**RC25385 (WAT1305-077) May 28, 2013**
**Computer Science**

# IBM Research Report

## A Hybrid LP/NLP Paradigm for Global Optimization Relaxations

**Aida Khajavirad**
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598
USA

**Nikolaos V. Sahinidis**
Carnegie Mellon University
Pittsburgh, PA
USA

# A hybrid LP/NLP paradigm for global optimization relaxations

**Aida Khajavirad · Nikolaos V. Sahinidis**

Received: date / Accepted: date

**Abstract** After their introduction in global optimization by Tawarmalani and Sahinidis [43, 44], polyhedral relaxations have been incorporated in a variety of solvers for the global optimization of nonconvex nonlinear programs and mixed-integer nonlinear programs. Currently, these relaxations constitute the dominant approach in global optimization practice. In this paper, we introduce a new relaxation paradigm for global optimization. The proposed framework combines polyhedral and convex nonlinear relaxations, along with fail-safe techniques, convexity identification at each node of the search tree, and learning strategies for automatically selecting and switching between different relaxations and between different local search algorithms in different parts of the search tree. We report computational experiments with the proposed methodology on widely-used test problem collections from the literature, including 369 problems from GlobalLib, 250 problems from MINLPLib, and 980 problems from PrincetonLib. Results show that incorporating the proposed techniques in the BARON software leads to significant reductions in execution time, and increases by 30% the number of problems that are solvable to global optimality within 500 s on a standard workstation.

**Keywords** Global optimization · Polyhedral relaxations · Nonlinear relaxations · Automatic convexity detection · Branch-and-reduce

**PACS** 65K05 · 90C30 · 90C26

## 1 Introduction

Research in global optimization has witnessed a significant growth at the algorithmic and software levels over the last few years. Several general-purpose deterministic global

---

This research was supported in part by National Science Foundation award CMII-1030168.

A. Khajavirad
Business Analytics and Mathematical Sciences, IBM T. J. Watson Research Center, Yorktown Heights, NY, USA, e-mail: aidakhajavi@us.ibm.com.

N. V. Sahinidis
Department of Chemical Engineering, Carnegie Mellon University, Pittsburgh, PA, USA, e-mail: sahinidis@cmu.edu. Address all correspondence to this author.

solvers have been developed for nonconvex nonlinear programs (NLPs) and mixed-integer nonlinear programs (MINLPs), and are maturing rapidly. Beginning with the appearance of `BARON` in the mid 1990s [40], the following branch-and-bound based global solvers were introduced in the past decade: `GlobSol` [27], `LindoGlobal` [29], `Couenne` [10], and `SCIP` [11]. Performance of these algorithms is highly dependent on an effective use of state-of-the-art solvers for linear programming (LP), mixed-integer programming (MIP), and convex programming, at various stages in the global search (cf. [42] for an exposition).

Branch-and-bound algorithms rely on relaxations of nonconvex problems over successively refined partitions of the feasible region. That is, at each node in the search tree, in order to compute a bound on the optimal value of the original problem over the corresponding partition element, a relaxation of the nonconvex problem is constructed and optimized by a suitable optimization solver. Various relaxation construction techniques for nonconvex problems have been proposed in the literature. These methods can be broadly categorized as nonlinear convex relaxations [39,29], polyhedral relaxations [43,44,10], and piece-wise linear relaxation [9,8]. Computational results reported over the past two decades demonstrate that, in comparison to LP solvers, local NLP solvers are slower and more susceptible to failure due to numerical difficulties. Motivated by this observation, Tawarmalani and Sahinidis [43] proposed the first branch-and-bound global optimization algorithm based on polyhedral relaxations. Using a factorable reformulation of the problem as the starting point, each univariate convex function was outer-approximated by supporting hyperplanes at a number of carefully selected points. These authors found that, while polyhedral relaxations introduce larger relaxation gaps compared to the convex relaxations from which they are derived, the superior performance and robustness of LP solvers significantly improves the reliability of LP relaxation-based global solvers. In a subsequent paper [44], Tawarmalani and Sahinidis addressed the question of constructing polyhedral relaxations for multivariate convex functions, and presented a branch-and-cut framework for global optimization of nonconvex NLPs and MINLPs based on the solution of LP relaxations. Furthermore, these authors showed that their proposed factorable-based polyhedral relaxations automatically exploit the convexity of intermediate expressions whose convexity follows from a recursive application of a set of well-known convexity preserving operations on a number of primitive functions. Due to its remarkable computational success, their proposed branch-and-cut framework was later adopted by other global solvers, currently including `Couenne`, `GlobSol`, and `SCIP`.

In Section 2 of this paper, we revisit the question of constructing sharp polyhedral relaxations for nonconvex problems. We take the polyhedral relaxation framework of [44] as the starting point, and enhance and refine it in several directions. We strengthen the existing relaxations by generating supporting hyperplanes for intermediate convex expressions whose convexity is not implied by factorable composition rules. In addition, we improve the reliability of the relaxation constructor in `BARON` by devising a more reliable cut generation scheme that examines the quality of the solution returned by the LP solver prior to utilizing it in global search.

In Section 3, we consider the case in which the continuous relaxation of a possibly nonconvex problems at a given node in the search tree is a convex optimization problem. Examples include convex NLPs, convex MINLPs, and nonconvex problems that become convex as a result of branching or range reduction operations. Current general-purpose global solvers based on LP relaxations employ a branch-and-cut algorithm even when the original problem is convex. While such an approach is quite stable and efficient for small problems, it is often several orders of magnitude slower than gradient-based algorithms. We present a hybrid lower bounding scheme that combines polyhedral and nonlinear convex relaxations to make the most of state-of-the-art LP and local NLP solvers. The main components of our implementation are (i) an efficient convexity detection tool that is embedded at every node in the branch-and-bound tree, (ii) a dynamic local solver selection strategy that can switch among various local solvers in the search tree based on their performance, (iii) a verification routine that examines the optimality of the solution returned by a local solver, and (iv) a hybrid relaxation constructor that alternates between polyhedral and nonlinear relaxations at every node based on their relative quality and numerical stability.

In Section 4, we present computational experiments on a variety of NLPs and MINLPs taken from `GlobalLib` [24], `MINLPLib` [14] and `PrincetonLib` [37]. Results show that the enhanced polyhedral relaxations and hybrid linear/nonlinear lower bounding scheme significantly improve the performance of `BARON`. In addition, a systematic comparison with a number of other solvers indicates the superiority of `BARON 12.0` on a wide range of problem types.

## 2 Polyhedral branch-and-cut

In this section, we consider the problem of constructing polyhedral relaxations for general nonconvex factorable programs. We first present some preliminary material on factorable programming relaxations as well as the outer-approximation method proposed in [44]. Subsequently, we discuss strengths and weaknesses of the existing framework and propose several enhancements.

### 2.1 Factorable-based polyhedral relaxations

Factorable programming relaxations [31] have formed the basis of currently popular techniques in global optimization for bounding nonconvex functions. A variant of this technique introduced by Ryoo and Sahinidis [39] starts by iteratively decomposing a nonconvex factorable function, through introduction of auxiliary variables and constraints for intermediate functional expressions, until all intermediate expressions can be outer-approximated by a convex feasible set. Current implementations of factorable relaxations in general-purpose global solvers [41, 29, 10] employ this nested decomposition to the extent that all intermediates are affine functions, univariate convex (concave) functions, bilinear expressions, or other functions whose convex/concave envelopes are known. Subsequently, bilinear relations are replaced by their polyhedral convex and concave envelopes, and convex (resp. concave) univariates are overestimated (resp. underestimated) by their affine envelopes (see [43] for further details). The resulting optimization problem is nonlinear and convex, possibly with many more variables than the original nonconvex problem, and its solution provides a lower bound for the optimal value of the original minimization problem.

To capitalize on the availability of highly efficient LP solvers, in [43], the authors proposed the use of entirely polyhedral relaxations by further underestimating (resp. overestimating) each convex (resp. concave) univariate via supporting hyperplanes at a number of points determined by a sandwich algorithm. The resulting LP is solved at each node in the branch-and-bound tree to generate a lower bound. This methodology was further refined in [44], where at each node additional cutting planes are generated in rounds and added to the polyhedral relaxation only if they violate the relaxation solution. In general, the nested decomposition and relaxation may introduce a large relaxation gap. Interestingly, there exist several important cases for which factorable decomposition exploits the structure of the original problem. In [44], the authors showed that the proposed factorable-based polyhedral relaxation framework automatically exploits the convexity of original functions, under certain assumptions. In the following, we briefly describe this result in a slightly different form (see also Theorem 1 in [44]). First, we recall a convexity-preserving operation, which is a powerful tool for detecting convexity of a wide class of functions (cf. Section 3.2 in [13]).

**Lemma 1** *Let $f : \mathcal{D} \to \mathbb{R}^n$ be a vector of functions $f_j$ , $j \in J = \{1, ..., n\}$, where $\mathcal{D} \subset \mathbb{R}^m$ is a convex set. Let $\bar{J}$ contain the elements of $J$ for which $f_j$ is not affine. Assume that $f_j$ is convex for $j \in J_1 \subset \bar{J}$ and concave for $j \in J_2 = \bar{J} \setminus J_1$. Let $g : \mathcal{C} \to \mathbb{R}$ be convex, where $\mathcal{C}$ is a convex set in $\mathbb{R}^n$ that contains the range of all $f_j$ over $\mathcal{D}$. Assume that $g(y_1, \ldots, y_n)$ is nondecreasing in $y_j$ , $j \in J_1$ and is nonincreasing in $y_j$ , $j \in J_2$. Then, the composite function $h(x) = g(f(x))$ is convex on $\mathcal{D}$.*

Various well-known convexity-preserving operations follow from the above result. For instance, if $f_j$, $j \in J$ are affine functions, then it follows that convexity is preserved under the affine mapping of the domain. Similarly, if we restrict $g$ to be a univariate convex function, then Lemma 1 simplifies to the following composition rule: a convex nondecreasing (resp. nonincreasing) function of a convex (resp. concave) function is convex.

Now, consider a factorable decomposition of $h(x) = g(f(x))$ defined as $y_j = f_j(x)$ for all $j \in J$, and $z = g(y)$. For simplicity, let $f$ and $g$ be differentiable and let $f$ denote a vector of convex functions, i.e., $J = J_1$. Suppose that convexity of $f(x)$ and $g(y)$ are recognizable by the relaxation constructor, whereas convexity of $h(x)$ is not known a priori. Our goal is to generate a polyhedral underestimator for $z$ at $(x, y) = (x_0, y_0)$, where $y_0 = f(x_0)$. By convexity and monotonicity assumptions on $f$ and $g$, a factorable relaxation of $z$ is given by $y_j \geq f_j(x_0) + \nabla f_j(x_0)^T(x - x_0)$ for all $j \in J$ and $z \geq g(y_0) + \nabla g(y_0)^T(y - y_0)$. By assumption, $g(y)$ is component-wise nondecreasing. Thus, $z \geq g(y_0) + (\nabla f(x_0)^T \nabla g(y_0))^T(x - x_0)$, where $\nabla f(x_0)$ denotes the jacobian of $f(x)$ at $x = x_0$. By the chain rule of differentiation, $\nabla h(x_0) = \nabla f(x_0)^T \nabla g(y_0)$. Therefore, $z \geq h(x_0) + \nabla h(x_0)^T(x - x_0)$, i.e., recursive relaxation automatically exploits the convexity of $h(x)$, in the sense that its projection onto the space of original variables corresponds to the supporting hyperplane of $h(x)$ at $x = x_0$.

In the remainder of the paper, by *primitive functions*, we refer to the set of functions whose convexity properties are recognizable by a factorable relaxation scheme. Moreover, we will refer to those convexity-preserving operations that are implied by Lemma 1, as *composition rules*. Hence, the above discussion implies that recursive factorable relaxations automatically exploit convexity of functions whose convexity can be described by a recursive application of composition rules on a set of primitive functions.

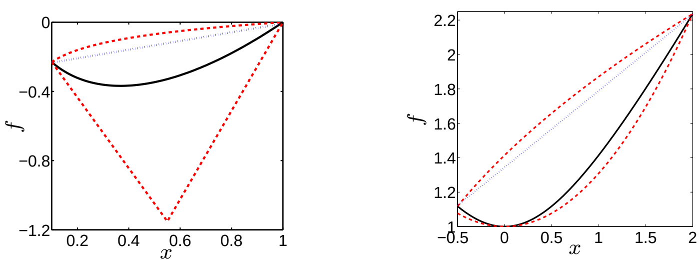
## 2.2 Exploiting convexity in constructing polyhedral relaxations

There exists a variety of functional classes whose convexity is not exploited by the conventional factorable relaxation scheme. For instance, in `BARON`'s factorable programming module, the list of primitive functions consist of affine expressions, monomials

($x^a$), powers ($a^x$), logarithmic functions, bilinears, and fractions. As a result, to generate a convex outer-approximation of the set $\mathcal{P} = \{(x, f) : f = x \log x, x \in [0.1, 1]\}$, a decomposed equivalent of $f$ is constructed as follows: $y = \log x$, $f = xy$. Subsequently, the nonconvex set defined by each equation is replaced by its convex hull to obtain a convex relaxation of $\mathcal{P}$, denoted by $\tilde{\mathcal{P}}$. It is then simple to show that the projection of $\tilde{\mathcal{P}}$ onto the original space $(x, f)$ is given by:

$$\mathcal{S} = \{(x, f) : \max\{-2.05x - 0.03, 2.56(x - 1.0)\} \leq f \leq \log x/10\}.$$

The convex set $\mathcal{S}$ is depicted in Figure 1(a), where it can be seen that the recursive relaxation introduces a large relaxation gap. As a second example, in Figure 1(b), a factorable relaxation of $\mathcal{P} = \{(x, f) : f = \sqrt{1 + x^2}, -0.5 \leq x \leq 2.0\}$ is shown. It is simple to verify that the function $f = \sqrt{1 + x^2}$ is convex. However, as a composite function, $f$ is a concave increasing function ($f = \sqrt{y}$) of a convex function ($y = 1 + x^2$), a structure that does not satisfy the assumptions of Lemma 1.




(a) $f = x \log x,\ 0.1 \leq x \leq 1.0$ (b) $f = \sqrt{1 + x^2},\ -0.5 \leq x \leq 2.0$

**Fig. 1** Conventional factorable relaxations for functions whose convexity does not follow from composition rules. The convex function $f$ is shown in solid black and its factorable outer-approximation is shown in dashed red. In both figures, the dotted blue line is the concave envelope of $f$.

Motivated by the above discussion, we present several important functional classes whose convexity or concavity are *not* exploited by the conventional factorable approach. Clearly, our list is by no means complete, and there exist many convex functions that we do not consider in our implementation. However, the following list is based upon an extensive survey of a large number of optimization problems that appear in widely-used test libraries as well as a variety of applications. We state the functional forms without proofs, as these proofs follow from elementary arguments.

1. **Products and ratios.** Consider the function

$$f(x) = \prod_{i \in I} f_i(x_i),\ I = \{1, \ldots, n\},\ n \geq 2,$$

where $f_i = x_i^{a_i}$, $a_i \in R \setminus \{0\}$ or $f_i = a_i^{x_i}$, $a_i > 0$ for all $i \in I$. Then, $f(x)$ is convex if one of the following conditions is satisfied:

(i) $f_i = a_i^{x_i}$, or $f_i = x_i^{a_i}$ with $x_i > 0$, $a_i < 0$ for all $i \in I$.

(ii) $f_i = x_i^{a_i}$ with $x_i > 0$, $a_i < 0$ for all $i \in I \setminus \{j\}$, and $f_j = x_j^{a_j}$ with $a_j > 1$, $f_j \geq 0$ and $\sum_{i \in I} a_i \geq 1$.

Moreover, $f(x) = \prod_{i \in I} x_i^{a_i}$ is concave if $a_i > 0$ for all $i \in I$ and $\sum_{i \in I} a_i \leq 1$.

2. **Perspective of functions.** The perspective operation preserves convexity, i.e., if $f(x)$ is convex, then its perspective $g(x, y) = y f(x/y)$, $y > 0$ is jointly convex in $x$ and $y$ (cf. Section 3.2.6 in [13]). Examples include perspective of negative entropy $g = x \log(x/y)$, and perspective of exponential $g = y \exp(x/y)$.

3. **Norms and norm-type functions.** Consider the function

$$f(x) = \left( a_0 + \sum_{i=1}^{n} a_i x_i^p \right)^q, \ x \in \mathbb{R}^n, \ p, q \in \mathbb{R}, \ n \geq 1,$$

where $a_i > 0$, $i = 0, 1, \ldots, n$. Then, $f(x)$ is convex if one of the following conditions is satisfied:
(i) $p > 1.0$, $q < 1.0$, and $pq \geq 1.0$; e.g. $f = \sqrt{x_1^2 + x_2^2}$
(ii) $p < 0.0$, and $0.0 < q < 1.0$; e.g. $f = \sqrt{1 + 1/x}$.
Moreover, $f(x)$ is concave if $p < 1.0$, $0.0 < pq < 1.0$, and $p(q-1) > 0$; e.g. $f = (1 + \sqrt{x})^2$.

4. **Quadratic functions and quadratic norms.** Consider the function $f(x) = x^T Q x$, $x \in \mathbb{R}^n$, $Q \in \mathcal{S}^n$, where $\mathcal{S}^n$ denotes the set of $n \times n$ symmetric matrices. Suppose that $Q$ is not diagonal. Then, $f(x)$ is convex (resp. concave) if and only if $Q$ is positive semidefinite (resp. negative semidefinite). Moreover, the $Q$-quadratic norm (resp. seminorm) defined as $||x||_Q = \sqrt{x^T Q x}$, where $Q$ is positive definite (resp. positive semidefinite), is a convex function.

5. **Log-sum-exp.** The function $f = \log(c_0 + c_1 a_1^{x_1} + \ldots + c_n a_n^{x_n})$, where $x \in \mathbb{R}^n$ and $c_i > 0$ for $i = 0, \ldots, n$ is convex.

6. **Negative entropy.** $f = x \log x$, $x > 0$ is a convex function.

*Remark 1* It is possible to construct many more types of functions by applying composition rules to the functions listed above. However, there is no need to characterize all such functions. A factorable relaxation automatically exploits such structures, provided that the above functions are included in the list of primitive functions of the factorable scheme. For example, consider $h(x) = (\prod_{i=1}^n f_i(x))^{1/n}$, $x \in \mathbb{R}^m$. Suppose that $f_i$, $i = 1, \ldots, n$ are concave and nonnegative. The geometric mean $g(y) = (\prod_{i=1}^n y_i)^{1/n}$, $y_i \geq 0$ belongs to Class 1 above, and thus is concave. In addition, $g(y)$ is nondecreasing in each argument. Hence, by Lemma 1, $h(x)$ is concave, and its concavity is automatically exploited by the recursive relaxation.

*Remark 2* Consider the set $\mathcal{S} = \{(x, f) : f = x_1 \sqrt{1 + (\log x_2)^2}, x \in [\underline{x}, \bar{x}]\}$, and suppose that the goal is replace $\mathcal{S}$ by a convex set. Clearly, as a function of $x$, $f$ does not belong to any of the functional types listed above. A factorable reformulation of this set is given by $\mathcal{S}' = \{(x, y, f) : y_1 = \log x_2, y_2 = y_1^2, y_3 = 1 + y_2, y_4 = \sqrt{y_3}, f = x_1 y_4, x \in [\underline{x}, \bar{x}]\}$. In the lifted space, we can deduce the relation $y_4 = \sqrt{1 + y_1^2}$, which, by Part (i) of Class 2, corresponds to the graph of a univariate convex function, and thus can be outer-approximated but its convex hull. It is important to note that, since factorable relaxations are already built in the lifted space, generating strong cuts for such intermediate relations can significantly enhance the convergence rate of a branch-and-bound algorithm.

*Remark 3* To identify convex/concave quadratics, we decompose a quadratic function into nonseparable components and compute the eigenvalues of each quadratic form to detect its convexity. As we detail in the following, we store both eigenvalues and eigenvectors of quadratics to generate additional classes of cutting planes. Furthermore, for indefinite quadratics with non-polyhedral envelopes, we generate cutting planes based on the separable programming approach of [38]. Quadratic forms with polyhedral envelopes are ignored here as `BARON` is equipped with a powerful relaxation constructor for such functions [7].

*Remark 4* Consider the second-order cone constraint given by

$$\sum_{i=1}^{n} a_i x_i^2 - a_0 x_0^2 \leq 0,\ x_i \in \mathbb{R},\ a_i > 0,\ \forall i \in \{0, 1, \ldots, n\}. \tag{1}$$

Clearly, the quadratic function $f = \sum_{i=1}^{n} a_i x_i^2 - a_0 x_0^2$ is not convex. However, inequality (1) defines a convex feasible region; i.e., the zero-sublevel set of $f$, denoted by $f_0$, is a convex set. It is simple to verify that factorable relaxations of $f$ do not exploit the convexity of $f_0$. Therefore, we reformulate constraints of the form (1) as

$$\Big(\sum_{i=1}^{n} a_i x_i^2\Big)^{0.5} - \sqrt{a_0} x_0 \leq 0. \tag{2}$$

Since the $l_2$-norm belongs to Class 3 functions above, the proposed factorable relaxation in the left-hand side of inequality (2) does not introduce any relaxation gap.

Now consider the function $f = x_1^{0.3} x_2^{0.4} x_3^{0.5}$, $x \geq 0$. This function is not concave. However, if at least one of the variables get fixed, then the resulting function is concave. In addition, if two of the variables become fixed, then concavity of the resulting univariate function is exploited by the conventional factorable scheme. As another example, consider $f = x_1^3 / \log x_2$, $x_1 \in \mathbb{R}$, $x_2 > 1$. This function is neither convex nor concave. However, if at some node in the search tree, we have $x_1 \geq 0$ (resp. $x_1 \leq 0$), then $f$ becomes convex (resp. concave). For functions of this type, we do not modify the conventional factorable reformulator. Instead, we apply the following two-step approach to generate tighter polyhedral relaxations at each node in the branch-and bound tree:

- **Recognition:** prior to the initialization of the branch-and-bound tree, we mark and classify all intermediate relations containing subexpressions whose convexity/concavity cannot be exploited by the factorable approach. The data structures required for generating and storing cutting planes are also allocated at this stage.
- **Cut generation:** at each node in the branch-and-bound tree, we first construct and solve a crude outer-approximation of the problem based on the conventional factorable reformulation. Subsequently, various classes of cutting planes are generated and added to the current relaxation iteratively, only if they violate the relaxation solution (see [44] for details). We do not add the proposed cutting planes to the initial outer-approximation but utilize them for the iterative cut generation scheme. At a given cut generation iteration, we scan all expressions $y_j = f_j(x)$, $j \in J$ stored at the recognition step. Denote by $(x^*, y_j^*)$ the projection of the current relaxation solution to the $(x, y_j)$ space. If any of the variables are fixed in the current node, then we eliminate them and update $f_j(x)$ accordingly. If $f_j(x)$

is a convex function (resp. concave function) that is not recognized as such by the factorable scheme and $y_j^* < f_j(x^*)$ (resp. $y_j^* > f_j(x^*)$), then a cut of the form $\nabla f_j(x^*)^T x - y_j \leq \nabla f_j(x^*)^T x^* - f_j(x^*)$ (resp. $\geq$ ) is added to the current relaxation.

In addition to generating sharp cutting planes, detecting convexity of intermediate expressions may result in inferring tighter bounds for both original and auxiliary variables of a nonconvex problem. Given lower and upper bounds on original variables, factorable programming-based global solvers employ a recursive bound propagation scheme based on interval arithmetic and monotonicity analysis to infer bounds on all intermediate variables. Subsequently, in a backward mode, bounds on the objective and constraint functions are used to infer tighter bounds on original and intermediate variables. Clearly, if an intermediate relation is convex (resp. concave), then we can compute its minimum (resp. maximum) to obtain a sharp lower bound (resp. upper bound) for the associated auxiliary variable. Accordingly, we tighten the bounds on all convex/concave intermediates, whenever such bounds are not implied by the existing bound propagation scheme. For instance, consider $f = x \log x$, $x \in [0.1, 2]$. By convexity of $f$, it follows that $f \in [-0.368, 1.386]$. However, in the recursive approach, we have $f = xy$, where $y = \log x$. Utilizing interval bounds for the bilinear term $f = xy$, $0.1 \leq x \leq 2$, $-2.3 \leq y \leq 0.69$, we obtain $f \in [-4.605, 1.386]$. Similarly, we exploit convexity (or concavity) of univariate quadratics of the form $ax^2 + bx$ to enhance linear-feasibility-based range reduction operator (cf. [17] for details).

As another example, consider a nonseparable convex quadratic function $f = x^T Qx + b^T x$, $x \in [\underline{x}, \bar{x}] \subset \mathbb{R}^n$, $Q \in \mathbb{R}^{n\times n}$, $b \in \mathbb{R}^n$, $n \geq 2$. In this case, we compute the minimum of $f$ over $\mathbb{R}^n$ by solving a system of linear equations and use it to bound $f$ in the relaxation. If $f$ is unbounded below over $\mathbb{R}^n$, i.e., if $b$ is not the in the column space of $Q$, then we ignore the linear part of $f$ and add the inequality $x^T Qx \geq 0$ to the relaxation. Now, suppose that a finite upper bound on $f$ is available, i.e., $x^T Qx + b^T x \leq \alpha$. The feasible region defined by this inequality is an ellipsoid (or a degenerate ellipsoid). In this case, we characterize the hyperplanes corresponding to the minimum-volume bounding box of this ellipsoid, and add them to the relaxation. In the following, we present the derivation of these inequalities for completeness. First, suppose that $f$ does not contain any linear term. Denote by $\lambda_i$ and $v_i$ the $i$th eigenvalue and eigenvector of $Q$, respectively. By convexity of $f$, we have $\lambda_i \geq 0$ for all $i = 1, \ldots, n$. Without loss of generality, suppose that the first $m$ eigenvalues of $Q$ are positive. Clearly, if $f$ is strictly convex, then $m = n$. It follows that $f = \sum_{i=1}^m \lambda_i (v_i^T x)^2$. By assumption, $\alpha$ is a finite upper bound on $f$. Thus, we obtain $2m$ inequalities given by $-\sqrt{\alpha/\lambda_i} \leq v_i^T x \leq \sqrt{\alpha/\lambda_i}$, for $i = 1, \ldots, m$. Now, suppose that $f$ contains linear terms. As in the previous case, we would like to reformulate $f$ as a sum of squares, i.e., we are interested in finding the vector $c \in \mathbb{R}^m$ such that $x^T Qx + b^T x = \sum_{i=1}^m \lambda_i (v_i^T x + c_i)^2 - \sum_{i=1}^m \lambda_i c_i^2$. After rearranging terms and using the relation $x^T Qx = \sum_{i=1}^m \lambda_i (v_i^T x)^2$, we obtain

$$\sum_{i=1}^m (\lambda_i v_{ij}) c_i = b_j/2, \ \forall j = 1, \ldots, n, \tag{3}$$

where $v_{ij}$ denotes the $j$th element of $v_i$. Two cases arise:

- The linear system (3) has a solution (clearly, if $Q$ is positive definite, i.e., $m = n$, then the system always has a solution). In this case, the minimum-volume bounding box of the feasible region is given by the following inequalities:

$$-\sqrt{\frac{\alpha + \sum_{j=1}^{m} \lambda_j c_j^2}{\lambda_i}} - c_i \leq v_i^T x \leq \sqrt{\frac{\alpha + \sum_{j=1}^{m} \lambda_j c_j^2}{\lambda_i}} - c_i, \quad i = 1, \ldots, m.$$

- The linear system (3) does not have any solutions (if $Q$ is positive semi-definite, then (3) has more equations than unknowns and may not have any solutions, i.e., the feasible region is an elliptic paraboloid). In this case, if a finite upper bound $\beta$ on the quadratic form $x^T Qx$ is available, then we ignore the linear part of $f$ and generate the bounding box of the region defined by $x^T Qx \leq \beta$.

### 2.3 Exploiting quasi-convexity in constructing polyhedral relaxations

Consider the inequality constraint $g$ defined as $x_1^2 x_2 \geq 1$. The function in the left-hand side, i.e., $f = x_1^2 x_2$ is neither convex nor concave. However, $g$ defines a convex feasible region since it is a superlevel set of the quasi-concave function $f$. Recall that a function is quasiconcave if and only if all of it superlevel sets $\mathcal{S}_\alpha = \{x : f(x) \geq \alpha\}$, for $\alpha \in \mathbb{R}$ are convex (cf. [5] for an exposition). To generate supporting hyperplanes of the convex region defined by $g$, we first reformulate it as $x_2 \geq 1/x_1^2$, and subsequently linearize it using first-order Taylor series approximation of the convex function $1/x_1^2$. In general, it is always possible to represent sublevel sets of quasiconvex functions (or superlevel sets of quasiconcave functions) via inequalities of convex functions (cf. Section 3.4 in [13]). To demonstrate the benefit of such reformulations for constructing tighter relaxations, in Figure 2, we show the factorable outer-approximation of $g$ projected onto the original space. This relaxation is obtained by first reformulating $g$ as $y_1 = x_1^2$, $y_2 = y_1 x_2$, $y_2 \geq 1$, and replacing the two equality constraints by their convex hulls. To construct this relaxation, finite lower and upper bounds on variables are required. We assume $x_1 \in [0.1, 1]$, $x_2 \in [1, 100]$.

Alternatively, a relaxation can be constructed by combining the two techniques, i.e., by reformulating $y_1 x_2 \geq 1$ as $x_2 \geq 1/y_1$, and employing the factorable relaxation of $y_1 = x_2^2$ as before. As can be seen in Figure 2, while stronger than the pure factorable approach, the hybrid scheme is not tight. For this example, it is simple to show that, if the exponent of $x_1$ is smaller than one, then the hybrid approach does not introduce any relaxation gap. We formalize this idea as follows. First, we recall the conditions under which a signomial term is quasiconvex or quasiconcave (see Chapter 5 of [5] for proofs). Clearly, convexity is a sufficient condition for quasi-convexity. Since convex/concave signomials are covered by the techniques discussed in the previous section, in the following, we only consider cases in which $f$ is merely quasiconvex (or merely quasi-concave).

**Lemma 2** *Consider the function $f = \prod_{i \in I} x_i^{a_i}$, $a_i \in \mathbb{R} \setminus \{0\}$ for all $i \in I = \{1, \ldots, n\}$. Define the subsets $I_p = \{i \in I : a_i > 0\}$, and $I_n = I \setminus I_p$. The function $f$ is defined over the domain $\mathcal{C} = \{x \in \mathbb{R}^n : x_i \geq 0 \ \forall i \in I_p,\ x_i > 0 \ \forall i \in I_n\}$. Then, $f$ is merely quasiconcave if one of the following conditions is satisfied:*

*(i) $a_i > 0$ for all $i \in I$, and $\sum_{i \in I} a_i > 1$,*
*(ii) $a_i > 0$ for all $i \in I \setminus \{j\}$, and $\sum_{i \in I} a_i \leq 0$.*

*Moreover, $f$ is merely quasiconvex if $a_i < 0$ for all $i \in I \setminus \{j\}$, and $0 \leq \sum_{i \in I} a_i < 1$.*

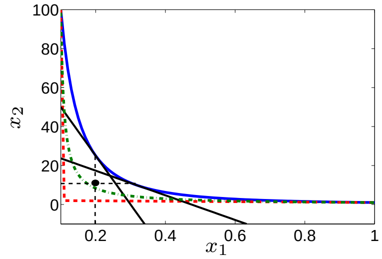



**Fig. 2** Alternative relaxation methods for the set $\mathcal{S} = \{(x_1, x_2) : x_1^2 x_2 \geq 1, x_1 \in [0.1, 1], x_2 \in [1, 100]\}$. The boundary of set $\mathcal{S}$ is shown in solid blue, its factorable outer-approximation is shown in dashed red, and a hybrid relaxation is shown in dotted green. The two tangent lines show the cutting planes associated with two convex reformulations of $x_1^2 x_2 \geq 1$, i.e., (i)$1/x_1^2 - x_2 \leq 0$ and, (ii) $1/\sqrt{x_2} - x_1 \leq 0$, which separate the point $(x_1, x_2) = (0.2, 1.0)$ from the enlarged feasible regions.

We now utilize the above result to construct tighter polyhedral relaxations for nonconvex problems that contain quasi-convex expressions. In the following, we assume that $f$ is defined over a box $[\underline{x}, \bar{x}] \subset \mathcal{C}$, where the set $\mathcal{C}$ is defined in the statement of Lemma 2. In addition, by $\underline{f}$ and $\bar{f}$, we imply interval lower and upper bounds on $f(x)$, respectively, i.e., $\underline{f} = \prod_{i \in I_n} \bar{x}_i^{a_i} \prod_{i \in I_p} \underline{x}_i^{a_i}$ and $\bar{f} = \prod_{i \in I_n} \underline{x}_i^{a_i} \prod_{i \in I_p} \bar{x}_i^{a_i}$.

**Lemma 3** *Consider the function $f(x)$, $x \in [\underline{x}, \bar{x}]$ defined in the statement of Lemma 2. Suppose that $y = f(x)$ is an intermediate relation introduced by the factorable reformulation of a nonconvex problem. Denote by $\underline{f}$ and $\bar{f}$ the interval bounds on $f(x)$, and let $\alpha$ and $\beta$ denote lower and upper bounds on $y$, respectively. Suppose that at least one of the following inequalities holds: (i) $\underline{f} < \alpha$, (ii) $\beta < \bar{f}$. Denote by $x^*$ the LP solution for a relaxation of the nonconvex problem. We have the following cases:*

*(i) if $f(x)$ is quasiconcave with $a_i > 0$ for all $i \in I$ and $f(x^*) < \alpha$, then an inequality of the form $l(x) - x_j \leq 0$ for some $j \in I$ cuts off the relaxation solution, where $l(x)$ is the supporting hyperplane of the convex function $z = \left(\prod_{i \in I \setminus \{j\}} x_i^{a_i} / \alpha\right)^{-1/a_j}$ at $x_i = x_i^*$ for all $i \in I \setminus \{j\}$.*

*(ii) if $f(x)$ is quasiconcave (resp. quasiconvex) with $a_j < 0$ (resp. $a_j > 0$) for some $j \in I$ and $f(x^*) < \alpha$ (resp. $f(x^*) > \beta$), then the inequality $x_j - l(x) \leq 0$ cuts off the relaxation solution, where $l(x)$ is the supporting hyperplane of the concave function $z = \left(\prod_{i \in I \setminus \{j\}} x_i^{a_i} / \gamma\right)^{-1/a_j}$ at $x_i = x_i^*$ for all $i \in I \setminus \{j\}$, and $\gamma = \alpha$ (resp. $\gamma = \beta$).*

*Proof* We state the proof for Part(i). The proof of Part(ii) follows from a similar line of arguments. Note that if $\alpha \leq \underline{f}$ and $\bar{f} \leq \beta$, then the relation $\alpha \leq f(x) \leq \beta$ is valid for all $x \in [\underline{x}, \bar{x}]$ and no useful cut can be generated. Now, suppose than $f(x^*) < \alpha$. Clearly, $\alpha > 0$. Hence $x_i > 0$ for all $i \in I$ at any point satisfying the inequality $f(x) \geq \alpha$. Consider $\prod_{i \in I} x_i^{a_i} \geq \alpha$. Since $a_i > 0$ and $x_i > 0$ for all $i \in I$, this inequality can be equivalently written as $(\prod_{i \in I \setminus \{j\}} x_i^{a_i} / \alpha)^{-1/a_j} \leq x_j$ for any $j \in I$. It is simple to show that the signomial term on the left-hand side of this inequality is convex (cf. Part (ii) of Class 1 functions in Section 2.2), and thus replacing it by a supporting hyperplane at $x = x^*$ results in a valid inequality that enforces $f(x^*) \geq \alpha$.

If a function $f(x)$, $x \in \mathbb{R}^n$ satisfies the conditions of Part(i) of Lemma 3, then $n$ distinct cutting planes can be generated, all of which violate the relaxation solution.

In Figure 2, we show the two cutting planes associated with the inequality $x_1x_2^2 \geq 1$ at $(x_1, x_2) = (0.2, 1.0)$. In our implementation, we generate one cutting plane by using a heuristic to select a reformulation that produces a safe cut. We will detail the concept of safe cuts in the next section. From Lemma 2, it follows that $f = (x_1/x_2)^a$, $a \in \mathbb{R}$ is both quasiconvex and quasiconcave. In this case, (i) if $f(x^*) < \alpha$, then we consider the inequality $(x_1/x_2)^a \geq \alpha$ and, (ii) if $f(x^*) > \beta$, then the inequality $(x_1/x_2)^a \leq \beta$ will be used as the starting point for cut generation. In principle, this reformulation technique can be used to generate cutting planes for any inequality of the form $f(x) \leq \alpha$, where $f$ is a quasi-convex function. However, at this stage, our implementation is restricted to the case where $f$ is either a signomial term or a function obtained by application of composition rules to signomials.

## 2.4 Other enhancements: safe lower bounds and reformulations

At each node in the branch-and-bound tree, a global optimization solver constructs and solves polyhedral relaxations to obtain lower bounds on the optimal value of the nonconvex problem. For many problems, these LP relaxations are ill-conditioned and contain constraints with very large and/or very small coefficients. This issue often arises for nonconvex problems that contain unbounded variables or nonlinear expressions that are not properly bounded. While the state-of-the-art LP solvers are quite reliable, there exist cases for which these solvers make false optimality or infeasibility claims (cf. [36]). For instance, we have observed that LP solvers occasionally return a solution that is slightly infeasible with respect to a number of cutting planes. As a result, the relaxation constructor generates identical (or very similar) cutting planes and adds them to the current relaxation. Subsequently, the solution returned by the LP solver remains unchanged, and the same set of cutting planes may be appended to the relaxation and sent to the LP solver for multiple rounds. As another example, for badly scaled LPs that are on the border of infeasibility, LP solvers tend to make false infeasibility claims. By simply accepting such false infeasibility claims and pruning the corresponding nodes, the global solver may fail to locate a global solution. To address these issues, we add cutting planes to the relaxation only if they are properly scaled. Additionally, we accept LP solver solutions only after verifying their optimality, and examine infeasibility certificates prior to pruning any node:

- **Safe cuts**: consider a cutting plane of the form $\sum_{i \in I} a_i x_i \leq b$, where $a_i \in \mathbb{R} \setminus \{0\}$, $x_i \in \mathbb{R}$ for all $i \in I = \{1, \ldots, n\}$, $b \in \mathbb{R}$. To avoid a poorly scaled model, we add this cut to the LP relaxation if the following conditions are satisfied: (i) $l \leq |a_i| \leq u$ for all $i \in I$, (ii) $l \leq |a_i/a_j| \leq u$ for all $i, j \in I$ such that $i < j$, and (iii) $|b| \leq \bar{b}$. Here, the implementation-specific constant $l > 0$ is acceptably small, and $0 < u < \bar{(}b)$ are acceptably large. To maintain the tightness of polyhedral relaxations, prior to the initial outer-approximation of each convex/concave univariate $f(x)$, $x \in \mathcal{C} \subseteq \mathbb{R}$, the algorithm locates an interval $\mathcal{L} \subseteq \mathcal{C}$, over which the resulting supporting hyperplanes satisfy the above conditions. Finally, prior to appending cutting planes to an existing relaxation, we scale the cuts so that our measure of violation agrees with the feasibility tolerance of the LP solver.
- **Safe LP bounds**: we verify the optimality of the primal-dual pair reported by the LP solver prior to utilizing them in the global search. In particular, we check for (i) primal feasibility, (ii) dual feasibility, and (iii) zero duality gap. We have observed

many cases for which the solution returned by the LP solver is slightly infeasible. This issue often arises if the original nonconvex problem contains (many) nonlinear equality constraints or if the polyhedral relaxation is poorly scaled. For such instances, if the LP solution is dual feasible, then by weak duality, the dual objective value serves as a valid lower bound for the problem. Linear programming solvers such as `CPLEX` and `CLP` provide Farkas certificates for infeasible models. Namely, if an infeasible LP has an unbounded dual, then finding a recession direction of the dual problem is sufficient to prove the infeasibility of the primal. Hence, to identify false infeasibility claims, we demand and examine the associated certificates.

For a nonlinear expression, there exist many equivalent reformulations each of which has different implications for convexification and range reduction purposes. As a preprocessing step, we employ the following basic reformulations, which are beneficial for constructing factorable relaxations as well as automatic convexity detection:

- Consider $y = (x^{a_1})^{a_2}$. We reformulate this expression as $y = x^{a_1 a_2}$ unless the following conditions are satisfied: (i) $a_1$ is an even number, (ii) $a_1 a_2$ is not an even number and (iii) $x \in [\underline{x}, \bar{x}]$ such that $\underline{x} < 0$. If these conditions hold, then we mark $y$ as a non-differentiable monomial expression and generate its convex and concave envelopes, as needed. More generally, we employ a similar reformulation for the expression $y = ((x^{a_1})^{\cdots})^{a_n}$.
- We reformulate $y = \log a^x$ as $y = x^{\log a}$.
- We reformulate $y = (a^x)^b$ as $y = (a^b)^x$.
- We reformulate $y = a^{\log x}$ as $y = x^{\log a}$.

## 3 Hybrid LP/NLP relaxation paradigm

Consider a convex optimization in standard form:

$$
\text{(CV)} \qquad \begin{aligned} \min_x \ & f_0(x) \\ \text{s.t.} \ & f_i(x) \leq 0, \quad i = 1, \ldots, m \\ & Ax - b = 0, \end{aligned}
$$

where $x \in \mathbb{R}^n$, $A \in \mathbb{R}^{p \times n}$, $b \in \mathbb{R}^p$, and $f_0, \ldots, f_m$ are convex functions. Various cutting plane-based algorithms have been proposed for solving convex NLPs by solving a sequence linear programs (cf. [28, 12]). These methods are used mostly for solving nonsmooth convex problems or large-scale structured convex problems such as nonlinear network flows. For general smooth convex problems, however, methods based on active set or interior point algorithms are often significantly faster, even though they are more prone to numerical difficulties. In fact, for some convex problems, state-of-the-art local solvers report sub-optimal solutions or make false infeasibility claims. It is well-known that, for differentiable convex problems, every primal-dual pair that satisfies KKT conditions is optimal. Therefore, a practical approach to solve convex problems could first utilize local NLP solvers and, if these solvers fail due to numerical issues, then the algorithm could utilize the more stable but slower polyhedral-based techniques.

In the context of global optimization, convex subproblems often appear as a result of relaxing or restricting the feasible region of the original nonconvex problem at various stages during the global search. For example, convex subproblems are obtained by relaxing integrality requirements in convex MINLPs or restricting the domain of a subset of variables in nonconvex NLPs, as a result of applying range reduction or branching operations. Clearly, an extreme yet important case is when the original NLP is convex. Global solvers such as `SCIP`, `LindoGlobal` and `GloMIQO` [32] detect convexity of quadratic programs (QPs) and quadratically constrained quadratic programs (QCQPs) prior to the initialization of the branch-and-bound tree, and solve convex problems with local solvers. However, for more general problem classes, current general-purpose global solvers are not equipped with automatic convexity detection facilities, and utilize the standard branch-and-bound algorithm to solve convex problems as well.

Motivated by the above discussion, in this section, we introduce a hybrid lower-bounding paradigm that aims to harness the advantages of both polyhedral and nonlinear convex relaxations by combining them in a dynamic fashion. The main components of our implementation are (i) an efficient convexity detection tool that checks if the continuous relaxation of the problem is convex at every node in the branch-and-bound tree, (ii) a dynamic local solver selection strategy that switches among various local solvers in the branch-and-abound tree based on their performance, (iii) an examiner that verifies if the solution returned by the local solver is optimal, and (iv) a hybrid relaxation constructor that alternates between polyhedral and nonlinear relaxations at every node based on their relative strength and numerical robustness. We detail each of these components below.

## 3.1 Automatic convexity detection

To expedite the global search, we embed an efficient convexity detector at every node in the branch-and-bound tree. As we describe in the following, our convexity verification approach is based on a number of basic principles in convex analysis, which have also been implemented in other computational environments [25, 22]. The main distinction between our method and earlier convexity assessment techniques stems from the different contexts in which the results of such analysis are utilized. Our starting point is a highly efficient polyhedral-based branch-and-cut framework, and our goal is to improve the global solver by exploiting convexity of subproblems at every node in the branch-and-bound tree. Clearly, such an approach is only beneficial if convexity verification can be conducted very fast; for instance, by performing all expensive computations at the root node. In the following, we briefly review existing automated convexity assessment techniques, and subsequently describe our implementation.

### *3.1.1 Background*

Due to its significant theoretical and algorithmic implications, assessing convexity properties of general optimization problems in a fully automated manner is of great interest. The problem of determining whether or not a general optimization problem is convex is NP-hard (cf. [4]). Nevertheless, several incomplete techniques have been proposed in the literature that can effectively prove or disprove convexity of many practical optimization problems. These approaches can be categorized in two main groups:

I. **Symbolic methods.** These techniques are based on the following observation. Given (i) a set of primitive functions whose convexity and monotonicity properties are known a priori, and (ii) a set of convexity-preserving operations, we can construct many more convex functions via a recursive application of these operations on the primitive set. Similarly, given a general nonlinear function, we can utilize this technique to possibly verify its convexity. This approach was introduced by Grant et al. [30] for constructing convex optimization problems called disciplined convex programs (DCP). The authors also developed a modelling framework called `CVX` to support the DCP methodology [25]. The `CVX` system verifies if the input problem is a valid DCP, converts it to a solvable form and sends it to a proper solver. This symbolic technique was later adopted by Fourer et al. [21] to verify the convexity of a given optimization problem. The resulting implementation is available as part of DrAmpl [22], a meta solver for analyzing structural properties of optimization problems including convexity. The main advantage of the symbolic approach is its low computational cost. However, it may fail to prove or disprove convexity for many cases, an outcome we will henceforth call *inconclusive*.

II. **Numerical methods.** Several techniques have been proposed to prove or disprove convexity of a given problem by verifying various properties of convex functions. Given a multivariate function, MProbe [15] checks the basic definition of convexity over randomly generated line segments in the feasible region. Clearly, this approach can only be used for disproving convexity. In DrAmpl [22], if the symbolic approach described above returns an inconclusive flag, then the algorithm proceeds to the numerical disproving phase in which it tries to find a direction of negative curvature by solving an auxiliary quadratic program. Nenov et al. [35] assess convexity by checking positive semi-definiteness of interval Hessians, an approach which in spite of generality is often quite expensive. In [33], the author proposes an efficient method to compute the bounds on eigenvalues of interval Hessians.

Our convexity detection technique is based on the symbolic approach with the exception of quadratic functions for which we check if the Hessian is positive semidefinite. In other words, we include quadratic functions in the list of primitive functions, and utilize a numerical approach to verify their convexity. As in the previous section, we decompose quadratics into nonseparable components, and assess convexity properties of each quadratic subexpression, independently. We will further detail our convexity detection algorithm for quadratics in the next section. In addition, our convexity-preserving ruleset consists of any operation implied by Lemma 1. For a given ruleset, generality of the symbolic detection approach depends on the list of primitive functions. As we described in Section 2, to generate sharp factorable-based polyhedral relaxations, we have enhanced the list of `BARON`'s primitive functions by including a variety of convex/concave functions that appear in applications. We utilize the same list of functions for our convexity detector. In addition to affine functions and basic univariate/bivariate functions of the form $f(x) = x^a$, $f(x) = a^x$, $f(x) = \log x$, $f(x) = x_1 x_2$, $f(x) = x_1/x_2$, we have support for the six classes of functions listed in Subsection 2.2.

To assess convexity of a general nonlinear function, we start from its factorable reformulation and associate the set of intermediate relations with a recursive application of composition rules on a number of primitive functions. In addition, to determine convexity and monotonicity of primitive functions, we make use of `BARON`'s bound propagation facilities. We illustrate this technique by a simple example and refer the reader to [25,6,21] for a detailed description of the symbolic convexity verification scheme. Consider the inequality constraint $(x_1^2 + x_2^2 + 1)^2 \leq 4$, $x \in \mathbb{R}^2$. A factorable reformulation of this constraint is given by $y_1 = x_1^2$, $y_2 = x_2^2$, $y_3 = y_1 + y_2 + 1$, $y_4 = y_3^2$,

$y_4 \leq 4$. Using interval arithmetic, `BARON` infers the following bounds: $x \in [-1,1]^2$, $(y_1, y_2) \in [0,1]^2$, $y_3 \in [1,2]$, $y_4 \in [1,4]$. To assess convexity properties of this constraint, we first identify convexity and monotonicity of all intermediate relations in the *lifted space*: (i) $y_1 = x_1^2$ and $y_2 = x_2^2$ are convex; since $y_1$ and $y_2$ are functions of $x$ variables, no monotonicity analysis is required for these functions, (ii) $y_3 = y_1 + y_2 + 1$ is affine and nondecreasing in each argument, (iii) $y_4 = y_3^2$ is convex and increasing over $y_3 \in [1,2]$. Next, we employ convexity-preserving operations to deduce convexity in the *original space*: (i) $y_3 = y_1 + y_2 + 1$ is affine and nondecreasing in both arguments, and $y_1$ and $y_2$ are convex functions of $x$; thus, by Lemma 1, $y_3$ is a convex function of $x$, (ii) $y_4$ is a convex increasing function of $y_3$, and $y_3$ is a convex function of $x$; again, by Lemma 1, $y_4$ is a convex function of $x$. It is important to note that `BARON`'s range reduction strategies are highly important for our convexity detection scheme as they utilize the feasible region of the original problem to infer tighter bounds for variables. For example, consider the function $f = (x_1 - x_2)^3$, $x \in [0,1]^2$ with a factorable reformulation given by $y = x_1 - x_2$, $f = y^3$, $y \in [-1,1]$. Clearly, $f(x)$ is nonconvex over the unit hypercube. Now, assume that we have an additional constraint of the form $x_1 \geq x_2$ in the model. Using this inequality, `BARON` infers $y \in [0,1]$. Then, it follows that $f$ is a convex increasing function of an affine function, and is therefore convex.

#### *3.1.2 Convexity detection for global optimization*

We embed our convexity detection technique in `BARON`'s preprocessor as well as every node in the branch-and-reduce algorithm. Our ultimate goal is to reduce the overall execution time of the global solver for a wide range of optimization problems. Hence, a highly efficient convexity verification algorithm is of crucial importance. Otherwise, the detection algorithm may deteriorate the performance of the global solver, specially for problems with a large number of nonconvex sub-problems in the branch-and-bound tree. The main steps of our convexity detection algorithm are outlined next.

I. **Reformulation.** Prior to the application of convexity detector, we utilize a number of simple transformations that are beneficial for the purpose of convexity assessment:
   (i) $\log(a^x)$ is replaced by $x \log a$,
   (ii) $a^{\log x}$ is replaced by $x^{\log a}$,
   (iii) $(x^a)^b$ is replaced by $x^{ab}$ with the exception of non-differentiable functions (see Subsection 2.4),
   (iv) $(a^x)^b$ is replaced by $(a^b)^x$,
   (v) $(x_1 \dots x_n)^a$ is replaced by $x_1^a \dots x_n^a$,
   (vi) quadratic functions are converted to an expanded form: (1) all products are disaggregated, i.e., $x_0(a_0 + a_1x_1 + \ldots + a_nx_n)$ is replaced by $a_0x_0 + a_1x_0x_1 + \ldots + a_nx_0x_n$; a similar reformulation is employed for the more general form $(a_0 + a_1x_1 + \ldots + a_nx_n)(b_0 + b_1y_1 + \ldots + b_ny_n)$, (2) monomials of the form $(a_0 + a_1x_1 + \ldots a_nx_n)^2$ are expanded only if such a reformulation is needed for convexity detection. If in the same constraint: (i) there exists a bilinear with a common variable; e.g. $(x_1 + x_2)^2 - 2x_1x_2 \leq 1$ is replaced by $x_1^2 + x_2^2 \leq 1$, (ii) there exists another monomial whose coefficient has the opposite sign and

the two monomials have common variables; e.g. $(x_1 + 2x_2)^2 + x_1^2 - 2x_2^2 \leq 1$ is replaced by $2x_1^2 + 4x_1x_2 + 2x_2^2 \leq 1$.

For instance, consider the reformulation (i) above. As a composite function, $f = \log(a^x)$ is a concave increasing function (log) of a convex function ($a^x$), a case whose convexity or concavity does not follow from Lemma 1. However, in the reformulated form, $f = x \log a$ is clearly an affine function. The above operations are applied in a recursive manner until we obtain a formulation for which none of these operations are applicable; for example $f = \sqrt{2^{\log(x+y)}}$ simplifies to $f = (x+y)^{0.35}$. To establish convexity, it suffices to verify that the objective function $f_0$ and all inequality constraint functions $f_i$, $i = 1, \ldots, m$ as defined in Problem (CV) are convex, and that the equality constraints are affine. In the following, we discuss various strategies to examine convexity properties of a multivariate function $f(x)$; all these techniques are used to establish convexity of objective and constraint functions.

II. **Initial convexity assessment.** With the objective of minimizing the overall computational cost of convexity detection, prior to the initialization of the branch-and-bound tree, we mark all convex expressions in the problem, identify various sources of nonconvexity, and store the information in proper data structures for later investigation. In particular, we classify variables that contribute to the nonconvexity of the problem as follows. Consider an optimization problem whose convexity is verifiable by the proposed detection scheme only if a subset of (original) variables become fixed. We refer to this subset as the set of *nonconvex variables*. For example, consider the constraint $x_1^2 + \log x_2 \leq 1$; in this case, $x_2$ is a nonconvex variable, whereas, for the constraint $x_1^2 + \log(x_1 + 2x_2) \leq 1$, both $x_1$ and $x_2$ are marked as nonconvex variables. Now, consider the function $f(x) = (x_1 - 2x_2)^3$, $x \in \mathbb{R}^2$. Clearly, $f$ is neither convex nor concave. However, if at some node in the branch-and-bound tree, we have $x_1 - 2x_2 \geq 0$ (resp. $x_1 - 2x_2 \leq 0$), then $f$ is convex (resp. concave) in that node. Accordingly, we define a *concavoconvex variable* as an original variable or an affine combination of original variables that appears in concavoconvex monomials, i.e., $f = x^{2k+1}$, $k = 1, 2, \ldots$, $x \in [\underline{x}, \bar{x}]$ such that $\underline{x} < 0 < \bar{x}$. In addition, with each concavoconvex variable $x_j$ (or $y_j = a^T x + b$), we associate a domain restriction flag $\eta_j$, defined as: (i) $\eta_j = 1$, if $x_j$ appears in concavoconvex monomials with positive coefficients, in which case the problem is convex only if $x_j$ takes nonnegative values, (i) $\eta_j = -1$, if $x_j$ appears in concavoconvex monomials with negative coefficients; in this case, if the problem becomes convex, then we have $x_j \leq 0$ and, (iii) $\eta_j = 0$, if $x_j$ appears in concavoconvex monomials with positive and negative coefficients; in this case, if $x_j$ takes nonzero values then the problem is nonconvex. We should remark that each variable can appear in at most one of the above lists based on the following dominance relations: (i) if a variable appears in concavoconvex terms with $\eta_j = 1$ (or $\eta_j = -1$ ), and nonconvex terms, then we append it to the list of nonconvex variables, (ii) if a variable appears in concavoconvex terms with $\eta_j = 0$ and nonconvex terms, then we mark it as a concavoconvex variable with $\eta_j = 0$. In the following, we describe the procedure for initial convexity assessment of a general factorable NLP.

Consider a constraint of the form $h_1(x_1) + \ldots + h_m(x_m) \leq \alpha$, $m \geq 1$, $x \in [\underline{x}, \bar{x}] \subseteq \mathbb{R}^n$, where each $h_j(x_j)$ is a primitive function or a composition of primitive functions, $x_j \in \mathbb{R}^{n_j}$, $n_j \leq n$ contains some components of $x$, $\alpha \in \mathbb{R}$ is a scalar, and $h_j(x_j)$ cannot be represented as $h_j = \sum_{k=1}^{K} h_{jk}$, where each $h_{jk}$ is a primitive function (or a composition of primitive functions). For example, consider the constraint $x_1^2+1.5(x_1\log x_1)^2+x_2^2+2x_1x_2-\exp(-x_2^2)\leq 1$ with $h_1(x_1,x_2)=x_1^2+2x_1x_2+x_2^2$, $h_2(x_1)=1.5(x_1\log x_1)^2$, and $h_3(x_2)=-\exp(-x_2^2)$. In this case, we do not further decompose $h_1$ to square and bilinear terms, as a nonseparable quadratic function belongs to the list of primitive functions. In addition, a decomposition of the form $h_2=1.5(x_1\log x_1)^2-\exp(-x_2^2)$ is not valid, since this expression is not a primitive function, and can be decomposed into two subfunctions $h_{21}=1.5(x_1\log x_1)^2$ and $h_{22}=-\exp(-x_2^2)$, each of which is obtained by an application of composition rules to primitive functions. First, suppose that $h_j(x_j)$ is not a quadratic function containing bilinear terms. We associate a *convexity flag* $\omega_j$ with each $h_j(x_j)$, defined as follows:

(i) $\omega_j=1$: the convexity detector verifies convexity of $h_j(x_j)$. We mark this function as convex, and do not re-examine its convexity in subsequent nodes.

(ii) $\omega_j=-1$: the convexity detector either verifies concavity of $h_j(x_j)$ or returns an inconclusive flag that is unchangeable unless all variables in $h_j(x_j)$ become fixed. In this case, we append $x_j$ to the list of nonconvex variables, and do not revisit $h_j$ later in the search tree. For example, consider $f=\log(x_1^2+x_2^2+1)$. As a composite function, $f$ is a concave increasing function of a (strictly) convex function; a structure that is not covered by Lemma 1, and will not change during the search tree unless both $x_1$ and $x_2$ are fixed.

(iii) $\omega_j=0$: the convexity detector returns an inconclusive flag; however, convexity properties of $h_j$ may change later, as a result of range reduction or branching operations. In this case, $h_j$ is marked for later check in the branch-and-bound tree. Moreover, if $h_j=(c^Tx_j+d)^{2k+1}$, $k=1,2,\ldots$ is concavoconvex, then we append $y=c^Tx_j+d$ to the list of concavoconvex variables. For example, consider $f=(x_1^2-x_2)^2$, $x\in[-1,1]^2$. In this case, $f$ is a convex nonmonotone function of a convex function, which is an inconclusive composition. In fact, it is simple to verify that, over $[-1,1]^2$, the function $f$ is neither convex nor concave. However, if at some node in the search tree $x_1$ is fixed or the inequality $x_1^2-x_2\geq 0$ is satisfied, then convexity of $f$ is verifiable by the proposed detection scheme.

Now, suppose that $h_j(x_j)$ is a nonseparable quadratic function containing bilinear terms. For notational simplicity, we drop the subscript $j$ from $x_j$ in the following. To assess convexity properties of $h_j(x)=x^TQx=\sum_{i=1}^n\sum_{k=1}^n q_{ik}x_ix_k$, we first check the sign of first- and second-order principal minors of $Q$. Namely, (i) if $h_j$ contains a square term $x_i^2$ with a negative coefficient, then we append $x_i$ to the list of nonconvex variables, (ii) for any bilinear term $x_ix_k$ in $h_j$ with $q_{ii}\geq 0$ and $q_{kk}\geq 0$, if $q_{ii}q_{kk}-q_{ik}^2<0$, then we add $x_ix_k$ to the list of *nonconvex bilinears*, i.e., if $h_j$ is convex at a certain node, then either $x_i$ or $x_k$ is fixed. Subsequently, we classify $h_j$ as follows:

(i) if all square terms in $h_j$ have negative coefficients, then $h_j$ does not become convex unless all $x$ variables are fixed. In this case, we let $\omega_j=-1$, and do not revisit $h_j$ later in the search tree.

(ii) if $Q$ has at least one negative first- or second-order principal minor, then we let $\omega_j=0$, and mark $h_j$ for later check in subsequent nodes.

(iii) if all first- and second-order principal minors of $Q$ are nonnegative, then we compute its eigenvalues. If all eigenvalues of $Q$ are nonnegative, we mark $h_j$ as convex, i.e., $\omega_j = 1$; otherwise, we let $\omega_j = 0$, and utilize the following criterion to determine whether it is necessary to recalculate eigenvalues of the Hessian in descendent nodes:

**Lemma 4** *Consider a quadratic function $h(x) = x^T Qx$, $x \in \mathbb{R}^n$. Suppose that $Q$ has $p$ negative eigenvalues. Let $h(\tilde{x})$ denote a convex quadratic function obtained by fixing a subset of variables in $h(x)$. Then, $h(\tilde{x})$ has at most $n - p$ variables.*

*Proof* Suppose that $h(\tilde{x})$ has $n - p + 1$ variables and, without loss of generality, assume that $h(\tilde{x})$ is obtained by fixing the last $p - 1$ components in $x$, i.e., $\tilde{x} = [x_1, \ldots, x_{n-p+1}]$. Denote by $\mathcal{S}$ the subspace spanned by all $\tilde{x} \in \mathbb{R}^{n-p+1}$. Let $V = \{v^i\}_{i=1}^n$ denote a set of $n$ orthonormal eigenvectors of $Q$, and let $V' = \{v^{i_1}, \ldots v^{i_p}\}$ be the set eigenvectors associated with negative eigenvalues ($\lambda$) of $Q$. Denote by $\mathcal{T}$ the $p$-dimensional subspace spanned by the eigenvectors in $V'$. It follows that $\mathcal{S} \cap \mathcal{T}$ contains at least one nonzero vector of the form $u = \sum_{j \in J} \alpha_j v^j$, $J = \{i_1, \ldots, i_p\}$, $\alpha \in \mathbb{R}^p$. By orthonormality of the vectors in $V'$, it follows that $h(u) = u^T Qu = \sum_{j \in J} \lambda_j \alpha_j^2 < 0$, which is in contradiction with the assumption that the restriction of $h(x)$ to $\mathcal{S}$ is convex. Thus, $h(\tilde{x})$ has at most $n - p$ variables.

Therefore, for each nonconvex quadratic $h_j$ with nonnegative first- and second-order principal minors, we store the number of negative eigenvalues $p_j$, and re-examine positive-semidefiniteness of the Hessian only if at least $p_j$ variables in $h_j$ are fixed. A similar procedure is applied for convexity assessment of quadratic forms that are composed with other nonlinear functions.

The techniques outlined above are utilized in the first call to convexity detector during `BARON`'s preprocessor. Subsequently, at the root node, we update the initial assessment, if applicable. For instance, if at the root node a nonconvex variable is fixed or a concavoconvex variable has the proper domain, then they are removed from the corresponding lists. The purpose of such updates is to minimize the computational cost of convexity detection in subsequent nodes in the branch-and-bound tree. Clearly, if during the initial assessment or at the root node it turns out that $\omega_j = 1$ for all $j$, then the problem is convex, and the detection algorithm terminates.

III. **Early termination tests.** Consider a bilinear program (P), i.e., the problem of minimizing a bilinear function subject to bilinear inequality constraints. Clearly, P becomes convex if and only if, for every bilinear term $x_i x_j$ in the formulation, at least one of $x_i$ and $x_j$ is fixed to a certain value. In other words, problem (P) is convex only when it reduces to an LP. The same argument holds for QCQPs consisting of component-wise concave quadratic functions. Clearly, for these examples, the global solver does not benefit from convexity detection. In general, we would like to identify cases for which convexity detection is not useful so as to avoid the extra computational cost in the branch-and-bound tree.

At every call to the convexity detector, we first mark all fixed variables and eliminate them from the factorable reformulation of the optimization problem. More precisely, we call $x_i \in [\underline{x}_i, \bar{x}_i]$, a fixed variable, if $\bar{x}_i - \underline{x}_i \leq \epsilon$, for some $\epsilon > 0$. Denote by $n_f$ the number of (original) variables that are fixed at the current node. Subsequently, we mark all variables that appear linearly in the problem; let $n_l$ denote the number of such variables. Define $n_r = n - n_f - n_l$, where $n$ is the number of variables in the original problem. Clearly, if $n_r = 0$, then this problem is an LP and the convexity detector terminates. Let $n_c$ denote the number of nonconvex variables and denote by $n_o$ the number of original concavoconvex variables with $\eta = 0$, at the current node. It follows that, if $n_r = n_c + n_o$, then the convexity detector is not beneficial as the problem remains nonconvex unless all variables that appear in nonlinear expressions become fixed. In particular, if this equality holds during preprocessing or at the root node, then the convexity detector will be deactivated for this problem.

IV. **Fast convexity detection strategies.** At every node in the branch-and-bound tree, we utilize the information stored during the initial convexity assessment to speed up the convexity detector. We start by examining the list of nonconvex and concavoconvex variables. If a nonconvex variable is not fixed in this node or a concavoconvex variable does not have the proper domain, then the problem is still nonconvex and the convexity detector terminates. For problems containing multivariate quadratic expressions, we employ the following tests: (i) if a nonconvex bilinear is not linear in this node or, (ii) if a quadratic function with $n$ variables and $k$ negative eigenvalues has more than $n - k$ unfixed variables, then the problem is still nonconvex. If the above conditions do not disprove the problem's convexity, we proceed to the next step and scan the list of nonlinear expressions with $\omega_j = 0$, as defined in Step II. If convexity of any of the functions is not verifiable or a function is found to be nonconvex, then the convexity detector terminates. Otherwise, the problem is marked as convex in this node.

Finally, since convexity of QCQPs does not depend on variable bounds, we define a list of *convexity patterns* $\mathcal{L}_C = \{l^k\}_{k \in K}$ for this problem class. Namely, once convexity of the QCQP is verified at a given node, we append the index set of fixed variables $\tilde{l}$ to the list of convexity patterns. In addition, we keep the size of $\mathcal{L}_C$ minimal by removing any $l^k$ from the list that is implied by $\tilde{l}$. In subsequent nodes, prior to the utilization of convexity detection tests, we compare the index set of fixed variables $l^0$ with those stored in $\mathcal{L}_C$ and, if $l^0 \supsetneq l^k$ for any $k \in K$, we conclude that the problem is convex.

With the exception of quadratic functions, our convexity detection tool relies on an efficient use of symbolic operations. In general, existing numerical verification algorithms are too expensive to be embedded in the branch-and-bound tree. However, an interesting possibility would involve utilizing the numerical approach of [33] to examine convexity properties of certain functional types for which the symbolic test is inconclusive.

### 3.2 Dynamic local solver selection

Once convexity of the continuous relaxation of a sub-problem is verified at a given node in the branch-and-bound tree, we employ a local solver to compute a lower bound by solving the convex NLP. Hence, a highly efficient and reliable local solver is key to the success of this approach. Through our experiments with several state-of-the-art local solvers, we have observed that each solver behaves quite differently on various problem types and, more importantly, there is no single solver that outperforms others on a wide range of optimization problems. In fact, even for a single problem in the course of the branch-and-bound tree, as the global solver searches through different parts of the feasible region, the most suitable local solver may vary. Consequently, to maximize the impact of convexity detection and increase the likelihood of finding better upper bounds on the global solution, we devised a dynamic local solver selection scheme that alternates among various solvers based on their performance in the branch-and-bound tree.

Suppose that we have access to a number of local solvers for the global search. Prior to the application of each local search in the branch-and-bound tree, we would like to employ a simple algorithm that selects a solver with the highest chance of success. To measure the success rate of a local solver, we associate a binary flag *win* with each call to that solver defined as follows: (i) for upper bounding, the local solver wins ($win = 1$), if it returns a solution that passes our feasibility test and improves the value of the best known upper bound, (ii) for lower bounding, the local solver wins ($win = 1$), if it returns a solution that satisfies the KKT conditions. We store this information in global and local data structures. For each solver, we store (i) the total number of wins in the branch-and-bound tree denoted by $N_{\text{wins}}$ and (ii) the number of consecutive wins/losses denoted by $N_{\text{gains}}$, where a positive number for $N_{\text{gains}}$ indicates the number of consecutive wins, whereas a negative number corresponds to the number of consecutive losses. Furthermore, if $m$ consecutive wins (resp. losses), are followed by a loss (resp. win), then $N_{\text{gains}}$ is reset to zero. Finally, we define a rank $r_s \in [1, \bar{r}]$ for each solver $s$. At a given node, solvers are selected for local search based on this rank. A smaller value for rank implies a higher chance of success. We initialize the rank of each solver based on our prior knowledge regarding solver average performance on a large number of test problems, and update $r_s$ during the global search as follows. If a solver fails $\eta$ consecutive times, then we decrease the frequency at which the solver is called by downgrading its rank: $r_s = \min(2r_s, \bar{r})$. Similarly, if a solver wins $\eta$ consecutive times, we upgrade its rank using the relation $r_s = \max(1, r_s/2)$. Prior to each local search, we employ the above learning procedure to select a local solver as follows. If all local solvers have failed too often, i.e., $r_s = \bar{r}$ for all solvers, then the solver with the largest total number of wins ($N_{\text{wins}}$) is selected; otherwise, a solver with the best rank is utilized for local search.

### 3.3 Combining polyhedral and nonlinear relaxations

If convexity of an NLP is verified during preprocessing or at the root node, then as soon as a local solver returns a solution that satisfies KKT conditions, `BARON` terminates. To avoid branching for convex problems, at the root node, we utilize all available local solvers based on their rank, and proceed to the branching step only if they all fail to find an optimal solution. In the following, we describe the integration of nonlinear relaxations into `BARON`'s branch-and-cut framework.

Suppose that the convexity detector verifies that a sub-problem in the branch-and-bound tree is convex. We select a local solver from the list of available solvers using the dynamic scheme outlined in the previous section. It is well known that, even for convex problems, a good starting point can highly affect the performance of local solvers. Indeed, providing a near-optimal starting point often expedites the convergence rate of Newton-type methods significantly. We construct a crude polyhedral relaxation of the convex NLP using `BARON`'s polyhedral relaxation constructor and utilize its solution as the starting point for the local solver. If the solution reported by the local solver satisfies the KKT conditions, then the optimal value of the convex NLP is used as the lower bound in this node; otherwise, the local solver's solution is discarded and `BARON` continues with the conventional polyhedral relaxation scheme as detailed in [44]. In addition, to avoid the extra cost of solving many NLPs for which the local solver fails to find an optimal solution, we adjust the frequency at which the NLP lower bounding scheme is used based on the performance of local solvers.

Now, consider the case where the primal-dual pair $(\tilde{x}, \tilde{\nu})$ returned by the local solver is not optimal but is dual feasible, i.e., the gradient of the Lagrangian with respect to $x$ vanishes at $(\tilde{x}, \tilde{\nu})$. By weak duality, the value of the Lagrangian function $L(x, \nu)$ at $(\tilde{x}, \tilde{\nu})$ serves as a lower bound to the optimal value of the convex NLP. In this case, we set the lower bound to $LB = \max(L(\tilde{x}, \tilde{\nu}), \hat{f})$, where $\hat{f}$ is the lower bound obtained by `BARON`'s polyhedral relaxation constructor. Finally, if both the LP and NLP solves fail, we resort to calculating a simple interval extension of the objective function in order to obtain a lower bound for the current node.

In this paper, we make use of local NLP solvers for lower bounding only if the continuous relaxation of the original problem is convex at the current node. More generally, local NLP solvers can be utilized to optimize nonlinear factorable relaxations of nonconvex problems of the type considered in [39]. However, these relaxations often have many more variables and constraints compared to the original problem and are more prone to solver failures. Designing efficient hybrid schemes for optimizing these nonlinear factorable relaxations is a subject of future research.

## 4 Numerical Experiments

The purpose of this section is to demonstrate the computational benefits of the proposed techniques by incorporating them in the branch-and-reduce global solver `BARON` [40, 44]. To this end, we consider a variety of NLPs and MINLPs from widely-used collections of test problems: 980 NLPs from `PrincetonLib` [37], 369 NLPs from `GlobalLib` [24], and 250 MINLPs from `MINLPLib` [14]. These test sets were obtained after eliminating problems involving trigonometrics, error functions and other expressions that `BARON` cannot handle from the original collections. In Table 1, we provide some statistics on the size of the problems used in our computations in terms of the numbers of constraints ($m$), variables ($n$), discrete variables ($n_d$), nonzero elements in the constraints and objective ($nz$), and nonlinear nonzero elements in the constraints and objective ($nnz$). Each test set contains a wide range of nonlinear problems, ranging from univariate optimization problems to problems with more than 20,000 variables, most of which have a significant nonlinear component.

Throughout this section, all experiments are performed with `GAMS 24.0.2` on a 64-bit Intel Xeon X5650 2.66Ghz processor; all implementations are single-threaded. In addition, all problems are solved with relative/absolute optimality tolerance of $10^{-6}$, and a CPU time limit of 500 seconds. Other algorithmic parameters are set to the default settings of the `GAMS` distribution for all solvers. When comparing the performance of different algorithms, we call a problem trivial if all algorithms take less than half a second to solve it to optimality. Denote by $f_{p,s},\ s \in \mathcal{S}$ the feasible solution returned by solver $s$ upon termination for model $p$, and let $f_p^*$ denote the best feasible solution among all solvers, i.e., $f_p^* = \min_{s \in \mathcal{S}} f_{p,s}$. To measure the relative quality of the solution returned by each algorithm, we define

$$\Delta_{p,s} = \begin{cases} f_{p,s} - f_p^*, & \text{if } |f_p^*| < 1, \\ (f_{p,s} - f_p^*)/|f_p^*|, & \text{otherwise.} \end{cases} \tag{4}$$

In particular, we say that solver $s$ finds the best solution for problem $p$, if $\Delta_{p,s} < 10^{-3}$. In addition, we say that problem $p$ is solved to global optimality by global solver $s$, if (i) the solver claims optimality upon termination, (ii) the final lower and upper bounds satisfy our optimality tolerances, and (iii) $\Delta_{p,s} < 10^{-3}$. As we detail in the following, to perform fair comparisons, we include additional safeguards to detect cases for which solvers make incorrect optimality claims.

To evaluate and compare the performance of different solvers on our test set, we make use of performance profiles, as described in [16]. The performance profile of a solver is a (cumulative) distribution function for a performance metric. Throughout this section, we use the ratio of the time that an algorithm takes to *solve* a problem versus the best time of all algorithms as the performance metric. For the purpose of constructing performance profiles, by 'solve' we imply that the algorithm finds the best solution among all solvers for a given problem within the time limit. We utilize the `GAMS` performance tools [1] to construct all performance profiles.

**Table 1** Size statistics for the test set. For each collection, we list the numbers of constraints ($m$), variables ($n$), discrete variables ($n_d$) (for MINLPs), nonzero elements in the constraints and objective ($nz$), and nonlinear elements in the constraints and objective ($nnz$).

(a) 980 NLPs from `PrincetonLib`

| | $m$ | $n$ | $nz$ | $nnz$ |
|---|---|---|---|---|
| minimum | 0 | 1 | 2 | 0 |
| first quartile | 0 | 3 | 7 | 4 |
| median | 3 | 10 | 41 | 14 |
| third quartile | 112 | 492 | 2347 | 691 |
| maximum | 14001 | 20201 | 7970301 | 120001 |

(b) 369 NLPs from `GlobalLib`

| | $m$ | $n$ | $nz$ | $nnz$ |
|---|---|---|---|---|
| minimum | 0 | 1 | 2 | 0 |
| first quartile | 4 | 5 | 20 | 6 |
| median | 10 | 14 | 62 | 20 |
| third quartile | 76 | 126 | 665 | 442 |
| maximum | 26758 | 19314 | 128411 | 31261 |

(c) 250 MINLPs from `MINLPLib`

| | $m$ | $n$ | $n_d$ | $nz$ | $nnz$ |
|---|---|---|---|---|---|
| minimum | 0 | 2 | 1 | 3 | 1 |
| first quartile | 10 | 11 | 6 | 52 | 10 |
| median | 63 | 91 | 30 | 338 | 21 |
| third quartile | 269 | 195 | 72 | 1441 | 127 |
| maximum | 24971 | 23826 | 10920 | 106857 | 61108 |

### 4.1 Impact of proposed relaxation techniques on the performance of BARON

The cut generation and range reduction schemes of Section 2 and the hybrid LP/NLP-based lower bounding framework of Section 3, are implemented in `BARON 12.0`. To demonstrate the impact of these techniques, we will compare `BARON 12.0` against the same version of `BARON` but with all these techniques disabled. We will refer to the latter version of the solver as `BARON X`. Additionally, `BARON X` will be restricted to utilize `MINOS 5.5` for local search, whereas `BARON 12.0` performs dynamic local search using `CONOPT 3` [18], `IPOPT 3.9.0` [45], `MINOS 5.5` [34], and `SNOPT 7.2.4` [23] as local NLP sub-solvers. Both `BARON 12.0` and `BARON X` use `CPLEX 12.4` [26] as the default LP solver.

To compare the performance of `BARON 12.0` and `BARON X`, we first eliminate from the test set all problems that are trivial and problems for which both algorithms fail to return a feasible solution or cannot guarantee global optimality due to numerical difficulties. Subsequently, for a given problem, we classify the relative performance of the two algorithms with respect to their computational time, as follows:

(i) Solver `A` is considered infinitely faster than Solver `B` if Solver `A` solves the problem to global optimality within the time limit, while Solver `B` fails due to numerical difficulties or does not return a feasible solution upon termination.
(ii) A solver is considered much faster than another if it is more than 50% faster.
(iv) A solver is considered faster than another if it is faster by less than 50%.
(v) Solution times are considered the same, if they are within 10% of each other.

Comparative statistics for the test set are listed in Table 2. The first line of Table 2 provides the number and, in parentheses, the percentage of problems for which `BARON 12.0` is infinitely faster than `BARON X` for each test library. The subsequent lines of the table provide similar statistics based on the classification described above. For more than 50% of the problems, the proposed relaxation techniques lead to significant performance improvement. For about one quarter of the problems, `BARON X` fails, whereas `BARON 12.0` returns a globally optimal solution within the time limit. For the second quarter of the problems, the proposed techniques reduce the CPU time of the global solver by at least a factor of two. We observe performance degradations for about 12% of the test problems, most of which are due to incorporating more conservative cut generation and bounding strategies, as described in Subsection 2.4. In Table 3, we compare the number of problems solved to global optimality with and without the proposed relaxation techniques. As can be seen, the new relaxations increase the number of solvable models by 37% for `PrincetonLib`, 15% for `GlobalLib`, and 8% for `MINLPLib`.

**Table 2** Effect of proposed relaxation techniques on the performance of `BARON` for 621 NLPs and MINLPs from `PrincetonLib`, `GlobalLib`, and `MINLPLib`.

| | `PrincetonLib` | `GlobalLib` | `MINLPLib` | Total models |
|---|---|---|---|---|
| `BARON 12.0` infinitely faster | 129 (32%) | 27 (23%) | 7 (7%) | 163 (26%) |
| `BARON 12.0` much faster | 117 (29%) | 18 (16%) | 23 (23%) | 158 (26%) |
| `BARON 12.0` faster | 14 (3%) | 6 (5%) | 5 (5%) | 25 (4%) |
| Solvers perform the same | 96 (24%) | 51 (44%) | 48 (48%) | 195 (32%) |
| `BARON X` faster | 17 (4%) | 8 (7%) | 8 (8%) | 33 (5%) |
| `BARON X` much faster | 25 (6%) | 5 (4%) | 8 (8%) | 38 (6%) |
| `BARON X` infinitely faster | 8 (2%) | 0 (0%) | 1 (1%) | 9 (1%) |

**Table 3** Number of problems solved to global optimality within 500 seconds, with and without the proposed relaxation techniques.

| Test library | `BARON 12.0` | `BARON X` |
|---|---|---|
| `PrincetonLib` | 726 | 532 |
| `GlobalLib` | 238 | 206 |
| `MINLPLib` | 136 | 126 |
| Total models | 1100 | 864 |

Next, we investigate the impact of the proposed techniques on *hard* problems, i.e., models that are not solvable by either of the two algorithms within the time limit, while a feasible solution is returned by at least one of the algorithms upon termination. In Table 4, we compare the quality of lower and upper bounds of `BARON 12.0` and `BARON X` after 500 seconds for hard problems from the test set. A solver is considered to provide a better upper bound (resp. lower bound), if the relative objective value difference (resp. relative lower bound difference) is greater than $\delta = 10^{-3}$. For objective values (resp. lower bounds) below one, we use absolute differences. Table 4(a) gives the numbers, and in parentheses the percentage, of problems for which `BARON 12.0` returns a better solution, the two algorithms return the same solution, and `BARON X` returns a better solution for each collection. Table 4(b) provides similar statistics with respect to the quality of final lower bounds. The results in this table demonstrate that the proposed convexification methodology has a positive impact on the quality of lower and upper bounds for hard problems from the three test libraries. Clearly, the proposed techniques are most beneficial for continuous problems.

**Table 4** Relative performance of `BARON 12.0` and `BARON X` for 328 NLPs and MINLPs that are not solvable to global optimality within 500 seconds.

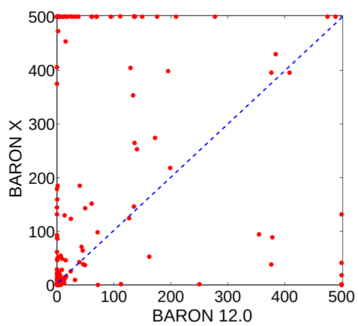
(a) Quality of best feasible solution

| Test library | `BARON 12.0` better | same solutions | `BARON X` better |
|---|---|---|---|
| `PrincetonLib` | 137 (74%) | 39 (21%) | 9 (5%) |
| `GlobalLib` | 49 (58%) | 29 (34%) | 7 (8%) |
| `MINLPLib` | 29 (50%) | 10 (20%) | 19 (30%) |
| Total models | 215 (66%) | 78 (24%) | 35 (10%) |

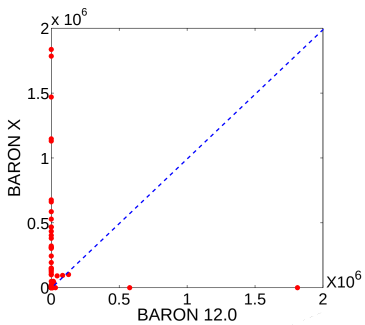
(b) Quality of lower bounds upon termination

| Test library | `BARON 12.0` better | same lower bounds | `BARON X` better |
|---|---|---|---|
| `PrincetonLib` | 136 (73%) | 29 (16%) | 20 (11%) |
| `GlobalLib` | 43 (51%) | 28 (33%) | 14 (16%) |
| `MINLPLib` | 31 (54%) | 6 (10%) | 21 (36%) |
| Total models | 210 (64%) | 64 (20%) | 54 (16%) |

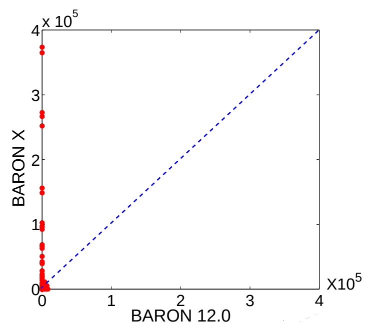
In Figure 3, we compare the performance of `BARON 12.0` and `BARON X` with respect to the following factors: (i) execution time, (ii) total number of nodes in the branch-and-bound tree (iterations), and (iii) maximum number of nodes stored in memory (memory). Comparisons are performed on nontrivial problems for which neither of the two algorithms fails due to numerical difficulties and at least one of the algorithms finds the global solution within the time limit. As seen in this figure, incorporating the proposed techniques in `BARON` results in average reductions of 40% in CPU time, and 35% in number of iterations as well as in maximum number of nodes in memory.




(a) CPU Time




(b) Iterations




(c) Memory

**Fig. 3** Performance of `BARON` with and without the proposed convexification methodology for 288 problems from `PrincetonLib`, `GlobalLib`, and `MINLPlib`. In these figures, nontrivial problems that are solved to global optimality by at least one of the two algorithms are compared with respect to (a) CPU time, (b) total number of nodes in the branch-and-bound tree, and (c) maximum number of nodes stored in memory.

In Table 5, we report the percentage of CPU time spent on generating cutting planes for convex intermediates as well as automatic convexity detection. By $\eta_{\text{cut}}$, we denote the average percentage of time spent on recognition and cut generation for the expressions described in Section 2, while $\eta_{\text{conv}}$ denotes the average percentage of time spent on convexity detection. In addition, the numbers in parentheses denote the standard deviations from each mean value. As can be seen from this table, on average, less than one percent of the total time is spent on convexity detection, while the average time spent on generating cutting planes is slightly higher. For large convex problems with high-dimensional dense quadratic expressions, convexity detection may take up to 5 seconds. However, without convexity detection, `BARON` is not able to solve such problems within hours. Therefore, a careful time allocation for convexity detection based on the size and structure of the problem is of crucial importance.

Next, we examine the impact of the hybrid lower-bounding scheme of Section 3 on the performance of `BARON` for convex problems. In particular, we consider those problems whose continuous relaxations are proved to be convex in the first call to the convexity detector; that is, during preprocessing. In the first line of Table 6, we list the number of problems in each collection that are not solved to global optimality prior to the first call to the convexity detector. The second line contains the number of these problems that are found to be convex during preprocessing. In subsequent lines, we report the number of convex models that are solved to global optimality by `BARON X` and `BARON 12.0`, respectively. Clearly, employing highly efficient local solvers along with an examiner facility to test the optimality of the reported solution is very effective for continuous problems. For convex MINLPs, however, the impact of nonlinear relaxations is less significant. This is mainly due to the hard combinatorial component of MINLP models, as we discuss below.

**Table 5** Relative computational cost of convexity exploitation operations in `BARON`.

| Test library | $\eta_{\text{cut}}$ (%) | $\eta_{\text{conv}}$ (%) |
|---|---|---|
| `PrincetonLib` | 1.50 (5.44) | 0.45 (4.45) |
| `GlobalLib` | 0.87 (1.54) | 0.07 (0.41) |
| `MINLPLib` | 1.87 (6.39) | 0.003 (0.02) |

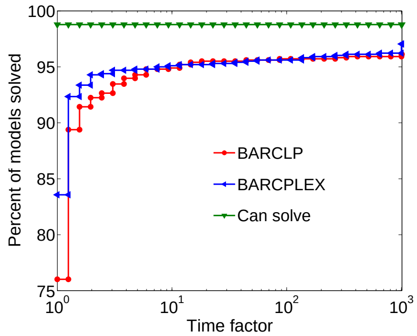
**Table 6** `BARON`'s performance for convex NLPs and MINLPs.

| Number of problems | `PrincetonLib` | `GlobalLib` | `MINLPLib` |
|---|---|---|---|
| nontrivial models | 740 | 337 | 232 |
| convex models | 250 | 30 | 86 |
| solved by `BARON X` | 137 | 17 | 43 |
| solved by `BARON 12.0` | 237 | 30 | 47 |

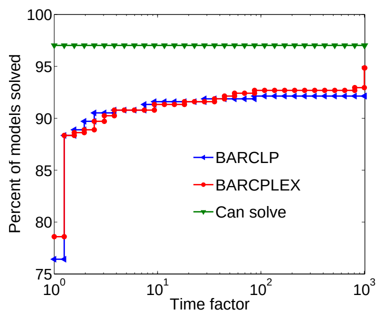
### 4.2 Impact of LP/NLP sub-solvers on the performance of `BARON`

Global solvers utilize LP/MIP and local NLP solvers at various stages during the global search. For instance, `BARON` uses LP codes for solving polyhedral relaxations of nonconvex problems as well as reducing the domain of variables via probing techniques (see [43] for details). Local NLP solvers are crucial for finding good feasible solutions, and they are used by `BARON 12.0` to construct hybrid LP/NLP relaxations. In this section, we examine the impact of various LP and local NLP solvers on the performance of `BARON`.

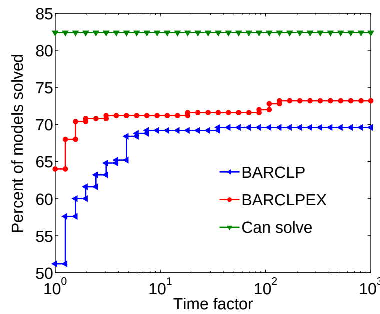
By default, `GAMS/BARON` uses the commercial code `CPLEX` [26] as the LP solver. Nonetheless, the Coin-OR LP solver `CLP` [20], is competitive with `CPLEX` on a wide range of problem types. Figure 4 shows the performance profiles of `BARON` with `CPLEX` and `CLP` as its LP solvers. As can be seen from this figure, performance of the two algorithms is similar for continuous models while, for MINLP models, `CPLEX` outperforms `CLP` by a relatively small margin. This difference could be due the fact that `CPLEX` exploits special structures that are often encountered in combinatorial optimization problems.




(a) PrincetonLib




(b) GlobalLib

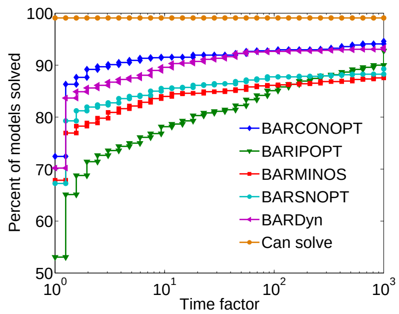



(c) MINLPLib

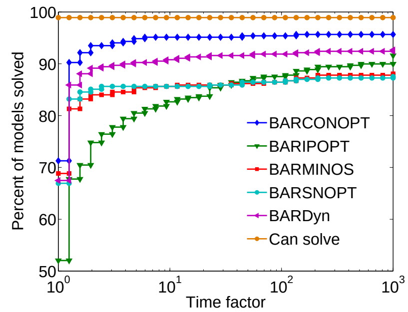
**Fig. 4** Performance profiles of BARON 12.0 using CPLEX as the LP solver (denoted by BARCPLEX) versus BARON 12.0 using CLP as the LP solver (denoted by BARCLP) for problems from PrincetonLib, GlobalLib, and MINLPLib.

In general, however, we conclude that CPLEX and CLP are equally effective when used under BARON to solve BARON's LP relaxations.

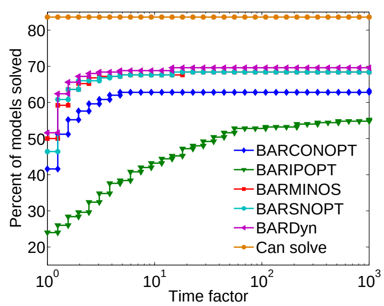
Next, we examine the impact of various local solvers on the performance of BARON. To this end, we consider the following cases: (i) BARMINOS: BARON with MINOS 5.5 [34] as the local solver, (ii) BARSNOPT: BARON with SNOPT 7.2.4 [23] as the local solver, (iii) BARCONOPT: BARON with CONOPT 3.0 [18] as the local solver, (iv) BARIPOPT: BARON with CoinIPOPT 3.9.0 [45] as the local solver, and (v) BARDyn: BARON with dynamic local solver selection strategy, which employs all aforementioned local solvers, as described in Subsection 3.2. Performance profiles for different local search strategies are shown in Figure 5. In Table 7, for each algorithm, we report the percentage of problems for which (i) global optimality is guaranteed within the time limit (optimal), (ii) the best solution is found among all solvers (best solution), (iii) a feasible solution is returned upon termination (feasible), and (iv) no feasible point is returned after 500 seconds due to solver failure or numerical difficulties (failure). While BARIPOPT is often slower than other solvers for all three collections, the choice of the best local solver varies depending on the problem type. For continuous models from PrincetonLib and GlobalLib, BARSNOPT and BARMINOS perform similarly, and they are outperformed by BARCONOPT. However, for MINLPs, we observe a notable performance degradation for BARCONOPT, while BARMINOS and BARSNOPT represent the best solvers. More importantly, it can be seen that `BARDyn` is competitive with and even better than the best static configurations for all test libraries. Comparing the results provided in Tables 2, 3, and 7, we find that the choice of LP/NLP solvers does not have a critical impact on the performance of `BARON`. For the most part, the superior performance of `BARON 12.0` is due to the lower-bounding facilities proposed in this paper.




(a) `PrincetonLib`




(b) `GlobalLib`




(c) `MINLPLib`

**Fig. 5** Impact of various local NLP solvers on the performance of `BARON` for problems from `PrincetonLib`, `GlobalLib` and, `MINLPLib`

Finally, we discuss the failure rate of LP and local NLP solvers in the branch-and-bound tree. As we alluded to in Section 2.4, both LP and local NLP solvers occasionally make false optimality claims. Hence, accepting their solution without verifying optimality may deteriorate the performance of global solvers. Figure 6 shows LP/NLP subsolver failure rates in the branch-and-bound tree, on a total number of 1599 NLPs and MINLPs from `PrincetonLib`, `GlobalLib`, and `MINLPLib`. All experiments are done with the default version of `BARON 12.0`, which utilizes `CPLEX` as the LP solver and dynamic local solver selection to optimize nonlinear sub-problems. By failure, we imply incorrect infeasibility and optimality claims for LP models, and incorrect local optimality claims for NLP sub-solvers. In particular, we say that a local solver fails if it declares the problem as locally optimal, while the reported solution does not pass the KKT test. At the time of this writing, `BARON` discards all infeasibility claims made by local NLP solvers. Verifying the infeasibility of convex NLPs requires the solution of certain auxiliary problems, and is the subject of future research. The distribution of the total number of LP/NLP calls over which the failure test is conducted is shown in

**Table 7** Comparative statistics for the impact of various local NLP solvers on the performance of BARON.

(a) 980 NLPs from PrincetonLib

| % problems | BARDyn | BARMINOS | BARSNOPT | BARCONOPT | BARIPOPT |
|---|---|---|---|---|---|
| optimal | 74% | 70% | 70% | 73% | 67% |
| best solution | 93% | 88% | 89% | 94% | 93% |
| feasible | 98% | 96% | 97% | 97% | 97% |
| failure | 2% | 4% | 3% | 3% | 3% |

(b) 369 NLPs from GlobalLib

| % problems | BARDyn | BARMINOS | BARSNOPT | BARCONOPT | BARIPOPT |
|---|---|---|---|---|---|
| optimal | 64% | 63% | 62% | 64% | 60% |
| best solution | 92% | 88% | 87% | 95% | 91% |
| feasible | 96% | 95% | 95% | 97% | 95% |
| failure | 4% | 5% | 5% | 3% | 5% |

(c) 250 MINLPs from MINLPLib

| % problems | BARDyn | BARMINOS | BARSNOPT | BARCONOPT | BARIPOPT |
|---|---|---|---|---|---|
| optimal | 54% | 54% | 54% | 54% | 51% |
| best solution | 70% | 68% | 68% | 63% | 55% |
| feasible | 80% | 78% | 76% | 71% | 63% |
| failure | 20% | 22% | 24% | 29% | 37% |

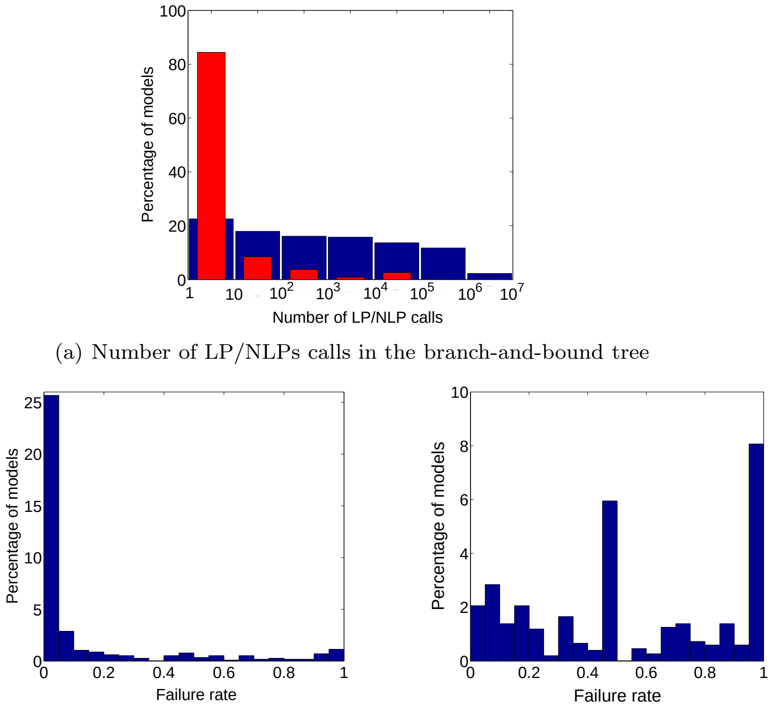
Figure 6(a). In Figures 6(b)-6(c), we have eliminated those models for which no failure has been reported. For 63% (resp. 67%) of the models, the LP solver (resp. dynamic local search) does not make incorrect claims. For the remaining problems, the failure rate of CPLEX is mostly under 5%, while, for a few models, the LP solver makes incorrect optimality/infeasiblity claims for a high percentage of LP sub-problems. For local solvers, however, the failure rate is more evenly distributed, highlighting the importance of optimality verification steps. It is important to note that the failure rates of LP vs. NLP solvers should be interpreted carefully as, on overage, the number of LP calls is significantly larger than the number of NLP calls in BARON's search tree (see Figure 6(a)).

## 4.3 Comparisons with other local and global solvers

In this section, we provide extensive comparisons with several state-of-the-art local and global algorithms. We consider the following solvers:

(i) Local NLP sovlers: CONOPT 3, IPOPT 3.9.0, MINOS 5.51, SNOPT 7.2.4.
(ii) Convex MINLP solvers: AlphaECP 2.09.02 [46], DICOPT [19], SBB [3]; all using CPLEX as the LP solver and CONOPT as the NLP solver.
(iii) Global solvers: Couenne 0.4 with CLP as the LP solver and IPOPT as the NLP solver, Lindo API v 7.0.1 with CONOPT as the NLP solver, SCIP 3.0.1 with CPLEX as the LP solver and IPOPT as the NLP solver.

We should remark that GAMS/IPOPT, which we use for the following comparisons, benefits from second-order derivatives and commercial factorization codes, neither of which are available in the version of CoinIPOPT that is incorporated by BARON as a local solver.




(a) Number of LP/NLPs calls in the branch-and-bound tree

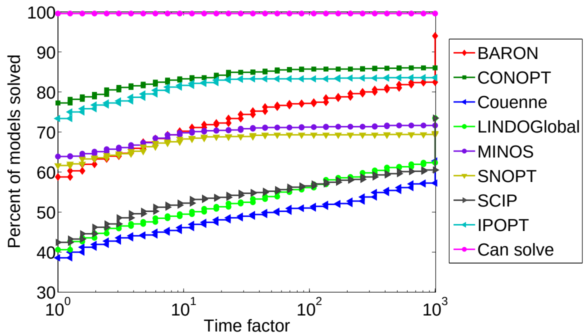
(b) Failure rate for `CPLEX`

(c) Failure rate for local solvers

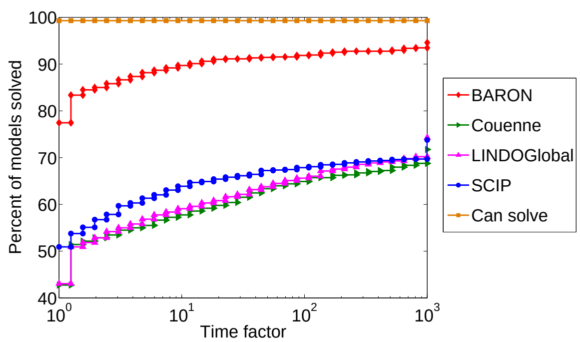
**Fig. 6** Failure rate of LP and local NLP sub-solvers in `BARON`'s search tree for 1599 NLPs and MINLPs from `PrincetonLib`, `GlobalLib`, and `MINLPLib`. In Figure 6(a), blue bars represent the distribution of LP calls, while red bars represent the distribution of NLP calls. In Figures 6(b) and 6(c), we have eliminated those models for which no sub-solver failure was diagnosed in the branch-and-bound tree by `BARON`'s examiner.

To conduct a fair comparison, optimal solutions returned by all solvers are further tested using `GAMS/EXAMINER` [2]. The optimality checks done in `EXAMINER` are first-order optimality conditions. `EXAMINER` takes the solution reported by the solver and tests for primal feasibility, dual feasibility, and complementarity slackness. While such a test is not sufficient to verify optimality for nonconvex problems, as we will see shortly, `EXAMINER` is an effective tool to identify many incorrect optimality claims made by different solvers. We employ `EXAMINER`'s first-order optimality test for `BARON` and local NLP solvers. However, since `SCIP` and `Couenne` do not return optimal dual values upon termination, we merely use `EXAMINER`'s primal feasibility check for `SCIP`, `Couenne`, `LindoGlobal`, and all convex MINLP solvers. In addition, we set `EXAMINER`'s feasibility tolerance to $10^{-5}$, which is larger than the default value for the feasibility tolerance of the solvers under consideration.

Performance profiles of local and global solvers for the 980 NLPs from `PrincetonLib`, and the 369 NLPs from `GlobalLib` are shown in Figures 7 and 8, respectively. In comparison to other global solvers, `BARON` is strongly dominant for both test libraries: for all performance ratios, `BARON` is ahead of the next best global solver by a 25% margin for `PrincetonLib`, and a 20% margin for `GlobalLib`. However, when local solvers are included in the solver pool, `BARON` is outperformed by `CONOPT` and `IPOPT` for




(a) local and global solvers

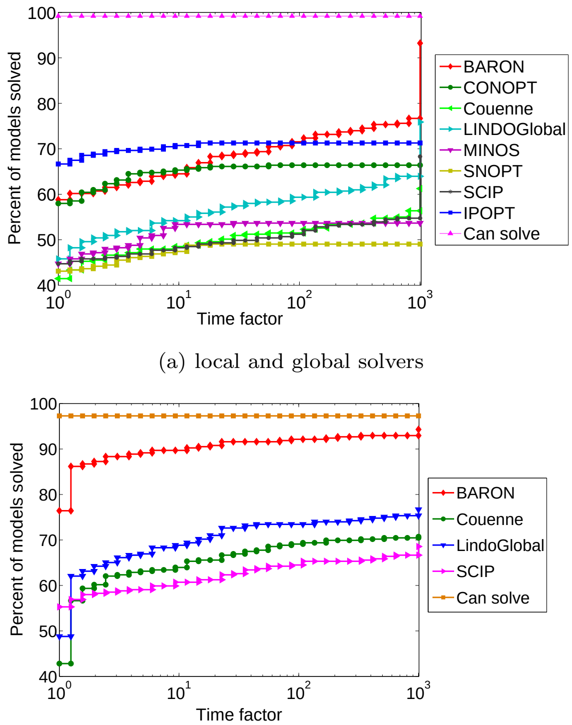



(b) global solvers

**Fig. 7** Performance profiles of local and global solvers for 980 NLPs from `PrincetonLib`

`PrincetonLib`, as this collection includes many large-scale problems for which `BARON` is not efficient at this point. For the `GlobalLib` collection, `BARON` outperforms `CONOPT` quickly, but requires about two orders of magnitude more CPU time to overtake `IPOPT`. It is interesting to note that, for both collections, `BARON` performs better than `MINOS` and `SNOPT`, both of which are highly efficient local solvers. This success is in part due to inability of local solvers to find global solutions of nonconvex problems and, more importantly, demonstrates the remarkable progress of global optimization algorithms and solvers over the past decade; `BARON` is capable of building on local solvers, such as `MINOS` and `SNOPT`, in ways that outperform the capabilities of each local solver in isolation.

Performance profiles of convex MINLPs and global solvers for `MINLPLib` are depicted in Figure 9. For this test set, `SCIP` dominates all other solvers, thanks to its MIP technologies. This result is not surprising since, at this stage, `BARON` does not include standard MIP features, such as cutting planes, heuristics, and branching rules tailored to integer programs. To confirm this hypothesis, next, we relax the integrality requirements for all problems in `MINLPLib`, and obtain a collection of 250 NLPs, which we denote by `RMINLPLib`. Figure 10 shows the performance profiles of local and global solvers for `RMINLPLib`, where again `BARON` is by far ahead of all other global solvers. The relative performance of `BARON` and `SCIP` for `MINLPLib` and `RMINLPLib` highlights the importance of incorporating standard MIP techniques in MINLP solvers.



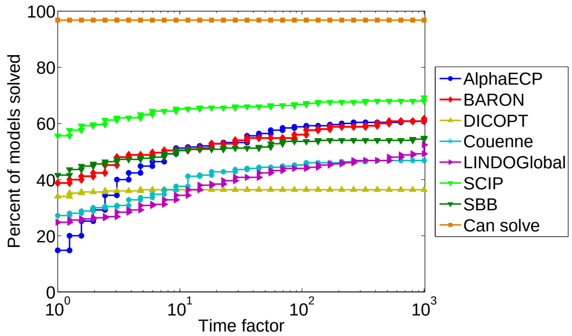

(a) local and global solvers



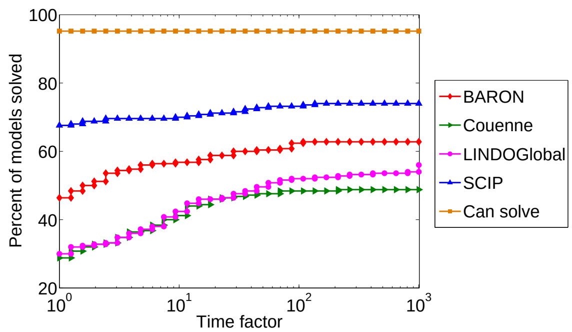
(b) global solvers

**Fig. 8** Performance profiles of local and global solvers for 369 NLPs from `GlobalLib`

In Table 8, we compare all solvers with respect to their solution quality. For each solver, we report the percentage of problems for which (i) global optimality is proved within the time limit (globally optimal), (ii) the best solution is found among all solvers (best solution), (iii) a feasible solution is returned upon termination (feasible), (iv) no feasible point is returned after 500 seconds due to the solver failure or numerical difficulties (failure), and (v) the reported solution does not pass `EXAMINER`'s test (examiner). By globally optimal, we refer to the cases for which a global solver (i) declares them as globally optimal and passes `EXAMINER`'s test, (ii) finds the best solution among all solvers and, (iii) closes the relaxation gap as defined by our optimality tolerances. As can be seen from Table 8, `BARON` is the only global solver for which no `EXAMINER` failure has been reported over the entire test set. `LINDOGlobal` takes the second place with a few incorrect claims, while `SCIP` and `Couenne` return infeasible solutions, for around 5% and 10% of the problems, respectively. In addition, we observe a high rate of false optimality claims for `MINOS` and `SNOPT`, while `IPOPT` and `CONOPT` are more reliable solvers. For continuous models, the lowest rate of solver failure belongs to `BARON`, followed by `IPOPT` and `CONOPT`, while other local and global solvers fail for about 10-20 percent of the models. For `MINLPs`, `SCIP` has the lowest rate of solver failure, followed by `AlphaECP` and `BARON`. `LINDOGlobal` has the highest number of solver failures for all three collections. Overall, we conclude that, for continuous models, `BARON` is the most robust solver, while `CONOPT` and `IPOPT` are often the fastest solvers, especially for large-




(a) local and global solvers




(b) global solvers

**Fig. 9** Performance profiles of local and global solvers for 250 MINLPs from `MINlPLib`

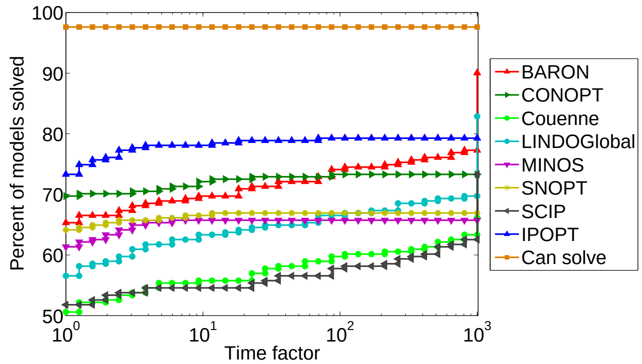
scale problems. For mixed-integer models, `SCIP` is ahead of all other MINLP solvers, local and global ones. Over the entire test test, `BARON` is ahead of all other global solvers by a 20% margin as seen in Figure 11.

## 5 Conclusions

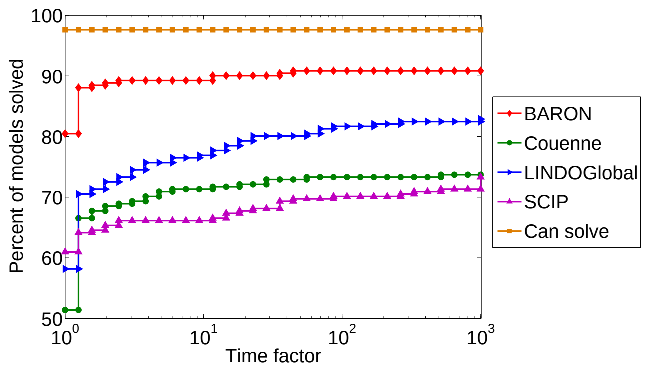
This paper demonstrates that convexity detection and exploitation is a powerful tool in global optimization of NLPs and MINLPs. We extended a widely-used polyhedral relaxation framework, by including cut generators for a variety of convex functions that appear frequently in applications. To capitalize on state-of-the-art in nonlinear programming, we developed a highly efficient convexity detector and proposed a hybrid LP/NLP-based lower bounding scheme that alternates between polyhedral and nonlinear relaxations at every node in the branch-and-bound tree. Results show that the proposed techniques significantly accelerate the branch-and-bound solver `BARON` and enable it to solve many more problems to global optimality.

## References

1. *GAMS Performance tools.* Available at `www.gamsworld.org/performance/tools.htm`.




(a) local and global solvers




(b) global solvers

**Fig. 10** Performance profiles of local and global solvers for 250 NLPs from `RMINLP`

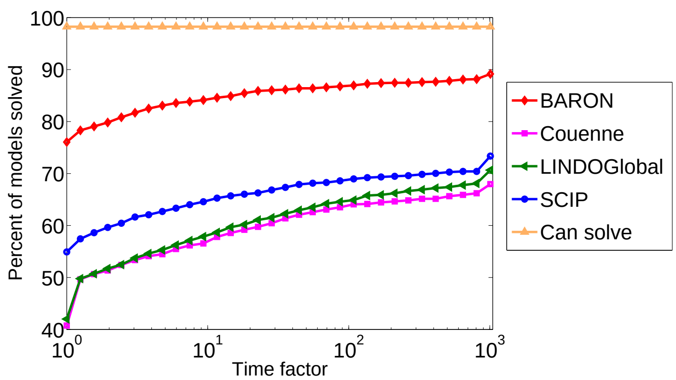



**Fig. 11** Performance profiles of global solvers for the entire test set consisting of 1599 NLPs and MINLPs (980 NLPs from `PrincetonLib`, 369 NLPs from `GlobalLib` and, 250 MINLPs from `MINLPLib`).

**Table 8** Comparative statistics for local and global solvers.

(a) 980 NLPs from `PrincetonLib`

| % problems | `BARON` | `Couenne` | `Lindo` | `SCIP` | `MINOS` | `SNOPT` | `CONOPT` | `IPOPT` |
|---|---|---|---|---|---|---|---|---|
| globally optimal | 75% | 48% | 59% | 54% | – | – | – | – |
| best solution | 94% | 73% | 74% | 74% | 69% | 67% | 86% | 83% |
| feasible | 99% | 78% | 80% | 84% | 74% | 74% | 94% | 91% |
| failure | 1% | 14% | 20% | 12% | 14% | 13% | 4% | 5% |
| examiner | 0% | 8% | 0% | 4% | 12% | 13% | 2% | 4% |

(b) 369 NLPs from `GlobalLib`

| % problems | `BARON` | `Couenne` | `Lindo` | `SCIP` | `MINOS` | `SNOPT` | `CONOPT` | `IPOPT` |
|---|---|---|---|---|---|---|---|---|
| globally optimal | 65% | 43% | 58% | 42% | – | – | – | – |
| best solution | 93% | 72% | 77% | 68% | 52% | 47% | 66% | 70% |
| feasible | 96% | 76% | 82% | 75% | 75% | 69% | 90% | 89% |
| failure | 4% | 14% | 17% | 16% | 14% | 11% | 9% | 7% |
| examiner | 0% | 10% | 1% | 9% | 11% | 20% | 1% | 4% |

(c) 250 MINLPs from `MINLPLib`

| % problems | `BARON` | `Couenne` | `Lindo` | `SCIP` | `DICOPT` | `SBB` | `AlphaECP` |
|---|---|---|---|---|---|---|---|
| globally optimal | 53% | 43% | 43% | 54% | – | – | – |
| best solution | 62% | 48% | 52% | 69% | 36% | 54% | 59% |
| feasible | 78% | 65% | 66% | 84% | 58% | 66% | 84% |
| failure | 20% | 24% | 31% | 10% | 40% | 34% | 15% |
| examiner | 0% | 12% | 3% | 6% | 2% | 0% | 1% |


2. *GAMS/EXAMINER,* User's Manual. Available at `www.gams.com/dd/docs/solvers/examiner.pdf`.
3. *GAMS/SBB,* User's Manual. Available at `www.gams.com/dd/docs/solvers/sbb.pdf`.
4. A. A. Ahmadi, A. Olshevsky, P. A. Parrilo, and J. N. Tsitsiklis. NP-hardness of deciding convexity of quartic polynomials and related problems. *Mathematical Programming*, 137:453–476, 2013.
5. M. Avriel, W. E. Diewert, S. Schaible, and I. Zang. *Generalized concavity.* Plenum Press, 1988.
6. X. Bao. Automatic convexity detection for global optimization. Master's thesis, Department of Chemical Engineering, University of Illinois at Urbana-Champaign, 2007.
7. X. Bao, A. Khajavirad, N. V. Sahinidis, and M. Tawarmalani. Global optimization of nonconvex problems with multilinear intermediates. *Mathematical Programming Computation, submitted*, 2012.
8. E. M. L. Beale and J. J. H. Forrest. Global optimization using special ordered sets. *Mathematical Programming*, 10:52–69, 1976.
9. E. M. L. Beale and J. A. Tomlin. Special facilities in a general mathematical programming system for nonconvex problems using ordered sets of variables. In J. Lawrence (ed.), *Proceedings of the Fifth International Conference on Operational Research,* Tavistock Publications, London, pages 447–454, 1970.
10. P. Belotti, J. Lee, L. Liberti, F. Margot, and A. Wächter. Branching and bounds tightening techniques for non-convex MINLP. *Optimization Methods and Software*, 24:597–634, 2009.
11. T. Berthold, G. Gamrath, G. Hendel, S. Heinz, T. Koch, M. Pfetsch, S. Vigerske, R. Waniek, M. Winkler, and K. Wolter. *SCIP 3.0.1,* User's Manual, 2012.
12. D. P. Bertsekas and H. Yu. A unifying polyhedral approximation framework for convex optimization. *SIAM Journal on Optimization*, 21:333–360, 2011.
13. S. Boyd and L. Vandenberghe. *Convex Optimization.* Cambridge University Press, 2004.
14. M. R. Bussieck, A. S. Drud, and A. Meeraus. MINLPLib–A collection of test models for mixed-integer nonlinear programming. *INFORMS Journal on Computing*, 15:114–119, 2003.

15. J. W. Chinneck. Analyzing mathematical programs using MProbe. *Annals of Operations Research*, 104:33–48, 2001.
16. E. Dolan and J. More. Benchmarking optimization software with performance profiles. *Mathematical programming*, 91:201–213, 2002.
17. F. Domes and A. Neumaier. Constraint propagation on quadratic constraints. *Constraints*, 15:404 – 429, 2010.
18. A. Drud. *CONOPT,* User's Manual. ARKI Consulting and Development A/S, Bagsvaerd, Denmark, 2011.
19. M. A. Duran and I. E. Grossmann. An outer-approximation algorithm for a class of mixed-integer nonlinear programs. *Mathematical Programming*, 36:307–339, 1986.
20. J. Forrest. *CLP* User Guide. Available at `http://www.coin-or.org/Clp/userguide/clpuserguide.html`.
21. R. Fourer, C. Maheshwari, A. Neumaier, D. Orban, and H. Schichl. Convexity and concavity detection in computational graphs: Tree walks for convexity assessment. *Informs journal on computing*, 22:26–43, 2010.
22. R. Fourer and D. Orban. DrAmpl: a meta solver for optimization problem analysis. *Computational Management Science*, 7:437–463, 2010.
23. P. E. Gill, W. Murray, and M. A. Saunders. User's Guide for SNOPT 7.2.4: A FORTRAN Package for Large-Scale Nonlinear Programming. Technical report, University of California, San Diego and Stanford University, CA, 2008.
24. GLOBAL Library. `http://www.gamsworld.org/global/globallib.htm`.
25. M. Grant and S. Boyd. CVX: Matlab software for disciplined convex programming, version 2.0 beta. `http://cvxr.com/cvx`, September 2012.
26. IBM. *CPLEX Optimizer*, 2011. `http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/`.
27. R. B. Kearfott. GlobSol user guide. *Optimization Methods and Software*, 24:687–708, 2009.
28. J. E. Kelley. The cutting plane method for solving convex programs. *Journal of the SIAM*, 8:703–712, 1960.
29. Y. Lin and L. Schrage. The global solver in the LINDO API. *Optimization Methods and Software*, 24:657–668, 2009.
30. S. Boyd M. Grant and Y. Ye. Disciplined convex programming. *Chapter in Global Optimization: From Theory to Implementation, L. Liberti and N. Maculan (eds.), Nonconvex Optimization and its Applications*, pages 155–210, 2006.
31. G. P. McCormick. Computability of global solutions to factorable nonconvex programs: Part I—Convex underestimating problems. *Mathematical Programming*, 10:147–175, 1976.
32. R. Misener and C. A. Floudas. GloMIQO: Global Mixed-Integer Quadratic Optimizer. *Journal of Global Optimization*, 2012. DOI: 10.1007/s10898-012-9874-7.
33. M Monnigmann. Efficient calculation of bounds on spectra of Hessian matrices. *SIAM Journal on Scientific Computing*, 30:2340–2357, 2008.
34. B. A. Murtagh and M. A. Saunders. MINOS 5.5 User's Guide. Technical Report SOL 83-20R, Systems Optimization Laboratory, Department of Operations Research, Stanford University, CA, 1995.
35. I. Nenov, D. Fylstra, and L. Kolev. Convexity determination in the microsoft excel solver using automatic differentiation techniques. *The 4th International Conference on Automatic Differentiation*, 2004.
36. A. Neumaier and O. Shcherbina. Safe bounds in linear and mixed-integer linear programming. *Mathematical Programming*, 99:283–296, 2004.
37. Princeton Library. `http://www.gamsworld.org/performance/princetonlib/princetonlib.htm`.
38. J. B. Rosen and P. M. Pardalos. Global minimization of large-scale constrained concave quadratic problems by separable programming. *Mathematical Programming*, 34:163–174, 1986.
39. H. S. Ryoo and N. V. Sahinidis. A branch-and-reduce approach to global optimization. *Journal of Global Optimization*, 8:107–139, 1996.
40. N. V. Sahinidis. BARON: A general purpose global optimization software package. *Journal of Global Optimization*, 8:201–205, 1996.
41. N. V. Sahinidis. *BARON 12.1.0: Global Optimization of Mixed-Integer Nonlinear Programs,* User's Manual, 2013.
42. M. Tawarmalani and N. V. Sahinidis. *Convexification and Global Optimization in Continuous and Mixed-Integer Nonlinear Programming: Theory, Algorithms, Software, and Applications*. Kluwer Academic Publishers, Dordrecht, 2002.

43. M. Tawarmalani and N. V. Sahinidis. Global optimization of mixed-integer nonlinear programs: A theoretical and computational study. *Mathematical Programming*, 99:563–591, 2004.
44. M. Tawarmalani and N. V. Sahinidis. A polyhedral branch-and-cut approach to global optimization. *Mathematical Programming*, 103:225–249, 2005.
45. A. Wächter and L. T. Biegler. On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106:25–57, 2006.
46. T. Westerlund and R. Porn. Solving pseudo-convex mixed integer optimization problems by cutting plane techniques. *Optimization and Engineering*, 3:253–280, 2002.