F is the intersection of a polyhedron with $\mathbb{R}^{|N \setminus N_I|} \times \mathbb{Z}^{|N_I|}$. A feasible solution to F is denoted an *integral feasible* point. With a slight abuse of notation, for a problem $\mathcal{P}$ and an inequality S, we denote by $\mathcal{P} \cap S$ the problem which is obtained by intersecting the feasible region of $\mathcal{P}$ with the points that satisfy S; in other words, we append S to the list of constraints. Given a point $y \in \mathbb{R}^n$, we denote by $\mathcal{Q}(y)$ the problem:

$$\left.\begin{array}{rr} \min & f(x) \\ \forall j \in M & g_j(x) \leq 0 \\ \forall i \in N \setminus N_I & x_i^L \leq x_i \leq x_i^U \\ \forall i \in N_I & x_i = y_i. \end{array}\right\} \mathcal{Q}(y) \tag{2}$$

$\mathcal{Q}(y)$ is the NLP that we obtain from $\mathcal{P}$ by fixing the integer variables of y; therefore, if y has integer components $y_j \in \mathbb{Z}\ \forall j \in N_I$ and $\mathcal{Q}(y)$ has a feasible solution z, then z is feasible for the original problem $\mathcal{P}$.

The heuristics described in this paper require an additional ingredient: a *no-good cut* $NG(\hat{x})$ for a given point $\hat{x}$. No-good cuts were originally introduced in [1], and have been used by the Constraint Programming community and in several other contexts (see e.g. [9,10,18]). In general, a no-good cut for $\hat{x}$ takes the form $NG(\hat{x}) = \|x - \hat{x}\| \geq \epsilon$, where the norm is typically the 1-norm, but is not restricted to be so. In this paper, for the most part we will employ canonical no-good cuts with 1-norm. However, to simplify notation, in the remainder we label $NG(\hat{x})$ *any* linear inequality violated by $\hat{x}$. In some cases (see Sect. 4), we use inequalities that do not take the form $\|x - \hat{x}\| \geq \epsilon$.

We now describe the main steps carried out by the heuristics proposed in this paper. Our idea is to start from a feasible solution x' to $\mathcal{Q}$, and look for an integral solution x^I which is close to x' and is in F. This can be achieved by solving the following problem:

$$\min_{x \in F} \|x - x'\|_1 \tag{3}$$

which is equivalent to rounding x' subject to the linear constraints that constitute a relaxation of the feasible region of $\mathcal{P}$. Note that (3) can be formulated as an MILP; it suffices to introduce an n-vector of extra variables and $2n$ constraints, to obtain the following problem, which can be easily shown to be equivalent to (3):

$$\left.\begin{array}{rl} \min & \sum_{i\in N} w_i \\ \forall i \in N & x_i - x'_i \leq w_i \\ \forall i \in N & x'_i - x_i \leq w_i \\ & x \in F \\ & w \in \mathbb{R}^n. \end{array}\right\} \tag{4}$$

Therefore, the point $x^I = \arg\min_{x\in F} \|x - x'\|_1$ can be obtained by solving (4) with an MILP solver; we call this a *rounding iteration*. x^I is certainly integral feasible, but it is not necessarily constraint feasible with respect to the original problem $\mathcal{P}$. To regain MINLP feasibility, we solve $\mathcal{Q}(x^I)$ with an NLP solver and hopefully obtain a feasible solution x^* to $\mathcal{P}$. If a termination criterion is not satisfied (which may depend on the feasibility of x^* or on different conditions), we iterate the algorithm by setting $F \leftarrow F \cap NG(x^I)$, and resolving (4), i.e. we try a different rounding of x'.

The basic scheme described above is formalized in Algorithm 1; we call this heuristic Iterative Rounding (IR). In Sects. 4, 5 and 6 we define the elements which are necessary for our algorithm: (i) the no-good cuts; (ii) how to compute the initial point x' to be rounded; and (iii) termination conditions. Depending on these details, and on additional constraints for the rounding phase, we obtain different heuristics. Note that all subproblems which have to be solved to apply Algorithm 1 need not be solved to optimality: in principle, only feasibility is required. However, typically $\mathcal{Q}(x^I)$ is solved to (at least) local optimality, because early termination could yield a solution with a worse objective function value.

Algorithm 1 Basic algorithm

```
1: Initialization: stop ← false,
2: Compute point x'
3: Set R ← F
4: while ¬stop do
5:    Compute x^I = arg min_{x∈R} ‖x − x'‖_1 with an MILP solver
6:    Solve Q(x^I) with an NLP solver and obtain point x*
7:    if Termination condition is satisfied then
8:       Set stop ← true
9:    else
10:      Set R ← R ∩ NG(x^I)
11: if x* feasible then
12:    return x*
13: else
14:    return failure
```

Some of the ideas that led to devising our IR scheme have also been employed by the recently developed Feasibility Pump (FP) for nonconvex MINLPs [8, Chapter 2;9], even though the aim of the FP is to find any feasible solution, whereas in our case, we want to find a *good* solution. We briefly highlight the main differences between the two approaches:

1. FP generates two trajectories of points, one of which is NLP feasible, and the other is integral on the integer variables, until the two trajectories converge to an MINLP feasible solution. It alternates between them in such a way that, at each

step, the point generated in one trajectory tries to get closer to the other trajectory. IR generates only *one* NLP feasible point x' (in different ways, see Sects. 5 and 6), and then generates several integer feasible points close to x'.
2. FP generates integer feasible points by solving an MILP involving the linear constraints included in the original problem $\mathcal{P}$, if present, and Outer Approximation cuts [5] that are computed from convex constraints by the heuristic during its execution. IR generates integer feasible points by solving an MILP involving a linearization of the original problem, and (in some cases) linear constraints that define a neighborhood for the search: see Sect. 6.
3. FP uses a tabu list [15] to avoid cycling, and resorts to no-good cuts if the tabu list fails. IR uses (different kinds of) no-good cuts at each iteration.

In general, IR focuses on the integer part of the problem: the nonlinearities (and the continuous variables) are dealt with only subsequently—the MILP solver does most of the work. Computational experiments (Sect. 8) show that IR is not as successful as FP in finding feasible solutions; on the other hand, it finds solutions of much better quality. Our explanation for this fact is that FP, unlike IR, generates a trajectory of points that possibly deteriorate the objective function value with the aim of reaching feasibility. IR keeps a point x' with good objective value fixed instead, i.e. it does not trade objective function value for feasibility, at least in the first iterations. Details on the different choices for the point x' will be discussed in Sects. 5 and 6.

Intuition tells us that the no-good cuts are likely to play an important role in practice, hence the following question arises: how do we *efficiently* cut off an integral point x^I from a polyhedron F by means of linear constraints? This will be discussed in the next section.

4 No-good cuts

The term *no-good cut* is used in the literature to indicate an inequality of the form $NG(\hat{x}) = \|x - \hat{x}\| \geq \epsilon$ (see [1]) that prevents x from getting too close to $\hat{x}$. This can be useful whenever we perform some kind of search, to avoid looking in a region that we know not to be promising (or already searched).

There are several issues that should be taken into account when designing the no-good cuts used within our heuristics. The most important aspect is the number of integer solutions that we want to cut off. On the one hand, there is an advantage if a no-good cut $NG(x^I)$ cuts off the given point x^I only, so that all remaining integer solutions can potentially be found in the subsequent rounding iterations. On the other hand, if we cannot regain MINLP feasibility by solving $\mathcal{Q}(x^I)$, then it is appealing to cut off a whole neighborhood of x^I, in order to diversify the computed integer solutions. Hence, we have to carefully balance these two objectives.

We employ two types of no-good cuts. They are described in detail below.

4.1 Type 1

Given an integer feasible point x^I that we want to cut off, let $B_L := \{i \in N_I | x_i^I = x_i^L\}$, $B_U := \{i \in N_I | x_i^I = x_i^U\}$ be the sets of integer variables at their lower and upper

bound, respectively. We assume that integer variables have integer bounds. The first type of no-good cut can only be used if $|B_L| + |B_U| \geq 1$; under this condition, we can employ the linear inequality:

$$\sum_{i \in B_U} (x_i^U - x_i) + \sum_{i \in B_L} (x_i - x_i^L) \geq \delta. \tag{5}$$

The effect of (5) is to cut off all points in a 1-norm neighborhood of x^I of size $\delta - 1$ (restricted to the space of the $B_L \cup B_U$ variables); therefore, this is a proper no-good cut. This type of constraint is very flexible, and it allows us to choose exactly the size of the neighborhood of x^I that we want to exclude at each rounding iteration. However, there are some drawbacks to this approach. If $B_L \cup B_U \neq N_I$, then we are cutting off feasible integer points of F based on the projection onto the space of the $B_L \cup B_U$ variables. This implies that we will cut off any integer solution y such that $y_i = x_i^I \ \forall i \in B_L \cup B_U$, regardless of the value of y_i for $i \in N_I \backslash (B_L \cup B_U)$. Hence, if $|B_L| + |B_U|$ is small compared to $|N_I|$, we may exclude more solutions than we want to. This suggests that no-good cuts of Type 1 should be employed when $|B_L| + |B_U|$ is of the same order of $|N_I|$. After some computational experimentation on a small set of instances (taken from MINLPLib [7]), we decided to use Type 1 no-good cuts only when:

$$|B_L| + |B_U| \geq \min\{50, \max\{|N_I|/10, 5\}\}. \tag{6}$$

Note that, if $\mathcal{P}$ has binary variables, then $B_L \cup B_U \neq \emptyset$. Thus, if a "sufficient" number of the integer variables are restricted to be binary, we can always employ no-good cuts of Type 1. If all integer variables are binary, (5) is a reverse local branching constraint [13], which requires δ variables to flip their value with respect to x^I. The inequality (5) can be dense if $|B_L| + |B_U|$ is large; however, it is a linear inequality with unitary coefficients and integer right hand side, and only a limited number of these cuts can be added to (4) at a given time (one per iteration of the main loop of the heuristic), thus it is not computationally burdensome.

How do we choose δ? Our intuition is that it should be problem-dependent. Indeed, consider a mixed binary problem (i.e. a problem where all integer constrained variables are binary): in this case, binary variables typically encode key decisions of the model, therefore even if only one binary variable is required to change its value with respect to x^I, we expect that this should be sufficient to diversify the solution obtained at the following rounding iteration. Now consider a problem with general integer variables with very loose bounds: in this case, if we set $\delta = 1$, then we can expect that there exists some integer feasible solution y to F that also satisfies (5) and is very "close" to x^I, i.e. they encode almost the same solution from a modeling standpoint. So we risk that, in the following rounding iteration, we obtain another point that is very similar to x^I, and most likely will not lead to an MINLP feasible point satisfying the termination condition of the heuristic. Our approach is to choose δ based on the bounds of the integer variables:

$$\delta = \left\lfloor \frac{\sum_{i \in B_L \cup B_U} (x_i^U - x_i^L)}{|B_L| + |B_U|} \right\rfloor, \tag{7}$$

i.e. δ is equal to the average distance between the bounds of the variables involved in (5), rounded to the nearest integer. For the mixed binary case, this translates to $\delta = 1$; when general integer variables are involved in the formulation, we believe that choosing δ this way should guarantee sufficient diversification of the integer points generated during the rounding phase. Note that efficient bound tightening techniques are very important for this approach whenever general integer variables are present: not only do they increase the chance that $B_L \cup B_U \neq \emptyset$, but they also allow a more meaningful estimation of δ. Finally, δ is always finite as we assume that all integer variables are bounded.

If the condition (6) is not satisfied, we employ a different scheme to generate no-good cuts; this will be discussed in the next section.

4.2 Type 2

No-good cuts of Type 1 (5) cannot be employed in a straightforward manner when none of the integer components of the point we wish to cut, x^I, is at one of its bounds. Note that this implies that $\mathcal{P}$ does not have any binary variable. It is possible to write a linear inequality similar to (5) for the case where no variables are at one of their bounds, following [13]. However, this would require the addition of one binary variable and two constraints for every integer variable involved in the inequality, because this is a more general case where the no-good cut is nonconvex. In other words, since we do not know in which direction each original variable is going to move, we need to introduce some kind of disjunctive constraint, hence the extra binary variables. This is not an efficient way of dealing with this issue: the MILP (4) would rapidly become very large, increasing running time.

We opted for a simpler approach. Whenever we cannot or decide not to employ Type 1 no-good cuts because (6) is not satisfied, we randomly choose one variable $i \in N_I$, and "branch" on x_i, i.e. impose the constraint $x_i \leq x_i^I - 1$ with probability $\mu = (x_i^I - x_i^L)/(x_i^U - x_i^L)$, or the constraint $x_i \geq x_i^I + 1$ with probability $1 - \mu = (x_i^U - x_i^I)/(x_i^U - x_i^L)$. Clearly, this cuts off x^I, but it can potentially cut a large number of feasible solutions to F, and the MILP (4) could rapidly become infeasible after the addition of no-good cuts of Type 2—even though unexplored integer feasible solutions exist.

To deal with this issue, we keep a list L of the variables that were "branched" on at previous iterations. $L = \emptyset$ when the heuristic starts. Throughout the procedure, the set of candidate branching variables is $N_I \setminus L$ if $N_I \setminus L \neq \emptyset$, or N_I otherwise. If x_i is the randomly selected candidate, we update $L \leftarrow L \cup \{i\}$. In other words, we never branch twice on the same variable, unless $|L| \geq |N_I|$, i.e. there are no more integer variables to branch on. Whenever we detect infeasibility of (4) and Type 2 no-good cuts were employed, we remove all Type 2 no-good cuts from the MILP (4), but we do not erase L; then, we branch on some variable in $N_I \setminus L$ from the last integral feasible solution obtained. If $|L| \geq |N_I|$, cycling could happen since the list of forbidden branchings L is ignored; however, this is an extremely rare occurrence in practice, and in general we eventually escape the cycle due to the random choice of the branching variable. Therefore, we did not take steps to prevent this type of cycling.

Note that both types of no-good cuts have pros and cons. The first one is more flexible, but can only be applied under certain conditions, whereas the second type can always be used, but is not as appealing as the first type. Both are linear inequalities with integer coefficients, and are therefore computationally efficient: the size of the LP relaxation of (4) does not increase substantially if we introduce a small number of no-good cuts, hence we can keep computing time under control. As remarked in Sect. 3, only the no-good cuts of Type 1 satisfy the definition of no-good cut given in [10], whereas those of Type 2 do not; we label both of them "no-good cuts" because this allows a significant simplification of the notation.

5 Feasibility-based Iterative Rounding

Based on Algorithm 1, we tailor a heuristic aimed at discovering a first feasible solution. Recall that we need to specify how to obtain an initial constraint feasible point x', and a stopping criterion. For this heuristic, we are satisfied with finding a feasible solution to the original problem $\mathcal{P}$; hence, we stop as soon as such a point is found.

How do we choose x'? Ideally, we would like to find a feasible solution to $\mathcal{P}$ that yields a good objective function value. Hence, it is a natural choice to set x' as the solution to $\mathcal{Q}$, i.e. a (locally) optimal solution of the continuous relaxation of the original problem. Note that, since $\mathcal{Q}$ is in general nonconvex, computing a global optimum would require a significant computational effort. Thus, we pick x' as the first local optimum of $\mathcal{Q}$ provided by an NLP solver.

Our intuition is that, if we are able to find an MINLP feasible point by rounding an x' with good objective function value for $\mathcal{Q}$, then the feasible solution discovered this way will likely yield a good objective value for $\mathcal{P}$ as well. However, we may not be able to find a suitable rounding. If this happens, we try to generate a new x', which satisfies the constraints of $\mathcal{Q}$ by a larger amount; we hope that this will yield better chances of discovering, through (4), a rounding that leads to an MINLP feasible point. In other words, we initially focus on the objective function value, and if we cannot discover MINLP feasible points, we switch our focus on feasibility.

More specifically, we try to round a sequence of h NLP feasible points $x'^{,1}, \ldots, x'^{,h}$, where each $x'^{,i}$, $i = 1, \ldots, h$ is obtained by solving with an interior point method (see e.g. [30]) the barrier problem associated with $\mathcal{Q}$ with a different *minimum* value $\mu'^{,i}$ for the barrier parameter μ. For $i = 1, \ldots, h$, we set $\mu'^{,i} = \omega(i-1)$, where $\omega > 0$ is a parameter; that is, we do not allow the log-barrier term in the objective function to go to zero, except during the computation of $x'^{,1}$ (which is a local optimum of $\mathcal{Q}$). This way, each point in the sequence $x'^{,1}, \ldots, x'^{,h}$ should be more in the interior of the feasible region with respect to its predecessors, at the expense of a worse objective function value.

We call this heuristic *Feasibility-based Iterative Rounding* (F-IR); we give a description in Algorithm 2. In practice, it is also reasonable to put a maximum time limit on the main loop, to avoid spending too much time running the heuristic if the auxiliary NLPs or MILPs that are solved turn out to be time-consuming.

Algorithm 2 Feasibility-based Iterative Rounding

Input: parameters $\omega, h, MaxIter$
Output: feasible solution x^*
Initialization: `stop` $\leftarrow$ false
for $j = 0, \ldots, h-1$ **do**
  Set `NumIter` $\leftarrow 0$
  Solve the log-barrier problem of $\mathcal{Q}$ with $\mu' = \omega j$ to obtain x'
  Set $R \leftarrow F$
  while $\neg$`stop` **do**
    Compute $x^I = \arg\min_{x \in R} \|x - x'\|_1$ with an MILP solver
    Solve $\mathcal{Q}(x^I)$ with an NLP solver and obtain point x^*
    if (x^* MINLP $-$ feasible) $\vee$ (`NumIter` $\geq MaxIter$) **then**
      Set `stop` $\leftarrow$ true
    else
      Set $R \leftarrow R \cap NG(x^*)$
    Set `NumIter` $\leftarrow$ `NumIter` $+ 1$
if x^* feasible **then**
  return x^*
else
  return failure

6 Improvement-based Iterative Rounding

Most BB solvers for MILPs employ *improvement* heuristics, i.e. heuristics that start from a feasible solution (the *incumbent*) and seek another feasible solution with a better objective function value. Typically, this is carried out by exploring a neighborhood of the incumbent. This neighborhood can be defined and explored in different ways, hence the number of existing improvement heuristics. Local branching [13] and RINS [11] are two among the most successful, within the domain of MILPs.

We have devised an improvement heuristic for nonconvex MINLPs using the scheme described in Sect. 3. In particular, given the incumbent $\bar{x}$, i.e. the solution that we aim to improve, we want to define a neighborhood $\mathcal{N}(\bar{x})$ of $\bar{x}$, and search for a better solution within this neighborhood with the machinery that we have introduced above. This means that the solution to the rounding problem (4) is restricted to be within $\mathcal{N}(\bar{x})$; hence, $\mathcal{N}(\bar{x})$ must be defined with linear constraints, so that we can still employ an MILP solver for the solution of (4). Note that the neighborhood should be small, so that it can be explored effectively, but also large enough that it has a good probability of comprising an improved solution. Therefore, we have to carefully balance this tradeoff.

We define the neighborhood to explore by means of a local branching constraint [13], if possible. Define, as in Sect. 4.1, B_L and B_U as the sets of variables at lower/upper bound at the incumbent $\bar{x}$. If either or both of them are nonempty, we define a local branching neighborhood through the following constraint:

$$\sum_{i \in B_U} (\bar{x}_i - x_i) + \sum_{i \in B_L} (x_i - \bar{x}_i) \leq k. \tag{8}$$

If all integer variables in the original problem are binary, we can always use this constraint, whose effect is to allow only k variables to flip their value with respect to $\bar{x}$.

This neighborhood has been shown to be very effective in practice on difficult MILP instances [13], in the sense that improved solutions can typically be found within this neighborhood even with a small right hand side, e.g. $k = 10$ or 15. It was also shown to be effective within the MINLP world [19,23]. Again, we face the problem of choosing k; for the binary case, we can follow the suggestion of [13] and use a small $k \in [10, 20]$; for general integer variables, we will choose a k based on the distance between the bounds involved in (8), as in Sect. 4.1. This will be discussed in the computational experiments, Sect. 8.

If the number of variables at their bounds in $\bar{x}$ is not large enough (which implies that there are few or no binary variables), then we define the neighborhood as a box on the integer variables only centered on $\bar{x}$, where the length of each side of the box is equal to half of the distance between the original bounds of the corresponding variable.

More formally: if $|B_L| + |B_U| \geq \min\{50, \max\{|N_I|/10, 5\}\}$, we define the neighborhood $\mathcal{N}(\bar{x})$ of $\bar{x}$ as $\mathcal{N}(\bar{x}) = F \cap \{x | \sum_{i \in B_U} (\bar{x}_i - x_i) + \sum_{i \in B_L} (x_i - \bar{x}_i) \leq k\}$, otherwise we define $\mathcal{N}(\bar{x}) = F \cap \{x | x_i^L + (\bar{x}_i - x_i^L)/2 \leq x_i \leq x_i^U - (x_i^U - \bar{x}_i)/2 \; \forall i \in N_I\}$.

We iterate the rounding phase until we find an MINLP feasible solution that has a better objective function value with respect to the incumbent $\bar{x}$, up to a maximum number of times. We have now defined all necessary ingredients; the improvement heuristic proceeds as indicated in Algorithm 3. We call this heuristic *Improvement-based Iterative Rounding* (I-IR). In practice, we will also set a maximum allowed time for the heuristic to run.

Algorithm 3 Improvement-based Iterative Rounding

1: **Input**: incumbent $\bar{x}$, parameters k, *MaxIter*
2: **Output**: improved solution x^*
3: Initialization: `stop` $\leftarrow$ false, `NumIter` $\leftarrow 0$
4: Solve $\mathcal{Q} \cap \mathcal{N}(\bar{x})$ with a NLP solver to obtain x'
5: Set $R \leftarrow F \cap \mathcal{N}(\bar{x})$
6: **while** $\neg$`stop` **do**
7: Compute $x^I = \arg\min_{x \in R} \|x - x'\|_1$ with an MILP solver
8: Solve $\mathcal{Q}(x^I)$ with an NLP solver and obtain point x^*
9: **if** ((x^* MINLP-feasible $\wedge$ $f(x^*) < f(\bar{x})$) $\vee$ (`NumIter` $\geq$ *MaxIter*)) **then**
10: Set `stop` $\leftarrow$ true
11: **else**
12: Set $R \leftarrow R \cap NG(x^*)$
13: Set `NumIter` $\leftarrow$ `NumIter` $+ 1$
14: **if** x^* feasible **then**
15: **return** x^*
16: **else**
17: **return** failure

For a mixed binary problem, if $|B|$ is the number of binary variables and k is the rhs of (8), Algorithm 3 will stop after at most $\sum_{i=1}^{k} \binom{|B|}{i}$ iterations, returning either an improved incumbent or no solution. Trivially, this follows from the fact that there are at most $\sum_{i=1}^{k} \binom{|B|}{i}$ different realizations of the vector of binary variables in the neighborhood of $\bar{x}$ defined by the local branching constraints (each vector differs on at most k components from $\bar{x}$), and each one is generated at most once.

7 Considerations on efficiency

Our heuristic scheme is mainly focused on dealing with the integer part of $\mathcal{P}$: no particular effort is made to make sure that the nonlinear part of the constraints is satisfied (except choosing a point to round x' which is NLP feasible). However, the nonlinear constraints, and in particular the nonconvex ones, can be a source of trouble, as we will see in the remainder of this section. We now state some basic facts on the theoretical behaviour of our heuristic, which depend on the convexity/nonconvexity of the constraints.

7.1 Nonconvexity and no-good cuts

In defining the no-good cuts (Sect. 4) we disregard the continuous variables of $\mathcal{P}$. Our no-good cuts impose a minimum amount of diversification of the integer variables with respect to the point x^I that is cut off. This implies that all integral feasible points whose components in N_I are equal to those of x^I, but the continuous variables are different, are cut off as well. Since we do not employ a global optimizer to solve the nonlinear nonconvex problem $\mathcal{Q}(x^I)$, an NLP solver may fail to find a feasible solution, but an MINLP feasible point x^* such that $x^*_i = x^I_i \ \forall i \in N_I$ may exist (note that if $\mathcal{Q}(x^I)$ is convex, i.e. $\mathcal{P}$ is convex when the integer variables are fixed, this situation cannot occur). If this is the case, we would generate a no-good cut $NG(x^I)$, preventing the possibility of discovering x^*.

Solving $\mathcal{Q}(x^I)$ to global optimality would require a significant computational effort and could be as difficult as the original problem $\mathcal{P}$; within a heuristic method such as the one described in this paper, with no guarantee of finding an MINLP feasible point even by globally solving $\mathcal{Q}(x^I)$ due to the use of no-good cuts, the approach would be time-consuming, therefore we are content with local solutions.

7.2 Termination for the convex mixed-binary case

If all integer variables of $\mathcal{P}$ are restricted to be binary, and $\mathcal{Q}$ is such that $\mathcal{Q}(y)$ is a convex NLP for each integral vector $y \in F$, then our Iterative Rounding heuristic eventually finds an MINLP feasible point if one exists, and indeed a global optimum, provided that the maximum number of iterations is set to $2^{|B|}$, where B is the set of binary variables. This follows from the fact that, in this case, the heuristic solves a sequence of convex NLP problems $\mathcal{Q}(y)$ for all possible assignments y of the integer variables which are feasible to F (which are at most $2^{|B|}$), and repetition is not allowed because of the no-good cuts of Type 1 (5) with $\delta = 1$. One of these NLPs must yield the global optimum.

8 Computational experiments

In this section, we provide computational experiments to assess the practical usefulness of the proposed heuristics. Detailed results for all the experiments can be found in the appendix. In the rest of this section, we only provide aggregated results.

Both F-IR and I-IR where implemented within `Couenne` [2], an open-source BB solver for nonconvex MINLPs. Through the rest of this section, the convexification of the feasible region of the original problem $\mathcal{P}$ is assumed to be the one provided by `Couenne` at the root node of the BB tree, after the bound tightening phase which is carried out with the default parameters. We used `Cplex 12.1` [17] as the MILP solver, and `Ipopt` [30] as the NLP solver. We set a time limit of 5 s and at most 50 nodes for the solution of all MILPs arising during the application of the heuristics; however, if no integral solution is found by Cplex after one of these two limits is hit, we continue the search (for another 5 s or 50 nodes). This is iterated until a solution is found, or until we hit the global time limit for the application of the heuristic. No time limit was given for the solution of NLPs, but the NLP solver is stopped after 3,000 iterations. All experiments were run on one core of a machine equipped with an Intel Xeon clocked at 3.20 GHz and 2.5 GB of RAM, running Linux.

Cplex was parameterized in the following way: `MIPEmphasis = HIDDENFEAS`, `FPHeur = 2`, `LBHeur = 1`, `RINSHeur = 99`. The node selection strategy was set to depth first (`NodeSel = 0`). These parameters are meant to drive the MILP solver towards finding an early feasible solution of good quality. Cut generation did not seem to help in this task, but it required additional time; since we are not interested in proving optimality of the computed solution, i.e. increasing the lower bound, cuts were disabled (`CutsFactor = 1.0`).

Recent versions of Cplex provide a solution pool that can be used to obtain multiple integer points x^I during the solution of (4). Each point could be used for the NLP searches $\mathcal{Q}(x^I)$. We decided not to pursue this approach for the following reasons. First, the solutions stored in the pool typically have larger 1-norm distance to the rounded point x' than the best solution found. This is because the objective function of (4) is the distance to x', and by default the solution pool stores the integer points discovered during branch-and-bound only. Since the additional integer candidates are farther from x' than the best solution found, we believe that they are less likely to lead to a good MINLP feasible solution. Solving the NLP subproblems $\mathcal{Q}(x^I)$ can be time-consuming, therefore we want to maximize the chances that their solution yields an MINLP feasible point. Another reason is that for problems with continuous and general integer variables, it is difficult to specify the amount of diversification that the solutions in the pool should satisfy. Finally, generating many integer solutions with good objective value requires modifications of Cplex's default parameters that result in an overall slowdown of the branch-and-bound search. For these reasons, we do not use solutions in the pool as integer candidates for the NLP subproblems $\mathcal{Q}(x^I)$. However, note that Cplex automatically takes advantage of these solutions in subsequent branch-and-bound searches after problem modification (e.g. addition of no-good cuts), for instance to derive solutions for the modified problem.

8.1 Test instances

Our test set initially consisted of the 251 instances included in MINLPLib [7] (http://www.gamsworld.org/minlp/minlplib.htm) in March 2010. MINLPLib is a freely available collection of convex and nonconvex MINLPs taken from different

sources; we reformulated them in AMPL format to make them readable for `Couenne`. We then eliminated instances because of the following reasons:

1. The instance cannot be read in AMPL due to unimplemented operators;
2. `Couenne` is not able to process the root node due to numerical problems or because the processing time exceeds 30 min;
3. The problem is found to be infeasible;
4. An MINLP-feasible solution is found at the root node by `Couenne` when processing the LP relaxation.

Note that in principle we could have kept instances that satisfy criterion 4), but these are mostly very easy instances, which are therefore uninteresting in this context: there is no need to apply sophisticated and possibly time-consuming heuristics if feasible solutions can be found at almost no cost using the simple heuristics incorporated in default `Couenne`.

99 instances satisfied one of the rejection criteria above; we are left with 152 instances, which are listed in Table 1. Details on these instances can be found at: http://www.gamsworld.org/minlp/minlplib/minlpstat.htm.

8.2 Number of rounding iterations

To determine the maximum number of rounding iterations that yields the best practical performance, we set up an experiment as follows. We applied F-IR and I-IR (if a feasible solution is found by F-IR) on all instances in the test set at the root node, where F-IR is parameterized with $h = 5$ (we try to round at most 5 different points x') and $\omega = 0.2$ (every time we solve the barrier problem to obtain a point x', we increase μ' by 0.2), and the rhs of (8) for I-IR is chosen as $k = \min\{15, \max\{1, |N_I|/2\}\} + \delta - 1$, where δ is equal to the average distance between the bounds of the variables involved in the inequality as in (7). Note that for a mixed binary case, this is equivalent to picking $k = 15$ except for very small problems; if general integer variables are involved, then the rhs is increased. These values for the parameters were dictated by our experience while developing the heuristics, and were found to perform well in a practical setting. With this setup, we test three values for the maximum number of rounding iterations *MaxIter*: 5, 10 and 20. We set a maximum running time for the combined heuristics of 300 s when $MaxIter = 5, 10$, and of 1,000 s for $MaxIter = 20$ (to make sure that we allow enough time to apply more rounding phases than for smaller values of this parameter). Note that, after the time limit is hit, the heuristic will not immediately exit if it is solving an NLP; in that case, it will return as soon as the NLP solution process ends.

Detailed results are reported in Table 6. A brief summary of the results is given in Table 2: we report, for each value of *MaxIter*, the number of instances for which a feasible solution is found (first column), the number of instances for which a solution at least as good as the remaining values of *MaxIter* is found (second column), the number of instances for which a solution strictly better than the remaining values of *MaxIter* is found (third column) and the average CPU time (fourth column). For the computation of the second column, instances such that no feasible solution was found (by any method) are excluded. The average time is a geometric average, computed

Table 1 Set of test instances

batchdes	fo7_ar25_1	no7_ar5_1	qapw
cecil_13	fo7_ar3_1	nuclear14a	ravem
contvar	fo7_ar4_1	nuclear14b	ravempb
csched1	fo7_ar5_1	nuclear24a	saa_2
deb10	fo7	nuclear24b	space25
deb6	fo8_ar2_1	nuclear25a	space960
deb7	fo8_ar25_1	nuclear25b	spectra2
deb8	fo8_ar3_1	nuclear49a	st_e29
deb9	fo8_ar4_1	nuclear49b	st_e31
detf1	fo8_ar5_1	nuclearva	st_e32
eg_disc2_s	fo8	nuclearvc	st_e35
eg_disc_s	fo9_ar2_1	nuclearvd	st_e40
eg_int_s	fo9_ar25_1	nuclearve	st_miqp2
elf	fo9_ar3_1	nuclearvf	st_test2
eniplac	fo9_ar4_1	nvs03	st_test3
enpro48	fo9_ar5_1	nvs10	st_test5
enpro48pb	fo9	nvs12	st_test6
enpro56	gastrans	nvs13	st_test8
enpro56pb	gear3	nvs15	st_testgr3
ex1224	hmittelman	nvs17	synheat
ex1225	lop97ic	nvs18	tln12
ex1233	lop97icx	nvs19	tln2
ex1243	m3	nvs23	tln4
ex1244	m6	nvs24	tln5
ex1263a	m7_ar2_1	o7_2	tln6
ex1263	m7_ar25_1	o7_ar2_1	tln7
ex1264a	m7_ar3_1	o7_ar25_1	tloss
ex1264	m7_ar4_1	o7_ar3_1	tls12
ex1265a	m7_ar5_1	o7_ar4_1	tls2
ex1265	m7	o7_ar5_1	tls4
ex1266a	netmod_dol1	o7	tls6
ex1266	netmod_dol2	o8_ar4_1	tls7
ex4	netmod_kar1	o9_ar4_1	tltr
fac2	netmod_kar2	oil2	uselinear
fac3	no7_ar2_1	ortez	var_con10
feedtray	no7_ar25_1	parallel	var_con5
fo7_2	no7_ar3_1	prob10	water4
fo7_ar2_1	no7_ar4_1	qap	waterz

adding 1 to each value before the calculation, and subtracting 1 from the result; this way, we avoid troubles when computing the logarithm of numbers close to zero.

These results suggest that increasing the number of rounding iterations does not have an effect on the number of feasible solutions found, but it has a major impact on

Table 2 Summary of the results of Table 6

	# of instances			CPU time
	Feasible	Obj $\leq$	Obj $<$	
$MaxIter = 5$	107	58	5	25.71
$MaxIter = 10$	107	70	13	38.51
$MaxIter = 20$	107	90	30	70.71

the quality of the solutions discovered. Indeed, setting $MaxIter = 20$ typically yields solution of better quality, as is shown by the fact that on 90 instances out of 107, our heuristics with $MaxIter = 20$ find a feasible point which is at least as good as the ones found by using a smaller value of the parameter, and in 30 cases, it is strictly better. On the other hand, this leads to longer CPU times: on average, one application of the heuristics with $MaxIter = 20$ takes more than 1 min. However, the runs that take the longest time are the *unsuccessful* ones, i.e., when no feasible solution is found: in only three cases a successful run terminates after 500 s.

It should be noted that increasing *MaxIter* does not always yield better objective values. This can be seen in Table 6 for example on the `fo7`, `m7`, `o7` instances. In most of these instances, all integer variables are general integers, and many Type 2 no-good cuts are used. The randomness involved in Type 2 no-good cuts partially explains this behaviour. In other cases, we found that the quality of the solutions is volatile because of the time limit imposed for solving the rounding MILP (4), which may lead to slightly different integer solutions in different runs. In our experiments this happens especially on instances involving products between variables. We do not have an explanation of why this is the case.

Another observation is that on some instances, the heuristic terminates very quickly without a solution (examples are `batchdes`, `eg_int_s`, `ex1225`, `nvs15`). There are several reasons for this. The rounding iterations take a short time for one or more of these factors: the instance is very small, there are no continuous variables (hence we skip solution of the NLP subproblem), or Cplex finds that the rounding MILP (4) is infeasible. In the latter case, the heuristic terminates immediately without a solution. In the remaining cases, it can happen that the subsequent NLP subproblem is quickly solved to optimality, but the solution returned by the NLP subsolver is deemed infeasible by `Couenne`. Hence, the heuristic's main loop takes little time but no feasible solution is found.

In the following, we keep $MaxIter = 10$, and set the maximum execution time of the heuristics at the root to 300 s, in order to balance CPU time and quality of the solutions; we remark that, from a practical point of view, smaller values of *MaxIter* yield a much faster heuristic, whereas larger values yield better solutions at the expense of computing times.

8.3 Feasibility-based Iterative Rounding

We now focus our attention on F-IR only, and analyse its results on our set of test instances, for different values of the parameters. There are two parameters whose influence should be assessed: the number h of different points x' that are successively

Table 3 Summary of the results reported in Table 7

	$\omega = 0.05$	$\omega = 0.10$	$\omega = 0.20$	$\omega = 0.25$
# Feasible solutions	101	104	109	108
# Times best solution	5	6	6	7
Avg. CPU time	14.28	13.61	13.08	13.59

In the first row we report the number of instances for which a feasible solutions is found. In the second row we report the number of instances for which an algorithm yields a solution that is *strictly better* than all remaining algorithms. The third row shows the geometric average of the CPU time

rounded and the factor ω by which the value of the minimum barrier parameter μ' is increased whenever we generate a new x' (see Sect. 5).

The parameter h has a considerable effect on the running times: each point x' is rounded up to *MaxIter* times through the solution of an MILP, which may be time consuming. Therefore, small values of h yield a faster heuristic, whereas larger values increase running times but allow more possibilities of finding a feasible point. Furthermore, h and ω should always be considered together: for instance, consider the case $h = 5$, $\omega = 0.2$ and the case $h = 10$, $\omega = 0.1$. It is easy to see that the sequence of points x' generated in the first case is a subsequence of the points generated in the second case (assuming that the solution of each log-barrier subproblem is unique). In general, h determines the amount of effort that we are willing to invest into generating a feasible solution, and ω determines how much we are willing to lose in terms of objective function value to favour feasibility.

Based on our previous experience [22] and on these observations, we picked $h = 5$ and varied ω. We tested four different values: $\omega = 0.05, 0.1, 0.2, 0.25$. Full results are reported in Table 7; a summary can be found in Table 3.

By looking at Tables 3 and 7, we can see that varying ω indeed has an effect on the solution returned by F-IR. In particular, increasing ω typically leads to more feasible solutions found (with the exception of switching from $\omega = 0.20$ to $\omega = 0.25$), which is expected: a larger ω means that we put our focus on feasibility, possibly disregarding the objective function value. However, it seems that the objective function value is usually good even for large values of ω. This is because, for a large number of instances, a feasible solution is found by rounding the solution to $\mathcal{Q}$, i.e. the continuous relaxation of $\mathcal{P}$, therefore the value of ω does not have any effect. There are cases where ω is important, though, as can be seen in all instances where the different algorithms compared in Table 7 find different solutions. There does not seem to be a clear winner in terms of solution quality. Since $\omega = 0.20$ yields the largest number of feasible solutions with shortest average CPU time, we use $\omega = 0.20$ in the rest of this paper.

8.4 Improvement-based Iterative Rounding

In this section we study the effects of applying an improvement phase (I-IR) on the solutions returned by F-IR. We design an experiment as follows: we apply F-IR with $h = 5$ and $\omega = 0.2$, with a time limit of 300 s. Then we apply I-IR (if a feasible

Table 4 Summary of the results reported in Table 8

	$k' = 10$	$k' = 15$	$k' = 20$	F-IR only
# Improved solutions	55	57	54	0
Geom. avg. distance from best (%)	0.87	0.76	0.97	4.17
Arithm. avg. distance from best (%)	3.52	3.25	3.95	22.12
Arithm. avg. # improving steps	1.21	1.24	1.07	0
Avg. CPU time	30.97	32.15	32.14	6.65

In the first row, we report the number of instances for which I-IR improves over F-IR alone. In the second and third row, we report the average objective function value increase of the solutions with respect to the best solution found for each instance. The fourth row contains the average number of times that I-IR returned an improved solution (note that I-IR can improve the solution more than once per instance), while the last row shows the average CPU time

solution is found), with different values of the rhs of the local branching constraint k; we keep applying I-IR as long as the incumbent improves, or until the time limit of 300 s is hit. *MaxIter* is set to 10 as usual, which means that after 10 failed rounding iterations, the heuristic returns with a failure (keeping the best solution found up to that point, if any).

These experiments are meant to assess the usefulness of the improvement heuristic I-IR. Note that the application of I-IR is in principle not restricted to using a solution found by F-IR: *any* feasible solution discovered during branch-and-bound can be employed as a starting point. However, for comparison purposes it is much easier for us to only consider solutions found by F-IR, as they are found at the root node in a short time. In our implementation of I-IR within `Couenne`, the heuristic is called after a new feasible solution is found, during branch-and-bound; in this section we initialize I-IR with the solutions discovered by F-IR only.

Here we only consider the 107 instances for which a feasible solution is found by F-IR with $h = 5$ and $\omega = 0.2$, so that I-IR can start at the root node. Our choice of k depends on the bounds of the integer variables involved in the constraint (8): we pick $k = \min\{k', \max\{1, |N_I|/2\}\} + \delta - 1$, where k' is a given value and δ is defined as in (7). For instances where only binary variables are involved, $k = k'$; otherwise, if general integer variables with loose bounds are present, then k is increased accordingly, to allow exploration of a larger neighbourhood around the incumbent. We test three different values of k': $k' = 10, k' = 15, k' = 20$, as suggested in [13].

Detailed results can be found in Table 8. Table 4 summarizes the results of the experiment. We report, for each value of k' (as well as for F-IR employed alone) the average CPU time and the average distance from the best solution found. To compute the latter, for each parameterization of the heuristic and for each instance, we calculate the increase in objective function value with respect to the best solution reported on that instance by any of the heuristic runs. This value is then averaged with a geometric mean as usual, and with an arithmetic mean as well. The reason for reporting the arithmetic mean is that in several cases the relative distance with respect to the best solution is 0% for all algorithms, therefore the geometric mean tends to be very

Table 5 Summary of the results reported in Table 9

	Iterative Rounding	Feasibility Pump	RECIPE
# Feasible solutions	107	122	132
# Times best solution	11	10	86
Avg. distance from best known (%)	4.56	2,989.77	0.35
Avg. CPU time	29.74	11.14	173.77

In the first row, we report the number of feasible solutions found. In the second row, we indicate the number of instances for which one of the heuristics discovers a solution strictly better than the other ones. In the third row, we report the average relative increase of objective function value with respect to the best known solutions. In the last row, we report average CPU times

small and thus misrepresents the differences. Additionally, we report the (arithmetic) average number of solution improvements through I-IR.

Table 4 shows that the improvement heuristic I-IR really pays off: the objective function value improves considerably with respect to applying F-IR alone, although this comes at the cost of some CPU time. In particular, for roughly half of the instances, I-IR is able to improve on the solution returned by F-IR, and on average F-IR finds a solution that is in relative terms 22% worse than that reported by one of the tested parameterizations of I-IR.

All three values for the parameter k' yield similar results, with a slight advantage for $k' = 15$, that is able to improve on a few more solutions than its competitors, achieving better average behaviour. In particular, I-IR $k' = 15$ is able to perform a marginally larger number of improving steps (i.e. moving from one incumbent to a better one).

Overall, I-IR seems very effective, even though CPU time increases on average by a factor of 5: we are able to greatly improve the quality of the solutions returned, while still requiring, on average, only ≈30 s at the root node.

8.5 Comparison with existing heuristics

We conclude our computational experiments with a comparison of the heuristics proposed in this paper with two existing heuristics for nonconvex MINLPs: the Feasibility Pump [9] and RECIPE [19]. The Feasibility Pump is discussed at the end of Sect. 3. RECIPE is based on the Variable Neighbourhood Search metaheuristic [16] and explores the solution space of a problem using a combination of different solvers. We compare the results obtained by: the Feasibility Pump; RECIPE (with `minlp_bb` and `Ipopt` as subsolvers, see [19]); an application of F-IR + I-IR at the root node (with a time limit of 300 s, parameters $h = 5, \omega = 0.2, k' = 15$). It is important to remark that a faster machine than ours is used for the computational experiments in [9,19].

Detailed results are reported in Table 9. A summary is given in Table 5. We report the number of instances for which each heuristic finds a feasible solution (over 152 instances), the number of instances for which a heuristic returns a solution that is strictly better than the one found by the other heuristics, the average relative distance

from the best solution known for the instance,[1] and the average CPU time.[2] We remark that on the four `netmod` instances, `saa_2` and `oil2`, we do not have results for the FP, as they are not reported in [9]. However, these instances are not taken into account when computing the averages. Note that on `saa_2` and `oil2`, our heuristic fails to find any feasible point, whereas close to optimal solutions are found on the four `netmod` instances.

Table 5 highlights the main practical differences between Iterative Rounding, Feasibility Pump and RECIPE. If we compare FP and IR, we see that FP finds more feasible solutions, and is on average much faster (almost 3 times faster—even though part of this difference is due to the more powerful machine used for the experiments). A comparison between RECIPE and IR shows that RECIPE finds more feasible solutions, but is considerably slower (more than 5 times slower even with a faster machine). In terms of solution quality, RECIPE comes out as the winner, with IR close behind. Indeed, solutions found by RECIPE and IR are very close to the best known solutions (the average increase in the objective function value is 0.35 and 4.56%, respectively), whereas FP does not make any effort to find good solutions (the average objective function increase is 2,989.77%). As a consequence, IR finds a better solution than FP on 92 instances out of 152. FP reports a better objective value than IR in 45 cases, the majority of which correspond to instances on which FP finds a feasible solution whereas IR does not. Overall, Table 5 shows that RECIPE is clearly the better algorithm in terms of solution quality, but it can take considerable CPU time. FP and IR are faster alternatives: FP excels at finding feasible solution, but of poor quality; IR finds fewer feasible solutions, but typically very close to the best known points.

9 Concluding remarks

We have presented two heuristics for finding feasible solutions to nonconvex MINLP problems, based on a common Iterative Rounding scheme. Our computational experience shows that these heuristics find solutions for the majority of the MINLPs in our test set. The main advantage of these heuristics over other heuristics for MINLP is the fact that they obtain a good solution in a relatively short time, and hence are a fair trade off between fast heuristics such as the Feasibility Pump, which does not aim at finding tight upper bounds, and procedures such as RECIPE that obtain good solutions but at a high computational cost. It is worth pointing out that the improvement heuristic I-IR can be applied during the whole branch-and-bound search, and is not limited to the root node as tested in this paper.

Appendix: detailed tables of results

We report here detailed results for the experiments discussed in Sect. 8. The tables follow the same order in which they are discussed in the paper. In order to facilitate comparison, in all tables below we omit rows that correspond to very similar results

[1] Computed as a geometric average by taking into account instances for which all heuristics find a feasible solution.

[2] Computed on instances for which CPU time for the Feasibility Pump is known.

and retain those where different methods obtain significantly different results, in terms of objective function value or CPU time. In each row, if one column has an objective value that is strictly better than the other ones, it is highlighted in boldface.

Number of rounding iterations

In Table 6 we report detailed results for the experiments discussed in Sect. 8.2. For each instance, we record the objective value of the solution returned by the heuristics (or "–" if none is found) and the corresponding CPU time in seconds. For clarity, we omit the results on those instances for which the objective function is the same for all three variants. Note that for all of these eliminated rows the CPU time increases with the number of iterations, and follows the same pattern shown in the table: for $MaxIter = 20$, the CPU time is between two and five times that required for $MaxIter = 5$.

Feasibility-based Iterative Rounding

In Table 7 we report detailed results for the experiments discussed in Sect. 8.3. Again, the results on those instances whose objective function is independent on ω are omitted. This time, however, the CPU time for all eliminated rows is also independent of ω.

Improvement-based Iterative Rounding

Table 8 presents detailed results for the experiments in Sect. 8.4, where we apply I-IR varying the value k'. Recall that k' is involved in the computation of the local branching constraint (8) because we pick $k = \min\{k', \max\{1, |N_I|/2\}\} + \delta - 1$, where δ is defined as in (7). In the last two columns, we report results obtained by applying F-IR only (i.e. the improvement heuristic I-IR is not employed) for comparison. The column labeled "# impr" indicates the number of times that I-IR was able to improve the solution on each instance; since the heuristic is applied as long as the solution keeps improving, this number can be greater than one. We omit all rows of instances where the resulting objective value is the same.

Comparison with existing heuristics

In Table 9 we report detailed results of the experiments discussed in Sect. 8.5. This table shows results for all instances, as each heuristic exhibits a radically different behavior on each instance. The results with the Feasibility Pump are taken from [9], the results with RECIPE are taken from [19]. Note that in [9], CPU times are rounded to the nearest second, and no more than two digits after the decimal point are reported for objective function values; therefore, we do the same for all the results. In the last column of Table 9, we report the best known solution for each instance, as obtained from the MINLPLib website. For the instances where the Feasibility Pump could not find a feasible solution, CPU time is not available and is indicated by "–". The experiments with RECIPE reported in [19] have a time limit of 2 h per instance. Whenever this time limit is hit, we report ">2 h" as CPU time, and we assume a value of 7,200 s when computing the average CPU time.

Table 6 Results obtained at the root node by applying F-IR followed by I-IR as long as the solution keeps improving, or up to the time limit

Instance	*MaxIter* = 5		*MaxIter* = 10		*MaxIter* = 20	
	Obj	Time	Obj	Time	Obj	Time
ex1243	125,705	1.47	125,705	3.11	**98,519**	15.00
ex1263a	26	16.24	**22**	45.28	24	39.14
ex1264	19	18.86	17	30.26	**11.1**	10.19
ex1266	24.5	51.11	23.5	81.38	**22.5**	167.20
fac3	3.6423×10^7	1.20	3.1982×10^7	4.49	3.1982×10^7	8.51
fo7_2	28.4047	102.31	**23.4379**	120.45	28.4047	215.71
fo7_ar2_1	28.0401	100.66	37.3996	101.65	**27.8046**	305.01
fo7_ar25_1	37.8891	51.02	37.8891	88.20	**28.9773**	385.30
fo7_ar3_1	34.3721	86.86	26.2713	162.68	**24.2226**	365.22
fo7_ar4_1	28.4095	41.73	25.8207	117.90	25.8207	185.88
fo7	28.091	52.23	**22.763**	249.44	22.8167	336.14
fo8_ar2_1	39.3033	86.70	38.7744	207.75	**35.9943**	398.97
fo8_ar25_1	39.3741	82.94	37.2123	132.50	**30.6629**	229.62
fo8_ar3_1	45.2436	54.33	**39.5087**	176.17	40.4129	359.81
fo8_ar4_1	42.1725	128.85	33.3922	105.06	**28.4667**	275.97
fo8_ar5_1	40.0166	54.21	37.2742	155.56	**34.1368**	321.90
fo9_ar2_1	47.485	237.99	46.984	269.30	**43.375**	946.19
fo9_ar25_1	50.841	150.54	41.5975	299.90	**37.4879**	322.70
fo9_ar3_1	**42.75**	99.06	42.7649	181.63	48.3489	392.62
fo9_ar4_1	52.5276	121.46	**38.7566**	120.81	41.3571	411.51
fo9_ar5_1	59.3221	75.90	**37.2903**	172.79	59.3221	175.29
fo9	42.4015	45.20	**40.75**	108.78	42.4015	156.63
lop97icx	4327.66	64.61	**4323.73**	66.03	4342.86	142.10
m7_ar2_1	210.403	63.72	**190.235**	205.05	195.035	325.34
m7_ar25_1	143.585	71.45	163.185	192.47	143.585	169.24
m7_ar3_1	228.403	59.41	228.403	76.02	**164.89**	194.40
m7_ar5_1	182.035	84.56	**136.871**	115.31	174.598	259.12
netmod_dol1	−0.454748	85.28	−0.454748	110.84	**−0.47079**	465.96
no7_ar2_1	122.364	56.58	131.871	113.81	122.364	176.68
no7_ar25_1	132.664	76.00	133.021	96.06	**124.238**	309.47
no7_ar3_1	139.184	66.02	**115.116**	145.69	139.184	155.75
no7_ar4_1	151.855	67.45	134.18	139.67	**108.513**	463.26
no7_ar5_1	119.726	98.05	104.558	96.14	**103.063**	330.50
nuclear14b	−1.09391	115.32	−1.11081	161.68	**−1.11676**	584.44
nuclear24b	−1.10724	96.98	−1.11563	295.50	**−1.11697**	408.52
nuclear25b	−1.09687	107.59	−1.08116	301.90	**−1.10294**	842.22
nuclearva	–	137.77	−1.01174	128.74	−1.01174	242.82
nuclearve	−1.02878	74.31	−1.02883	130.44	−1.02883	244.22

Table 6 continued

Instance	*MaxIter* = 5		*MaxIter* = 10		*MaxIter* = 20	
	Obj	Time	Obj	Time	Obj	Time
o7_2	140.826	88.45	139.345	160.09	139.345	215.88
o7_ar2_1	169.798	90.98	161.528	135.93	**143.132**	479.97
o7_ar25_1	160.693	70.60	159.493	72.14	**142.229**	253.59
o7_ar3_1	163.234	138.90	161.471	145.39	**158.822**	298.08
o7_ar4_1	150.918	71.34	**144.108**	135.93	150.918	151.73
o7_ar5_1	**143.306**	65.72	152.821	85.88	143.336	129.79
o7	157.671	63.28	157.671	93.74	**148.1**	326.86
o8_ar4_1	304.773	117.88	341.553	109.08	**301.287**	206.16
o9_ar4_1	320.375	102.31	**289.404**	102.30	291.821	270.65
tln12	**305.8**	147.57	378.8	119.29	367.8	222.11
tln5	17.7	44.41	18.5	91.69	**16.5**	105.60
tln6	**26.1**	72.25	–	136.75	–	360.25
tls2	12.3	0.72	10.3	2.55	**8.3**	6.55
tls4	18.6	12.02	17.6	27.04	17.6	44.39
tls6	41.1	62.40	41.1	137.12	**40.1**	206.15
tls7	**47.8**	82.05	55.8	275.18	48.8	260.23
tltr	129.412	30.17	68.2792	75.89	**61.1333**	126.60
var_con10	452.69	15.44	444.214	56.97	444.214	109.76
var_con5	285.874	9.28	285.874	29.39	**278.145**	158.59

Table 7 Results obtained at the root node by applying F-IR with $h = 5$ and different values of ω

Instance	$\omega = 0.05$		$\omega = 0.10$		$\omega = 0.20$		$\omega = 0.25$	
	Obj	Time	Obj	Time	Obj	Time	Obj	Time
ex1263a	24	26.11	24	24.74	26	26.45	27	26.20
ex1264	–	43.63	–	43.46	**17**	26.38	18	23.37
ex1265a	–	94.50	**18.5**	94.98	22.5	49.10	20.5	26.74
ex1266a	22.5	150.49	**20.5**	54.61	22.5	35.24	22.5	40.79
ex1266	–	187.83	**22.5**	160.05	23.5	71.82	23.5	74.13
fo8_ar2_1	41.2522	111.39	42.1832	16.75	47.0345	20.98	**40.6316**	71.28
fo8_ar25_1	40.8052	19.50	**38.7791**	23.62	38.9245	26.25	43.9553	20.63
fo8_ar4_1	52.8443	66.36	54.1692	6.51	54.1692	6.49	52.8443	66.30
fo9_ar2_1	55	79.72	**38.6**	20.68	47.9808	15.47	51.875	158.68
fo9_ar25_1	52.75	20.82	52.971	21.21	52.971	46.18	**48.6825**	15.39
fo9_ar3_1	70.0139	15.64	54.5696	15.51	64.1514	10.03	**45.4966**	20.11
m7_ar2_1	281.177	5.38	281.177	5.37	281.177	5.41	281.177	5.41
m7_ar25_1	187.185	31.00	224.833	30.68	187.185	30.85	224.833	30.72
m7_ar3_1	254.499	11.19	254.499	11.12	254.499	11.20	254.499	11.18
m7_ar4_1	148.62	7.71	148.62	7.70	148.62	7.72	148.62	7.71

Table 7 continued

Instance	$\omega = 0.05$		$\omega = 0.10$		$\omega = 0.20$		$\omega = 0.25$	
	Obj	Time	Obj	Time	Obj	Time	Obj	Time
m7_ar5_1	257.513	42.47	190.33	20.61	190.33	20.76	257.513	42.93
m7	200.93	7.35	200.93	7.39	200.93	7.41	200.93	7.40
no7_ar3_1	139.184	15.08	127.53	14.64	127.53	14.75	139.184	15.27
nuclear14a	**−1.12967**	12.01	−1.11997	12.92	−1.11997	12.94	−1.11997	11.97
nuclear14b	−1.09088	77.30	−1.06989	37.15	**−1.11081**	68.74	−1.10362	53.29
nuclear24a	−1.12967	11.93	−1.11997	12.80	−1.11997	12.92	−1.12967	12.04
nuclear24b	−1.04799	54.45	−1.06989	36.20	−1.09321	77.28	**−1.10362**	51.99
nuclear25a	–	299.30	–	302.67	–	299.23	–	300.39
nuclear25b	−1.05817	53.58	−1.06736	53.74	−1.06736	38.89	−1.06736	39.30
nuclearva	–	224.83	–	226.01	**−1.01174**	128.61	–	296.19
nuclearvd	−1.03371	5.45	−1.03336	4.40	−1.03336	4.43	−1.03371	5.37
nuclearve	–	220.97	–	218.89	**−1.02719**	128.95	−1.02704	75.82
nuclearvf	–	226.81	–	226.97	−1.0105	131.76	−1.0105	79.89
nvs17	**−1,097**	10.91	−1,086.2	43.26	−1,086.2	21.43	−1,086.2	21.15
nvs18	−771	0.42	−771	0.41	−771	0.45	−771	0.44
nvs19	−1,068.6	13.78	−1,095.4	27.77	−1,095.4	14.25	**−1,096.8**	42.49
nvs23	**−1,119.6**	19.20	−1,118.2	18.88	−1,109.4	19.42	−1,101.2	19.03
nvs24	**−1,029.6**	26.26	−1,021	25.67	−1,026.2	26.53	−1,023.4	25.98
o7_ar2_1	**145.319**	26.12	177.836	5.05	177.836	5.04	177.836	5.04
o7_ar3_1	166.636	10.26	188.998	51.23	191.027	15.11	**163.013**	68.21
o9_ar4_1	345.941	15.58	297.202	30.88	**289.404**	15.50	326.648	35.55
tln12	364.8	63.75	**294.8**	115.02	378.8	70.36	353.8	80.84
tln2	5.3	0.17	5.3	0.18	5.3	0.18	5.3	0.17
tln4	–	49.94	–	48.38	–	52.52	–	52.80
tln5	–	90.74	–	81.90	**18.5**	80.38	21.5	84.79
tloss	23.1	165.63	23.1	63.45	23.1	35.36	26.1	65.01
tls7	49.8	172.55	55.8	60.74	55.8	87.66	**44.8**	87.11
tltr	–	105.98	68.075	83.89	74.8125	44.17	68.075	21.90

Table 8 Results obtained at the root node by applying F-IR with $h = 5$ and $\omega = 0.2$, followed by I-IR with different valued of k'

Instance	$k' = 10$			$k' = 15$			$k' = 20$			F-IR only	
	Obj	Time	# Impr	Obj	Time	# Impr	Obj	Time	# Impr	Obj	Time
elf	0.19166	2.02	4	0.19166	1.54	3	0.19166	1.53	3	1.67499	0.31
ex1263a	23	37.99	1	22	45.28	1	22	44.65	1	26	26.45
ex1264	**11.6**	33.11	1	17	30.26	0	17	30.95	0	17	26.38
fac3	3.1982×10^7	4.48	1	3.1982×10^7	4.49	1	3.1982×10^7	4.50	1	3.6423×10^7	0.21

Table 8 continued

Instance	$k' = 10$			$k' = 15$			$k' = 20$			F-IR only	
	Obj	Time	# Impr	Obj	Time	# Impr	Obj	Time	# Impr	Obj	Time
fo7_2	29.0094	105.89	2	23.4379	120.45	3	**17.7493**	87.27	1	36.5729	7.39
fo7_ar2_1	33.2223	113.17	1	37.3996	101.65	1	**28.0401**	116.88	1	39.3889	30.46
fo7_ar25_1	31.1553	135.02	3	37.8891	88.20	0	**28.0401**	108.33	2	37.8891	7.23
fo7_ar3_1	35.3501	66.93	0	**26.2713**	162.68	3	27.8046	103.94	2	35.3501	7.36
fo7_ar4_1	28.4095	76.74	0	**25.8207**	117.90	1	27.8046	94.40	1	28.4095	7.68
fo7	28.091	161.65	3	**22.763**	249.44	5	24.3794	187.25	6	34.7516	7.37
fo8_ar2_1	39.1862	131.03	2	38.7744	207.75	1	**38.1266**	137.07	1	47.0345	20.98
fo8_ar25_1	43.9553	105.58	0	**37.2123**	132.50	2	41.0169	148.48	1	38.9245	26.25
fo8_ar3_1	40.7348	164.14	1	**39.5087**	176.17	2	43.5048	212.85	1	45.2436	6.01
fo8_ar4_1	**31.6739**	186.01	2	33.3922	105.06	2	37.7757	168.06	3	54.1692	6.49
fo8_ar5_1	38.1066	90.68	1	37.2742	155.56	2	**31.2631**	117.66	1	41.4533	7.38
fo8	34.0253	96.36	1	**32.425**	159.43	3	32.4705	111.62	2	38.0277	7.54
fo9_ar2_1	52.1968	301.31	2	46.984	269.30	1	**43.375**	302.97	1	47.9808	15.47
fo9_ar25_1	43.8228	154.31	1	41.5975	299.90	3	**39.1346**	227.23	2	52.971	46.18
fo9_ar3_1	**41.7981**	179.24	1	42.7649	181.63	2	47.0615	147.00	2	64.1514	10.03
fo9_ar4_1	44.3698	261.78	4	**38.7566**	120.81	2	63.3674	104.12	1	66.732	30.14
fo9_ar5_1	**36.5714**	215.14	3	37.2903	172.79	4	40.9286	167.00	2	63.2228	23.40
fo9	42.4015	82.53	0	**40.75**	108.78	1	42.4015	82.22	0	42.4015	7.52
lop97icx	4,354.65	108.30	2	4,323.73	66.03	1	**4,291.36**	229.57	9	4,471.64	5.19
m3	37.8	2.65	1	37.8	2.68	1	37.8	2.64	1	46.3063	0.14
m6	82.2569	89.35	1	82.2569	79.35	1	82.2569	79.42	1	106.257	5.36
m7_ar2_1	196.028	86.58	2	**190.235**	205.05	3	195.035	88.66	1	281.177	5.41
m7_ar25_1	172.86	130.02	1	163.185	192.47	3	**153.26**	183.04	2	187.185	30.85
m7_ar3_1	**152.579**	172.28	3	228.403	76.02	1	180.504	183.86	4	254.499	11.20
m7_ar5_1	190.778	206.64	3	**136.871**	115.31	2	190.778	114.25	1	190.33	20.76
m7	106.757	113.76	2	106.757	143.43	4	130.757	124.89	3	200.93	7.41
netmod_dol1	**−0.475417**	161.79	13	−0.454748	110.84	8	−0.468264	178.16	10	−0.225944	5.20
netmod_dol2	−0.463025	192.19	7	**−0.475425**	91.29	3	−0.474371	64.00	1	−0.445448	6.00
netmod_kar1	−0.39908	25.39	3	−0.39908	37.58	3	**−0.402038**	55.99	4	−0.344592	5.89
netmod_kar2	−0.39908	25.38	3	−0.39908	37.46	3	**−0.402038**	56.15	4	−0.344592	5.91
no7_ar2_1	132.283	82.83	0	**131.871**	113.81	1	132.283	79.82	0	132.283	5.72
no7_ar25_1	**118.639**	110.15	1	133.021	96.06	1	129.208	182.51	4	166.026	15.05
no7_ar3_1	**112.912**	172.73	2	115.116	145.69	2	132.3	94.30	1	127.53	14.75
no7_ar4_1	**118.372**	206.32	2	134.18	139.67	3	142.203	121.05	2	151.855	32.86
no7_ar5_1	110.038	149.68	5	104.558	96.14	1	**104.194**	78.80	2	138.871	7.34
nuclear14a	−1.12967	12.18	0	−1.12967	40.46	2	−1.12054	300.50	1	−1.11997	12.94
nuclear14b	**−1.11908**	256.29	5	−1.11081	161.68	0	−1.10724	242.27	3	−1.11081	68.74
nuclear24a	−1.12967	109.68	2	−1.12967	40.34	2	−1.12054	298.99	1	−1.11997	12.92
nuclear24b	−1.10593	236.22	3	**−1.11563**	295.50	8	−1.10362	110.43	0	−1.09321	77.28
nuclear25b	**−1.10039**	298.08	4	−1.08116	301.90	4	−1.07008	168.18	1	−1.06736	38.89
nuclearva	**−1.01183**	130.22	0	−1.01174	128.74	0	−1.01174	134.63	0	−1.01174	128.61

Table 8 continued

Instance	$k' = 10$			$k' = 15$			$k' = 20$			F-IR only	
	Obj	Time	# Impr	Obj	Time	# Impr	Obj	Time	# Impr	Obj	Time
nuclearvd	−1.03371	5.55	0	−1.03369	5.26	1	−1.03371	5.55	0	−1.03336	4.43
nuclearve	−1.02804	137.28	2	**−1.02883**	130.44	2	−1.02814	136.34	1	−1.02719	128.95
nuclearvf	−1.0132	137.68	2	−1.0132	133.34	2	−1.0132	129.90	1	−1.0105	131.76
o7_2	145.38	99.53	4	139.345	160.09	6	**138.154**	128.91	4	183.735	5.05
o7_ar2_1	**157.76**	124.51	1	161.528	135.93	1	169.847	92.24	1	177.836	5.04
o7_ar25_1	**154.531**	118.17	3	159.493	72.14	2	158.342	91.23	2	179.303	5.23
o7_ar3_1	**154.53**	102.70	1	161.471	145.39	3	155.774	130.88	1	191.027	15.11
o7_ar4_1	154.174	113.17	2	**144.108**	135.93	2	176.182	80.65	2	181.841	16.16
o7_ar5_1	**134.563**	71.46	1	152.821	85.88	1	143.135	77.42	1	196.798	10.41
o7	**147.867**	92.78	2	157.671	93.74	3	148.536	84.20	1	171.283	5.14
o8_ar4_1	313.08	146.60	1	341.553	109.08	0	**309.257**	118.03	1	341.553	49.83
o9_ar4_1	**285.153**	177.65	4	289.404	102.30	0	317.75	131.51	0	289.404	15.50
synheat	154,997	11.22	1	154,997	11.24	1	154,997	11.16	1	196,206	0.58
tln12	407.8	120.34	0	**378.8**	119.29	0	407.8	124.32	0	378.8	70.36
tls2	10.3	2.58	4	10.3	2.55	4	10.3	2.48	4	14.3	0.05
tls4	19.6	12.28	0	**17.6**	27.04	2	18.6	31.14	1	19.6	1.95
tls7	**46.8**	113.36	0	55.8	275.18	0	47.1	299.51	1	55.8	87.66
tltr	74.8125	57.54	0	**68.2792**	75.89	1	74.8125	60.56	0	74.8125	44.17
var_con10	444.214	60.03	1	444.214	56.97	1	444.214	59.33	1	452.69	1.56

Table 9 Comparison between the Iterative Rounding heuristic applied at the root node with a time limit of 300 s, the Feasibility Pump for nonconvex MINLPs, and RECIPE

	Iterative Rounding		Feasibility Pump		RECIPE		Best
Instance	Obj	Time	Obj	Time	Obj	Time	Known
batchdes	–	1	228,396.00	0	**167,427.67**	2	167,428.00
cecil_13	**−115,657.00**	59	–	–	−114,379.07	>2 h	−115,570.00
contvar	–	252.40	19,442,300.00	608	**809,149.54**	>2 h	769,977.00
csched1	–	13	−21,049.20	0	**−30,639.26**	156	−30,639.30
deb10	**209.43**	8	–	–	–	674	209.43
deb6	–	505	**237.10**	4	–	1,217	201.74
deb7	–	413	**369.83**	13	–	4,544	116.58
deb8	–	405	**1,451,450.00**	2	–	4,482	116.58
deb9	–	533	**426.57**	18	–	4,920	116.58
detf1	–	300	**15,976.00**	128	–	0	12.16
eg_disc2_s	–	303	100,004.00	7	**5.64**	554	5.64
eg_disc_s	–	303	100,005.00	9	**5.76**	635	5.76
eg_int_s	–	2	–	–	**6.45**	196	6.45

Table 9 continued

	Iterative Rounding		Feasibility Pump		RECIPE		Best
Instance	Obj	Time	Obj	Time	Obj	Time	Known
elf	0.19	1	2,399,200.00	0	0.19	165	0.19
eniplac	–	23	−102,095.00	2	**−132,117.03**	>2 h	−132,117.00
enpro48	–	296	1,642,170.00	609	**187,277.16**	40	187,277.00
enpro48pb	–	297	1,642,180.00	610	**187,277.16**	560	187,277.00
enpro56	–	152	707,303.00	607	**263,428.36**	62	263,428.00
enpro56pb	–	152	707,303.00	607	**263,428.36**	2,515	263,428.00
ex1224	−0.94	0	−0.30	0	−0.94	2	−0.94
ex1225	–	0	34.00	0	**31.00**	1	31.00
ex1233	200,571.00	3	253,382.00	1	**155,010.67**	0	155,011.00
ex1243	125,705.00	3	168,498.00	0	**83,402.51**	10	83,402.50
ex1244	96,380.60	10	95,415.20	0	**82,042.91**	102	82,042.90
ex1263a	22.00	45	31.00	1	**19.60**	7	19.60
ex1263	–	61	121.00	66	**19.60**	233	19.60
ex1264a	12.00	4	12.00	0	**8.60**	5	8.60
ex1264	17.00	30	18.30	29	**8.60**	35	8.60
ex1265a	22.50	54	16.50	2	**11.50**	10	10.30
ex1265	–	88	**12.30**	158	15.10	8	10.30
ex1266a	22.50	58	**10.30**	0	16.30	5	16.30
ex1266	23.50	81	34.60	628	**16.30**	19	16.30
ex4	–	14	2,556,590.00	0	**−8.06**	21	−8.06
fac2	331,837,000.00	4	1,951,970,000.00	1	331,837,498.18	7	331,838,000.00
fac3	31,982,300.00	4	104,966,000.00	0	31,982,309.85	9	31,982,300.00
feedtray	–	1	−12.41	0	**−13.41**	46	−13.41
fo7_2	23.44	120	1,200,000.00	11	**17.75**	4,169	17.75
fo7_ar2_1	37.40	101	–	–	**24.84**	>2 h	24.84
fo7_ar25_1	37.89	88	1,199,990.00	3,066	**23.09**	>2 h	23.09
fo7_ar3_1	26.27	162	1,200,000.00	8	**22.52**	>2 h	22.52
fo7_ar4_1	25.82	117	1,200,000.00	141	**20.73**	>2 h	20.73
fo7_ar5_1	17.75	74	1,200,000.00	7	17.75	>2 h	17.75
fo7	22.76	249	–	–	**20.73**	>2 h	20.73
fo8_ar2_1	38.77	207	–	–	**30.34**	>2 h	30.34
fo8_ar25_1	37.21	132	–	–	**33.31**	>2 h	28.05
fo8_ar3_1	39.51	176	–	–	**30.57**	>2 h	23.91
fo8_ar4_1	**33.39**	105	1,399,990.00	430	37.04	>2 h	22.38
fo8_ar5_1	37.27	155	1,399,990.00	13	**23.91**	>2 h	22.38
fo8	32.42	159	1,400,000.00	1,330	**22.38**	>2 h	22.38
fo9_ar2_1	46.98	269	–	–	**36.67**	>2 h	32.62
fo9_ar25_1	**41.60**	299	–	–	54.08	>2 h	32.19
fo9_ar3_1	42.76	181	1,599,990.00	417	**37.01**	>2 h	24.82

Table 9 continued

	Iterative Rounding		Feasibility Pump		RECIPE		Best
Instance	Obj	Time	Obj	Time	Obj	Time	Known
fo9_ar4_1	**38.76**	120	1,599,990.00	3,986	45.42	>2 h	23.46
fo9_ar5_1	**37.29**	172	1,599,990.00	15	–	>2 h	23.46
fo9	**40.75**	108	1,600,000.00	196	–	>2 h	23.46
gastrans	89.09	4	–	–	89.09	19	89.09
gear3	0.00	0	0.73	0	0.00	1	0.00
hmittelman	13.00	0	21.00	0	13.00	4	13.00
lop97ic	–	298	–	–	**4,232.69**	>2 h	4,284.59
lop97icx	4,323.73	66	–	–	**4,099.06**	5,381	4,099.06
m3	37.80	2	2,400,000.00	0	37.80	5	37.80
m6	82.26	79	6,480,000.00	0	82.26	1,414	82.26
m7_ar2_1	190.24	205	7,880,030.00	1	190.24	>2 h	190.24
m7_ar25_1	163.19	192	7,879,970.00	1	**143.59**	2,919	143.59
m7_ar3_1	228.40	76	7,880,030.00	0	**143.59**	>2 h	143.59
m7_ar4_1	148.62	71	7,880,040.00	0	**106.76**	>2 h	106.76
m7_ar5_1	136.87	115	7,880,040.00	0	**106.46**	>2 h	106.46
m7	106.76	143	7,880,000.00	0	106.76	2,251	106.76
netmod_dol1	**−0.45**	110	–	–	–	>2 h	−0.56
netmod_dol2	−0.48	91	–	–	**−0.55**	>2 h	−0.56
netmod_kar1	−0.40	37	–	–	**−0.42**	>2 h	−0.42
netmod_kar2	−0.40	37	–	–	**−0.42**	>2 h	−0.42
no7_ar2_1	131.87	113	–	–	**107.82**	>2 h	107.81
no7_ar25_1	133.02	96	3,999,990.00	2,986	**107.82**	>2 h	107.81
no7_ar3_1	115.12	145	3,999,980.00	14	**107.82**	>2 h	107.81
no7_ar4_1	134.18	139	3,999,990.00	138	**98.52**	>2 h	98.52
no7_ar5_1	104.56	96	3,999,990.00	7	**90.62**	>2 h	90.62
nuclear14a	−1.13	40	−1.11	1,839	−1.13	2,732	−1.13
nuclear14b	−1.11	161	−1.10	670	−1.11	>2 h	−1.11
nuclear24a	−1.13	40	−1.11	1,826	−1.13	2,769	−1.13
nuclear24b	**−1.12**	295	−1.10	668	−1.11	>2 h	−1.11
nuclear25a	–	299	**−1.09**	902	–	0	−1.12
nuclear25b	−1.08	301	−1.08	627	**−1.10**	>2 h	−1.10
nuclear49a	–	299	–	–	–	>2 h	−1.15
nuclear49b	–	372	–	–	–	0	−1.11
nuclearva	−1.01	128	−1.01	132	−1.01	1,323	−1.01
nuclearvc	–	240	−0.99	119	**−1.00**	1,370	−1.00
nuclearvd	−1.03	5	−1.03	424	−1.03	1,806	−1.03
nuclearve	−1.03	130	−1.03	406	−1.03	2,013	−1.04
nuclearvf	−1.01	133	−1.02	373	−1.02	1,911	−1.02
nvs03	17.00	0	16.00	0	16.00	1	16.00

Table 9 continued

	Iterative Rounding		Feasibility Pump		RECIPE		Best
Instance	Obj	Time	Obj	Time	Obj	Time	Known
nvs10	−310.80	0	−102.40	0	−310.80	1	−310.80
nvs12	−481.20	0	−188.00	0	−481.20	1	−481.20
nvs13	−573.80	0	−166.40	0	**−585.20**	1	−585.20
nvs15	–	0	1.00	0	1.00	1	1.00
nvs17	−1,086.20	21	−279.00	0	**−1,100.40**	2	−1,100.40
nvs18	−771.00	0	−209.00	0	**−778.40**	1	−778.40
nvs19	−1,095.40	14	−282.40	0	**−1,098.40**	2	−1,098.40
nvs23	−1,109.40	19	−454.80	1	**−1,125.20**	4	−1,125.20
nvs24	−1,026.20	26	−536.20	1	**−1,033.20**	4	−1,033.20
o7_2	139.34	160	4,800,000.00	10	**116.95**	>2 h	116.95
o7_ar2_1	161.53	135	–	–	**140.41**	>2 h	140.41
o7_ar25_1	159.49	72	4,799,980.00	2,987	**141.62**	>2 h	140.41
o7_ar3_1	161.47	145	4,799,980.00	7	**138.86**	>2 h	137.93
o7_ar4_1	144.11	135	4,799,990.00	136	**131.65**	>2 h	131.65
o7_ar5_1	152.82	85	4,799,990.00	8	**116.95**	>2 h	116.95
o7	157.67	93	–	–	**131.65**	>2 h	131.65
o8_ar4_1	341.55	109	8,199,970.00	622	**254.98**	>2 h	243.07
o9_ar4_1	**289.40**	102	8,199,960.00	3,953	316.01	>2 h	236.14
oil2	–	9	–	–	–	0	−0.73
ortez	−9,532.04	39	−0.39	0	−9,532.04	68	−9,532.04
parallel	–	104	4×10^{10}	0	**924.30**	1	924.30
prob10	–	0	4.79	0	**3.45**	1	3.45
qap	–	300	499,512.00	0	**388,870.00**	1,541	388,214.00
qapw	–	300	460,118.00	610	**388,988.00**	4,410	388,214.00
ravem	269,590.00	28	764,412.00	171	269,590.27	2	269,590.00
ravempb	269,590.00	27	764,412.00	185	269,590.27	57	269,590.00
saa_2	–	301	–	–	–	0	12.78
space25	485.57	50	650.69	446	**484.33**	1,146	484.33
space960	–	300	–	–	–	>2 h	7,610,310.00
spectra2	–	10	304.79	0	**13.98**	228	13.98
st_e29	−0.94	0	−0.30	0	−0.94	2	−0.94
st_e31	−2.00	0	−0.42	1	−2.00	24	−2.00
st_e32	–	0	−1.43	0	−1.43	0	−1.43
st_e35	–	6	**132,703.00**	0	–	0	64,868.10
st_e40	–	0	–	–	**30.41**	4	30.41
st_miqp2	–	0	7.00	0	**2.00**	7	2.00
st_test2	−9.25	0	0.00	0	−9.25	1	−9.25
st_test3	−7.00	0	0.00	0	−7.00	1	−7.00
st_test5	−110.00	0	−110.00	0	−110.00	1	−110.00

Table 9 continued

	Iterative Rounding		Feasibility Pump		RECIPE		Best
Instance	Obj	Time	Obj	Time	Obj	Time	Known
st_test6	471.00	0	567.00	0	471.00	1	471.00
st_test8	−29,605.00	0	24,728.00	0	−29,605.00	2	−29,605.00
st_testgr3	−20.06	2	0.00	0	**−20.59**	2	−20.59
synheat	154,997.00	11	248,724.00	0	154,997.33	11	154,997.00
tln12	378.80	119	–	–	**106.80**	>2 h	90.50
tln2	5.30	0	28.30	1	5.30	7	5.30
tln4	–	52	12.00	1	**8.30**	541	8.30
tln5	18.50	91	16.50	0	**10.30**	442	10.30
tln6	–	136	25.10	1	**15.40**	>2 h	15.30
tln7	–	212	107.80	1,072	**15.60**	>2 h	15.00
tloss	23.10	62	24.10	5	**16.30**	5	16.30
tls12	–	302	–	–	–	>2 h	–
tls2	10.30	2	5.30	1	5.30	7	5.30
tls4	17.60	27	10.00	22	**8.80**	>2 h	8.30
tls6	41.10	137	–	–	**16.40**	>2 h	16.60
tls7	**55.80**	275	–	–	–	>2 h	27.30
tltr	68.28	75	**48.07**	0	–	2	48.07
uselinear	–	531	**1,951.37**	51	–	816	−1,050.34
var_con10	444.21	56	463.17	10	444.10	2,550	444.21
var_con5	285.87	29	315.17	7	**278.04**	2,293	278.14
water4	945.85	31	3,335,260.00	5	**907.02**	298	913.16
waterz	–	127	3,355,460.00	3	**910.88**	>2 h	1,037.64

In the last column we report the best solution known for the instance as reported on the MINLPLib website

References

1. Balas, E., Jeroslow, R.: Canonical cuts on the unit hypercube. SIAM J. Appl. Math. **23**(1), 61–69 (1972)
2. Belotti, P.: Couenne: a user's manual. Tech. Rep., Lehigh University (2009). http://www.coin-or.org/Couenne
3. Belotti, P., Lee, J., Liberti, L., Margot, F., Wächter, A.: Branching and bounds tightening techniques for non-convex MINLP. Optim. Methods Softw. **24**(4–5), 597–634 (2008)
4. Biegler, L., Grossmann, I., Westerberg, A.: Systematic Methods of Chemical Process Design. Prentice Hall, Upper Saddle River (NJ) (1997)
5. Bonami, P., Cornuéjols, G., Lodi, A., Margot, F.: A feasibility pump for Mixed Integer Nonlinear Programs. Math. Program. **119**(2), 331–352 (2009)
6. Bonami, P., Gonçalves, J.: Primal heuristics for mixed-integer nonlinear programs. Tech. Rep. RC24639, IBM (2008)
7. Bussieck, M.R., Drud, A.S., Meeraus, A.: MINLPLib—a collection of test models for Mixed-Integer Nonlinear Programming. INFORMS J. Comput. **15**(1) (2003). http://www.gamsworld.org/minlp/minlplib.htm
8. D'Ambrosio, C.: Application oriented Mixed Integer Nonlinear Programming. Ph.D. thesis, DEIS, Università di Bologna (2009)

9. D'Ambrosio, C., Frangioni, A., Liberti, L., Lodi, A.: Experiments with a Feasibility Pump approach for nonconvex MINLPs. In: Festa, P. (ed.) Proceedings of the 9th Symposium on Experimental Algorithms (SEA 2010), Lecture Notes in Computer Science, vol. 6049. Springer, Berlin (2010)
10. D'Ambrosio, C., Frangioni, A., Liberti, L., Lodi, A.: On interval-subgradient and no-good cuts. Oper. Res. Lett. **38**(5), 341–345 (2010)
11. Danna, E., Rothberg, E., Le Pape, C.: Exploring relaxation induced neighborhoods to improve MIP solutions. Math. Program. A **102**, 71–90 (2005)
12. Fischetti, M., Glover, F., Lodi, A.: The feasibility pump. Math. Program. A **104**(1), 91–104 (2005)
13. Fischetti, M., Lodi, A.: Local branching. Math. Program. **98**, 23–37 (2003)
14. Floudas, C.: Global optimization in design and control of chemical process systems. J. Process Control **10**, 125–134 (2001)
15. Glover, F.W.: Tabu search—part I. ORSA J. Comput. **1**(3), 190–206 (1989)
16. Hansen, P., Mladenović, N.: Variable neighbourhood search: principles and applications. Eur. J. Oper. Res. **130**, 449–467 (2001)
17. IBM ILOG: IBM ILOG CPLEX 12.1 User's Manual. IBM ILOG, Gentilly, France (2010)
18. Kilinç Karzan, F., Nemhauser, G.L., Savelsbergh, M.W.P.: Information-based branching schemes for binary linear mixed-integer programs. Math. Program. Comput. **1**, 249–293 (2009)
19. Liberti, L., Mladenović, N., Nannicini, G.: A good recipe for solving MINLPs. Math. Program. Comput. (2011). doi:10.1007/s12532-011-0031-y
20. Lindo Systems: LINDO Solver Suite: user manual. http://www.gams.com/solvers/lindoglobal.pdf
21. McCormick, G.: Computability of global solutions to factorable nonconvex programs: Part i — convex underestimating problems. Math. Program. **10**, 146–175 (1976)
22. Nannicini, G., Belotti, P.: Rounding-based heuristics for nonconvex MINLPs. In: Bonami, P., Liberti, L., Miller, A., Sartenaer, A. (eds.) Proceedings of the European Workshop on MINLP. CIRM, Marseille, France (2010)
23. Nannicini, G., Belotti, P., Liberti, L.: A local branching heuristic for MINLPs. Tech. Rep. 0812.2188 (2008). http://arxiv.org/abs/0812.2188
24. Nemhauser, G., Wolsey, L.: Integer and Combinatorial Optimization. Wiley, New York (1988)
25. Sahinidis, N.: BARON: a general purpose global optimization software package. J. Glob. Optim. **8**(2), 201–205 (1996)
26. Shectman, J., Sahinidis, N.: A finite algorithm for global minimization of separable concave programs. J. Glob. Optim. **12**, 1–36 (1998)
27. Smith, E.: On the optimal design of continuous processes. Ph.D. thesis, Imperial College of Science, Technology and Medicine, University of London (1996)
28. Smith, E., Pantelides, C.: A symbolic reformulation/spatial branch-and-bound algorithm for the global optimisation of nonconvex MINLPs. Comput. Chem. Eng. **23**, 457–478 (1999)
29. Tawarmalani, M., Sahinidis, N.: Global optimization of mixed integer nonlinear programs: a theoretical and computational study. Math. Program. **99**, 563–591 (2004)
30. Wächter, A., Biegler, L.T.: On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. Math. Program. **106**(1), 25–57 (2006)
31. Wolsey, L.: Integer Programming. Wiley, New York (1998)