Noname manuscript No.
(will be inserted by the editor)

A recipe for finding good solutions to MINLPs

Leo Liberti · Nenad Mladenović · Giacomo Nannicini

the date of receipt and acceptance should be inserted later

Abstract Finding good (or even just feasible) solutions for Mixed-Integer Nonlinear Programming problems independently of the specific problem structure is a very hard but practically important task, especially when the objective and/or the constraints are nonconvex. With this goal in mind, we present a general-purpose heuristic based on Variable Neighborhood Search, Local Branching, a local Nonlinear Programming algorithm and Branch-and-Bound. We test the proposed approach on MINLPLib, comparing with several existing heuristic and exact methods. An implementation of the proposed heuristic is freely available and can employ all NLP/MINLP solvers with an AMPL interface as the main search tools.

1 Introduction

We address the mathematical programming formulation $\min\{f(x) \mid g(x) \leq 0\}$ where f, g involve nonlinear functions and the vector x includes some integer variables. This kind of problem is called a Mixed-Integer Nonlinear Program (MINLP), which is often also categorized according to the convexity of objective function and constraints. Solving nonconvex MINLPs involves difficulties

This paper extends [41]

Leo Liberti
LIX, École Polytechnique, Palaiseau, France
E-mail: liberti@lix.polytechnique.fr

Nenad Mladenović
Brunel University, London, UK and *Institute of Mathematics, Academy of Sciences, Belgrade, Serbia*
E-mail: nenad.mladenovic@brunel.ac.uk

Giacomo Nannicini (corresponding author)
Singapore University of Technology and Design, Singapore
E-mail: nannicini@sutd.edu.sg

arising from both nonconvexity and integrality, and it has proven to be a very hard challenge from a practical standpoint. From the modelling point of view, however, nonconvex MINLPs are the most expressive mathematical programs, and therefore attract a lot of interest from the mathematical programming community. Currently, optimal solutions of MINLPs in general form are obtained by using the spatial Branch-and-Bound (BB) algorithm [2,35,48,49]; but guaranteed optima can only be obtained for relatively small-sized MINLPs. MINLPs arising in real-world applications, with hundreds or thousands of variables and nonconvex constraints, represent a difficult challenge for spatial BB algorithms, and even finding a feasible solution is hard. Some practically efficient solvers targeting convex MINLPs exist in the literature [1,6,8,23,24,34]; they can all be used on nonconvex MINLPs as well, but they offer no guarantee of returning a good or even feasible solution. Some heuristic methods have been proposed, but very few can be applied to nonconvex MINLPs: the Feasibility Pump (FP) [21] idea was extended to convex [7] and very recently nonconvex MINLPs [15]; heuristics based on Iterative Rounding are discussed in [45]. An earlier version of the algorithm presented in this paper, described in [41], was among the first general-purpose heuristic methods for nonconvex MINLPs proposed in the literature.

In this paper, we propose a MINLP heuristic called the Relaxed-Exact Continuous-Integer Problem Exploration (RECIPE) algorithm. The MINLPs we address are cast in the following general form:

$$\left.\begin{array}{rc} \min\limits_{x\in\mathbb{R}^n} & f(x) \\ \text{s.t.} & l \leq g(x) \leq u \\ & x^L \leq x \leq x^U \\ & x_i \in \mathbb{Z} \qquad \forall\, i \in Z \end{array}\right\} \tag{P}$$

In the above formulation, x are the decision variables (x_i is integer for each $i \in Z$ and continuous for each $i \notin Z$, where $Z \subseteq \{1,\ldots,n\}$). $f : \mathbb{R}^n \to \mathbb{R}$ is a possibly nonlinear function, $g : \mathbb{R}^n \to \mathbb{R}^m$ is a vector of m possibly nonlinear functions (assumed to be differentiable), $l, u \in \mathbb{R}^m$ are the constraint bounds, and $x^L, x^U \in \mathbb{R}^n$ are the variable bounds (also called *box constraints*); the bounds may be set to $\pm\infty$. Note that we do not assume convexity of f, g. We define $B = \{i \in Z \mid x_i^L = 0 \wedge x_i^U = 1\}$ as the set of binary variables.

RECIPE puts together a global search phase based on Variable Neighborhood Search (VNS) [28] and a local search phase based on a MINLP heuristic. The VNS global phase rests on neighborhoods defined as hyperrectangles for the continuous and general integer variables [37] and by Local Branching (LBR) constraints for the binary variables [22]. The local phase employs a heuristic algorithm for finding feasible solutions, which in this paper is a BB algorithm for convex MINLPs [24]; a NLP solver supplies an initial constraint-feasible solution to be employed by the BB as starting point. RECIPE is an effective and reliable general-purpose algorithm for finding good solutions to complex MINLPs of small and medium scale. Its code is freely distributed, and it can be used with any local NLP and (heuristic) MINLP solvers that

provide an AMPL interface. In particular, it can be employed to increase the effectiveness of local or heuristic MINLP solvers, by guiding the exploration of the solution space.

The original contribution of this paper is the way a set of well-known and well-tested tools are combined into making a global optimization method. This paper does not contribute theoretical knowledge but rather the description of a practically useful algorithm whose easy implementation rests on existing off-the-shelf software tools complemented by relatively few lines of code.

The rest of this paper is organized as follows. Section 2 presents the overall approach, describing all the basic components on which RECIPE is based as well. In Section 3 we report extensive computational results obtained over MINLPLib, comparing our heuristic to several existing solvers, and analyzing the strengths and weaknesses of the proposed method. Section 4 concludes the paper.

2 The RECIPE algorithm

Our main algorithm is a heuristic exploration of the solution space by means of an alternating search between the relaxed NLP and the exact MINLP. This is a two-phase global optimization method. The global phase of the algorithm is given by the Variable Neighborhood Search metaheuristic [28,30] using two separate neighborhoods for continuous and general integer variables, and for binary variables. The former neighborhoods have hyper-rectangular shape; the latter are based on a single LBR [22] constraint involving all binary variables. The local phase needs two components: the first one is a local NLP solver, which is used to identify promising regions of the solution space (using a solution to the continuous relaxation to guide the search for an integer solution has also been used in the RINS heuristic for MILPs [17]); the second one is a MINLP heuristic, which is used to find a MINLP feasible solution (i.e. both integral and constraint feasible). In principle, any kind of algorithm that (heuristically) finds MINLP feasible solutions can be employed for the local phase. Our method of choice is a Branch-and-Bound search using an algorithm designed for convex MINLPs; therefore, no guarantee of optimality is provided. The reason for the effectiveness of RECIPE is that the each local search is performed on a problem which is easier than the original one, because of the way neighborhoods are defined: the LBR constraint reduces the number of feasible realizations of the vector of binary variables, and the hyper-rectangular neighborhood enforces a smaller search interval for general integer variables. Thus, the solution space that has to be searched by the local phase is smaller than that of the original problem, and we quickly discover new local optima, coordinated by VNS. RECIPE can be employed to enhance the performance of existing local/heuristic MINLP solution algorithms, by coordinating the local and global search phases, in order to efficiently explore the feasible region.

2.1 The basic ingredients

Before describing RECIPE, we need to introduce VNS and the neighborhood structure (i.e. hyper-rectangles and LBR).

2.1.1 Variable neighborhood search

VNS relies on iteratively exploring neighborhoods of growing size to identify better local optima [9,28–30]. Suppose that a local minimum x^* is found. VNS escapes from the current local minimum x^* by initiating local searches from starting points sampled from a neighborhood of x^* which increases its size iteratively, until a local minimum better than x^* is found. These steps are repeated until a given termination condition is met. This can be based on CPU time, number of non-improving steps and other configurable parameters. Typically, a maximum neighborhood size $k_{\max}$ is given; thus, each neighborhood is associated with its size $k = 1, \ldots, k_{\max}$, and the algorithm terminates if the neighborhood of size $k_{\max}$ has been explored without yielding an improved solution.

VNS has been applied to a wide variety of problems both from combinatorial and continuous optimization [3,10,19,33,38,39,46]. Its early applications to continuous problems were based on a particular problem structure [10,28]. The first VNS algorithm targeted at problems with fewer structural requirements, namely, box-constrained nonconvex NLPs, was given in [43] (the paper focuses on a particular class of box-constrained NLPs, but the proposed approach is general). Its implementation is described in [18]. Since the problem is assumed to be box-constrained, the neighborhoods arise naturally as hyperrectangles of growing size centered at the current local minimum x^*. The same neighborhoods were used in [37,42], an extension of VNS to constrained NLPs.

2.1.2 Local branching

Local Branching is an effective heuristic for finding good solutions to difficult Mixed-Integer Linear Programs (MILPs) [22]. Given an integer $s > 0$, LBR explores s-neighborhoods of the incumbent x^* with a BB algorithm by allowing at most s of the binary variables to change their value; this condition is enforced by means of the *LBR constraint*:

$$\sum_{i \in B : x_i^* = 1} (1 - x_i) + \sum_{i \in B : x_i^* = 0} x_i \leq s, \tag{1}$$

where B is the set of binary variables. (1) defines a 1-norm neighborhood of radius s with respect to the binary variables of P, centered at the current solution x^*. LBR updates the incumbent as it finds better solutions. When this happens, the LBR procedure is called iteratively with the new x^*. We remark that LBR was successfully used in conjunction with VNS in [31].

For general MINLPs, it is difficult to define a maximum neighborhood size to be employed by VNS without any knowledge of the specific problem. We opted for the following approach: for any problem, the neighborhood of maximum size (i.e. the neighborhood of size $k_{\max}$) coincides with the full initial search space of problem P. Smaller, concentric neighborhoods are defined for each $k = 1, \ldots, k_{\max}$, so that $k_{\max}$ actually defines the *number* of concentric neighborhood which are defined, as opposed to defining the *size* of the largest neighborhood.

Therefore, a neighborhood of size $k \leq k_{\max}$ on the binary variables is defined as follows:

$$\sum_{i \in B} (x_i^*(1 - x_i) + (1 - x_i^*)x_i) \leq \left\lceil k \frac{|B|}{k_{\max}} \right\rceil .$$

Observe that for $k = k_{\max}$, the constraint is redundant.

2.1.3 Hyperrectangular neighborhood structure

For a given $k \leq k_{\max}$ and point x^*, we define hyperrectangles $H_k(x^*)$, centered at $x^* \in \mathbb{R}^n$, and proportional to the hyperrectangle $x^L \leq x \leq x^U$ given by the original variable bounds, so that $H_{k-1}(x) \subset H_k(x)$ for each $k \leq k_{\max}$. More formally, let $H_k(x^*)$ be the hyperrectangle $y^L \leq x \leq y^U$, where for all $i \notin Z$:

$$y_i^L = x_i^* - \frac{k}{k_{\max}}(x_i^* - x_i^L)$$
$$y_i^U = x_i^* + \frac{k}{k_{\max}}(x_i^U - x_i^*),$$

for all $i \in Z \setminus B$:

$$y_i^L = \lfloor x_i^* - \frac{k}{k_{\max}}(x_i^* - x_i^L) + 0.5 \rfloor$$
$$y_i^U = \lfloor x_i^* + \frac{k}{k_{\max}}(x_i^U - x_i^*) + 0.5 \rfloor,$$

and for all $i \in B$:

$$y^L = 0, y^U = 1.$$

We define the neighborhood of size $1 \leq k \leq k_{\max}$ with respect to x^* as $N_k(x^*) = H_k(x^*) \backslash H_{k-1}(x^*)$. This neighborhood structure defines a set of hyperrectangular nested shells with respect to continuous and general integer variables. We define $N_0(x^*) = x^*$.

Note that, if one component of the original variable bounds x^L, x^U is $\pm\infty$, then the corresponding hyperrectangle bound is $\pm\infty$ regardless of x^* or k. Thus, the neighborhood remains the same at each iteration, and we would always sample in the same interval. To avoid this problem, we artificially substitute a finite but relatively large value $\bar{\infty}$ for ∞ when computing $H_k(x^*)$; this ensures that the sampling region depends on x^* and k. The value $\bar{\infty}$

is a parameter of our algorithm, and can affect the practical performance of RECIPE. However, in our computational tests we observed no difference in the results as long as $\bar{\infty}$ is set to a "reasonable" large value.

The VNS metaheuristic (and therefore RECIPE) requires a way of sampling random points in the neighborhoods. We now describe our method for this task. Let τ be the affine map sending the hyperrectangle $H_k(x^*)$ into the unit L_∞ ball (i.e., hypercube) $\mathcal{B}$ centered at 0, i.e., $\mathcal{B} = \{x : |x_i| \leq 1 \forall i\}$. In order to sample a random vector $\tilde{x}$ in $N_k(x^*)$ we proceed as in Alg. 1 [37].

Algorithm 1 Sampling in the shell neighborhoods.

INPUT: $k, k_{\max}$.
OUTPUT: A point $\tilde{x}$ sampled in $N_k(x^*)$.
Sample a random direction vector $d \in \mathbb{R}^n$ in the unit L_∞ ball
Normalize d: set $d \leftarrow \frac{d}{||d||_\infty}$
Sample a random radius $r \in [\frac{k-1}{k}, k]$ yielding a point in the outer part of the L_∞ ball
Let $\tilde{x} = \tau^{-1}(rd)$

Furthermore, we round $\tilde{x}_j$ to the nearest integer for $j \in Z$, i.e. we set $\tilde{x}_j \leftarrow \lfloor \tilde{x}_j + 0.5 \rfloor$. This is rather ineffective with the binary variables x_j, which would keep the same value $\tilde{x}_j = x^*_j$ for each $k \leq \frac{k_{\max}}{2}$. Binary variables are best dealt with via a LBR constraint.

2.2 Algorithm description

The input of RECIPE is the following: the maximum number of neighborhoods to explore $k_{\max}$, the maximum number of local searches in the same neighrbouhood L, a local NLP solution algorithm $\mathcal{N}$, and a MINLP solution algorithm $\mathcal{M}$. Here, $\mathcal{M}$ can be *any* method that is capable of producing as output a feasible solution to a MINLP (i.e. a solution that satisfies both the integrality requirements and the nonlinear constraints), meaning that $\mathcal{M}$ could be a heuristic. The output of RECIPE, if the algorithm is successful, is a feasible solution x^* for P.

Before describing the algorithm, we need some definitions. Consider a nonconvex MINLP given by formulation P, with its continuous relaxation $\bar{P}$. Let x^* be a feasible solution to P. For $1 \leq k \leq k_{\max}$, we define the problem:

$$\left.\begin{array}{rc} \min\limits_{x \in \mathbb{R}^n} & f(x) \\ \text{s.t.} & l \leq g(x) \leq u \\ & \sum\limits_{i \in B} (x^*_i(1-x_i) + (1-x^*_i)x_i) \leq \left\lceil k \frac{|B|}{k_{\max}} \right\rceil \\ & x^L \leq x \leq x^U \\ & x_i \in H_k(x^*) \; \forall \; i \in Z \setminus B \\ & x_i \in \mathbb{Z} \; \forall \; i \in Z \end{array}\right\} \qquad Q_k(x^*)$$

By comparing $Q_k(x^*)$ to P, we see that the differences are the introduction of a LBR constraint (1) centered on x^*, and tighter box constraints for general integer variables ($x_i \in H_k(x^*)$). In particular, general integer variables are forced to take values within the (outer bounds of the) hyperrectangular neighborhood of size k, which is smaller than the box defined by x^L, x^U for each $k < k_{\max}$. We denote by $\bar{Q}_k(x^*)$ the continuous relaxation of $Q_k(x^*)$. Finally, we use the convention that $f(x) = +\infty$ is x is not feasible for P. We can now describe RECIPE. The algorithm has two steps: an Initialization step, where a first feasible solution is sought, and a Main step.

In the Initialization step, RECIPE sets $k \leftarrow 0$, and each component of x^* is set to the midpoint of its lower/upper bounds if they are finite, or to zero if at least one of the bounds is infinite and zero is within the bounds. In the remaining case, it is set to the only finite bound. Formally:

$$x_i^* = \begin{cases} (x_i^L + x_i^U)/2 & \text{if } x_i^L > -\infty \wedge x_i^U < \infty, \\ x_i^L & \text{if } x_i^L > 0, \\ x_i^U & \text{if } x_i^L < 0, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

At each iteration, with a certain associated parameter k, we sample a point $\tilde{x}$ in $N_k(x^*)$, which is the outer region of a hyperrectangular neighborhood of radius k (at the first iteration, $k = 0$ which implies $\tilde{x} = x^*$), rounding where necessary for $i \in Z$: see Section 2.1.3 for details. Then, we apply $\mathcal{N}$ on P from the point $\tilde{x}$; this hopefully provides a constraint-feasible point $\bar{x}$. One of the reasons for performing this step is to find a good starting point for $\mathcal{M}$. Depending on the MINLP solution algorithm, this is sometimes necessary: for instance, if the MINLP solver is a BB algorithm for convex MINLPs (as is the case for the computational experiments in this paper), then the root node can be pruned by infeasibility if no point satisfying g is available. An additional reason is to make sure that we are exploring the solution space using our point sampling strategy, instead of relying on the MINLP solver to find an initial point. If $\mathcal{N}$ successfully returns $\bar{x}$, the MINLP solution algorithm $\mathcal{M}$ is initiated from $\bar{x}$; if $\mathcal{M}$ finds a feasible solution x', we set $x^* \leftarrow x', k \leftarrow 1$ and proceed to the Main step of the algorithm. Otherwise (i.e. one of the two solver fails), we return to the random point sampling step, and increase k if too many local searches have already been performed in the current neighborhood (more details on this are provided below).

The Main step of the algorithm is entered as soon as a feasible solution x^* is known. As before, we sample a random point $\tilde{x}$ in $N_k(x^*)$. We then solve the continuous relaxation $\bar{Q}_k(x^*)$ with $\mathcal{N}$ using $\tilde{x}$ as a starting point, and obtain $\bar{x}$. Finally, if $\mathcal{N}$ returns with success (i.e. $l \leq g(\bar{x}) \leq u$), $\mathcal{M}$ is used for solving $Q_k(x^*)$, taking $\bar{x}$ as input, hopefully obtaining a feasible solution x' to the original problem P. If x' improves on the incumbent x^*, then x^* is replaced by x' and k is reset to 1. Otherwise, i.e. if x' is worse than x^* or one of the solvers fails, we try another local search in the same neighborhood, or increase k in a VNS-like fashion. The algorithm is stopped after the last neighborhood $k = k_{\max}$ is explored, or if a time-based termination condition is met.

The number of local searches ℓ performed within a given neighborhood before k is increased depends on the input parameter L. We set $\ell = \min(k, L)$ at each iteration. This way, for small neighborhood sizes only a few local searches are carried out, whereas up to L are performed in larger neighborhoods. This is because we assume that in small neighborhoods, very few local searches are needed to discover a good solution, if any exists. In this paper, we set $L = 5$ in all the tests. Our computational experience for the conference version of this paper [41] confirmed that performing too many local searches in the same neighborhood does not bring advantages.

We can now describe the RECIPE algorithm formally in Alg. 2.

Algorithm 2 The RECIPE algorithm.

INPUT: number of neighborhoods $k_{\max}$;
maximum number of local searches L;
NLP solution algorithm $\mathcal{N}$;
MINLP solution algorithm $\mathcal{M}$.
OUTPUT: Best solution found x^*.
Set x^* as in (2)
Set $k \leftarrow 0$
Set *Step* $\leftarrow$ `Init`
while ($\neg$(time-based termination condition) $\wedge$ ($k \leq k_{\max}$)) **do**
  for ($i = 1, \ldots, \min(k, L)$) **do**
    Sample a random point $\tilde{x}$ in $N_k(x^*)$.
    if (*Step* = `Init`) **then**
      Solve $\bar{P}$ using $\mathcal{N}$ from initial point $\tilde{x}$ obtaining $\bar{x}$
      if ($l \leq g(\bar{x}) \leq u$) **then**
        Solve P using $\mathcal{M}$ from initial point $\bar{x}$ obtaining x'
      end if
      if ($f(x') < +\infty$) **then**
        Set *Step* $\leftarrow$ `Main`
      end if
    else {in this case *Step* = `Main`}
      Solve $\bar{Q}_k(x^*)$ using $\mathcal{N}$ from initial point $\tilde{x}$ obtaining $\bar{x}$
      if ($l \leq g(\bar{x}) \leq u$) **then**
        Solve $Q_k(x^*)$ using $\mathcal{M}$ from initial point $\bar{x}$ obtaining x'
      end if
    end if
    if ($f(x') < f(x^*)$) **then**
      Set $x^* \leftarrow x'$
      Set $k \leftarrow 0$
      Exit the FOR loop
    end if
  end for
  Set $k \leftarrow k + 1$.
end while

From Alg. 2 and the definition of $Q_k(x^*)$, it can be seen that $k_{\max}$ balances the tradeoff between solving a short sequence of large problems, or a long sequence of small problems. Indeed, for small values of $k_{\max}$ each neighborhood is relatively large, but we explore a smaller number of them; on the other hand,

for large values of $k_{\max}$ we perform more local searches, but each of them is executed on a smaller (hopefully easier) problem.

2.3 Local search phase

The local search phase of RECIPE requires both NLP and MINLP, possibly heuristic, solution algorithms. In this section we describe the methods employed in this paper, namely:

- The constraint feasibility enforcing local solution algorithm, that is used to quickly identify promising areas of the feasible region: Sequential Quadratic Programming (SQP) or Interior Point (IP) method;
- The constraint and integral feasibility enforcing local solution algorithm: Branch-and-Bound for convex MINLPs.

2.3.1 Sequential quadratic programming

SQP methods find local solutions to nonconvex NLPs. They solve a sequence of quadratic approximations of the original problem. The quadratic approximation is obtained by a convex model of the objective function Hessian at a current solution point, subject to a linearization of the (nonlinear) constraints around the current point. The SQP solver used in this paper [25] implements a trust region algorithm with a filter [26] to promote global convergence. In a trust region method, the quadratic approximation of the original problem is solved within a trust region, defined as a L_∞ norm ball of radius ρ around the current point. Whenever the step for the current iterate is rejected, the radius of the trust region is set to a smaller value; on the other hand, if the step is accepted, ρ increases.

2.3.2 Interior point algorithms

IP algorithms aim to solve NLPs in the form:

$$\left.\begin{array}{rl} \min\limits_{x\in\mathbb{R}^n} & f(x) \\ \text{s.t.} & g(x) = 0 \\ & x \geq 0. \end{array}\right\} \tag{3}$$

It is not difficult to show that every P with $Z = \emptyset$ (i.e. NLP) can be reformulated exactly to (3) via the introduction of slack variables and, if necessary, some translations. A typical IP algorithm generates a sequence of points x^i satisfying $g(x^i) = 0$ and $x^i > 0$ for all $i \in \mathbb{N}$, such that the limit x of the sequence minimizes f and satisfies $x \geq 0$. Each point x^i is obtained as the solution of the *barrier subproblem* (parametrized by a real $\mu > 0$):

$$\left.\begin{array}{rl} \min\limits_{x\in\mathbb{R}^n} & f(x) - \mu \sum\limits_{j\leq n} \ln(x_j) \\ \text{s.t.} & g(x) = 0. \end{array}\right\} \tag{4}$$

As $\mu \to 0$, x^i converges to the sought solution. IP implementations vary considerably in the way they solve the barrier subproblem (4). If (3) is an LP, IP methods converge to an optimal solution in polynomial time [32]; for general NLPs, only convergence to a local optimum is guaranteed.

2.3.3 Branch-and-Bound for convex MINLPs

Solving convex MINLPs (i.e. MINLPs where the objective function and constraints are convex — the terminology is confusing as all MINLPs are actually nonconvex problems because of the integrality constraints) is conceptually not much more difficult than solving MILPs: as the continuous relaxation of the problem is convex, obtaining lower bounds is easy. The existing tools, however, are still far from the quality attained by modern MILP solvers. The problem is usually solved by BB, where only the integer variables are selected for branching. A restricted (continuous) convex NLP is formed and solved at each node, where the variable ranges have been restricted according to the node's definition. Depending on the algorithm, the lower bounding problem at each node may either be the original problem with relaxed integrality constraints [13, 24] (in which case the BB becomes a recursive search for a solution that is both integer feasible and a local optimum in continuous space), or its linear relaxation by outer approximation [1,6,20,23].

These approaches guarantee an optimal solution if the objective and constraints are convex, but may be used as a heuristic even in presence of nonconvexity. Within this paper, we employ these methods in order to find local optima of nonconvex MINLPs. The problem of finding an initial feasible starting point (so that the continuous relaxation of the root node is not asserted to be infeasible) is addressed by supplying the method with a constraint-feasible (although not integer-feasible) starting point found by a continuous NLP solver.

3 Computational Experiments

In this section we describe our implementation of RECIPE, and discuss computational results. RECIPE can be freely downloaded at the following address: `http://www.lix.polytechnique.fr/~liberti/recipe.tar.gz`.

All computational results were obtained on an Intel Xeon X3353 2.66 GHz with 24 GB RAM running Linux. We compare RECIPE to some of the most reliable existing solvers for MINLPs available, namely: `AlphaECP` [51] version 1.65, `Baron` [47,49] version 8.1.5, `Couenne` [4,5] revision 422 and `sBB` [13] revision 009. Note that among these solvers, only `Couenne` provides an AMPL interface; the rest is available through GAMS [11]. `AlphaECP` and `sBB` find provably optimal solutions to convex MINLPs only, but can be heuristically employed on nonconvex MINLPs as well. They are included in the comparison because, according to the web page of the MINLPLib collection of test

instances (see Section 3.2), they discovered several best known feasible solutions, therefore proving to be effective on both convex and nonconvex MINLPs. `Baron` and `Couenne` are exact solvers for nonconvex MINLPs. In order to compensate for the loss of an optimality guarantee, an heuristic method such as RECIPE should find better solutions than exact solution methods, or it should be faster (if the solutions found are of similar quality). Additionally, we compare RECIPE with the Feasibility Pump for nonconvex MINLPs [15] on a limited number of instances (the ones for which we have detailed results obtained with FP).

3.1 Implementation

Alg. 2 presents many implementation difficulties: the problem must be reformulated iteratively with the addition of a different LBR constraint at each iteration; different solvers acting on different problem formulations must be used. All this must be coordinated by the outermost VNS at the global level. We chose to implement RECIPE as an AMPL [27] script because it makes it very easy to interface to many external solvers. Since AMPL cannot generate the reformulation $Q_k(x^*)$ of P iteratively independently of the problem structure, we employed a C++ program that reads an AMPL output `.nl` file in flat form [35,36] and outputs the required reformulation as an AMPL-readable `.mod` file.

The greatest advantage of coding RECIPE as an AMPL script is that we can plug into the algorithm all solvers that provide an AMLP interface – which represents the majority of them. RECIPE employs a NLP solver and a MINLP solver as local search black-box tools. In this paper, we report computational experiments obtained using `filterSQP` [25] version 20010807, which is an SQP trust region method, and `Ipopt` [50] revision 1694, which is an IP method, as NLP solvers, and `minlp_bb` [34] version 0.3.2 20100607 as the MINLP solver. Note that `minlp_bb` is a NLP-based BB solver for convex MINLP, therefore it offers no guarantee of optimality or even of finding a feasible solution when applied to nonconvex problems, as is the case. We found `minlp_bb` to be extremely fast in practice, significantly more than some of its competitors (in particular, the NLP-based BB algorithm included in `BonMin` [8] — `AlphaECP` and `sBB` were not tested as subsolvers because we are not aware of an AMPL interface). For our heuristic scheme, where several (typically small) problems have to be iteratively solved, speed is one of the key factors.

RECIPE has many customizable options and parameters: a small number of algorithmic parameters (the few that are necessary to configure VNS), and several parameters that affect the underlying subsolvers, such as tolerances. Our implementation within AMPL fully supports the use of customized options for each of the subsolvers employed, so that each one can be parameterized by the user as desired. The user can specify a template options file for each solver, and RECIPE takes care of setting the parameters to their correct value based on the template provided. Since there is no standardized way of setting

solver parameters (e.g. time limit) in AMPL, and we want to have maximum flexibility in the choice of solvers, having template option files is a necessary step to ensure the coordination between RECIPE and each subsolver.

In its current implementation, RECIPE supports the following parameters (in brackets, we report the name of the option in the software and the value used in this paper):

- $k_{\max}$. (`kmax` $= 20$)
- Time limit. (`timelimit` $= 7200$)
- Integrality tolerance. (`int-precision` $= 10^{-5}$)
- Constraint feasibility tolerance. (`epsilon` $= 10^{-6}$)
- Maximum absolute value for variable bounds, i.e. $\bar{\infty}$ as introduced in Section 2.1.3. (`max-bound` $= 10^{10}$)
- Maximum number η of solutions with the same objective value that are accepted before an improvement is required. (`equal-max` $= 3$)
- Required objective function improvement between consecutive solutions. (`obj-improvement` $= 10^{-5}$)
- Maximum number L of local searches performed in the same neighborhood. (`max-searches` $= 5$)

In our current implementation, if a solution is discovered which has the same objective value than the current incumbent, it is accepted up to a maximum of η times. However, since the MINLP solver is given a cutoff value which aims for a minimum improvement of 10^{-5} with respect to the current incumbent, this is a rare occurence, which only happens when the NLP solver returns a solution which is both constraint- and integer-feasible. Additionally, the user can set the following algorithmic options (default value in brackets):

- Require a constraint (not integral) feasible solution for running the MINLP solver? (`need-constr-feas` $=$ `yes`)
- At iteration k, use hyperrectangle $H_k(x^*)$ instead of the original variable bounds as box constraints for the continuous variables? (`bound-cont` $=$ `no`)
- At iteration k, use hyperrectangle $H_k(x^*)$ instead of the original variable bounds as box constraints for the integer (not binary) variables? (`bound-int` $=$ `yes`)

The default values for these options yield RECIPE as described in Section 2.2, but the user can change them to modify the behaviour of the algorithm. In particular, the first option can be used to start the MINLP solver even when the NLP solver fails and no point that satisfied the nonlinear constraints is available; depending on the particular MINLP solver, this could be desirable (note that in this case, the point provided as input to the MINLP is $\tilde{x}$, since $\bar{x}$ is not defined). The remaining two options decide whether the hyperrectangles are enforced as variable bounds at each iteration when solving problem $Q_k(x^*)$ (the default version of RECIPE only restricts the bounds of the integer variables).

Instead of relying solely on the subsolvers to verify the constraint and integrality feasibility of the solutions, we run additional checks to ensure that

feasibility is attained with the required tolerances. We employ the AMPL tool `evalchk`, freely available through `http://www.netlib.org`, to avoid the errors (and program crashes) that would occur whenever the argument of an operator is outside its domain (e.g. logarithm of a negative number).

3.2 MINLPLib

The MINLPLib [12] is a collection of Mixed Integer Nonlinear Programming models which can be searched and downloaded freely. Statistics for the instances in the MINLPLib are available from `http://www.gamsworld.org/minlp/minlplib/minlpstat.htm`. The instance library is available at `http://www.gamsworld.org/minlp/minlplib.htm`. The MINLPLib is distributed in GAMS [11] format, so we employed an automatic translator to cast the files in AMPL format.

At the time of downloading (Feb. 2010), the MINLPLib consisted of 270 MINLP instances contributed by the scientific and industrial OR community. However, 19 of these problems cannot be successfully read by AMPL due to the presence of unimplemented functions, and had to be excluded from our test set. These instances are:

`blendgap`, `dosemin2d`, `dosemin3d`, `hda`, `meanvarxsc`, `pb302035`, `pb302055`, `pb302075`, `pb302095`, `pb351535`, `pb351555`, `pb351575`, `pb351595`, `water3`, `waterful2`, `watersbp`, `waters`, `watersym1`, `watersym2`.

The performance of RECIPE was evaluated on the remaining 251 instances.

3.3 Subsolver comparison

In this section we compare the results obtained by RECIPE using either `Ipopt` or `filterSQP` as NLP subsolvers; the MINLP solver of choice is `minlp_bb` in both cases. Additionally, we include in the comparison the results obtained by using `minlp_bb` to directly solve the test instances; that is, `minlp_bb` used as a stand-alone solver outside RECIPE. We want to assess whether RECIPE is more powerful than just its basic components.

In these experiments, we set the $k_{\max}$ parameter of RECIPE to 20. This choice was based on computational experiments and on our previous experience [41]. We additionally tested the values $k_{\max} = 10$ and $k_{\max} = 30$; for space reasons, we do not report full computational results obtained with these values. We summarize them briefly: with $k_{\max} = 10$, RECIPE is on average 33% faster (we compared the geometric mean of the average CPU time), and finds one less feasible solution; solution quality is only slightly affected, with an advantage for $k_{\max} = 20$. With $k_{\max} = 30$, RECIPE is approximately 20% slower on average, but we did not observe any improvement in the quality of the solutions. $k_{\max} = 20$ turned out to be a good compromise between speed and quality. The value of $k_{\max}$ can be reduced to favour speed.

Results are reported in Table 1; for each solver, we report the objective function value of the best solution found (∞ if no feasible solution was discovered), and the CPU time required, in seconds. In each row, if one of the solutions is strictly better than the remaining two, then it is highlighted in boldface. Whenever two solutions have an objective function value within a relative difference, they are considered equal; this relative difference is set to 0.1% of the smallest objective function (absolute) value if it is inside the interval $[10^{-5}, 10^{3}]$, or to the closest bound of the interval whenever it is outside. All solvers were given a time limit of 2 hours, and `Ipopt` is parameterized with the option `bound_relax_factor = 0`, to ensure that the point it returns satisfies the box constraints. A summary of the results is given in Table 2.

	`minlp_bb`		RECIPE with `Ipopt`		RECIPE with `filterSQP`	
Instance	f^*	time	f^*	time	f^*	time
`4stufen`	∞	0.01	∞	0.79	∞	0.61
`alan`	2.9250	0.00	2.9250	1.95	2.9250	0.41
`batchdes`	167427.6571	0.01	167427.6697	2.74	167427.6697	1.49
`batch`	285506.5082	0.06	285506.3123	10.69	285506.3123	5.81
`beuster`	∞	0.01	∞	0.79	∞	0.62
`cecil_13`	-114379.0758	7200.15	-114379.0669	7200.15	-114379.0669	7200.30
`chp_partload`	∞	11.22	26.9316	7212.52	**25.2226**	7254.09
`contvar`	809149.8272	115.62	809149.5423	7201.72	∞	99.69
`csched1a`	∞	0.00	**-30430.1780**	4.33	∞	0.68
`csched1`	-30639.2578	0.64	-30639.2607	156.50	-30639.2607	1.73
`csched2a`	∞	0.01	**-165398.7013**	3286.04	∞	72.76
`csched2`	∞	0.07	**-166101.9928**	7315.06	∞	18.06
`deb10`	∞	0.10	∞	674.73	∞	4.97
`deb6`	**201.7393**	12.98	∞	1217.27	∞	3.41
`deb7`	∞	121.30	∞	4544.21	∞	10.89
`deb8`	∞	91.30	∞	4482.77	∞	17.06
`deb9`	∞	119.40	∞	4920.32	∞	11.05
`detf1`	∞	67.11	∞	0.00	∞	7233.56
`du-opt5`	8.0737	0.03	8.0740	4.58	8.0740	3.78
`du-opt`	3.5563	0.03	3.5566	4.41	3.5566	3.52
`eg_all_s`	7.6578	2.64	7.6578	326.04	7.6578	201.42
`eg_disc2_s`	5.9333	1.19	5.6421	554.37	5.6421	293.71
`eg_disc_s`	6.8390	1.37	5.7606	635.73	5.7606	168.19
`eg_int_s`	6.4531	1.19	6.4533	196.52	6.4533	142.98
`elf`	0.4347	0.21	0.1917	165.27	0.1917	5.01
`eniplac`	-131863.6349	25.27	-132117.0254	7200.05	-132117.0254	477.89
`enpro48`	187277.2594	4.00	187277.1571	40.08	187277.1571	6.65
`enpro48pb`	187277.2594	2.57	187277.1571	560.52	187277.1571	14.08
`enpro56`	263428.3010	37.77	263428.3589	62.54	263428.3589	43.36
`enpro56pb`	263428.3010	20.36	263428.3589	2515.05	263428.3589	49.50
`ex1221`	7.6672	0.00	7.6672	2.41	7.6672	0.51
`ex1222`	1.0765	0.00	1.0765	1.37	1.0765	0.64
`ex1223a`	4.5796	0.00	4.5796	1.43	4.5796	0.71
`ex1223b`	4.5796	0.00	4.5796	1.59	4.5796	0.85
`ex1223`	4.5796	0.00	4.5796	1.85	4.5796	0.94
`ex1224`	-0.9316	0.00	-0.9435	2.59	-0.9435	1.20
`ex1225`	31.0000	0.00	31.0000	1.51	31.0000	0.79
`ex1226`	∞	0.00	-17.0000	2.41	-17.0000	0.81
`ex1233`	155522.4622	0.08	155010.6713	0.47	155010.6713	0.71
`ex1243`	83402.5064	0.10	83402.5069	10.58	83402.5069	0.83
`ex1244`	82042.9052	0.07	82042.9052	102.00	82042.9052	1.22
`ex1252a`	128893.7410	0.18	128894.0029	7.82	128894.0029	5.72
`ex1252`	128893.7410	0.11	128894.0029	23.70	128894.0029	18.01
`ex1263a`	19.6000	0.16	19.6000	7.37	19.6000	2.94
`ex1263`	19.6000	3.27	19.6000	233.24	19.6000	147.61
`ex1264a`	8.6000	0.33	8.6000	5.88	8.6000	3.55
`ex1264`	8.6000	10.66	8.6000	35.66	8.6000	128.84
`ex1265a`	15.1000	0.01	11.5000	10.65	**10.3000**	4.70
`ex1265`	15.1000	0.04	15.1000	8.92	**10.3000**	6.40

	minlp_bb		RECIPE with Ipopt		RECIPE with filterSQP	
Instance	f^*	time	f^*	time	f^*	time
ex1266a	16.3000	0.03	16.3000	5.25	16.3000	3.32
ex1266	16.3000	0.16	16.3000	19.24	16.3000	7.52
ex3	68.0097	0.02	68.0097	0.76	68.0097	0.49
ex3pb	68.0097	0.03	68.0097	5.09	68.0097	2.62
ex4	-8.0641	0.33	-8.0641	21.40	-8.0641	19.44
fac1	160912612.3502	0.01	160912612.3502	2.78	160912612.3502	1.10
fac2	331837498.1768	0.07	331837498.1768	7.85	331837498.1768	3.23
fac3	31982309.8480	0.06	31982309.8480	9.78	31982309.8480	4.85
feedtray2	0.0000	2.53	0.0000	378.54	0.0000	66.26
feedtray	-13.4060	0.05	-13.4114	46.74	-13.4114	58.72
fo7_2	17.7493	937.56	17.7493	4169.08	17.7493	7200.09
fo7_ar2_1	24.8398	4019.28	24.8398	7200.05	24.8398	7200.12
fo7_ar25_1	23.0936	1276.11	23.0936	7200.04	23.0936	7200.05
fo7_ar3_1	22.5175	1762.36	22.5175	7200.04	22.5175	7200.25
fo7_ar4_1	20.7298	3490.48	20.7298	7200.05	20.7298	6424.18
fo7_ar5_1	17.7493	436.03	17.7493	7200.05	17.7493	1866.79
fo7	20.7298	7200.09	20.7298	7200.16	20.7298	3696.15
fo8_ar2_1	34.2990	7200.03	30.3406	7200.05	30.3406	7200.06
fo8_ar25_1	37.8614	7200.62	33.3122	7200.05	**32.4477**	7200.06
fo8_ar3_1	33.5290	7200.03	30.5745	7200.05	**23.9101**	7200.14
fo8_ar4_1	22.3819	7200.02	37.0390	7200.30	22.3819	7200.70
fo8_ar5_1	22.3819	6446.94	23.9101	7200.06	22.3819	7200.66
fo8	23.9101	7200.06	22.3819	7200.05	22.3819	7200.05
fo9_ar2_1	39.3141	4297.48	36.6681	7200.07	**32.6250**	7200.60
fo9_ar25_1	53.5665	2836.67	54.0811	7200.05	**43.7722**	7200.06
fo9_ar3_1	60.4636	2571.03	37.0128	7200.10	**24.8155**	7201.24
fo9_ar4_1	44.5500	7200.24	45.4231	7200.31	**43.6116**	7202.36
fo9_ar5_1	63.9902	3536.50	∞	7200.08	**28.6727**	7200.10
fo9	39.6169	7200.04	∞	7200.06	**30.7500**	7200.11
fuel	8566.1190	0.00	8566.1190	16.10	8566.1190	1.13
fuzzy	**-0.5015**	3579.59	∞	0.00	∞	0.00
gasnet	7004607.8064	6.60	**6999391.6436**	117.70	7045336.9264	153.55
gastrans	∞	0.15	89.0858	19.55	89.0858	1.22
gbd	2.2000	0.00	2.2000	2.09	2.2000	0.71
gear2	0.0000	0.01	0.0000	1.51	0.0000	1.49
gear3	0.0000	0.01	0.0000	1.21	0.0000	0.58
gear4	**1.6434**	0.56	1.9682	15.68	1.9682	14.59
gear	0.0000	0.00	0.0000	1.18	0.0000	0.67
gkocis	-1.9231	0.00	-1.9231	3.13	-1.9231	0.94
hmittelman	∞	0.01	13.0000	4.57	13.0000	1.12
johnall	-224.7302	3.40	-224.7302	69.25	-224.7302	7.00
lop97ic	4153.7109	7200.45	4232.6891	7212.59	**4138.3292**	7200.79
lop97icx	4099.0600	4029.98	4099.0600	5381.36	0.0000	5517.73
m3	37.8000	0.05	37.8000	5.42	37.8000	3.58
m6	82.2569	37.64	82.2569	1414.40	82.2569	2094.19
m7_ar2_1	190.2350	538.72	190.2350	7200.04	190.2350	7200.44
m7_ar25_1	143.5850	254.73	143.5850	2919.44	143.5850	2769.84
m7_ar3_1	143.5850	1430.29	143.5850	7200.04	143.5850	6889.18
m7_ar4_1	106.7569	1370.36	106.7569	7200.05	106.7569	5771.79
m7_ar5_1	106.4600	1138.74	106.4600	7200.04	106.4600	4112.68
m7	106.7569	397.35	106.7569	2251.72	106.7569	7200.03
mbtd	4.6667	4014.52	5.5833	7203.37	4.6667	7203.56
meanvarx	14.3692	0.00	14.3692	3.06	14.3692	0.71
minlphix	∞	0.00	316.6928	54.76	316.6928	3.38
netmod_dol1	∞	9698.77	∞	9720.59	∞	9716.51
netmod_dol2	-0.5600	2470.45	-0.5535	8930.19	-0.5600	9008.98
netmod_kar1	-0.4198	97.09	-0.4198	7214.51	-0.4198	257.50
netmod_kar2	-0.4198	97.36	-0.4198	7213.04	-0.4198	257.50
no7_ar2_1	107.8153	1524.09	107.8153	7200.05	107.8153	7204.43
no7_ar25_1	107.8153	3592.06	107.8153	7200.11	107.8153	7200.09
no7_ar3_1	107.8695	7200.01	107.8153	7204.28	107.8153	7200.05
no7_ar4_1	98.5184	7200.29	98.5184	7200.04	98.5184	7203.95
no7_ar5_1	90.6227	4836.94	90.6227	7200.04	90.6227	7203.33
nous1	**1.6521**	0.06	∞	21.96	∞	1.15
nous2	**0.6260**	0.01	∞	21.99	∞	1.09
nuclear104	∞	0.00	∞	7267.51	∞	10509.10
nuclear10a	∞	0.00	∞	0.00	∞	0.00

Instance	minlp_bb		RECIPE with Ipopt		RECIPE with filterSQP	
	f^*	time	f^*	time	f^*	time
nuclear10b	∞	0.00	∞	7223.50	∞	750.72
nuclear14a	-1.1280	160.73	**-1.1296**	2732.95	-1.1280	1270.87
nuclear14b	-1.0896	7221.04	**-1.1093**	7200.37	-1.0936	7200.60
nuclear14	∞	41.10	**-1.1257**	6062.20	∞	1378.68
nuclear24a	-1.1280	160.40	**-1.1296**	2769.73	-1.1280	1272.52
nuclear24b	-1.0896	7212.66	**-1.1093**	7201.06	-1.0936	7205.14
nuclear24	∞	41.04	**-1.1257**	6006.97	∞	1383.67
nuclear25a	**-1.1193**	622.00	∞	0.00	-1.1000	7217.52
nuclear25b	-1.0851	7200.35	**-1.0977**	7201.25	∞	7202.16
nuclear25	∞	64.80	**-1.1171**	7225.31	∞	1840.91
nuclear49a	**-1.1513**	4185.18	∞	8293.68	∞	8188.66
nuclear49b	∞	0.00	∞	0.00	∞	0.00
nuclear49	∞	865.66	∞	7200.83	∞	7298.06
nuclearva	∞	7.74	**-1.0109**	1323.10	∞	183.79
nuclearvb	∞	7.09	**-1.0289**	1408.14	∞	189.93
nuclearvc	∞	1.97	**-0.9972**	1370.61	∞	188.91
nuclearvd	∞	13.74	**-1.0315**	1806.26	∞	183.95
nuclearve	∞	4.38	**-1.0317**	2013.02	∞	187.36
nuclearvf	∞	2.52	**-1.0225**	1911.18	∞	212.76
nvs01	12.4697	0.00	12.4697	1.28	12.4697	0.57
nvs02	6.1587	0.00	5.9642	3.49	5.9642	1.99
nvs03	16.0000	0.00	16.0000	1.11	16.0000	0.55
nvs04	2.1200	0.00	0.7200	2.09	0.7200	0.75
nvs05	∞	0.00	**5.4709**	42.91	∞	0.08
nvs06	1.7703	0.00	1.7703	1.15	1.7703	0.59
nvs07	4.0000	0.00	4.0000	1.40	4.0000	0.68
nvs08	23.4497	0.00	23.4497	1.55	23.4497	0.59
nvs09	-43.1343	0.00	-43.1343	0.95	-43.1343	0.46
nvs10	-310.8000	0.00	-310.8000	1.28	-310.8000	0.55
nvs11	-431.0000	0.00	-431.0000	1.20	-431.0000	0.58
nvs12	-481.2000	0.00	-481.2000	1.43	-481.2000	0.65
nvs13	-585.2000	0.00	-585.2000	1.53	-585.2000	0.74
nvs14	-38413.2312	0.00	-40358.1142	3.32	-40358.1142	1.97
nvs15	1.0000	0.00	1.0000	1.30	1.0000	0.64
nvs16	0.7031	0.00	0.7031	1.63	0.7031	0.58
nvs17	-1100.4000	0.01	-1100.4000	2.27	-1100.4000	1.34
nvs18	-778.4000	0.01	-778.4000	1.90	-778.4000	0.99
nvs19	-1098.4000	0.03	-1098.4000	2.95	-1098.4000	1.99
nvs20	230.9222	0.01	230.9222	1.83	230.9222	1.19
nvs21	0.0000	0.00	-5.6848	1.56	-5.6848	0.96
nvs22	∞	0.00	6.0582	57.65	6.0582	0.58
nvs23	-1125.2000	0.04	-1125.2000	4.63	-1125.2000	3.32
nvs24	-1033.2000	0.05	-1033.2000	4.65	-1033.2000	3.24
o7_2	118.8593	7200.01	116.9459	7200.04	116.9459	7200.07
o7_ar2_1	140.4120	6695.43	140.4120	7200.04	140.4120	7204.95
o7_ar25_1	140.4120	7200.03	141.6231	7202.21	140.4120	7200.07
o7_ar3_1	142.0855	4467.70	138.8649	7200.04	**137.9318**	7200.06
o7_ar4_1	131.6531	7200.03	131.6531	7200.04	131.6531	7200.07
o7_ar5_1	116.9458	7200.03	116.9458	7200.55	116.9458	7200.41
o7	137.9318	5644.42	131.6531	7203.14	131.6531	7200.35
o8_ar4_1	**253.8707**	2196.35	254.9776	7200.06	273.5999	7201.03
o9_ar4_1	327.8660	3530.38	316.0128	7200.08	**297.0642**	7200.06
oaer	-1.9231	0.00	-1.9231	2.72	-1.9231	0.36
oil2	-0.7333	5.33	∞	0.00	-0.7333	282.63
oil	-0.9325	7200.27	-0.9325	7202.21	∞	130.58
ortez	-9532.0391	0.07	-9532.0398	68.70	-9532.0398	0.77
parallel	924.2956	0.59	924.2986	1.14	924.2986	3.31
prob02	112235.0000	0.00	112235.0000	1.45	112235.0000	0.79
prob03	10.0000	0.00	10.0000	1.07	10.0000	0.62
prob10	3.4455	0.00	3.4455	1.60	3.4455	0.79
procsel	-1.7210	0.00	-1.7210	0.66	-1.7210	0.40
product2	∞	1.21	**-2096.7699**	7201.63	∞	1067.69
product	-2142.9286	3016.50	-2142.9275	7200.50	-2140.8409	7201.34
pump	128893.7410	0.18	128894.0029	7.75	128894.0029	5.61
qap	396024.0000	25.65	388870.0000	1541.90	**388214.0000**	644.64
qapw	394908.0000	122.94	388988.0000	4410.31	**388214.0000**	5187.32
ravem	269590.2193	0.36	269590.2650	2.93	269590.2650	1.48

	minlp_bb		RECIPE with Ipopt		RECIPE with filterSQP	
Instance	f^*	time	f^*	time	f^*	time
ravempb	269590.2193	0.43	269590.2650	57.39	269590.2650	19.68
risk2b	∞	0.02	**-56.8208**	305.67	∞	4.20
risk2bpb	-55.8761	3.36	-56.8208	1631.20	-56.8208	25.52
saa_2	∞	67.32	∞	0.00	∞	7230.22
sep1	-470.1301	0.01	-470.1301	15.26	-470.1301	1.07
space25a	484.3286	23.57	484.3278	302.06	484.3278	32.22
space25	484.3286	72.29	484.3278	1146.59	484.3278	223.88
space960	∞	241.62	∞	7200.65	∞	0.00
spectra2	13.9783	0.12	13.9783	228.87	13.9783	5.90
spring	0.8462	0.01	0.8462	28.17	0.8462	0.48
st_e13	2.0000	0.00	2.0000	2.85	2.0000	0.47
st_e14	4.5796	0.00	4.5796	1.84	4.5796	0.88
st_e15	7.6672	0.00	7.6672	2.43	7.6672	0.49
st_e27	2.0000	0.00	2.0000	1.22	2.0000	0.60
st_e29	-0.9316	0.00	-0.9435	2.57	-0.9435	1.20
st_e31	-2.0000	0.19	-2.0000	24.51	-2.0000	13.46
st_e32	-1.4304	0.05	-1.4304	0.19	-1.4304	0.15
st_e35	∞	0.26	∞	0.21	∞	0.32
st_e36	-166.4440	0.00	-246.0000	2.40	-246.0000	3.17
st_e38	7197.7271	0.00	7197.7216	1.14	7197.7216	0.50
st_e40	42.1421	0.00	**30.4142**	4.58	31.0710	0.69
st_miqp1	281.0000	0.00	281.0000	1.33	281.0000	0.67
st_miqp2	2.0000	0.00	2.0000	7.37	2.0000	0.68
st_miqp3	-6.0000	0.00	-6.0000	1.19	-6.0000	0.47
st_miqp4	-4574.0000	0.00	-4574.0000	1.32	-4574.0000	0.68
st_miqp5	-333.8889	0.00	-333.8900	2.49	-333.8900	0.49
stockcycle	119948.6883	31.34	119948.7606	5124.47	119948.7606	3515.12
st_test1	0.0000	0.00	-0.0000	1.27	0.0000	0.70
st_test2	-9.2500	0.00	-9.2500	1.66	-9.2500	0.64
st_test3	-7.0000	0.00	-7.0000	1.59	-7.0000	0.57
st_test4	-7.0000	0.00	-7.0000	1.61	-7.0000	0.67
st_test5	-110.0000	0.00	-110.0000	1.45	-110.0000	0.68
st_test6	471.0000	0.00	471.0000	1.28	471.0000	0.69
st_test8	-29605.0000	0.00	-29605.0000	2.46	-29605.0000	0.73
st_testgr1	-12.8116	0.01	-12.8116	1.47	-12.8116	0.79
st_testgr3	-20.5900	0.01	-20.5900	2.54	-20.5900	1.44
st_testph4	-80.5000	0.00	-80.5000	1.33	-80.5000	0.59
super1	∞	51.04	**9.6438**	7224.29	9.8913	7207.42
super2	∞	33.80	**5.2468**	7221.22	5.2907	7210.68
super3	∞	45.77	**12.9385**	7205.99	13.4772	7210.18
super3t	**-0.6744**	7206.93	-0.6684	7206.03	-0.6673	7201.32
synheat	154997.8251	0.09	154997.3349	11.13	154997.3349	1.56
synthes1	6.0098	0.00	6.0098	1.37	6.0098	0.65
synthes2	73.0353	0.00	73.0353	3.90	73.0353	0.93
synthes3	68.0097	0.01	68.0097	2.48	68.0097	1.38
tln12	137.1000	732.36	106.8000	7200.04	**102.3000**	7200.04
tln2	5.3000	0.01	5.3000	7.15	5.3000	0.95
tln4	8.3000	14.42	8.3000	541.30	8.3000	578.15
tln5	10.3000	50.01	10.3000	442.67	10.3000	480.93
tln6	15.3000	44.29	15.4000	7200.03	15.3000	7200.03
tln7	23.6000	56.55	**15.6000**	7200.03	15.9000	7200.03
tloss	16.3000	0.04	16.3000	5.30	16.3000	2.92
tls12	∞	7200.11	∞	7200.23	∞	7200.65
tls2	5.3000	0.21	5.3000	7.56	5.3000	4.11
tls4	11.7000	835.46	**8.8000**	7200.04	9.0000	7200.04
tls5	14.0000	5937.37	14.2000	7200.67	**11.0000**	7200.04
tls6	**15.9000**	2683.52	16.4000	7200.10	18.1000	7200.08
tls7	∞	7200.01	∞	7200.16	∞	7200.14
tltr	∞	0.01	∞	2.67	∞	1.76
uselinear	∞	311.43	∞	816.29	∞	20046.10
util	999.5788	2.00	999.5790	313.65	999.5790	4.31
var_con10	∞	0.54	**444.1032**	2550.53	∞	240.81
var_con5	∞	0.73	**278.0384**	2293.98	∞	238.78
waste	732.2534	7200.18	**679.0943**	7208.15	903.8701	7200.66
water4	907.0170	66.48	907.0174	298.20	907.0174	63.23
waterx	926.5789	3.27	**914.1837**	142.69	919.0510	9.07
waterz	913.1571	2767.08	910.8823	7200.08	**907.0174**	3005.99

Instance	minlp_bb		RECIPE with Ipopt		RECIPE with filterSQP	
	f^*	time	f^*	time	f^*	time
windfac	∞	0.00	∞	4.87	**0.2545**	3.43

Table 1: Comparison of the results obtained by minlp_bb used as a stand-alone solver, by RECIPE with Ipopt as NLP subsolver, and by RECIPE with filterSQP as NLP subsolver (the MINLP subsolver for RECIPE is minlp_bb in both cases).

	minlp_bb	RECIPE Ipopt	RECIPE filterSQP
# feasible solutions	204	221	206
# at least as good solutions	177	214	208
# strictly better solutions	10	31	20
Average CPU time	14.63	112.96	53.45
CPU time:			
< 600	192	147	169
< 1800	11	11	7
< 7200	23	20	13
7200+	25	73	62

Table 2 Summary of the results in Table 1. For each algorithm, we report: the number of instances for which a feasible solution is found, the number of instances for which it finds a solution with an objective value (∞ included) that is at least as good as the other methods, the number of instances for which it finds a solution strictly better than the remaining methods, the average CPU time in seconds (geometric mean), and the number of instances that require a given CPU time.

From Table 2, we see that there is a clear winner in terms of solution quality: RECIPE with Ipopt as the NLP subsolver. Not only does it find more feasible solutions (221 instances out of 251, 88%), but they are almost always at least as good as the solutions found by the competitors. In particular, for 214 instances out of 251 (85%) it returns a solution which is equal to or better than those found by minlp_bb alone or RECIPE with filterSQP. On the other hand, Ipopt is slower than filterSQP. This is consistently observed in our experiments. Employing Ipopt instead of filterSQP slows down RECIPE also as a side-effect of finding more feasible solutions: RECIPE with filterSQP may fail to solve an instance and quickly report a failure, whereas RECIPE with Ipopt may take a longer time but eventually find a feasible point. We can also see that minlp_bb used as a stand-alone solver seems to perform very well on average; however, this was expected. The reason for this is that the MINLPLib contains several instances which are convex or nonconvex but very easy; for these instances, minlp_bb typically finds the optimal (or a very good) solution, and using RECIPE does not bring additional benefits. However, on the more difficult nonconvex instances, RECIPE is able to improve considerably over minlp_bb, in particular when Ipopt is used as the NLP solver; examples are the csched, the nuclear and the super instances. In a very limited number of cases (10), minlp_bb stand-alone finds a better solution than RECIPE; for most of these cases (6 out of 10), the NLP solver employed by RECIPE always fails, therefore the MINLP solver is not run. In this case, setting the option need-constr-feas to no might help in solving the issue. For

the remaining cases, our intuition is that we were just unlucky in the sampling of the points from which the local searches are initiated.

3.4 Comparison with existing solvers on the full test set

In this section we compare RECIPE to other, existing solvers, in order to evaluate the quality of the solutions it finds, its speed, and its reliability. RECIPE is designed for finding good solutions, whereas the solvers that we compare to are primarily designed for proving optimality of solutions (of convex or non-convex MINLPs). Therefore, in terms of solution quality we want RECIPE to be at least as effective as these solvers.

We ran RECIPE (with `Ipopt` as NLP solver and $k_{\max} = 20$, as suggested by the results in Section 3.3), `AlphaECP`, `Baron`, `Couenne` and `sBB` on all instances of our test set with a time limit of 2 hours, with the same integrality/constraint tolerances as RECIPE. For all solvers that we compare RECIPE to, we set the absolute gap tolerance for optimality to 10^{-9}, and the relative gap tolerance to 10^{-6}. `Baron` uses Cplex as LP solver. `sBB` requires an iteration limit, which was set to 10^9, and a node limit, set to 10^7; none of these limits was hit during our experiments. The branching strategy in `Couenne` was set to strong branching. All other parameters are left to their default values. In Table 3 we report the solution reported by each of these methods, and the total CPU time. As before, if an algorithm found a solution which is better than those discovered by all other methods, the solution is highlighted in boldface.

	RECIPE		AlphaECP		Baron		Couenne		sBB	
Instance	f^*	time	f^*	time	f^*	time	f^*	time	f^*	time
4stufen	∞	0.79	∞	48.26	116820.4719	7200.01	∞	7331.55	**116329.6706**	0.26
alan	2.9250	1.95	2.9250	0.00	2.9250	0.04	2.9250	0.15	2.9250	0.00
batchdes	167427.6697	2.74	167427.6571	0.01	167427.6571	0.05	∞	0.12	167427.6571	0.00
batch	285506.3123	10.69	285506.5082	1.76	285506.7753	1.37	∞	1.00	285506.5082	0.00
beuster	∞	0.79	∞	0.66	∞	0.77	∞	0.00	**116329.6706**	12.85
cecil_13	-114379.0669	7200.15	-107458.2928	432.65	-110871.2829	7200.02	∞	196.29	**-115592.7795**	7200.00
chp_partload	26.9316	7212.52	∞	7200.29	∞	7204.42	∞	0.00	**23.5545**	7199.99
contvar	**809149.5423**	7201.72	813386.9455	7200.60	811036.2023	7200.03	∞	7236.98	∞	0.60
csched1a	-30430.1780	4.33	-30430.1768	0.11	-30430.1768	1.42	-30430.1757	0.81	-30430.1768	0.00
csched1	-30639.2607	156.50	∞	0.90	-30639.2578	42.94	∞	3.98	-30639.2578	0.00
csched2a	-165398.7013	3286.04	-159250.1724	7200.69	-127191.6054	7200.01	-156477.8176	7259.72	-165398.7013	1.13
csched2	-166101.9928	7315.06	-158480.1624	7200.34	-155750.4335	7200.01	∞	7272.93	-166101.9964	13.59
deb10	∞	674.73	∞	268.32	∞	0.00	∞	0.00	**209.4278**	0.25
deb6	∞	1217.27	∞	4.08	∞	0.00	∞	7615.47	**201.7393**	3.51
deb7	∞	4544.21	∞	7200.02	∞	0.00	∞	7433.64	∞	0.94
deb8	∞	4482.77	∞	7200.30	∞	0.00	∞	0.00	∞	0.94
deb9	∞	4920.32	∞	4830.73	∞	0.00	∞	0.00	**116.5846**	11.33
detf1	∞	0.00	12.8818	7200.79	∞	7478.11	**12.7651**	8441.15	12.8818	7200.04
du-opt5	8.0740	4.58	8.0737	8.42	8.0737	144.37	8.0737	38.06	8.0737	8.40
du-opt	3.5566	4.41	3.5637	1.36	3.5563	7200.00	3.5563	75.02	3.5563	1.25
eg_all_s	**7.6578**	326.04	8.9427	0.17	7.9828	7200.21	7.8218	8704.59	7.9202	12.60
eg_disc2_s	**5.6421**	554.37	7.8872	0.17	5.9370	7200.78	∞	9999.92	5.9364	6.55
eg_disc_s	5.7606	635.73	7.3802	0.11	5.7605	7200.40	6.9093	12113.11	5.7605	11.38
eg_int_s	6.4533	196.52	8.0262	0.21	6.4531	7200.19	6.4531	9116.95	6.4531	5.15
elf	0.1917	165.27	1.0570	0.93	0.1917	9.67	0.3000	5.56	0.4347	0.01
eniplac	-132117.0254	7200.05	-132117.0830	1.51	-132117.0563	7200.01	∞	7573.27	-130450.7699	0.07
enpro48	187277.1571	40.08	187277.2594	1.27	187277.1867	9.28	∞	24.37	187277.2594	0.04
enpro48pb	187277.1571	560.52	187277.2594	1.24	187277.2306	14.73	∞	30.14	187277.2594	0.06
enpro56	263428.3589	62.54	263428.3010	25.09	263428.2818	30.89	∞	14.10	263428.3010	0.09
enpro56pb	263428.3589	2515.05	263428.3010	43.51	263428.2818	30.74	∞	20.50	263428.3010	0.14

	RECIPE		AlphaECP		Baron		Couenne		sBB	
Instance	f^*	time	f^*	time	f^*	time	f^*	time	f^*	time
ex1221	7.6672	2.41	∞	7200.02	7.6672	0.00	7.6672	0.04	∞	0.00
ex1222	1.0765	1.37	1.0765	0.00	1.0765	0.01	1.0765	0.04	1.0765	0.00
ex1223a	4.5796	1.43	4.5796	0.01	4.5796	0.01	4.5796	0.04	4.5796	0.00
ex1223b	4.5796	1.59	4.5796	0.01	4.5796	0.05	4.5796	0.07	4.5796	0.00
ex1223	4.5796	1.85	4.5796	0.00	4.5796	0.07	∞	0.17	4.5796	0.00
ex1224	-0.9435	2.59	-0.7191	0.01	-0.9435	0.04	-0.9435	0.25	-0.9435	0.00
ex1225	31.0000	1.51	34.0000	0.00	31.0000	0.02	31.0000	0.08	31.0000	0.00
ex1226	-17.0000	2.41	-17.0000	0.00	-17.0000	0.06	-17.0000	0.08	∞	0.00
ex1233	155010.6713	0.47	155010.6713	41.75	155010.6713	266.26	∞	23.63	155010.6713	0.00
ex1243	83402.5069	10.58	515069.7614	0.00	83402.5064	1.24	∞	2.65	83402.5064	0.00
ex1244	82042.9052	102.00	82042.9052	0.14	82042.9052	2931.47	82042.9052	38.21	82042.9052	0.00
ex1252a	128894.0029	7.82	∞	7200.02	∞	0.03	∞	4.25	128893.7410	0.00
ex1252	128894.0029	23.70	∞	7200.06	151115.7300	0.03	∞	12.51	128893.7410	0.00
ex1263a	19.6000	7.37	21.3000	0.02	19.6000	0.48	19.6000	1.84	19.6000	0.00
ex1263	19.6000	233.24	20.6000	0.40	19.6000	2.18	19.6000	5.01	19.6000	0.00
ex1264a	8.6000	5.88	8.6000	0.02	8.6000	0.70	8.6000	0.70	8.6000	0.00
ex1264	8.6000	35.66	9.3000	0.03	8.6000	3.86	8.6000	3.74	8.6000	0.00
ex1265a	11.5000	10.65	10.6000	0.08	10.3000	0.39	10.3000	1.93	15.1000	0.00
ex1265	15.1000	8.92	11.3000	0.08	10.3000	2.23	10.6000	6.92	10.3000	0.00
ex1266a	16.3000	5.25	16.3000	0.03	16.3000	0.33	16.3000	2.74	16.3000	0.00
ex1266	16.3000	19.24	16.3000	0.20	16.3000	1.67	16.3000	11.65	16.3000	0.00
ex3	68.0097	0.76	77.1043	0.14	68.0097	0.10	68.0097	0.17	68.0097	0.00
ex3pb	68.0097	5.09	68.0097	0.31	68.0097	0.09	68.0097	0.17	68.0097	0.00
ex4	-8.0641	21.40	-8.0641	0.38	-8.0641	3.82	-8.0641	11.54	-8.0641	0.08
fac1	160912612.3502	2.78	160912612.3502	0.00	160912612.3500	0.04	∞	0.08	160912612.3502	0.00
fac2	331837498.1768	7.85	331837498.1768	0.19	331837498.1770	0.40	331837498.1768	1.09	331837498.1768	0.02
fac3	31982309.8480	9.78	31982309.8480	0.03	31982309.8480	0.45	31982309.8482	0.48	31982309.8480	0.00
feedtray2	0.0000	378.54	0.0000	0.10	0.0000	0.43	0.0000	1.34	∞	0.01
feedtray	-13.4114	46.74	∞	7200.82	-13.4060	0.77	∞	7271.11	∞	0.01
fo7_2	17.7493	4169.08	17.7493	23.20	17.7493	3871.55	∞	355.06	17.7493	7199.99
fo7_ar2_1	24.8398	7200.05	24.8398	91.81	24.9721	7200.01	24.8398	287.89	38.1605	7200.00
fo7_ar25_1	23.0936	7200.04	23.0936	68.41	23.1210	7200.01	∞	688.65	25.6784	7199.97
fo7_ar3_1	22.5175	7200.04	22.5175	85.81	22.5175	7200.01	∞	579.37	30.2619	1777.24
fo7_ar4_1	20.7298	7200.05	20.7298	130.62	20.7298	7200.01	20.7298	1272.72	28.4706	7200.00
fo7_ar5_1	17.7493	7200.05	17.7493	43.44	17.7493	1085.21	∞	244.97	17.7493	7200.00
fo7	20.7298	7200.16	20.7298	237.13	22.3925	7200.01	20.7298	1139.76	27.1480	7200.00
fo8_ar2_1	30.3406	7200.05	30.3406	493.52	42.1429	7200.00	∞	7631.57	53.3354	7199.99
fo8_ar25_1	33.3122	7200.05	**28.0452**	580.26	46.0481	7200.00	∞	7592.76	53.3663	7200.00
fo8_ar3_1	30.5745	7200.05	23.9101	190.92	23.9101	7200.00	∞	7688.21	41.9649	7200.00
fo8_ar4_1	37.0390	7200.30	**22.3819**	293.67	23.9976	7200.00	∞	2325.73	39.8196	7199.99
fo8_ar5_1	23.9101	7200.06	**22.3819**	497.73	30.8551	7200.00	23.4680	7611.87	38.0818	7200.00
fo8	22.3819	7200.05	22.3819	545.00	34.6407	7200.01	22.3819	7633.72	51.7311	7200.00
fo9_ar2_1	**36.6681**	7200.07	43.6474	1855.35	∞	7200.00	∞	7623.11	∞	715.44
fo9_ar25_1	54.0811	7200.05	**32.1864**	7200.30	∞	7200.00	32.2500	7658.16	53.1887	7101.40
fo9_ar3_1	37.0128	7200.10	**24.8155**	552.82	∞	7200.01	34.7500	7662.94	∞	7200.00
fo9_ar4_1	45.4231	7200.31	**23.4643**	2067.83	∞	7200.01	31.9533	7653.29	∞	0.07
fo9_ar5_1	∞	7200.08	**23.4643**	2789.53	48.9429	7200.00	∞	7643.03	60.0000	7131.38
fo9	∞	7200.06	**23.4643**	4147.19	51.9927	7200.01	31.2111	7658.19	42.1176	7199.99
fuel	8566.1190	16.10	8566.1190	0.04	8566.1190	0.04	8566.1190	0.16	8566.1190	0.00
fuzzy	∞	0.00	**-0.2961**	88.97	∞	0.00	∞	0.00	∞	0.43
gasnet	**6999391.6436**	117.70	7096273.4570	7200.22	∞	1.34	∞	7674.49	7004607.8064	0.02
gastrans	89.0858	19.55	89.0858	8.13	89.0858	0.41	∞	3.04	89.0858	0.01
gbd	2.2000	2.09	2.2000	0.00	2.2000	0.01	2.2000	0.04	∞	0.00
gear2	0.0000	1.51	0.0000	0.01	0.0000	7200.01	0.0000	0.26	∞	0.00
gear3	0.0000	1.21	0.0000	0.00	0.0000	0.25	∞	0.08	∞	0.00
gear4	1.9682	15.68	945.9914	0.00	1.6434	0.45	1.6434	0.39	1.6434	0.00
gear	0.0000	1.18	0.0000	0.00	0.0000	0.03	0.0000	0.05	∞	0.00
gkocis	-1.9231	3.13	-1.9231	0.01	-1.9231	0.02	-1.9231	0.06	-1.9231	0.00
hmittelman	13.0000	4.57	13.0000	7200.00	13.0000	0.07	13.0000	0.22	13.0000	0.00
johnall	-224.7302	69.25	-224.7302	0.04	-224.7302	6.27	-224.7302	628.64	-224.7302	1.03
lop97ic	4232.6891	7212.59	∞	7200.08	8011.5855	7200.48	∞	7441.10	**4130.3959**	7200.00
lop97icx	**4099.0600**	5381.36	∞	7200.90	4377.0599	7200.01	∞	7437.87	4156.3775	6886.10
m3	37.8000	5.42	37.8000	0.04	37.8000	0.06	37.8000	0.75	37.8000	0.04
m6	82.2569	1414.40	82.2569	3.15	82.2569	155.04	82.2569	16.56	82.2569	749.99
m7_ar2_1	190.2350	7200.04	190.2350	8.75	190.2350	42.96	190.2350	73.25	∞	6785.00
m7_ar25_1	143.5850	2919.44	143.5850	0.92	143.5850	39.90	143.5850	31.73	143.5850	6832.63
m7_ar3_1	143.5850	7200.04	143.5850	17.00	143.5850	175.93	143.5850	66.95	156.0380	6840.07

	RECIPE		AlphaECP		Baron		Couenne		sBB	
Instance	f^*	time	f^*	time	f^*	time	f^*	time	f^*	time
m7_ar4_1	106.7569	7200.05	106.7569	4.79	106.7569	191.70	106.7569	34.68	124.9053	6819.66
m7_ar5_1	106.4600	7200.04	106.4600	18.47	106.4600	416.31	106.4600	154.76	207.7267	6803.37
m7	106.7569	2251.72	106.7569	13.31	106.7569	1082.99	106.7569	118.91	106.7569	6559.48
mbtd	5.5833	7203.37	8.0000	5639.64	∞	7200.90	7.6667	8924.79	**5.4167**	7200.59
meanvarx	14.3692	3.06	14.3692	0.00	14.3692	0.27	∞	1.91	14.3692	0.00
minlphix	316.6928	54.76	316.6927	0.04	424.4954	576.04	316.6927	0.67	641.8714	0.00
netmod_dol1	∞	9720.59	-0.5561	7200.20	-0.1277	7200.23	-0.5600	7287.63	-0.5598	7199.99
netmod_dol2	-0.5535	8930.19	-0.5600	475.40	-0.0000	7200.04	-0.5600	5450.07	-0.5600	279.49
netmod_kar1	-0.4198	7214.51	-0.4198	149.82	-0.4198	895.53	-0.4198	258.70	-0.4198	14.16
netmod_kar2	-0.4198	7213.04	-0.4198	149.86	-0.4198	897.10	-0.4198	258.69	-0.4198	14.43
no7_ar2_1	107.8153	7200.05	107.8153	94.53	107.8153	7200.01	∞	972.41	146.7258	7199.99
no7_ar25_1	107.8153	7200.11	107.8153	453.37	112.3227	7200.01	∞	4626.31	136.2852	7200.00
no7_ar3_1	107.8153	7204.28	107.8153	636.73	113.7244	7200.00	∞	5251.87	135.0166	7200.00
no7_ar4_1	98.5184	7200.04	98.5184	525.84	107.8715	7200.00	∞	7720.82	142.4666	7200.00
no7_ar5_1	90.6227	7200.04	90.6227	204.49	90.6227	7200.01	90.6227	1407.96	120.0595	7200.00
nous1	∞	21.96	1.5671	7200.03	1.5671	254.59	∞	7479.14	1.5671	0.00
nous2	∞	21.99	1.3843	7200.05	0.6260	1.01	0.6260	1.28	0.6260	0.00
nuclear104	∞	7267.51	∞	7200.20	∞	7283.30	∞	0.00	∞	1053.89
nuclear10a	∞	0.00	∞	0.00	∞	7255.57	∞	13563.55	∞	7200.02
nuclear10b	∞	7223.50	∞	0.00	∞	7232.08	∞	8407.36	**-1.1333**	7200.35
nuclear14a	**-1.1296**	2732.95	∞	7200.51	∞	7200.32	-1.1144	7290.36	∞	0.46
nuclear14b	-1.1093	7200.37	∞	9.04	-1.1096	7200.24	-1.1002	7363.34	**-1.1128**	7199.99
nuclear14	**-1.1257**	6062.20	∞	142.72	-1.1237	66.75	∞	7237.89	-1.1199	1.64
nuclear24a	**-1.1296**	2769.73	∞	7200.23	∞	7200.42	-1.1144	7283.95	∞	0.46
nuclear24b	-1.1093	7201.06	∞	8.72	-1.1096	7200.35	-1.1002	7358.12	**-1.1128**	7200.00
nuclear24	**-1.1257**	6006.97	∞	142.51	-1.1237	68.24	∞	7233.55	-1.1199	1.63
nuclear25a	∞	0.00	∞	1.68	∞	7200.08	∞	7303.84	**-1.1198**	2.69
nuclear25b	-1.0977	7201.25	∞	4.05	-1.0749	7200.14	-1.0975	7356.74	-1.0954	7200.00
nuclear25	-1.1171	7225.31	∞	7200.08	∞	7200.22	∞	7233.25	**-1.1197**	72.90
nuclear49a	∞	8293.68	∞	11.97	-1.1509	7204.33	∞	4951.04	-1.1512	4.73
nuclear49b	∞	0.00	∞	24.79	-1.1180	7200.87	∞	2427.45	**-1.1217**	7199.99
nuclear49	∞	7200.83	∞	7200.10	∞	7208.12	∞	7217.00	**-1.1511**	77.89
nuclearva	**-1.0109**	1323.10	∞	5728.20	∞	1804.66	∞	7595.48	-1.0095	0.12
nuclearvb	-1.0289	1408.14	∞	53.30	∞	2988.50	∞	7429.02	-1.0281	0.24
nuclearvc	-0.9972	1370.61	∞	7200.82	-0.9968	3677.68	∞	7382.91	∞	0.06
nuclearvd	-1.0315	1806.26	∞	290.33	**-1.0330**	3810.13	∞	7391.59	∞	0.07
nuclearve	-1.0317	2013.02	∞	451.96	∞	2445.54	∞	7388.95	-1.0322	0.20
nuclearvf	**-1.0225**	1911.18	∞	303.31	-1.0176	3960.56	∞	7313.94	-1.0209	0.13
nvs01	12.4697	1.28	∞	0.11	12.4697	0.05	12.4697	0.06	12.4697	0.00
nvs02	5.9642	3.49	∞	7200.03	5.9642	0.01	5.9642	0.04	5.9642	0.00
nvs03	16.0000	1.11	16.0000	0.00	16.0000	0.01	16.0000	0.05	16.0000	0.00
nvs04	0.7200	2.09	2.1200	0.00	0.7200	0.01	0.7200	0.06	∞	0.00
nvs05	5.4709	42.91	∞	7200.11	5.4709	5.79	∞	5.15	5.4709	0.00
nvs06	1.7703	1.15	1.7703	0.00	1.7703	0.01	1.7703	0.03	1.7703	0.00
nvs07	4.0000	1.40	4.0000	0.00	4.0000	0.02	4.0000	0.04	4.0000	0.00
nvs08	23.4497	1.55	23.4497	0.01	23.4497	0.05	23.4497	0.05	23.4497	0.00
nvs09	-43.1343	0.95	-43.1343	0.00	-43.1343	0.00	-43.1343	0.99	∞	0.00
nvs10	-310.8000	1.28	-310.8000	0.01	-310.8000	0.03	-310.8000	0.06	-310.8000	0.00
nvs11	-431.0000	1.20	-431.0000	0.02	-431.0000	0.06	-431.0000	0.12	-431.0000	0.00
nvs12	-481.2000	1.43	-481.2000	0.01	-481.2000	0.07	-481.2000	0.29	-481.2000	0.00
nvs13	-585.2000	1.53	-585.2000	0.02	-585.2000	0.23	-585.2000	0.82	-585.2000	0.00
nvs14	-40358.1142	3.32	∞	7200.04	-40358.1548	0.01	-40358.1548	0.05	-40358.1548	0.00
nvs15	1.0000	1.30	1.0000	0.00	1.0000	0.02	1.0000	0.05	1.0000	0.00
nvs16	0.7031	1.63	0.7031	0.00	0.7031	0.01	0.7031	0.05	0.7031	0.00
nvs17	-1100.4000	2.27	-1100.4000	2.31	-1100.4000	4.45	-1100.4000	14.35	-1100.4000	0.00
nvs18	-778.4000	1.90	-778.4000	0.32	-778.4000	0.91	-778.4000	2.17	-778.4000	0.00
nvs19	-1098.4000	2.95	-1098.4000	9.19	-1098.4000	15.46	∞	0.00	-1098.4000	0.00
nvs20	230.9222	1.83	230.9222	1.24	230.9222	1.34	230.9222	3.37	230.9222	0.00
nvs21	-5.6848	1.56	-0.0001	0.01	-5.6848	0.03	-4.4552	0.05	-5.6848	0.00
nvs22	6.0582	57.65	40.3053	7200.09	6.0582	0.65	6.0582	0.14	6.0582	0.00
nvs23	-1125.2000	4.63	-1125.2000	10.30	-1125.2000	61.63	-1125.2000	649.69	-1125.2000	0.02
nvs24	-1033.2000	4.65	-1033.2000	104.49	-1033.2000	314.06	∞	7764.79	-1033.2000	0.10
o7_2	116.9459	7200.04	116.9459	967.56	123.9855	7200.00	118.7666	7620.36	123.9808	7200.00
o7_ar2_1	140.4120	7200.04	140.4120	193.21	146.2676	7200.01	∞	1571.35	163.5146	7199.99
o7_ar25_1	141.6231	7202.21	**140.4120**	461.57	141.1893	7200.00	∞	7482.83	172.4345	7199.93
o7_ar3_1	138.8649	7200.04	**137.9318**	1366.73	138.8649	7200.01	∞	7631.79	165.5136	2584.10
o7_ar4_1	131.6531	7200.04	131.6531	2526.52	138.8649	7200.01	∞	7630.42	170.6387	7199.99

	RECIPE		AlphaECP		Baron		Couenne		sBB	
Instance	f^*	time	f^*	time	f^*	time	f^*	time	f^*	time
o7_ar5_1	116.9458	7200.55	116.9458	457.25	123.9807	7200.00	∞	7636.28	133.5345	7199.99
o7	131.6531	7203.14	131.6531	3235.39	137.8693	7200.01	134.1667	7638.95	159.2485	6243.61
o8_ar4_1	254.9776	7200.06	**251.3129**	7200.81	∞	7200.00	∞	7675.80	313.4221	7200.00
o9_ar4_1	316.0128	7200.08	**311.9231**	7200.06	∞	7200.00	∞	7683.12	∞	7200.00
oaer	-1.9231	2.72	0.0000	0.00	-1.9231	0.01	-1.9231	0.08	-1.9231	0.00
oil2	∞	0.00	∞	52.59	-0.7333	7200.01	∞	1825.65	-0.7333	0.20
oil	-0.9325	7202.21	-0.8514	520.03	-0.8367	7201.62	∞	0.00	-0.9325	1406.28
ortez	-9532.0398	68.70	-3668.8535	0.18	-9532.0391	0.26	∞	0.77	-9532.0391	0.01
parallel	924.2986	1.14	924.2956	0.12	924.2947	51.72	∞	0.00	924.2956	0.07
prob02	112235.0000	1.45	112235.0000	0.00	112235.0000	0.01	112235.0000	0.04	112235.0000	0.00
prob03	10.0000	1.07	10.0000	0.00	10.0000	0.00	10.0000	0.04	10.0000	0.00
prob10	3.4455	1.60	∞	0.89	∞	0.00	3.4455	0.05	3.4455	0.00
procsel	-1.7210	0.66	-1.9231	0.00	-1.9231	0.03	-1.9231	0.06	-1.9231	0.00
product2	-2096.7699	7201.63	∞	1.98	∞	15.70	∞	7472.33	**-2100.0774**	7200.00
product	-2142.9275	7200.50	-1922.0353	7200.05	∞	23.13	∞	2400.41	-2142.9481	1355.45
pump	**128894.0029**	7.75	∞	3.65	∞	0.02	∞	5.40	134263.5853	0.00
qap	**388870.0000**	1541.90	400528.0000	26.65	390374.0000	7200.83	∞	7764.23	395738.0000	7.53
qapw	**388988.0000**	4410.31	402948.0000	33.71	440710.0000	7200.11	403182.0000	7393.13	394086.0000	1.81
ravem	269590.2650	2.93	269590.2193	0.85	269590.0793	3.91	∞	16.21	269590.2193	0.01
ravempb	269590.2650	57.39	269590.2193	1.22	269590.0793	3.78	∞	18.26	269590.2193	0.01
risk2b	**-56.8208**	305.67	-55.8761	0.61	-55.8761	2.26	∞	27.17	-55.7362	0.07
risk2bpb	**-56.8208**	1631.20	-55.8761	0.62	-55.8761	2.43	∞	28.06	-55.8761	0.06
saa_2	∞	0.00	12.8818	7200.16	∞	7474.95	**12.7651**	8371.30	12.8818	7200.10
sep1	-470.1301	15.26	-510.0810	0.02	-510.0810	0.04	∞	0.29	-510.0810	0.00
space25a	484.3278	302.06	∞	7201.24	499.1340	7200.00	∞	7628.21	484.3286	2.14
space25	484.3278	1146.59	∞	7200.55	∞	7200.01	∞	7641.96	484.3286	6.30
space960	∞	7200.65	∞	7200.23	∞	7216.49	∞	7369.59	∞	7199.99
spectra2	13.9783	228.87	13.9783	94.59	13.9783	17.14	13.9783	3.99	13.9783	0.16
spring	0.8462	28.17	∞	107.95	0.8462	0.23	∞	0.35	0.8462	0.00
st_e13	2.0000	2.85	2.0000	0.00	2.0000	0.01	2.0000	0.04	∞	0.00
st_e14	4.5796	1.84	4.5796	0.01	4.5796	0.06	∞	0.17	4.5796	0.00
st_e15	7.6672	2.43	∞	7200.02	7.6672	0.01	7.6672	0.04	∞	0.00
st_e27	2.0000	1.22	2.0000	0.00	2.0000	0.01	2.0000	0.05	∞	0.00
st_e29	-0.9435	2.57	-0.7191	0.01	-0.9435	0.03	-0.9435	0.25	-0.9435	0.00
st_e31	-2.0000	24.51	-2.0000	0.05	-2.0000	0.92	∞	5.13	∞	0.00
st_e32	-1.4304	0.19	-1.2404	3046.57	-1.4304	4.31	-1.2404	4.48	-1.4304	0.01
st_e35	∞	0.21	301340.5026	0.00	**64868.0769**	4.30	∞	3.27	∞	0.00
st_e36	-246.0000	2.40	-226.1825	0.01	-246.0000	0.04	-246.0000	0.53	∞	0.02
st_e38	7197.7216	1.14	7197.7271	0.00	7197.7271	0.02	∞	0.06	∞	0.00
st_e40	30.4142	4.58	∞	0.85	30.4142	0.04	30.4142	0.37	∞	0.00
st_miqp1	281.0000	1.33	281.0000	0.00	281.0000	0.01	281.0000	0.04	281.0000	0.00
st_miqp2	2.0000	7.37	2.0000	0.00	2.0000	0.02	2.0000	0.05	2.0000	0.00
st_miqp3	-6.0000	1.19	-6.0000	0.00	-6.0000	0.00	-6.0000	0.04	∞	0.00
st_miqp4	-4574.0000	1.32	-4574.0000	0.00	-4574.0000	0.01	-4574.0000	0.05	∞	0.00
st_miqp5	-333.8900	2.49	-333.8889	0.00	-333.8889	0.01	-333.8889	0.06	∞	0.00
stockcycle	119948.7606	5124.47	123932.6350	7200.87	119948.6883	566.94	119948.6883	108.87	119948.6883	11.35
st_test1	-0.0000	1.27	0.0000	0.00	0.0000	0.01	0.0000	0.05	0.0000	0.00
st_test2	-9.2500	1.66	-9.2500	0.00	-9.2500	0.02	-9.2500	0.04	∞	0.00
st_test3	-7.0000	1.59	-7.0000	0.00	-7.0000	0.03	-7.0000	0.07	∞	0.00
st_test4	-7.0000	1.61	-7.0000	0.00	-7.0000	0.01	-7.0000	0.04	∞	0.00
st_test5	-110.0000	1.45	-110.0000	0.00	-110.0000	0.02	-110.0000	0.07	-110.0000	0.00
st_test6	471.0000	1.28	471.0000	0.00	471.0000	0.04	471.0000	0.10	471.0000	0.00
st_test8	-29605.0000	2.46	-29605.0000	0.00	-29605.0000	0.03	-29605.0000	0.07	-29605.0000	0.00
st_testgr1	-12.8116	1.47	-12.8116	0.01	-12.8116	0.07	-12.8116	0.26	-12.8116	0.00
st_testgr3	-20.5900	2.54	-20.5900	0.00	-20.5900	0.80	-20.5900	0.81	-20.5900	0.00
st_testph4	-80.5000	1.33	-80.5000	0.00	-80.5000	0.01	-80.5000	0.04	-80.5000	0.00
super1	**9.6438**	7224.29	∞	7200.30	∞	26.40	∞	0.00	∞	0.53
super2	**5.2468**	7221.22	∞	7200.64	∞	35.57	∞	0.00	∞	0.75
super3	**12.9385**	7205.99	∞	7200.54	∞	35.48	∞	0.00	∞	0.63
super3t	**-0.6684**	7206.03	-0.6414	7200.08	∞	7200.34	∞	0.00	∞	0.22
synheat	154997.3349	11.13	154997.3349	7.36	154997.3354	266.64	∞	98.46	154997.3349	0.00
synthes1	6.0098	1.37	6.0098	0.00	6.0098	0.02	∞	0.07	6.0098	0.00
synthes2	73.0353	3.90	73.0353	0.01	73.0353	0.03	73.0353	0.13	73.0353	0.00
synthes3	68.0097	2.48	68.0097	0.03	68.0097	0.07	68.0903	0.12	68.0097	0.00
tln12	106.8000	7200.04	**99.8000**	7200.76	∞	7200.02	∞	0.00	∞	7200.00
tln2	5.3000	7.15	5.3000	0.00	5.3000	0.03	5.3000	0.09	5.3000	0.00
tln4	8.3000	541.30	8.8000	0.47	8.3000	4.70	8.3000	2.64	8.3000	0.00

Instance	RECIPE		AlphaECP		Baron		Couenne		sBB	
	f^*	time	f^*	time	f^*	time	f^*	time	f^*	time
tln5	10.3000	442.67	11.1000	1.17	10.3000	194.45	10.3000	32.25	11.3000	7200.00
tln6	15.4000	7200.03	16.3000	3.96	15.3000	7200.00	15.3000	7511.64	15.7000	7200.00
tln7	15.6000	7200.03	16.4000	709.71	15.4000	7200.01	**15.0000**	7535.61	∞	7200.00
tloss	16.3000	5.30	16.3000	0.69	16.3000	0.48	16.3000	5.17	16.3000	0.00
tls12	∞	7200.23	∞	7200.04	∞	7200.21	∞	7309.00	∞	7200.00
tls2	5.3000	7.56	5.3000	0.00	5.3000	0.52	5.3000	1.25	5.3000	0.00
tls4	8.8000	7200.04	8.3000	1987.74	8.3000	520.37	8.3000	3327.67	8.5000	7200.00
tls5	14.2000	7200.67	15.9000	7200.74	**10.6000**	7200.00	∞	7339.42	13.6000	7200.00
tls6	16.4000	7200.10	∞	7200.37	**16.3000**	7200.01	∞	7281.92	∞	7200.00
tls7	∞	7200.16	∞	7200.87	∞	7200.01	∞	7256.41	∞	7200.00
tltr	∞	2.67	48.0667	0.00	48.0667	0.16	48.0667	1.32	∞	0.00
uselinear	∞	816.29	∞	90.77	∞	7200.21	∞	0.00	∞	226.65
util	999.5790	313.65	999.5788	0.78	999.5788	0.57	∞	0.00	999.5788	0.00
var_con10	444.1032	2550.53	∞	2.93	∞	0.00	∞	7583.87	444.2140	0.98
var_con5	278.0384	2293.98	∞	6.78	∞	0.00	∞	7578.81	278.1449	0.88
waste	**679.0943**	7208.15	1056.3587	7200.19	711.5902	1650.71	∞	7944.02	∞	0.66
water4	**907.0174**	298.20	1115.6358	3788.09	1051.8529	7200.00	∞	7600.85	910.8822	7.23
waterx	914.1837	142.69	917.3819	1.70	**911.2815**	7200.01	∞	7543.11	938.8628	0.25
waterz	910.8823	7200.08	1129.9849	18.05	∞	7200.01	∞	7584.73	**907.0170**	2101.60
windfac	∞	4.87	0.2545	7200.01	∞	0.00	0.2545	0.62	0.2545	0.00

Table 3: Comparison of the results obtained by RECIPE with Ipopt as NLP subsolver, AlphaECP, Baron, Couenne and sBB.

	RECIPE	AlphaECP	Baron	Couenne	sBB
# feasible solutions	221	192	203	136	199
# at least as good solutions	195	146	160	121	149
# strictly better solutions	25	14	5	3	18
Average CPU time	112.44	29.28	55.63	52.65	13.03
CPU time:					
< 600	147	184	155	157	183
< 1800	11	4	5	8	6
< 7200	20	12	8	9	11
7200+	73	51	83	77	51

Table 4 Summary of the results in Table 3. For each algorithm, we report: the number of instances for which a feasible solution is found, the number of instances for which it finds a solution with an objective value (∞ included) that is at least as good as the other methods, the number of instances for which it finds a solution strictly better than the remaining methods, the average CPU time in seconds (geometric mean), and the number of instances that require a given CPU time.

As can be seen from Table 4, RECIPE is very effective. Not only does it find more feasible solutions (18 solutions more than its closest competitor, Baron), but they are of better quality: for 195 instances out of 251 (78%), RECIPE with Ipopt finds a solution which is at least as good as the remaining methods, and comes out as the winning algorithm in the comparison with respect to this criterion. A more detailed analysis of Table 3 indicates that AlphaECP is typically the fastest method on convex instances (such as the fo, m7, no7 and o7 instances); RECIPE is more effective on nonconvex instances such as the nuclear, super and water problems. On the majority of our test problems, RECIPE returns a solution which is at least of the same quality as the ones found by the exact BB solvers for nonconvex MINLPs Baron and Couenne, and

the convex MINLP solvers `AlphaECP` and `sBB`; in several cases the solution is strictly better. For applications where no guarantee of optimality is required, and the problem is not known to be convex, RECIPE seems to be a better choice.

In terms of CPU time, RECIPE is the slowest algorithm on average, even though `Baron` and `Couenne` hit the time limit on a larger number of instances. There are two reasons for this. First, on instances that run in just a few seconds ($< \approx 5$ seconds), the overhead for employing AMPL as a scripting language is not negligible. Second, RECIPE cannot stop because a solution is proven to be optimal. There are several instances on which the solvers that we compare RECIPE to are able to quickly prove optimality, whereas RECIPE keeps running even after finding the optimal solution.

We observe that `Couenne` exceeds the 2 hours time limit on some instances, sometimes by a large amount. This is due to the solution of time-consuming NLPs or LPs: `Couenne` uses no or very loose upper bounds on the time consumption for solving subproblems, therefore it may happen that the time limit is exceeded while the main algorithm is waiting for a subproblem to be solved. From Table 4, we can also see that `Couenne` seems to be the weakest solver among the ones that were tested. Computational testing revealed that this is due to small infeasibilities in the reported solutions. Indeed, `Couenne` reports feasible solutions on most instances, but many of those are discarded by the post-processing scripts that we employed to ensure that all solutions satisfied our feasibility criteria (the same used by RECIPE). In our tests, `Couenne` seemed to have troubles with constraints involving exponential functions, and it often reported solutions violating those constraints by small amounts. If we trust `Couenne`'s feasibility test, we obtain a different ranking of the solvers employed in this section. In Table 5, we report the analogue of Table 4 with the difference that now we accept all solutions found to be feasible by `Couenne`, regardless of the result of our post-processing feasibility check. In this case, it can be seen that `Couenne` finds more feasibile solutions that any other method. However, RECIPE is still the best method in terms of solution quality.

3.5 Comparison with the Feasibility Pump

We now compare RECIPE with another heuristic for nonconvex MINLPs: the recently proposed Feasibility Pump [15,14, Chapter 2]. Results for FP are taken from [16]; since computational experiments with several versions of the heuristic are reported, we keep the best one available (i.e. best objective function value or best CPU time if the objective value is tied). Likewise, we consider the best solution found by RECIPE using either `Ipopt` or `filterSQP` as the NLP solver. [16] reports detailed results on a much smaller test set than ours (they only consider hard instances), and rounds CPU times to the nearest second; we conform to that paper.

	RECIPE	AlphaECP	Baron	Couenne	sBB
# feasible solutions	221	192	203	234	199
# at least as good solutions	183	136	151	168	136
# strictly better solutions	24	9	5	18	16
Average CPU time	112.44	29.28	55.63	52.65	13.03
CPU time:					
< 600	147	184	155	157	183
< 1800	11	4	5	8	6
< 7200	20	12	8	9	11
7200+	73	51	83	77	51

Table 5 Summary of the results when trusting the feasibility checks of Couenne. For each algorithm, we report: the number of instances for which a feasible solution is found, the number of instances for which it finds a solution with an objective value (∞ included) that is at least as good as the other methods, the number of instances for which it finds a solution strictly better than the remaining methods, the average CPU time in seconds (geometric mean), and the number of instances that require a given CPU time.

	RECIPE		Feasibility Pump	
Instance	f^*	time	f^*	time
beuster	∞	0.00	∞	7200.00
csched2a	**-165398.7013**	3286.00	-112174.7256	624.00
csched2	**-166101.9928**	7315.00	-120066.0152	241.00
deb10	∞	4.00	**223.2910**	25.00
deb6	∞	3.00	**234.7799**	197.00
deb7	∞	10.00	**345.7648**	10.00
deb8	∞	17.00	**416332.3238**	3.00
deb9	∞	11.00	**425.3353**	16.00
detf1	∞	0.00	**8455.7458**	961.00
eg_all_s	**7.6578**	201.00	223.1413	27.00
eg_disc2_s	**5.6421**	293.00	65822.9573	7.00
eg_disc_s	**5.7606**	168.00	94165.4165	8.00
eg_int_s	**6.4533**	142.00	94167.1250	10.00
fo8_ar25_1	**32.4477**	7200.00	994207.0612	185.00
fo8_ar3_1	**23.9101**	7200.00	994235.3292	784.00
fo8	**22.3819**	7200.00	894678.4196	9.00
fo9_ar2_1	**32.6250**	7200.00	1136279.4869	1.00
fo9_ar25_1	**43.7722**	7200.00	1136997.7323	635.00
fo9_ar4_1	**43.6116**	7202.00	9959.6821	202.00
fo9_ar5_1	**28.6727**	7200.00	1428148.2023	17.00
fo9	**30.7500**	7200.00	1006964.2111	61.00
johnall	**-224.7302**	6.00	-221.9212	618.00
lop97ic	**4138.3292**	7200.00	∞	7200.00
mbtd	**4.6667**	7203.00	91.3293	4.00
nuclear104	∞	7267.00	∞	7200.00
nuclear10a	∞	0.00	∞	7200.00
nuclear10b	∞	750.00	∞	7200.00
nuclear14a	**-1.1296**	2732.00	-1.1134	2.00
nuclear14b	**-1.1093**	7200.00	-1.1031	646.00
nuclear14	**-1.1257**	6062.00	-1.1156	1.00
nuclear24a	**-1.1296**	2769.00	-1.1134	2.00
nuclear24b	**-1.1093**	7201.00	-1.0904	1.00
nuclear24	**-1.1257**	6006.00	-1.1195	1.00
nuclear25a	**-1.1000**	7217.00	-1.0614	6.00
nuclear25b	**-1.0977**	7201.00	-1.0451	707.00
nuclear49a	∞	8188.00	**-1.1109**	6.00
nuclear49b	∞	0.00	**-1.1319**	4.00
nuclear49	∞	7200.00	**-1.1373**	1.00
nuclearva	**-1.0109**	1323.00	-1.0087	496.00
nuclearvb	**-1.0289**	1408.00	-1.0252	614.00
nuclearvc	-0.9972	1370.00	-0.9971	2.00
nvs08	**23.4497**	0.00	24116.9437	0.00
nvs20	**230.9222**	1.00	138691481.6708	0.00
nvs24	**-1033.2000**	3.00	-536.2000	1.00

Instance	RECIPE		Feasibility Pump	
	f^*	time	f^*	time
o8_ar4_1	**254.9776**	7200.00	5822973.4498	22.00
o9_ar4_1	**297.0642**	7200.00	6877522.8246	198.00
qapw	**388214.0000**	5187.00	460118.0000	637.00
saa_2	∞	0.00	**8455.7458**	978.00
space25a	**484.3278**	32.00	650.6916	245.00
space25	**484.3278**	223.00	650.6916	773.00
space960	∞	0.00	**24070000.0017**	3.00
super1	**9.6438**	7224.00	∞	7200.00
super2	**5.2468**	7221.00	∞	7200.00
super3	**12.9385**	7205.00	∞	7200.00
super3t	**-0.6684**	7206.00	∞	7200.00
tln12	**102.3000**	7200.00	∞	7200.00
tls12	∞	7200.00	∞	7200.00
tls2	5.3000	4.00	5.3000	1.00
tls5	**11.0000**	7200.00	22.5000	58.00
tls6	**16.4000**	7200.00	∞	7200.00
tls7	∞	7200.00	**37.8000**	2.00
uselinear	∞	816.00	**1951.3743**	187.00
var_con10	**444.1032**	2550.00	463.1678	12.00
var_con5	**278.0384**	2293.00	315.1640	6.00
waste	**679.0943**	7208.00	62025.7781	50.00

Table 6: Comparison of RECIPE to the Feasibility Pump for nonconvex MINLPs.

	RECIPE	Feasibility Pump
# feasible solutions	47	53
# at least as good solutions	52	20
# strictly better solutions	45	13
Average CPU time	578.76	76.16
CPU time:		
< 600	22	42
< 1800	5	11
< 7200	8	0
7200+	30	12

Table 7 Summary of the results in Table 6. For each algorithm, we report: the number of instances for which a feasible solution is found, the number of instances for which it finds a solution with an objective value (∞ included) that is at least as good as the other methods, the number of instances for which it finds a solution strictly better than the remaining methods, the average CPU time in seconds (geometric mean), and the number of instances that require a given CPU time.

As expected, FP is more successful in terms of feasibility: for most instances (53 out of 65 – 6 instances more than RECIPE), at least one of the variants of FP is able to find a feasible solution. FP is also faster, hitting the time limit only 12 times, as opposed to RECIPE's 30 (this explains the large difference in the geometric mean). However, FP returns as soon as a first feasible solution is found, while RECIPE keeps trying to improve the solution: as a result, RECIPE finds solution of much better quality in most of the instances. Indeed, whenever RECIPE returns a feasible solution, it is at least as good as the one returned by FP. These differences reflect the purposes for which the heuristics were designed: quickly finding feasible solutions (FP) or finding solutions of good quality (RECIPE).

3.6 New optima

Our computational experiments with RECIPE led to the discovery of some solutions for MINLPLib instances that, to the best of our knowledge, improve on the currently best known objective values (taken from the MINLPLib webpage and amended with better values found on recent papers or during our experiments with MINLP solvers), and were *not* found by any other of the tested solvers. These new solutions have the following values (we report the previously known best feasible solutions in brackets):

- `nuclear14a`: -1.1296 (-1.1280)
- `nuclear24a`: -1.1296 (-1.1280)
- `nuclearvf`: -1.0225 (-1.0194)
- `risk2b`: -56.8208 (-55.8761)
- `risk2bpb`: -56.8208 (-55.8761)

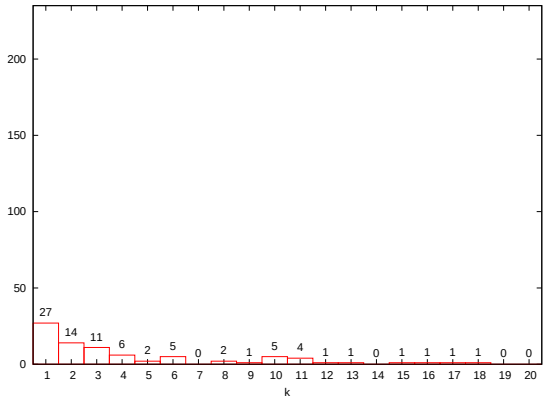
3.7 Why does RECIPE work?

We now try to investigate the reason behind the practical effectiveness of RECIPE. The key algorithmic idea of RECIPE is to run several local searches on smaller, easier subproblems (see the definition of $Q_k(x^*)$), instead of always searching in the original space; RECIPE would not work if solving these subproblems did not yield an improved solution. It turns out that in practice, very often improved solutions can be found by solving $Q_k(x^*)$ with a small value of k, and they are found in a very short CPU time. We provide data to support our claims; all data reported in this section is obtained by running RECIPE with `Ipopt` as NLP solver and $k_{\max} = 20$ on the 221 instances for which a feasible solution is found.

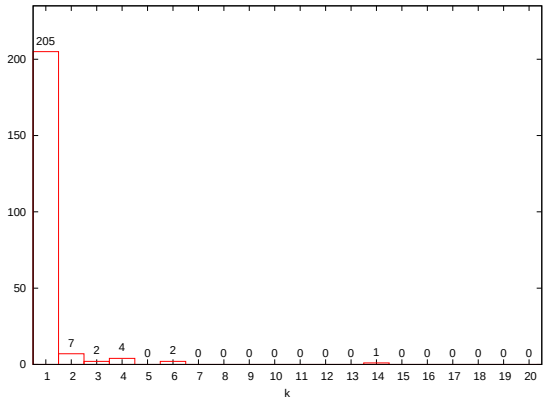
First, we note that solving $Q_k(x^*)$ is faster than solving the original problem P. On the instances for which more than one iteration of RECIPE is run, we recorded the average CPU time for a local search iteration (i.e. running the NLP and the MINLP subsolvers) before a feasible solution is found, and after. We found that before a feasible solution is known, each iteration takes 647.32 seconds on average; but after, i.e. when the solvers are applied on $Q_k(x^*)$ instead of P, the time per iteration decreases to 54.96. Therefore, each problem $Q_k(x^*)$ can be solved considerably faster than P.

Additionally, improved solutions are typically found for small k. In Figure 1(a), we report the number of times that RECIPE finds an improved solution while exploring a neighborhood of size $k = 1, \ldots, 20$, on the instances in the test set for this section. We can see that most solutions are found for very small values of k. This suggests that our neighborhood definition is effective in practice, and leads RECIPE to a quick discovery of good feasible solutions.

We now want to assess the importance of running the NLP solver to find a starting point for the MINLP solver, and of using the neighborhood structure for sampling points and constraining the search. These are two components



(a) Discovery of improved solution



(b) Discovery of first feasible solution

Fig. 1 Number of times that an improved solution (Fig. 1(a)) and the first feasible solution (Fig. 1(b)) is found during the exploration of a neighborhood of size $k = 1, \ldots, 20$.

that are not present in a simple multistart algorithm, and we want to verify whether they are helpful or not. To do so, we ran two additional experiments. First, we ran RECIPE without the NLP solver. Second, we ran a simple multistart algorithm that uses `minlp_bb` as local subsolver, where the initial point is chosen uniformly at random within the initial box constraints (i.e. ignoring the neighborhood definition of RECIPE) and we use the same time and iteration limit as RECIPE. Results are given in Table 8; for space reasons, we do not report detailed results. We observe the following. Skipping the NLP subsolver during the local search decreases performance in terms of feasibility and solution quality. This loss of performance occurs on $\approx 5\%$ of our set of test instances. Therefore, finding a constraint-feasible point with an NLP solver to start the MINLP solver is very important on some of the instances. We also note that skipping the NLP solver brings an advantage in terms of average

	RECIPE	RECIPE no NLP	`minlp_bb` multistart
# feasible solutions	221	210	195
# at least as good solutions	221	212	196
# strictly better solutions	28	10	10
Average CPU time	112.96	78.26	78.20
CPU time:			
< 600	147	154	151
< 1800	11	5	4
< 7200	20	12	10
7200+	73	80	86

Table 8 Comparison of default RECIPE (`Ipopt` and `minlp_bb` as subsolvers, $k_{\max} = 20$, 2 hours time limit) with RECIPE *without* the NLP subsolver and with a multistart algorithm implemented on top of `minlp_bb` (2 hours time limit). For each algorithm, we report: the number of instances for which a feasible solution is found, the number of instances for which it finds a solution with an objective value (∞ included) that is at least as good as the other methods, the number of instances for which it finds a solution strictly better than the remaining methods, the average CPU time in seconds (geometric mean), and the number of instances that require a given CPU time.

CPU time, but the advantage is mostly gained on small, easy instances. Indeed, the number of (difficult) instances that hit the time limit increases. If we now remove from RECIPE the neighborhood structure (as well as the NLP solver), essentially turning it into a simple multistart algorithm, we observe (Table 8) a further loss of performance: $\approx 10\%$ of the instances are affected, in terms of feasibility and solution quality. These experiments confirm that all the components of RECIPE contribute to the effectiveness of the algorithm.

Finally, in Figure 1(b) we report the number of times that the first feasible solution for an instance is found while exploring a neighborhood of size $k = 1, \ldots, 20$ of the initial (infeasible) point x^*, defined as in (2). On most instances, a feasible solution is found in the very first iteration, which allows RECIPE to switch to problems of the form $Q_k(x^*)$ in order to improve the incumbent. For particular classes of structured problems, for which an initial feasible solution is known in advance or can be easily constructed, we believe that RECIPE could be made more effective by skipping the initialization phase, and moving directly to the solution improvement part of the algorithm, which represents its core. An example of such an application is given in [44, Chapter 9.3]. Another possibility to decrease the CPU time required before switching to the solution improvement part of RECIPE is to use a MINLP solver which provides the option of terminating as soon as a solution is available (this is not possible with `minlp_bb`).

4 Conclusion

This paper describes a heuristic approach for finding solutions to nonconvex MINLPs. Our approach, called RECIPE, combines several existing exact and heuristic techniques, resulting in a method that can successfully find solutions of very good quality to many difficult MINLPs. Such a reliable solver would be particularly useful in industrial applications, where proving optimality is of relative importance, and the possibility of plugging into the heuristic any NLP and MINLP solution algorithm available is a great advantage.

Extensive computational experiments show that RECIPE performs better than existing solvers (both heuristic and exact) for MINLPs in terms of solution quality. A more detailed analysis of the results indicates that the reason for RECIPE's effectiveness is the rapidity with which feasible solutions are improved, exploring small neighborhoods.

Acknowledgements

We are very grateful to Prof. Tapio Westerlund for carefully checking all the computational results and informing us of some misprints on the MINLPLib website, to Claudia D'Ambrosio for providing us with the latest computational results from the Feasibility Pump, to Prof. Pierre Hansen and to the anonymous referees for providing useful suggestions that greatly improved the presentation of the paper.

References

1. Abhishek, K., Leyffer, S., Linderoth, J.: Filmint: An outer-approximation based solver for nonlinear mixed-integer programs. Tech. Rep. ANL/MCS-P1374-0906, Argonne National Laboratory (2007)
2. Adjiman, C., Androulakis, I., Floudas, C.: Global optimization of MINLP problems in process synthesis and design. Computers & Chemical Engineering **21**, S445–S450 (1997)
3. Aouchiche, M., Bonnefoy, J., Fidahoussen, A., Caporossi, G., Hansen, P., Hiesse, L., Lacheré, J., Monhait, A.: VNS for extremal graphs 14: The AGX 2 system. In: Liberti and Maculan [40], pp. 281–308
4. Belotti, P.: Couenne: a user's manual. Tech. rep., Lehigh University (2009). URL `http://www.coin-or.org/Couenne`
5. Belotti, P., Lee, J., Liberti, L., Margot, F., Wächter, A.: Branching and bounds tightening techniques for non-convex MINLP. Optimization Methods and Software **24**(4-5), 597–634 (2008)
6. Bonami, P., Biegler, L., Conn, A., Cornuéjols, G., Grossmann, I., Laird, C., Lee, J., Lodi, A., Margot, F., Sawaya, N., Wächter, A.: An algorithmic framework for convex Mixed Integer Nonlinear Programs. Tech. Rep. RC23771, IBM Corporation (2005)
7. Bonami, P., Cornuéjols, G., Lodi, A., Margot, F.: A feasibility pump for Mixed Integer Nonlinear Programs. Mathematical Programming **119**(2), 331–352 (2009)
8. Bonami, P., Lee, J.: `BONMIN` user's manual. Tech. rep., IBM Corporation (2007)
9. Brimberg, J., Hansen, P., Mladenović, N.: Attraction probabilities in variable neighborhood search. 4OR **8**, 181–194 (2010)
10. Brimberg, J., Mladenović, N.: A variable neighbourhood algorithm for solving the continuous location-allocation problem. Studies in Location Analysis **10**, 1–12 (1996)

11. Brook, A., Kendrick, D., Meeraus, A.: Gams, a user's guide. ACM SIGNUM Newsletter **23**(3-4), 10–11 (1988)
12. Bussieck, M.R., Drud, A.S., Meeraus, A.: MINLPLib — a collection of test models for Mixed-Integer Nonlinear Programming. INFORMS Journal on Computing **15**(1) (2003). URL `http://www.gamsworld.org/minlp/minlplib.htm`
13. Consulting, A., Development: SBB Release Notes (2002)
14. D'Ambrosio, C.: Application oriented Mixed Integer Nonlinear Programming. Ph.D. thesis, DEIS, Università di Bologna (2009)
15. D'Ambrosio, C., Frangioni, A., Liberti, L., Lodi, A.: Experiments with a Feasibility Pump approach for nonconvex MINLPs. In: P. Festa (ed.) Proceedings of the 9th Symposium on Experimental Algorithms (SEA 2010), *Lecture Notes in Computer Science*, vol. 6049. Springer, Berlin (2010)
16. D'Ambrosio, C., Frangioni, A., Liberti, L., Lodi, A.: A storm of Feasibility Pumps for nonconvex MINLP. Tech. Rep. OR-10-13, DEIS, Università di Bologna (2010)
17. Danna, E., Rothberg, E., Le Pape, C.: Exploring relaxation induced neighborhoods to improve MIP solutions. Mathematical Programming A **102**, 71–90 (2005)
18. Dražic, M., Kovačević-Vujčić, V., Čangalović, M., Mladenović, N.: Glob — a new VNS-based software for global optimization. In: Liberti and Maculan [40], pp. 135–154
19. Drazić, M., Lavor, C., Maculan, N., Mladenović, N.: A continuous VNS heuristic for finding the tridimensional structure of a molecule (2004)
20. Duran, M., Grossmann, I.: An outer-approximation algorithm for a class of mixed-integer nonlinear programs. Mathematical Programming **36**, 307–339 (1986)
21. Fischetti, M., Glover, F., Lodi, A.: The feasibility pump. Mathematical Programming A **104**(1), 91–104 (2005)
22. Fischetti, M., Lodi, A.: Local branching. Mathematical Programming **98**, 23–37 (2003)
23. Fletcher, R., Leyffer, S.: Solving Mixed Integer Nonlinear Programs by outer approximation. Mathematical Programming **66**, 327–349 (1994)
24. Fletcher, R., Leyffer, S.: Numerical experience with lower bounds for MIQP branch-and-bound. SIAM Journal of Optimization **8**(2), 604–616 (1998)
25. Fletcher, R., Leyffer, S.: User manual for filter. Tech. rep., University of Dundee, UK (1999)
26. Fletcher, R., Leyffer, S.: Nonlinear programming without a penalty function. Mathematical Programming **91**, 239–269 (2002)
27. Fourer, R., Gay, D.: The AMPL Book. Duxbury Press, Pacific Grove (2002)
28. Hansen, P., Mladenović, N.: Variable neighbourhood search: Principles and applications. European Journal of Operations Research **130**, 449–467 (2001)
29. Hansen, P., Mladenović, N., Brimberg, J., Moreno Pérez, J.: Variable neighbourhood search. In: M. Gendreau, J.Y. Potvin (eds.) Handbook of Metaheuristics, 2nd edition. Kluwer, Dordrecht (2010)
30. Hansen, P., Mladenović, N., Moreno Pérez, J.: Variable neighbourhood search: methods and applications. 4OR **6**, 319–360 (2008)
31. Hansen, P., Mladenović, N., Urošević, D.: Variable neighbourhood search and local branching. Computers and Operations Research **33**(10), 3034–3045 (2006)
32. Karmarkar, N.: A new polynomial time algorithm for linear programming. Combinatorica **4**(4), 373–395 (1984)
33. Lavor, C., Liberti, L., Maculan, N.: Computational experience with the molecular distance geometry problem. In: J. Pintér (ed.) Global Optimization: Scientific and Engineering Case Studies. Springer, Berlin (2006)
34. Leyffer, S.: User manual for MINLP_BB. Tech. rep., University of Dundee, UK (1999)
35. Liberti, L.: Writing global optimization software. In: Liberti and Maculan [40], pp. 211–262
36. Liberti, L., Cafieri, S., Savourey, D.: Reformulation optimization software engine. In: K. Fukuda, J. van der Hoeven, M. Joswig, N. Takayama (eds.) Mathematical Software, *LNCS*, vol. 6327, pp. 303–314. Springer, New York (2010)
37. Liberti, L., Dražic, M.: Variable neighbourhood search for the global optimization of constrained NLPs. In: Proceedings of GO Workshop, Almeria, Spain (2005)
38. Liberti, L., Lavor, C., Maculan, N.: Double VNS for the molecular distance geometry problem. In: Proc. of Mini Euro Conference on Variable Neighbourhood Search, Tenerife, Spain (2005)

39. Liberti, L., Lavor, C., Maculan, N., Marinelli, F.: Double variable neighbourhood search with smoothing for the molecular distance geometry problem. Journal of Global Optimization (accepted for publication)
40. Liberti, L., Maculan, N. (eds.): Global Optimization: from Theory to Implementation. Springer, Berlin (2006)
41. Liberti, L., Nannicini, G., Mladenović, N.: A good recipe for solving MINLPs. In: V. Maniezzo, T. Stützle, S. Voss (eds.) Matheuristics: Hybridizing metaheuristics and mathematical programming, *Annals of Information Systems*, vol. 10, pp. 231–245. Springer (2009)
42. Mladenović, N., Drazic, M., Kovacevic-Vujcic, V., Cangalovic, M.: General variable neighborhood search for the continuous optimization. European Journal of Operational Research **191**(3), 753 – 770 (2008)
43. Mladenović, N., Petrović, J., Kovačević-Vujčić, V., Čangalović, M.: Solving a spread-spectrum radar polyphase code design problem by tabu search and variable neighbourhood search. European Journal of Operations Research **151**, 389–399 (2003)
44. Nannicini, G.: Point-to-point shortest paths in dynamic time-dependent road networks. Ph.D. thesis, Ecole Polytechnique, Palaiseau, France (2009)
45. Nannicini, G., Belotti, P.: Rounding-based heuristics for nonconvex MINLPs. In: P. Bonami, L. Liberti, A. Miller, A. Sartenaer (eds.) Proceedings of the European Workshop on MINLP. CIRM, Marseille, France (2010)
46. Puchinger, J., Raidl, G.: Relaxation guided variable neighbourhood search. In: Proc. of Mini Euro Conference on Variable Neighbourhood Search, Tenerife, Spain (2005)
47. Sahinidis, N.: BARON: A general purpose global optimization software package. Journal of Global Optimization **8**(2), 201–205 (1996)
48. Smith, E., Pantelides, C.: A symbolic reformulation/spatial branch-and-bound algorithm for the global optimisation of nonconvex MINLPs. Computers & Chemical Engineering **23**, 457–478 (1999)
49. Tawarmalani, M., Sahinidis, N.: Global optimization of mixed integer nonlinear programs: A theoretical and computational study. Mathematical Programming **99**, 563–591 (2004)
50. Wächter, A., Biegler, L.T.: On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. Mathematical Programming **106**(1), 25–57 (2006)
51. Westerlund, T., Skrifvars, H., Harjunkoski, I., Pörn, R.: An extended cutting plane method for a class of non-convex MINLP problems. Computers & Chemical Engineering **22**(3), 357–365 (1998)