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

Automatic Reformulations for Convex Mixed-Integer Nonlinear Optimization: Perspective and Separability

Meenarli Sharma · Ashutosh Mahajan

Received: date / Accepted: date

Abstract Tight reformulations of combinatorial optimization problems like Convex Mixed-Integer Nonlinear Programs (MINLPs) enable one to solve these problems faster by obtaining tight bounds on optimal value. We consider two techniques for reformulation: perspective reformulation and separability detection. We develop routines for automatic detection of problem structures suitable for these reformulations, and implement new extensions. Since detecting all 'on-off' sets for perspective reformulation in a problem can be as hard as solving the original problem, we develop heuristic methods to automatically identify them. The LP/NLP branch-and-bound method is strengthened via 'perspective cuts' derived from these automatic routines. We also provide methods to generate tight perspective cuts at different nodes in the branch-and-bound tree. The second structure, i.e., separability of nonlinear functions, is detected by means of the computational graph of the function. Our routines have been implemented in the open-source Minotaur solver for general convex MINLPs. Computational results show an improvement of up to 45% in the solution time and the size of the branch-and-bound tree for convex instances from benchmark library MINLPLib. On instances where reformulation using function separability induces structures that are amenable to perspective reformulation, we observe an improvement of up to 88% in the solution time.

Keywords Convex MINLP, perspective reformulation, branch-and-bound, outer approximation, function separability

M. Sharma
Institute of Mathematics, University of Augsburg, Germany and A. Mahajan
Indian Institute of Technology Bombay, Mumbai, MH 400076, India
E-mail: meenarli.sharma@math.uni-augsburg.de, amahajan@iitb.ac.in

1 Introduction

We study convex MINLPs that are optimization problems of the form

$$\left.\begin{aligned} &\underset{x}{\text{minimize}} && c^T x \\ &\text{subject to} && g_i(x) \leq \ 0, \ i = 1, \ldots, m, \\ & && x_i \in \mathbb{Z}, \ i \in \mathcal{I}. \end{aligned}\right\} \tag{P}$$

Here, variables with indices in set $\mathcal{I}$ are restricted to take only integer values and constraint functions $g_i : \mathbb{R}^n \to \mathbb{R}, \ i = 1, \ldots, m$ are convex and twice continuously differentiable. Convex MINLPs arise in a several real-world applications and are also solved as subproblems in nonconvex MINLPs [19, 22], and mixed-integer PDE constrained optimization problems [25].

In a branch-and-bound framework for solving a convex MINLP (P), iteratively tightened relaxations of the problem are solved to obtain lower bounds on optimal objective value, Z^*. Since tight relaxations often give good bounds, one seeks to generate tight relaxations at different nodes in a tree-search. Reformulation of the problem is one of the ways to tighten relaxations. In this work, we consider two useful reformulations (i) Perspective Reformulation (PR) [9, 13] (ii) and a reformulation using the 'separability' property of functions in nonlinear constraints.

It is shown in [9, 13] that, for some special disjunctive sets ('on-off' sets) convex hull description can be given in the space of the original variables using the perspective function. Problems to which PR can be applied occur in many applications, and they are shown to be solved better in terms of solution time and branch-and-bound tree size by using PR [2, 8, 13]. Although PR is useful, a bottleneck in its implementation is detecting the on-off sets in a given problem. Moreover, the reformulation involves a nonlinear constraint that can cause numerical difficulty due to possible division by zero. Depending on how this nonlinear constraint is handled, there are different ways to solve the reformulated problem [10, 12, 13]. We first introduce structures (in the form of collections of constraints) that indicate disjunctions suitable for PR, and then provide computationally economical ways to automatically detect these structures in a problem. More specifically, we present some new structures that imply 'semi-continuous' variables, an important component in defining on-off sets. To obtain tight linear inequalities that outer-approximate the perspective reformulation, we propose novel line search approaches.

In the context of outer-approximation based algorithms, it is demonstrated in [17, 27] that, if a convex function is decomposed into its convex sub-expressions then outer-approximating these decomposed components separately gives better approximation of the original function than outer-approximating the original function directly. Exploiting separability in functions defining nonlinear convex constraints allows such a decomposition. The effectiveness of exploiting separability in specific models is further demonstrated in [14], where such algorithms are shown to require orders of magnitude fewer cuts to converge using separability. We implement routines to automatically detect separable functions using their 'computational graphs' that are available in some solvers to store nonlinear functions.

All presented methods have been implemented within the open-source solver Minotaur [1] [20]. Our computational experiments show the improved performance of Minotaur on convex instances in MINLPLib [5] achieved by these reformulation techniques. To the best of our knowledge, PR has only recently been implemented in SCIP [3] for both convex and nonconvex problems. Also, a reformulation based on separability exists in the convex MINLP solver SHOT [18].

The rest of the paper is organized as follows. In Section 2 and Section 3, we present the perspective reformulation and the separability based reformulation, respectively, and their impact on the performance of a branch-and-cut algorithm in Minotaur. The combined effect of these reformulations is reported in Section 4. Section 5 presents our conclusions.

2 Perspective Reformulation

A disjunctive set of the form (S)

$$\left\{(x,z) \in \mathbb{R}^n \times \{0,1\} \;\middle|\; \begin{array}{l} x \in \Gamma_0, \text{ if } z = 0 \\ x \in \Gamma_1, \text{ if } z = 1 \end{array}\right\}, \tag{S}$$

where Γ_0 is a singleton set and Γ_1 is a bounded convex set, is called an 'on-off' set. The roles of $z = 0$ and $z = 1$ in (S) can be swapped without losing generality. The binary variable z 'controls' variables x in the sense that when $z = 0$, x takes a fixed value $\hat{x} \in \mathbb{R}^n$, and $z = 1$ implies that x lies in a compact convex set. Such variables x are called semi-continuous variables [9,10] and they appear in many real-world applications [11,15,29].

The convex hull of an 'on-off' set can be represented using a function $\check{f}$ in the space of original variables [7]. Given a function $f(x) : \mathbb{R}^n \to \mathbb{R}$, $\check{f}(x,\lambda) : \mathbb{R}^{n+1} \to \mathbb{R}$ is defined as

$$\check{f}(x,\lambda) = \begin{cases} \lambda f\Big(\dfrac{x-(1-\lambda)\hat{x}}{\lambda}\Big), & \text{if } \lambda > 0, \\ 0, \ \text{ if } \lambda = 0, \\ \infty, \ \text{otherwise}, \end{cases} \tag{PF}$$

where $\hat{x}$ is some fixed vector. It can be easily shown that if f is convex, then $\check{f}$ is also convex. When $\hat{x} = 0$, the function $\check{f}$ is well known as the perspective function of f. Two sets that conform to the form (S) are as follows.

1. This set referred to as (S_1), equals $\Gamma_0 \cup \Gamma_1$ with

$$\left.\begin{array}{l} \Gamma_0 := \{(x,z) \in \mathbb{R}^p \times \{0,1\} : x = \hat{x},\ z = 0\} \\ \Gamma_1 := \{(x,z) \in \mathbb{R}^p \times \{0,1\} : g_i(x) \le 0,\ A(x,z) \le a, z = 1\} \end{array}\right\} \tag{S_1}$$

[1] Available at http://github.com/minotaur-solver/minotaur

2. This set, referred to as (S_2), is defined as $\Gamma_0 \cup \Gamma_1$, with

$$\left.\begin{aligned} \Gamma_0 &:= \{(x,v,z) \in \mathbb{R}^{p+q} \times \{0,1\} : x = \hat{x}, d^T v \le 0, z = 0\}. \\ \Gamma_1 &:= \{(x,v,z) \in \mathbb{R}^{p+q} \times \{0,1\} : g_i(x) + d^T v \le 0, A(x,z) \le a, z = 1\}. \end{aligned}\right\} \tag{S_2}$$

   Even though (S_2) is not an on-off set by definition, its convex hull can still be described by a perspective function like an on-off set in the space of original variables.

In both sets $\hat{x}$ is a fixed vector, g_i is a convex nonlinear function, and Γ_1 is a compact convex set. In the first set Γ_0 is a singleton and in the second it is a halfspace. The polyhedral set defined by $A(x,z) \le a$ is compact (A and a are a matrix and a vector of the corresponding dimension) and contains $(\hat{x}, 0)$, enforcing the on-off relation between x and z. The convex hulls ($conv(.)$) of these sets (Lemmas 1–2) can be shown to lie in the space of the original variables x and z [13].

Lemma 1 *$conv(S_1) = closure(\tilde{S}_1)$, where*

$$\tilde{S}_1 = \Big\{(x,z) \in \mathbb{R}^{p+1} : z g_i\Big(\frac{x-(1-z)\hat{x}}{z}\Big) \le 0, \qquad A(x-(1-z)\hat{x}) \le az,\ 0 < z \le 1\Big\}.$$

Lemma 2 *$conv(S_2) = closure(\tilde{S}_2)$, where*

$$\tilde{S}_2 = \Big\{(x,v,z) \in \mathbb{R}^{p+q+1} : z g_i\Big(\frac{x-(1-z)\hat{x}}{z}\Big) + d^T v \le 0, \qquad A(x-(1-z)\hat{x}) \le az,\ 0 < z \le 1\Big\}.$$

In sets (S_1) and (S_2), x are semi-continuous variables. Given a convex MINLP, to find if x_i is a semi-continuous variable controlled by z, where $z \in \{0,1\}$, two MINLPs have to be solved to check if x_i can be fixed to $\hat{x}_i$. In these two MINLPs, z is fixed to 0 and the objective functions are *max* x_i and *min* x_i, respectively (the rest remains same as the original problem). If the optimal value of these two MINLPs is equal to $\hat{x}_i$, then it implies $z = 0$ fixes x_i to $\hat{x}_i$. Since detecting semi-continuous variables in a given problem requires solving MINLPs (which can be as difficult as solving the original problem), there is a trade-off between the number of (S_1) and (S_2) sets detected and the time spent in detecting them. A less time-consuming alternative is to heuristically find collections of constraints during presolve, that indicate semi-continuous variables and binary variables controlling them. Some such collections ((C_1), (C_2), and (C_3)) are presented below, and our computational study show that these collections appear as small blocks in many optimization problems. The techniques to detect them are similar to probing for MILPs [24]. Henceforth, $\hat{x}$ indicates the fixed value that semi-continuous variables x takes when the corresponding binary variable has value 0.

1. Linear inequalities in at most two variables of the form,

$$\left.\begin{aligned} &l^1 z + l^0(1-z) \le x \le u^1 z + u^0(1-z), \\ &z \in \{0,1\},\ x \in \mathbb{R}, \end{aligned}\right\} \tag{C_1}$$

   where $l^0, l^1, u^0, u^1 \in \mathbb{R}$ and $l^j \le u^j, j = 0,1$. If $l^0 = u^0$, then x is a semi-continuous variable controlled by z. Similarly, if $l^1 = u^1$, then $(1-z)$ controls x. A simple example of (C_1) that appears in many problems is, $lz \le x \le uz, z \in \{0,1\},\ x \in \mathbb{R}$.
2. Single constraint with an indicator

$$\left.\begin{aligned} &a^T x + d_1 z \le d_2, \\ &l \le x \le u, \\ &z \in \{0,1\}, \end{aligned}\right\} \tag{C_2}$$

   where d_1 and d_2 are scalars, and $l, u \in \mathbb{R}^p$ with $l_i \le u_i, \forall i$, with the additional property that if $a_i > 0$, then $l_i = 0$, and if $a_i < 0$, then $u_i = 0$.
   (a) If $d_2 = 0$ and $d_1 < 0$, every component of x is semi-continuous variable controlled by z such that $\hat{x} = 0$. If $d_2 = 0$ and $d_1 > 0$, $z = 1$ is infeasible and therefore, z can be fixed to 0.
   (b) When $d_1 = d_2$, x is semi-continuous variable controlled by $1 - z$ and $\hat{x} = 0$. Additionally, if $d_1 < 0$, then $z = 0$ becomes infeasible and z can be fixed to 1. A simple example of this case is $\sum_{i=1}^{p} z_i \le 1, z \in \{0,1\}^p$ where any $(1 - z_i)$ controls all the other variables.
3. (C_1) or (C_2) with an extra equality constraint

$$\left.\begin{aligned} &d^T \overline{x} + d_3 \tilde{x} = d_4, \\ &\tilde{x} \in \mathbb{R},\ \overline{x} \in C_1 \text{ or } C_2, \end{aligned}\right\} \tag{C_3}$$

   where $d \in \mathbb{R}^p$, and d_3 and d_4 are any scalars. If $\overline{x}$ is controlled by z or $(1-z)$ ($\hat{\overline{x}}$ being the corresponding fixed value of $\overline{x}$), then so is $\tilde{x}$ if $\tilde{l} \le \dfrac{d_4 - d^T \hat{\overline{x}}}{d_3} \le \tilde{u}$, where $\tilde{l}$ and $\tilde{u}$ are lower and upper bounds respectively on $\tilde{x}$, otherwise, z can be fixed to 1 or 0, respectively.

Out of 374 convex MINLP instances in MINLPLib, 274 instances have at least one binary variable remaining after Minotaur's presolve routine. Table 9 in Appendix B reports the number of instances out of these 274 with above mentioned collections. Out of these 274 instance, 220 have at least one of these three collections, and set of these instances is referred to as TS_c, which is used for detecting sets amenable to PR. Details of instances in test set TS_c are presented in the Table 6 in Appendix A. Note that these are the instances that have semi-continuous variables, and some of them may not be suitable for the perspective reformulation.

2.1 Structures Amenable to Perspective Reformulation

Given the problem (P), following structures conform with sets of the form (S_1) or (S_2), and thus, are amenable to perspective reformulation.

1. A constraint in which all variables in the nonlinear function are semi-continuous, that is,

$$\left.\begin{array}{l} g_i(x) \leq 0, \\ (x,z) \in \bar{C}, \end{array}\right\} \qquad (PS_1)$$

   where $\bar{C}$ is a union of at least one of C_1 or C_2 or C_3. This structure conforms with (S_1).
2. A constraint in which all variables in only the nonlinear part of the function are semi-continuous. Let $\widetilde{g}_i$ and $\overline{g}_i$ denote nonlinear and linear parts of function g_i in disjoint set of variables $\tilde{x}$ and $\overline{x}$, respectively. If z exists in the constraint, it should be considered a part of $\tilde{g}_i$. This structure conforms with set (S_2).

$$\left.\begin{array}{l} \widetilde{g}_i(\tilde{x}) + \overline{g}_i(\overline{x}) \leq 0, \\ (\tilde{x},z) \in \bar{C}, \end{array}\right\} \qquad (PS_2)$$

The PR amenable structure specified in [3] is of form (PS_2), with semi-continuous variables recognized by the constraints (C_1). A reformulation of the problem that results by replacing structures (PS_1) and (PS_2) by their convex hull description (as mentioned in Lemma 1 and Lemma 2, respectively) is referred to as perspective reformulation of the problem.

2.2 Detecting Structures (PS_1) and (PS_2)

Given a problem (P), we have a straightforward two-phase algorithm for detecting nonlinear constraints amenable to PR. In the first phase, the algorithm iterates through linear inequalities to find blocks of constraints (C_1) and (C_2). Then it iterates through all the linear equalities to detect (C_3). The outcome of the first phase is either a set of semi-continuous variables (and binary variables controlling them) or an indication that there is none. If there are semi-continuous variables, then in the second phase, the algorithm iterates through nonlinear constraints and checks if it has form required for (PS_1) or (PS_2). In case a nonlinear constraint conforms to either of the forms, it is declared amenable to PR.

Our computational results show that 104 instances (all mixed-binary nonlinear programs) in the test set TS_c have structures amenable to PR. We refer to the set of these 104 instances as TS_{pr} and Table 7 in Appendix A reports more details of these instances. Out of these, 103 instances have all PR amenable constraints of type (PS_1), and instance `synthes3` has one constraint each of type (PS_1) and (PS_2). Moreover, we found that all instances (except `synthes2` and `synthes3`) in TS_{pr} have all nonlinear constraints amenable to PR.

As this algorithm iterates through linear constraints for finding semi-continuous variables, it might take more time on instances with a large number of linear constraints. In our experiments, the time taken to detect these structures (including detection of semi-continuous variables) in any instance in the set TS_{pr} is negligible (less than half a second).

2.3 Solving Perspective Reformulation

We solve the reformulated problem using perspective cuts in the LP/NLP based branch-and-bound method [23]. This method, also known as the QG algorithm, is based on a branch-and-cut framework and is a state-of-art method for convex MINLPs. It is implemented and practically enhanced in many MINLP solvers [1,4,16,21,26].

Perspective cuts (PCs) are outer-approximation cuts to constraints after PR is applied to them [13]. For the nonlinear constraint in structure (PS_1), the outer-approximation cut at (x', z') and $\left(\frac{x'}{z'}, 1\right)$ are the same and is given by

$$x^\top s + z\Big(g_i\Big(\frac{x'}{z'}\Big) + s\Big(\hat{x} - \frac{x'}{z'}\Big)\Big) \leq s^\top \hat{x},\;\; s \in \partial_x g_i\Big(\frac{x'}{z'}\Big). \tag{1}$$

Adding infinitely many PCs to the defining function in a PR amenable structure gives its convex hull. Also, note that all PCs pass through the point $(\hat{x}, 0)$. Similarly, a PC for a reformulated constraint in (PS_2), a perspective cut is given by

$$\tilde{x}^\top s + z\Big(\widetilde{g}_i\Big(\frac{\tilde{x}'}{z'}\Big) + \tilde{s}\Big(\hat{\tilde{x}} - \frac{\tilde{x}'}{z'}\Big)\Big) + \overline{g}_i(\overline{x}) \leq s^T \hat{\tilde{x}},\;\; s \in \partial_{\tilde{x}} \widetilde{g}_i\Big(\frac{x'}{z'}\Big). \tag{2}$$

In QG, cuts (gradient inequalities) are added at nodes where associated linear programs yield integer optimal solutions. Traditionally, the solution to the continuous relaxation of the root node, say (x^0, z^0), is used to create the initial linear relaxation by linearizing the nonlinear constraints at (x^0, z^0). Here, z^0 represents the vector of binary variables associated with semi-continuous variables appearing in the structures amenable to PR. These linearizations to the constraints active at (x^0, z^0) are supporting for P^c (the feasible region of the continuous relaxation of problem), but not necessarily for P^r (the feasible region of the continuous relaxation of the perspective reformulated problem). This scenario arises when some $z_i^0 \in (0, 1)$ satisfies the original nonlinear constraint but not the reformulated constraint. Moreover, this can also happen at other nodes in the branch-and-bound tree.

We found that in 68 instances in TS_{pr}, at least one reformulated nonlinear constraint is violated at (x^0, z^0) and 20 of them have more than 50% of the reformulated constraints violated. This observation motivated us to generate tight PCs for the reformulated problem at a point (x', z') that is not in P^r. We study the problem of generating perspective cuts at such a point (x', z') under the following two cases.

1. $(x', z') \in P^c$ and $(x', z') \notin P^r$: This happens when (x^0, z^0) does not lie in P^r.
2. $(x', z') \notin P^c$ and $(x', z') \notin P^r$: This can happen at nodes other than root node yielding fractional optimal solutions.

Given such a point $(x', z') \notin P^r$, we propose the following two methods that find another point (x'', z'') in P^r (or at least at the boundary of the violated constraint) such that the linearizations at (x'', z'') cut off (x', z').

1. SimLS Method: This is a simple line search that considers each violated constraint and search for a point that satisfies the reformulated constraint at equality. That is, given

$$z_i' g_i\Big(\frac{x' - (1 - z_i')\hat{x}}{z_i'}\Big) > 0,$$

this method finds a point (x'', z'') such that $z_i'' g_i\Big(\frac{x'' - (1 - z_i'')\hat{x}}{z_i''}\Big) = 0$. Given the point $(x', z') \in P^c$, if $(x', 1) \in P^r$, then (x'', z'') is such that $x'' = x'$ and $z_i'' = (1 - \lambda)z_i' + \lambda$ for some $\lambda \in (0, 1]$.
Also, if $(\hat{x}, 1) \in P^c$ (and thus, in P^r), then for every $(x', z') \in P^c$, $(x', 1) \in P^c$ (and thus, in P^r). Verifying $(\hat{x}, 1) \in P^c$ amounts to evaluating whether the nonlinear constraint satisfies at $(x', 1)$. Also, for the structure (PS_1), if the associated binary variable does not exist in the defining nonlinear constraint, then $(\hat{x}, 1) \in P^c$.
We found that in 98 instances in TS_{pr}, $(\hat{x}, 1)$ belongs to P^c for all the PR amenable constraints and in 50 of them, the binary variables controlling the semi-continuous variables do not appear in the constraint functions. The 6 instances in which this condition is not satisfied for any of the PR amenable constraints are of the type `clay*`.
2. CenLS Method: This method performs a line search between the given point and (x^C, z^C) (an approximation of the center of P^c) to obtain a point (x'', z'') at the boundary. The point (x^C, z^C) is obtained by solving the following nonlinear problem (NLPI), in which all the nonlinear inequalities in the original problem are modified using an auxiliary variable, ν, which also forms the objective of (NLPI). All the linear constraints remain unchanged. Let the optimal solution of (NLPI) be $(\tilde{\nu}, \tilde{x}, \tilde{z})$. If $\tilde{\nu} < 0$, then we set $(x^C, z^C) = (\tilde{x}, \tilde{z})$. If $\tilde{\nu} = 0$, then no point in the feasible region of the original problem exists at which all the nonlinear constraints are inactive. In this case, we terminate the method. If (NLPI) is unbounded, then we add ν to the linear inequalities in the same way as the nonlinear constraints and then re-solve.

$$\left.\begin{array}{ll}\underset{x,\nu}{\text{minimize}} & \nu \\ \text{subject to} & g_i(x) \leq \nu, \; i \in M, \\ & \nu \leq 0. \end{array}\right\} \qquad \text{(NLPI)}$$

If (x^C, z^C) is obtained, then it lies in P^r.

The following two sections present the computational experiments that compare the default implementation of QG in Minotaur, referred to as *qg*, to *qg* with PCs on overall solution time and size of the tree (in terms of the number of nodes processed). All the computational experiments have been carried out on a system with two 64-bit Intel(R) Xeon(R) E5-2670 v2, 2.50GHz CPUs having 10 cores each and sharing 128GB RAM. Our schemes are available in the development version of Minotaur[2]. All codes are complied with GCC-4.9.2 compiler. IPOPT-3.12 with MA27 linear-systems solver is used as the NLP solver. CPLEX-12.8 has been used as the LP solver. We have set a time limit of one hour for all our experiments and reported all the solution times in seconds.

2.3.1 Adding Perspective Cuts at Root Node

First we add PCs only at the root node with the following three settings.

1. *root_reg*: adds PCs to PR amenable constraints violated at (x^0, z^0).
2. *root_cenls*: adds PCs to violated PR amenable using CenLS method in addition to PCs from *root_reg*.
3. *root_bothls*: adds additional PCs using SimLS method, wherever applicable, in addition to PCs from *root_reg* and *root_cenls*.

In these experiments, we commonly add the following cuts:

1. PCs to PR constraints at corresponding points $(\hat{x}, 0)$, where $z = 0$ implies $x = \hat{x}$.
2. If the perspective reformulated constraint is inactive at (x^0, z^0) and $(x^0, 1)$ does not lie in P^r, then we find a point on the boundary by moving along the direction $-e_z$ (which is always feasible), a vector whose components associated with z are -1 and the rest are 0.

Table 1 and Table 2 show a comparison of default *qg* and *qg* with settings $s \in \{$*root_reg*, *root_cenls*, *root_bothls*$\}$ for instances in test set TS_{pr}. These results show the distribution of performance across instances with varying difficulty. Each row corresponds to an experimental setting (s). Each row in Table 1 (Top) corresponds to the results of instances solved by *qg* and *qg* with setting s, and in Table 1 (Bottom), Table 2, it corresponds to instances that are solved by both, but where at least one of the methods takes more than

[2] Available at http://github.com/minotaur-solver/minotaur

10, 100, and 500 seconds, respectively. The first column under the headings 'time' and 'nodes' shows the shifted geometric mean (SGM) of these measures reported by the reference solver (*qg* in this case) for the instances solved by both. The second column under these headings show the relative SGM ('rel.') under the setting s for the same instances. The relative SGM of a measure is computed as the ratio of the SGM value of the proposed scheme (here, *qg* under setting s) to the SGM value of the reference solver (*qg*). If this ratio, say r, is less than one, it implies that the proposed solver has performed better than the reference solver. More specifically, the proposed solver has shown an improvement over the reference solver with a factor $(1 - r)$ on the considered performance measure. One instance (`rsyn0830m04m`) on which *qg* reached the time limit took 56.22s with the setting *root_reg*, 53.93s with *root_cenls*, and 47.17s with *root_bothls*.

Table 1 (Top) Comparison of *qg* and *qg* with setting s on 103 instances that are solved by both the methods. (Bottom) Performance on 26 instances for *qg* and *qg* with first two settings, and 25 instances with *root_bothls* that are solved by both methods, but at least one method took more than 10 seconds.

	time		nodes	
setting (s)	*qg*	rel.	*qg*	rel.
root_reg	8.60	0.67	505.44	0.69
root_cenls	8.60	0.63	505.44	0.69
root_bothls	8.60	0.57	505.44	0.59

	time		nodes	
setting (s)	*qg*	rel.	*qg*	rel.
root_reg	67.23	0.48	14956.08	0.41
root_cenls	67.23	0.43	14956.08	0.41
root_bothls	72.39	0.34	17005.06	0.29

Table 2 (Top) Comparison of *qg* and *qg* with setting s on 9 instances that are solved by both the methods but at least one method took more than 100 seconds. (Bottom) Similar comparison on 2 instances that are solved by both the methods, but at least one method took more than 500 seconds.

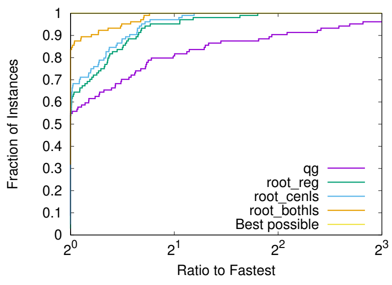
time		nodes	
qg	rel.	*qg*	rel.
294.93	0.28	55751.27	0.24
294.93	0.24	55751.27	0.26
294.93	0.19	55751.27	0.19

time		nodes	
qg	rel.	*qg*	rel.
1249.47	0.10	595448.0	0.09
1249.47	0.10	595448.0	0.09
1249.47	0.05	595448.0	0.04

Our computational results show improvements in both the considered measures under all three settings. The highest improvement is reported by *qg* with *root_bothls*. Overall, it improved the solution time and tree size by about 43.19% and 41.45%, respectively. Even higher improvements (about 81% and 95%, respectively) are observed for instances in with default *qg* took more than 100 seconds and 500 seconds, respectively.

Furthermore, we use performance profiles [6] that graphically demonstrate the relative performance of different solvers for a particular performance measure over a given set of instances.

Figure 1 shows the performance profiles of *qg* and *qg* with settings *root_reg*, *root_cenls*, *root_bothls* using the solution times of the instances in test set TS_{pr}. It shows that on nearly 85% of these instances, *root_cenls* is not slower than the rest and it solved all instances within 2 times of the best solvers



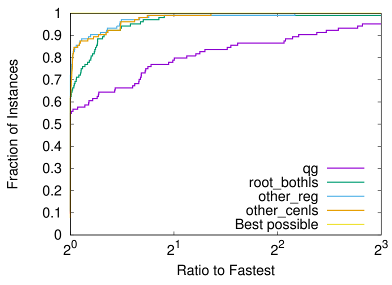


Fig. 1 Performance profiles comparing solution times of *qg* and *qg* with *root_reg*, *root_cenls*, *root_bothls* (on left), and *qg* and *qg* with *root_bothls*, *other_reg*, *other_cenls* (on right).

among the considered ones. On the other hand, on 20% of the instances *qg* took more than double the time taken by the fastest method.

2.3.2 Adding Perspective Cuts at Other Nodes

Next, we generate PCs at other nodes yielding integer feasible solutions in addition to the root node. The nodes that we have selected for generating perspective cuts are the same as in default *qg* (the ones yielding integer optimal LP solution). But using the fixed-NLP solution as in QG algorithm may not produce tight inequalities for the reformulated problem for the same reason as mentioned for the case of the root node. Here, we employ CenLS method for finding points for generating tighter perspective cuts. Let (x', z') be an integer solution of the fixed-NLP at any node. If the fixed-NLP is infeasible, (x', z') is a solution to the feasibility problem. When the fixed-NLP is optimal, a reformulated constraint is always feasible. However, some constraints could be inactive. In the test set TS_{pr}, in 6 instances, at least one reformulated nonlinear constraint is violated at some node yielding an integer optimal solution. In 10 instances, at least one reformulated constraint is inactive at the fixed-NLP optimal solution. Thus, keeping the best setting at the root node, following computational experiments have been performed for adding PCs at integer optimal nodes.

– *other_reg*: This setting generates perspective cuts to every nonlinear constraint in the reformulated problem at (x', z') if $z' \neq 0$.
– *other_cenls*: This setting employs CenLS method for generating perspective cuts.

We have experimented *other_cenls* method with and without adding constraints for inactive perspective amenable constraints in the same manner as in the root node. We found better results by not adding additional constraints for inactive constraints and thus report the same. Table 3 and Table 4 summarize the results of *qg* with these settings in comparison to the default *qg* on instances in the test set TS_{pr}.

Our computational results show improvements in both the considered measures under all the three settings. The highest improvement is reported by *qg*

Table 3 (Top) Comparison of *qg* and *qg* with setting *s* on 103 instances that are solved by both the methods. (Bottom) Performance on instances (25 for *qg* and *qg* with first two settings, and 26 with *other_cenls*) that are solved by both but at least one method took more than 10 seconds.

	time		nodes	
setting (s)	*qg*	rel.	*qg*	rel.
root_bothls	8.60	0.57	505.44	0.59
other_reg	8.60	0.53	505.44	0.56
other_cenls	8.60	0.53	505.44	0.55

	time		nodes	
setting (s)	*qg*	rel.	*qg*	rel.
root_bothls	72.39	0.34	17005.06	0.29
other_reg	72.39	0.30	17005.06	0.26
other_cenls	67.61	0.31	15645.84	0.27

Table 4 (Top) Comparison of *qg* and *qg* with setting *s* on 9 instances that are solved by both the methods but at least one method took more than 100 seconds. (Bottom) Similar comparison on 2 instances that are solved by both the methods, but at least one method took more than 500 seconds.

time		nodes	
qg	rel.	*qg*	rel.
294.93	0.19	55751.27	0.19
294.93	0.18	55751.27	0.17
294.93	0.19	55751.27	0.17

time		nodes	
qg	rel.	*qg*	rel.
1249.47	0.05	595448.0	0.04
1249.47	0.06	595448.0	0.05
1249.47	0.06	595448.0	0.05

with *other_cenls*. Overall, it improved the solution time and the tree size by about 47.40% and 44.62%, respectively, but even higher improvements (around 82% and 95% for both the measures) are observed for instances in with default *qg* took more than 100 seconds and 500 seconds, respectively. Figure 1 shows the performance profiles of *qg* and *qg* with settings *root_bothls*, *other_reg*, *other_cenls* using the solution times of the instances in test set TS_{pr}.

One can also add the perspective cuts at the fractional nodes in the tree. However, since we compare with traditional QG, we limit our PR related computational experiments to the root node and the nodes yielding integer optimal solutions only.

3 Reformulation Based on Function Separability

A function $f : \mathbb{R}^n \to \mathbb{R}$ is called 'group separable' or separable if there exist functions $f^i : \mathbb{R}^{n_i} \to \mathbb{R},\ i = 1, \dots, m$, such that

$$f(x) = \sum_{i=1}^{m} f^i(x^i), \tag{3}$$

where f^i and f^j for $i \neq j$ have no variables in common [28]. That is, f can be written as a sum of functions with a disjoint set of variables. A function is said to be fully separable if every f^i is a univariate function and partially separable if some f^i are not univariate functions.

Proposition 1 *If f as defined in (3) is convex, then every $f^i(x^i), i = 1, \dots, m$ is also convex.*

Given a nonlinear separable constraint of the form $\sum_{i=1}^{m} f^i(x^i) \leq b$, where $b \in \mathbb{R}$ is a scalar, using function separability it can be reformulated as

$$\left.\begin{array}{l} \sum_{i=1}^{m} \tilde{\gamma}_i \leq b, \\ f^i(x^i) \leq \tilde{\gamma}_i,\ i = 1, \ldots, m, \\ \tilde{\gamma}_i \in \mathbb{R}, i = 1, \ldots, m. \end{array}\right\} \tag{SepCon}$$

Proposition 1 ensures that the reformulated problem is also a convex MINLP.

To detect separability of a nonlinear function, we use its computational graph (CG) that represents the function as a directed acyclic graph for computational purposes. A CG of a nonlinear function f is constructed as a combination of unary, binary, or other operations carried out on the input variables, constants, and intermediate variables, which themselves are created using these operations.

A node in a CG represents either a variable, a constant, or an operation. An edge e_{ij} from node i to node j implies that i is a parent of j, or j is an operand of operation represented by i. A node with no child is called an independent (or leaf) node and it represents either a constant or a variable. Other nodes are called dependent nodes. A node that represents a binary operation has two child nodes. A node representing a unary operation has only one child. Let E_i^o and E_i^t denote the sets of edges originating from a node i and terminating at node i, respectively. Node i with $E_i^t = \emptyset$ is called the root node and is unique in a CG. If $E_i^o = \emptyset$ then i is a leaf node. For an edge e_{ij}, let N_{ij}^o and N_{ij}^t represent the origin and terminal nodes, respectively. In our implementation, a node representing a constant can have only one parent.

Computational Subgraph

We use a notion of computational subgraph (CSG) in finding separable parts in the CG of a nonlinear function. Let $G_f(C, E)$ be a CG of a nonlinear function f where C and E refer to the sets of nodes and edges in the CG, respectively. A graph $G_f^s(V, F)$ is called a subgraph of $G_f(C, E)$ if the following conditions hold.

1. $V \subseteq C$ and $F \subseteq E$.
2. For each $v \in V$, $E_v^o \in F$, and for each $e_{ij} \in F$, $N_{ij}^t \in V$ and $N_{ij}^o \in V$.
3. A node with no parent node in V should not represent operations $+$, $-$, or unary minus.
4. $G_f^s(V, F)$ is connected.

A subgraph can have more than one node with no parent. Every CG is also its CSG. We define 'maximal subgraph' as a CSG that is not a part of any CSG other than the original CG. Figure 2 shows the CG of a separable function and its maximal subgraphs. Let f be a separable function (which cannot be further simplified symbolically) and let G_f be its CG.

Proposition 2 *The number of maximal subgraphs in G_f is equal to the number of separable parts f and vice-versa.*

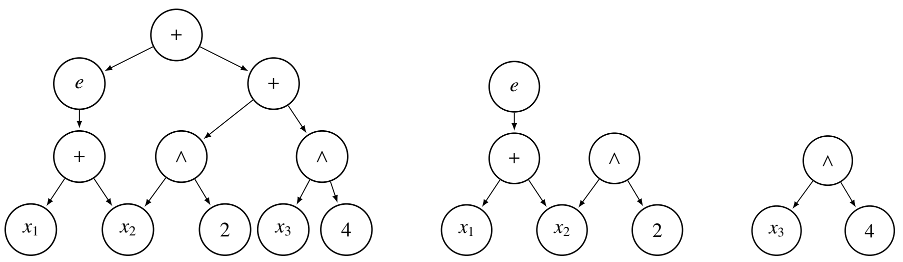


Fig. 2 Computational graph of $f = e^{x_1+x_2}+{x_2}^2+{x_3}^4$ (left) and its two maximal subgraphs.

Proposition 3 *f is not separable if and only if G_f has only one maximal subgraph.*

3.1 Detection of Function Separability

Given a nonlinear convex function $f : \mathbb{R}^n \to \mathbb{R}$ and its CG G_f, checking whether f is not separable is easier than checking otherwise. We start with employing simple rules to check if a function is not separable. If a function does not follow these rules, we use more extensive checking for separability.

Let r be the root node of G_f. Function f is not considered separable if any of the following conditions is met:

1. r represents a unary operation like log, exp, sin, cos, $\sqrt{(.)}$, etc., other than unary minus.
2. r represents the binary operation $\times$ and both its children represent a nonconstant expression.
3. r represents the binary operation $\div$ and its right child (divisor) represents a nonconstant expression.

If r represents the operation $\times$ with any of its children representing a constant, we compute the tree rooted at the nonconstant node and check again. Similarly, if r represents $\div$ with the right child representing a constant, we further analyze the graph rooted at its left child.

If a function does not satisfy the above conditions, G_f is iteratively searched for its maximal subgraphs. If there is only one maximal subgraph, the function is not separable. Otherwise, it is separable into as many parts as the number of maximal subgraphs.

3.2 Some Implementation Details

As our algorithm for detecting separability relies on the CG of the function, we prefer to express the function as explicitly as possible. Different separable expressions (in an explicit form) that can be identified by our algorithms include (1) $a \times (\sum_{i=1}^{m} f^i(x^i))$ (2) $\sum_{i=1}^{m} a_i \times f^i(x^i)$ (3) $\sum_{i=1}^{m} \dfrac{f^i(x^i)}{a_i}$, where

$a,\ a_i,\ \ b \in \mathbb{R}, \forall i = 1, \dots, m$. However, if an expression can be simplified symbolically, e.g., $\sqrt{(x_1^2 + x_2^2)^2}$, then the proposed algorithm cannot recognize it as separable, even if it is technically a separable function.

In some instances of MINLPLib, we found that different separable constraints have common separable parts (f^i). Our implementation reuses variables corresponding to different separable parts in different constraint expressions, thus avoids creating an extra variable and an additional constraint. For example, the set of constraints of the form

$$a_1 f^1(x^1) + a_2 f^2(x^2) \le b_1, \quad d_1 f^1(x^1) + d_2 f^3(x^3) \le b_2,$$

is reformulated as

$$\begin{aligned} &a_1\gamma_1 + a_2\gamma_2 \le b_1, \quad d_1\gamma_1 + d_2\gamma_3 \le b_2, \\ &f^1(x^1) \le \gamma_1, \quad f^2(x^2) \le \gamma_2, \quad f^3(x^3) \le \gamma_3. \end{aligned}$$

In Minotaur, we carry out separability detection before the presolving step. We have found that 126 instances have at least one separable nonlinear function (either in constraint or objective) out of 374 convex instances in MINLPLib. Out of 126 such instances, 79 have separability only in the nonlinear objective function. In 45 of the remaining 47 instances, all the nonlinear constraints are separable, and 2 have 40% of the nonlinear constraints with separability property. Also, 108 out of these 126 instances have at least one integer constrained variable. These 108 instances constitute our test set, TS_{sep} for analyzing the impact of exploiting separability in QG. More details on these instances are provided in the Table 8 in Appendix A. Using the same performance measures as before, Table 5 reports a comparison of default *qg* and *qg* with separability based reformulation (denoted *qgsep*) on instances in test set TS_{sep}. The time required by the proposed routine to detect function separability is a very small fraction of the total solution time in all instances of the test set. Overall, we achieve about 40% improvement in the solution time

Table 5 Comparison of *qg* and *qgsep* on instances in TS_{sep}. The second column indicates the number of instances solved by both methods, where at least one method took more than the number of seconds indicated in the first column.

		time		nodes	
time	# of inst.	*qg*	rel.	*qg*	rel.
>= 0	76	13.68	0.60	700.04	0.50
>= 10	26	94.54	0.42	5902.23	0.32
>= 100	11	714.61	0.20	24311.92	0.28
>= 500	7	1900.22	0.12	54480.5	0.26

and the tree size; even better improvements are obtained for difficult instances. Using this reformulation, 8 instances that reached the time limit earlier with *qg* could be solved. Figure 3 in Appendix B shows the performance profiles of *qg* and *qgsep* using the solution times of the instances in test set TS_{sep}.

4 Combined Effects of the Two Reformulations

Reformulation using separability sometimes results in structures amenable to PR. For example, consider the following uncapacitated facility location problem.

$$\left.\begin{array}{ll}\underset{x,\ z,\ \eta}{\text{minimize}} & \eta \\ \text{subject to} & \sum_{i\in\mathcal{F}} c_i z_i + \sum_{i\in\mathcal{F}, j\in\mathcal{C}} t_{ij}x_{ij}^2 \leq \eta, \\ & 0 \leq x_{ij} \leq z_i,\ i \in \mathcal{F},\ j \in \mathcal{C}, \\ & \sum_{i\in\mathcal{F}} x_{ij} = 1,\ j \in \mathcal{C}, \\ & x_{ij} \geq 0, z_i \in \{0,1\},\ i \in \mathcal{F},\ j \in \mathcal{C}. \end{array}\right\} \qquad (UFL_1)$$

The function in the nonlinear constraint is separable and on reformulating (UFL_1), we get

$$\left.\begin{array}{ll}\underset{x,\ z,\ \eta}{\text{minimize}} & \eta \\ \text{subject to} & \sum_{i\in\mathcal{F}} c_i z_i + \sum_{i\in\mathcal{F}, j\in\mathcal{C}} t_{ij}\gamma_{ij} \leq \eta, \\ & x_{ij}^2 \leq \gamma_{ij}, i \in \mathcal{F},\ j \in \mathcal{C}, \\ & 0 \leq x_{ij} \leq z_i,\ i \in \mathcal{F},\ j \in \mathcal{C}, \\ & \sum_{i\in\mathcal{F}} x_{ij} = 1,\ j \in \mathcal{C}, \\ & x_{ij} \geq 0, z_i \in \{0,1\},\ i \in \mathcal{F},\ j \in \mathcal{C}. \end{array}\right\} \qquad (UFL_2)$$

(UFL_2) now has structures of the form (PS_2) and thus, becomes amenable to PR.

Our results show that 26 instance in TS_{sep} become amenable to PR after separability based reformulation. These instances comprise test set TS_{ps} and are reported in Table 8 in Appendix A. Results on TS_{ps} using *qg*, *qgsep*, and *qgprsep* (*qg* with both separability and perspective reformulations) are reported in Table 10 and Table 11, respectively, in Appendix B. Overall, there is a significant improvement of about 88% in both the solution time and the tree size, and 4 more instances that reached the time limit with even separability based reformulation could now be solved. Figure 3 in Appendix B shows the performance profiles of *qg*, *qgsep*, and *qgprsep* using the solution times of the instances in test set TS_{ps}.

5 Conclusions

Our study concludes that perspective reformulation and exploitation of separability of nonlinear constraint functions help generate better polyhedral-approximations of the feasible region. This is observed even when these reformulations are deployed using automatic routines that detect corresponding structures heuristically. We see improvement in both the solution time and the size of the tree in the branch-and-cut framework of the QG method on our test instances. The improvements are even higher for difficult instances and for those that became amenable to PR after separability based reformulation. We believe that such automatic routines can also reduce the effort required to model convex MINLPs.

References

1. K. Abhishek, S. Leyffer, and J. T. Linderoth. FilMINT: An outer-approximation-based solver for nonlinear mixed integer programs. Preprint ANL/MCS-P1374-0906, Mathematics and Computer Science Division, Argonne National Laboratory, 2006.
2. S. Aktürk, A. Atamtürk, and S. Gürel. A strong conic quadratic reformulation for machine-job assignment with controllable processing times. *Operations Research Letters*, 37:187–191, 2009.
3. Ksenia Bestuzheva, Ambros Gleixner, and Stefan Vigerske. A computational study of perspective cuts. *arXiv preprint arXiv:2103.09573*, 2021.
4. Pierre Bonami and Jon Lee. BONMIN user's manual. *Numer Math*, 4:1–32, 2007.
5. Michael R Bussieck, Arne Stolbjerg Drud, and Alexander Meeraus. MINLPLib - a collection of test models for mixed-integer nonlinear programming. *INFORMS Journal on Computing*, 15(1):114–119, 2003.
6. Elizabeth Dolan and Jorge Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91:201–213, 2002.
7. A. Frangioni and C. Gentile. Perspective cuts for a class of convex 0-1 mixed integer programs. *Mathematical Programming*, 106:225–236, 2006.
8. Antonio Frangioni, Fabio Furini, and Claudio Gentile. Approximated perspective relaxations: a project and lift approach. *Computational Optimization and Applications*, 63(3):705–735, 2016.
9. Antonio Frangioni and Claudio Gentile. Perspective cuts for a class of convex 0–1 mixed integer programs. *Mathematical Programming*, 106(2):225–236, 2006.
10. Antonio Frangioni and Claudio Gentile. A computational comparison of reformulations of the perspective relaxation: SOCP vs. cutting planes. *Operations Research Letters*, 37(3):206–210, 2009.
11. Antonio Frangioni, Claudio Gentile, and Fabrizio Lacalandra. Solving unit commitment problems with general ramp constraints. *International Journal of Electrical Power & Energy Systems*, 30(5):316–326, 2008.
12. Kevin C Furman, Nicolas W Sawaya, and Ignacio E Grossmann. A computationally useful algebraic representation of nonlinear disjunctive convex sets using the perspective function. *Computational Optimization and Applications*, pages 1–26, 2020.
13. Oktay Günlük and Jeff Linderoth. Perspective reformulations of mixed integer nonlinear programs with indicator variables. *Mathematical programming*, 124(1-2):183–205, 2010.
14. Hassan Hijazi, Pierre Bonami, and Adam Ouorou. An outer-inner approximation for separable mixed-integer nonlinear programs. *INFORMS Journal on Computing*, 26(1):31–44, 2014.
15. Norbert J Jobst, Michael D Horniman, Cormac A Lucas, Gautam Mitra, et al. Computational aspects of alternative portfolio selection models in the presence of discrete asset choice constraints. *Quantitative finance*, 1(5):489–501, 2001.
16. KNITRO. *KNITRO Documentation*. Ziena Optimization., dec. 2012.
17. Jan Kronqvist, Andreas Lundell, and Tapio Westerlund. Reformulations for utilizing separability when solving convex MINLP problems. *Journal of Global Optimization*, 71(3):571–592, 2018.
18. Andreas Lundell, Jan Kronqvist, and Tapio Westerlund. The supporting hyperplane optimization toolkit for convex minlp. *Journal of Global Optimization*, pages 1–41, 2022.
19. Andreas Lundell and Tapio Westerlund. Solving global optimization problems using reformulations and signomial transformations. *Computers & Chemical Engineering*, 116:122–134, 2018.

20. Ashutosh Mahajan, Sven Leyffer, Jeff Linderoth, James Luedtke, and Todd Munson. Minotaur: A mixed-integer nonlinear optimization toolkit. *Mathematical Programming Computation*, pages 1–38, 2020.
21. Wendel Melo, Marcia Fampa, and Fernanda Raupp. An overview of minlp algorithms and their implementation in muriqui optimizer. *Annals of Operations Research*, 286(1):217–241, 2020.
22. Ivo Nowak, Norman Breitfeld, Eligius MT Hendrix, and Grégoire Njacheun-Njanzoua. Decomposition-based inner-and outer-refinement algorithms for global optimization. *Journal of Global Optimization*, 72(2):305–321, 2018.
23. Ignacio Quesada and Ignacio E Grossmann. An LP/NLP based branch and bound algorithm for convex MINLP optimization problems. *Computers & chemical engineering*, 16(10-11):937–947, 1992.
24. M. W. P. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal on Computing*, 6:445–454, 1994.
25. Meenarli Sharma, Mirko Hahn, Sven Leyffer, Lars Ruthotto, and Bart van Bloemen Waanders. Inversion of convection–diffusion equation with discrete sources. *Optimization and Engineering*, pages 1–39, 2020.
26. Meenarli Sharma, Prashant Palkar, and Ashutosh Mahajan. Linearization and parallelization schemes for convex mixed-integer nonlinear optimization. *Computational Optimization and Applications*, pages 1–56, 2022.
27. Mohit Tawarmalani and Nikolaos V Sahinidis. A polyhedral branch-and-cut approach to global optimization. *Mathematical Programming*, 103(2):225–249, 2005.
28. Stephen J Wright, Robert D Nowak, and Mário AT Figueiredo. Sparse reconstruction by separable approximation. *IEEE Transactions on Signal Processing*, 57(7):2479–2493, 2009.
29. Juan M Zamora and Ignacio E Grossmann. A global MINLP optimization algorithm for the synthesis of heat exchanger networks with no stream splits. *Computers & Chemical Engineering*, 22(3):367–384, 1998.

A Description of Test Sets

Table 6 Description of instances with collections (C_1), (C_2), and (C_3). First column shows instance name and the entries $(bv, tv, fv, b0, b1, b01, v0, v1, v01)$ in the second column are: bv denotes the number of binary variables, tv indicates the total number of variables, fv reports the number of binary variables that are fixed as part of structure identification, $b0$ and $b1$ represent the number of binary variables z and $1 - z$, respectively, controlling at least one other variable, $b01$ denotes number of binary variables z such that both z and $1 - z$ control another variable, $v0$ and $v1$ report the number of variables controlled by a binary variable z and $1 - z$ respectively, $v01$ is the number of variables controlled by a binary variable z and also $1 - z$.

Instance	$(bv, tv, fv, b0, b1, b01, v0, v1, v01)$
alan	(4, 4, 0, 4, 0, 0, 4, 0, 0)
batch0812	(60, 60, 28, 0, 32, 0, 0, 70, 0)
batchdes	(9, 9, 1, 0, 8, 0, 0, 12, 0)
batch	(24, 24, 2, 0, 22, 0, 0, 30, 0)
batchs101006m	(120, 120, 0, 0, 120, 0, 0, 140, 0)
batchs121208m	(191, 191, 0, 0, 191, 0, 0, 215, 0)
batchs151208m	(188, 188, 0, 0, 188, 0, 0, 212, 0)
batchs201210m	(225, 225, 0, 0, 225, 0, 0, 249, 0)
clay0203h	(18, 18, 0, 0, 0, 18, 60, 12, 6)
clay0203m	(18, 18, 0, 0, 12, 6, 0, 12, 6)
clay0204h	(32, 32, 0, 0, 0, 32, 112, 24, 8)
clay0204m	(32, 32, 0, 0, 24, 8, 0, 24, 8)
clay0205h	(50, 50, 0, 0, 0, 50, 180, 40, 10)
clay0205m	(50, 50, 0, 0, 40, 10, 0, 40, 10)
clay0303h	(21, 21, 0, 0, 0, 21, 66, 21, 0)
clay0303m	(21, 21, 0, 0, 21, 0, 0, 21, 0)
clay0304h	(36, 36, 0, 0, 0, 36, 120, 36, 0)
clay0304m	(36, 36, 0, 0, 36, 0, 0, 36, 0)
clay0305h	(55, 55, 0, 0, 0, 55, 190, 55, 0)
clay0305m	(55, 55, 0, 0, 55, 0, 0, 55, 0)
color_lab2_4x0	(28920, 28920, 0, 0, 28680, 240, 28680, 240, 0)
color_lab6b_4x20	(27730, 27730, 0, 0, 27495, 235, 27495, 235, 0)
enpro48pb	(92, 92, 0, 0, 92, 0, 0, 108, 0)
enpro56pb	(73, 73, 0, 0, 73, 0, 0, 85, 0)
fac1	(6, 6, 0, 2, 0, 4, 16, 0, 4)
fac2	(12, 12, 0, 3, 0, 9, 54, 0, 9)
fac3	(12, 12, 0, 3, 0, 9, 54, 0, 9)
flay02h	(4, 4, 0, 0, 0, 4, 32, 4, 0)
flay02m	(4, 4, 0, 0, 4, 0, 0, 4, 0)
flay03h	(12, 12, 0, 0, 0, 12, 96, 12, 0)
flay03m	(12, 12, 0, 0, 12, 0, 0, 12, 0)
flay04h	(24, 24, 0, 0, 0, 24, 192, 24, 0)
flay04m	(24, 24, 0, 0, 24, 0, 0, 24, 0)
flay05h	(40, 40, 0, 0, 0, 40, 320, 40, 0)
flay05m	(40, 40, 0, 0, 40, 0, 0, 40, 0)
flay06h	(60, 60, 0, 0, 0, 60, 480, 60, 0)
flay06m	(60, 60, 0, 0, 60, 0, 0, 60, 0)
gams01	(110, 110, 0, 0, 100, 0, 0, 100, 0)
hybriddynamic_fixed	(1, 1, 0, 0, 0, 1, 8, 0, 2)
ibs2	(1500, 1500, 0, 0, 0, 1500, 1500, 1500, 0)
meanvarx	(12, 12, 0, 2, 0, 10, 12, 10, 0)
meanvarxsc	(22, 22, 0, 12, 0, 10, 12, 10, 0)
netmod_dol1	(462, 462, 0, 0, 0, 462, 1524, 462, 0)
netmod_dol2	(455, 455, 0, 0, 79, 367, 973, 440, 6)
netmod_kar1	(136, 136, 0, 0, 15, 121, 255, 136, 0)
netmod_kar2	(136, 136, 0, 0, 15, 121, 255, 136, 0)
pedigree_ex1058	(49386, 49386, 0, 112, 48387, 865, 48387, 112, 865)
pedigree_ex485_2	(7136, 7136, 0, 110, 6710, 294, 6710, 110, 294)
pedigree_ex485	(7136, 7136, 0, 110, 6710, 294, 6710, 110, 294)
pedigree_sim400	(11226, 11226, 0, 51, 11076, 99, 11076, 51, 99)
pedigree_sp_top4_250	(11694, 11694, 0, 243, 10981, 414, 10981, 243, 414)
pedigree_sp_top4_300	(5969, 5969, 0, 160, 5496, 244, 5496, 160, 244)
pedigree_sp_top4_350tr	(3100, 3100, 0, 105, 2838, 145, 2838, 105, 145)
pedigree_sp_top5_200	(32120, 32120, 0, 336, 30862, 871, 30862, 336, 871)
pedigree_sp_top5_250	(17028, 17028, 0, 243, 16193, 536, 16193, 243, 536)
portfol_buyin	(8, 8, 0, 8, 0, 0, 8, 0, 0)
portfol_card	(8, 8, 0, 8, 0, 0, 8, 0, 0)
portfol_classical050_1	(50, 50, 0, 50, 0, 0, 50, 0, 0)
procurement2mot	(60, 60, 0, 19, 3, 38, 77, 18, 23)
ravempb	(53, 53, 0, 0, 53, 0, 0, 65, 0)
risk2bpb	(12, 12, 0, 0, 12, 0, 0, 183, 0)
rsyn0805h	(37, 37, 0, 3, 0, 34, 84, 32, 26)
rsyn0805m02h	(148, 148, 0, 3, 0, 145, 171, 37, 166)
rsyn0805m02m	(148, 148, 0, 3, 64, 81, 19, 53, 118)
rsyn0805m03h	(222, 222, 0, 3, 0, 219, 255, 42, 264)
rsyn0805m03m	(222, 222, 0, 3, 96, 123, 27, 66, 192)
procurement2mot	(60, 60, 0, 19, 3, 38, 77, 18, 23)
ravempb	(53, 53, 0, 0, 53, 0, 0, 65, 0)
risk2bpb	(12, 12, 0, 0, 12, 0, 0, 183, 0)
rsyn0805h	(37, 37, 0, 3, 0, 34, 84, 32, 26)
rsyn0805m02h	(148, 148, 0, 3, 0, 145, 171, 37, 166)
rsyn0805m02m	(148, 148, 0, 3, 64, 81, 19, 53, 118)
rsyn0805m03h	(222, 222, 0, 3, 0, 219, 255, 42, 264)
rsyn0805m03m	(222, 222, 0, 3, 96, 123, 27, 66, 192)
rsyn0805m04h	(296, 296, 0, 3, 0, 293, 339, 47, 362)
rsyn0805m04m	(296, 296, 0, 3, 128, 165, 35, 79, 266)
rsyn0805m	(37, 37, 0, 3, 32, 2, 8, 32, 2)
rsyn0810h	(41, 41, 0, 3, 0, 38, 95, 34, 26)
rsyn0810m02h	(166, 166, 0, 3, 0, 163, 187, 47, 182)
rsyn0810m02m	(166, 166, 0, 3, 64, 99, 35, 63, 134)
rsyn0810m03h	(249, 249, 0, 3, 0, 246, 278, 57, 289)
rsyn0810m03m	(249, 249, 0, 3, 96, 150, 50, 81, 217)
rsyn0810m04h	(332, 332, 0, 3, 0, 329, 369, 67, 396)
rsyn0810m04m	(332, 332, 0, 3, 128, 201, 65, 99, 300)
rsyn0810m	(41, 41, 0, 3, 32, 6, 19, 34, 2)
rsyn0815h	(44, 44, 0, 3, 0, 41, 105, 35, 27)
rsyn0815m02h	(182, 182, 0, 3, 0, 179, 204, 57, 197)
rsyn0815m02m	(182, 182, 0, 3, 64, 115, 52, 73, 149)
rsyn0815m03h	(273, 273, 0, 3, 0, 270, 303, 72, 312)
rsyn0815m03m	(273, 273, 0, 3, 96, 174, 75, 96, 240)
rsyn0815m04h	(364, 364, 0, 3, 0, 361, 402, 87, 427)
rsyn0815m04m	(364, 364, 0, 3, 128, 233, 98, 119, 331)
rsyn0815m	(44, 44, 0, 3, 32, 9, 29, 35, 3)
rsyn0820h	(49, 49, 0, 3, 0, 46, 116, 35, 29)
rsyn0820m02h	(202, 202, 0, 3, 0, 199, 223, 67, 214)
rsyn0820m02m	(202, 202, 0, 3, 64, 135, 71, 83, 166)
rsyn0820m03h	(303, 303, 0, 3, 0, 300, 330, 87, 339)
rsyn0820m03m	(303, 303, 0, 3, 96, 204, 102, 111, 267)
rsyn0820m04h	(404, 404, 0, 3, 0, 401, 437, 107, 464)
rsyn0820m04m	(404, 404, 0, 3, 128, 273, 133, 139, 368)
rsyn0820m	(49, 49, 0, 3, 32, 14, 40, 35, 5)
rsyn0830h	(58, 58, 0, 6, 0, 52, 136, 37, 30)
rsyn0830m02h	(240, 240, 0, 6, 0, 234, 259, 90, 243)
rsyn0830m02m	(240, 240, 0, 6, 64, 170, 107, 106, 195)
rsyn0830m03h	(360, 360, 0, 6, 0, 354, 381, 120, 387)
rsyn0830m03m	(360, 360, 0, 6, 96, 258, 153, 144, 315)
rsyn0830m04h	(480, 480, 0, 6, 0, 474, 503, 150, 531)
rsyn0830m04m	(480, 480, 0, 6, 128, 346, 199, 182, 435)
rsyn0830m	(58, 58, 0, 6, 32, 20, 60, 37, 6)
rsyn0840h	(66, 66, 0, 6, 0, 60, 157, 38, 33)
rsyn0840m02h	(276, 276, 0, 6, 0, 270, 295, 110, 275)
rsyn0840m02m	(276, 276, 0, 6, 64, 206, 143, 126, 227)
rsyn0840m03h	(414, 414, 0, 6, 0, 408, 433, 150, 437)
rsyn0840m03m	(414, 414, 0, 6, 96, 312, 205, 174, 365)
rsyn0840m04h	(552, 552, 0, 6, 0, 546, 571, 190, 599)
rsyn0840m04m	(552, 552, 0, 6, 128, 418, 267, 222, 503)
rsyn0840m	(66, 66, 0, 6, 32, 28, 81, 38, 9)
slay04h	(24, 24, 0, 0, 0, 24, 96, 24, 0)
slay04m	(24, 24, 0, 0, 24, 0, 0, 24, 0)
slay05h	(40, 40, 0, 0, 0, 40, 160, 40, 0)
slay05m	(40, 40, 0, 0, 40, 0, 0, 40, 0)
slay06h	(60, 60, 0, 0, 0, 60, 240, 60, 0)
slay06m	(60, 60, 0, 0, 60, 0, 0, 60, 0)
slay07h	(84, 84, 0, 0, 0, 84, 336, 84, 0)
slay07m	(84, 84, 0, 0, 84, 0, 0, 84, 0)
slay08h	(112, 112, 0, 0, 0, 112, 448, 112, 0)
slay08m	(112, 112, 0, 0, 112, 0, 0, 112, 0)
slay09h	(144, 144, 0, 0, 0, 144, 576, 144, 0)
slay09m	(144, 144, 0, 0, 144, 0, 0, 144, 0)
slay10h	(180, 180, 0, 0, 0, 180, 720, 180, 0)
slay10m	(180, 180, 0, 0, 180, 0, 0, 180, 0)
squfl010-025	(10, 10, 0, 10, 0, 0, 250, 0, 0)

Instance	$(bv, tv, fv, b0, b1, b01, v0, v1, v01)$
squfl010-040	(10, 10, 0, 10, 0, 0, 400, 0, 0)
squfl010-080	(10, 10, 0, 10, 0, 0, 800, 0, 0)
squfl015-060	(15, 15, 0, 15, 0, 0, 900, 0, 0)
squfl015-080	(15, 15, 0, 15, 0, 0, 1200, 0, 0)
squfl020-040	(20, 20, 0, 20, 0, 0, 800, 0, 0)
squfl020-050	(20, 20, 0, 20, 0, 0, 1000, 0, 0)
squfl020-150	(20, 20, 0, 20, 0, 0, 3000, 0, 0)
squfl025-025	(25, 25, 0, 25, 0, 0, 625, 0, 0)
squfl025-030	(25, 25, 0, 25, 0, 0, 750, 0, 0)
squfl025-040	(25, 25, 0, 25, 0, 0, 1000, 0, 0)
squfl030-100	(30, 30, 0, 30, 0, 0, 3000, 0, 0)
squfl030-150	(30, 30, 0, 30, 0, 0, 4500, 0, 0)
squfl040-080	(40, 40, 0, 40, 0, 0, 3200, 0, 0)
sssd08-04	(44, 44, 0, 0, 32, 12, 12, 44, 0)
sssd12-05	(75, 75, 0, 0, 60, 15, 15, 75, 0)
sssd15-04	(72, 72, 0, 0, 60, 12, 12, 72, 0)
sssd15-06	(108, 108, 0, 0, 90, 18, 18, 108, 0)
sssd15-08	(144, 144, 0, 0, 120, 24, 24, 144, 0)
sssd16-07	(133, 133, 0, 0, 112, 21, 21, 133, 0)
sssd18-06	(126, 126, 0, 0, 108, 18, 18, 126, 0)
sssd18-08	(168, 168, 0, 0, 144, 24, 24, 168, 0)
sssd20-04	(92, 92, 0, 0, 80, 12, 12, 92, 0)
sssd20-08	(184, 184, 0, 0, 160, 24, 24, 184, 0)
sssd22-08	(200, 200, 0, 0, 176, 24, 24, 200, 0)
sssd25-04	(112, 112, 0, 0, 100, 12, 12, 112, 0)
sssd25-08	(224, 224, 0, 0, 200, 24, 24, 224, 0)
st_miqp2	(2, 2, 0, 2, 0, 0, 2, 0, 0)
st_miqp4	(2, 2, 0, 2, 0, 0, 2, 0, 0)
stockcycle	(432, 432, 0, 0, 432, 0, 0, 480, 0)
st_test3	(5, 5, 0, 3, 0, 0, 3, 0, 0)
syn05h	(5, 5, 0, 3, 0, 2, 8, 0, 2)
syn05m02h	(20, 20, 0, 3, 0, 17, 19, 13, 14)
syn05m02m	(20, 20, 0, 3, 0, 17, 19, 13, 14)
syn05m03h	(30, 30, 0, 3, 0, 27, 27, 18, 24)
syn05m03m	(30, 30, 0, 3, 0, 27, 27, 18, 24)
syn05m04h	(40, 40, 0, 3, 0, 37, 35, 23, 34)
syn05m04m	(40, 40, 0, 3, 0, 37, 35, 23, 34)
syn05m	(5, 5, 0, 3, 0, 2, 8, 0, 2)
syn10h	(9, 9, 0, 3, 0, 6, 19, 2, 2)
syn10m02h	(38, 38, 0, 3, 0, 35, 35, 23, 30)
syn10m02m	(38, 38, 0, 3, 0, 35, 35, 23, 30)
syn10m03h	(57, 57, 0, 3, 0, 54, 50, 33, 49)
syn10m03m	(57, 57, 0, 3, 0, 54, 50, 33, 49)
syn10m04h	(76, 76, 0, 3, 0, 73, 65, 43, 68)
syn10m04m	(76, 76, 0, 3, 0, 73, 65, 43, 68)
syn10m	(9, 9, 0, 3, 0, 6, 19, 2, 2)
syn15h	(12, 12, 0, 3, 0, 9, 29, 3, 3)

Instance	$(bv, tv, fv, b0, b1, b01, v0, v1, v01)$
syn15m02h	(54, 54, 0, 3, 0, 51, 52, 33, 45)
syn15m02m	(54, 54, 0, 3, 0, 51, 52, 33, 45)
syn15m03h	(81, 81, 0, 3, 0, 78, 75, 48, 72)
syn15m03m	(81, 81, 0, 3, 0, 78, 75, 48, 72)
syn15m04h	(108, 108, 0, 3, 0, 105, 98, 63, 99)
syn15m04m	(108, 108, 0, 3, 0, 105, 98, 63, 99)
syn15m	(12, 12, 0, 3, 0, 9, 29, 3, 3)
syn20h	(17, 17, 0, 3, 0, 14, 40, 3, 5)
syn20m02h	(74, 74, 0, 3, 0, 71, 71, 43, 62)
syn20m02m	(74, 74, 0, 3, 0, 71, 71, 43, 62)
syn20m03h	(111, 111, 0, 3, 0, 108, 102, 63, 99)
syn20m03m	(111, 111, 0, 3, 0, 108, 102, 63, 99)
syn20m04h	(148, 148, 0, 3, 0, 145, 133, 83, 136)
syn20m04m	(148, 148, 0, 3, 0, 145, 133, 83, 136)
syn20m	(17, 17, 0, 3, 0, 14, 40, 3, 5)
syn30h	(26, 26, 0, 6, 0, 20, 60, 5, 6)
syn30m02h	(112, 112, 0, 6, 0, 106, 107, 66, 91)
syn30m02m	(112, 112, 0, 6, 0, 106, 107, 66, 91)
syn30m03h	(168, 168, 0, 6, 0, 162, 153, 96, 147)
syn30m03m	(168, 168, 0, 6, 0, 162, 153, 96, 147)
syn30m04h	(224, 224, 0, 6, 0, 218, 199, 126, 203)
syn30m04m	(224, 224, 0, 6, 0, 218, 199, 126, 203)
syn30m	(26, 26, 0, 6, 0, 20, 60, 5, 6)
syn40h	(34, 34, 0, 6, 0, 28, 81, 6, 9)
syn40m02h	(148, 148, 0, 6, 0, 142, 143, 86, 123)
syn40m02m	(148, 148, 0, 6, 0, 142, 143, 86, 123)
syn40m03h	(222, 222, 0, 6, 0, 216, 205, 126, 197)
syn40m03m	(222, 222, 0, 6, 0, 216, 205, 126, 197)
syn40m04h	(296, 296, 0, 6, 0, 290, 267, 166, 271)
syn40m04m	(296, 296, 0, 6, 0, 290, 267, 166, 271)
syn40m	(34, 34, 0, 6, 0, 28, 81, 6, 9)
synthes1	(3, 3, 0, 0, 1, 1, 1, 2, 0)
synthes2	(5, 5, 0, 1, 1, 3, 3, 2, 2)
synthes3	(8, 8, 0, 3, 1, 4, 8, 3, 2)
tls12	(489, 489, 0, 12, 465, 12, 0, 504, 129)
tls2	(31, 31, 0, 2, 29, 0, 0, 18, 17)
tls4	(85, 85, 0, 4, 81, 0, 0, 76, 25)
tls5	(131, 131, 0, 5, 126, 0, 0, 125, 31)
tls6	(165, 165, 0, 6, 159, 0, 0, 156, 45)
tls7	(278, 278, 0, 7, 271, 0, 0, 266, 61)
unitcommit1	(427, 427, 0, 9, 235, 179, 310, 196, 83)
unitcommit_200_100_1_mod_8	(4380, 4380, 0, 3843, 0, 537, 13245, 398, 190)
unitcommit_200_100_2_mod_8	(4400, 4400, 0, 3969, 0, 431, 13148, 530, 230)
unitcommit_50_20_2_mod_8	(1093, 1093, 0, 991, 0, 102, 3259, 132, 58)
watercontamination0202	(7, 7, 0, 7, 0, 0, 521, 0, 0)
watercontamination0202r	(7, 7, 0, 7, 0, 0, 188, 0, 0)
watercontamination0303	(14, 14, 0, 14, 0, 0, 1046, 0, 0)
watercontamination0303r	(14, 14, 0, 14, 0, 0, 370, 0, 0)

Table 7 Description of test set T_{pr} of 104 instances with structures, (PS_1) and (PS_2), amenable to perspective reformulation in the Section 2. The entry in the first column is the instance name and for each instance the entries in the second column are as follows: ts denotes total number of nonlinear constraints, pc shows the number of PR amenable constraints, $s1$ and $s2$ report number of constraints (out of pc) of type (S_1) and (S_2), respectively, and the last entry ub denotes the number of unique variables associated with PR amenable constraints.

Instance	$(tc, pc, s1, s2)$
clay0203h	$(24, 24, 24, 0)$
clay0204h	$(32, 32, 32, 0)$
clay0205h	$(40, 40, 40, 0)$
clay0303h	$(36, 36, 36, 0)$
clay0304h	$(48, 48, 48, 0)$
clay0305h	$(60, 60, 60, 0)$
rsyn0805h	$(3, 3, 3, 0)$
rsyn0805m02h	$(6, 6, 6, 0)$
rsyn0805m02m	$(6, 6, 6, 0)$
rsyn0805m03h	$(9, 9, 9, 0)$
rsyn0805m03m	$(9, 9, 9, 0)$
rsyn0805m04h	$(12, 12, 12, 0)$
rsyn0805m04m	$(12, 12, 12, 0)$
rsyn0805m	$(3, 3, 3, 0)$
rsyn0810h	$(6, 6, 6, 0)$
rsyn0810m02h	$(12, 12, 12, 0)$
rsyn0810m02m	$(12, 12, 12, 0)$
rsyn0810m03h	$(18, 18, 18, 0)$
rsyn0810m03m	$(18, 18, 18, 0)$
rsyn0810m04h	$(24, 24, 24, 0)$
rsyn0810m04m	$(24, 24, 24, 0)$
rsyn0810m	$(6, 6, 6, 0)$
rsyn0815h	$(11, 11, 11, 0)$
rsyn0815m02h	$(22, 22, 22, 0)$
rsyn0815m02m	$(22, 22, 22, 0)$
rsyn0815m03h	$(33, 33, 33, 0)$
rsyn0815m03m	$(33, 33, 33, 0)$
rsyn0815m04h	$(44, 44, 44, 0)$
rsyn0815m04m	$(44, 44, 44, 0)$
rsyn0815m	$(11, 11, 11, 0)$
rsyn0820h	$(14, 14, 14, 0)$
rsyn0820m02h	$(28, 28, 28, 0)$
rsyn0820m02m	$(28, 28, 28, 0)$
rsyn0820m03h	$(42, 42, 42, 0)$
rsyn0820m03m	$(42, 42, 42, 0)$
rsyn0820m04h	$(56, 56, 56, 0)$
rsyn0820m04m	$(56, 56, 56, 0)$
rsyn0820m	$(14, 14, 14, 0)$
rsyn0830h	$(20, 20, 20, 0)$
rsyn0830m02h	$(40, 40, 40, 0)$
rsyn0830m02m	$(40, 40, 40, 0)$
rsyn0830m03h	$(60, 60, 60, 0)$
rsyn0830m03m	$(60, 60, 60, 0)$
rsyn0830m04h	$(80, 80, 80, 0)$
rsyn0830m04m	$(80, 80, 80, 0)$
rsyn0830m	$(20, 20, 20, 0)$
rsyn0840h	$(28, 28, 28, 0)$
rsyn0840m02h	$(56, 56, 56, 0)$
rsyn0840m02m	$(56, 56, 56, 0)$
rsyn0840m03h	$(84, 84, 84, 0)$
rsyn0840m03m	$(84, 84, 84, 0)$
rsyn0840m04h	$(112, 112, 112, 0)$
rsyn0840m04m	$(112, 112, 112, 0)$
rsyn0840m	$(28, 28, 28, 0)$
syn05h	$(3, 3, 3, 0)$
syn05m02h	$(6, 6, 6, 0)$
syn05m02m	$(6, 6, 6, 0)$
syn05m03h	$(9, 9, 9, 0)$
syn05m03m	$(9, 9, 9, 0)$
syn05m04h	$(12, 12, 12, 0)$
syn05m04m	$(12, 12, 12, 0)$
syn05m	$(3, 3, 3, 0)$
syn10h	$(6, 6, 6, 0)$
syn10m02h	$(12, 12, 12, 0)$
syn10m02m	$(12, 12, 12, 0)$
syn10m03h	$(18, 18, 18, 0)$
syn10m03m	$(18, 18, 18, 0)$
syn10m04h	$(24, 24, 24, 0)$
syn10m04m	$(24, 24, 24, 0)$
syn10m	$(6, 6, 6, 0)$
syn15h	$(11, 11, 11, 0)$
syn15m02h	$(22, 22, 22, 0)$
syn15m02m	$(22, 22, 22, 0)$
syn15m03h	$(33, 33, 33, 0)$
syn15m03m	$(33, 33, 33, 0)$
syn15m04h	$(44, 44, 44, 0)$
syn15m04m	$(44, 44, 44, 0)$
syn15m	$(11, 11, 11, 0)$
syn20h	$(14, 14, 14, 0)$
syn20m02h	$(28, 28, 28, 0)$
syn20m02m	$(28, 28, 28, 0)$
syn20m03h	$(42, 42, 42, 0)$
syn20m03m	$(42, 42, 42, 0)$
syn20m04h	$(56, 56, 56, 0)$
syn20m04m	$(56, 56, 56, 0)$
syn20m	$(14, 14, 14, 0)$
syn30h	$(20, 20, 20, 0)$
syn30m02h	$(40, 40, 40, 0)$
syn30m02m	$(40, 40, 40, 0)$
syn30m03h	$(60, 60, 60, 0)$
syn30m03m	$(60, 60, 60, 0)$
syn30m04h	$(80, 80, 80, 0)$
syn30m04m	$(80, 80, 80, 0)$
syn30m	$(20, 20, 20, 0)$
syn40h	$(28, 28, 28, 0)$
syn40m02h	$(56, 56, 56, 0)$
syn40m02m	$(56, 56, 56, 0)$
syn40m03h	$(84, 84, 84, 0)$
syn40m03m	$(84, 84, 84, 0)$
syn40m04h	$(112, 112, 112, 0)$
syn40m04m	$(112, 112, 112, 0)$
syn40m	$(28, 28, 28, 0)$
synthes2	$(3, 1, 1, 0)$
synthes3	$(4, 2, 1, 1)$

Table 8 Description of the instances in the test set TS_{sep} for separability based reformulation in the Section 3. First column shows the instance name and the entries (nc, sp, os, us, rs) in the second column are: nc and sc number of nonlinear constraints and number of separable nonlinear constraints, respectively, os indicates whether objective function is separabe (1) or not (0), us is the number of unique separable parts considering all separable constraints and objective function, rs is the number of separable parts that are repeated. 26 instances also belonging to the test set TS_{ps} (that became amenable to perspective reformulation after the reformulation based on separability of nonlinear constraints and objective) are highlighted in bold.

Instance	(nc, sp, os, us, rs)
ball_mk2_10	$(1, 1, 0, 10, 0)$
ball_mk2_30	$(1, 1, 0, 30, 0)$
ball_mk3_10	$(1, 1, 0, 10, 0)$
ball_mk3_20	$(1, 1, 0, 20, 0)$
ball_mk3_30	$(1, 1, 0, 30, 0)$
ball_mk4_05	$(1, 1, 0, 5, 0)$
ball_mk4_10	$(1, 1, 0, 10, 0)$
ball_mk4_15	$(1, 1, 0, 15, 0)$
batch0812	$(2, 2, 0, 20, 0)$
batchdes	$(2, 2, 0, 5, 0)$
batch	$(2, 2, 0, 11, 0)$
batchs101006m	$(2, 2, 0, 29, 0)$
batchs121208m	$(2, 2, 0, 35, 0)$
batchs151208m	$(2, 2, 0, 38, 0)$
batchs201210m	$(2, 2, 0, 43, 0)$
clay0203m	$(24, 24, 0, 24, 24)$
clay0204m	$(32, 32, 0, 32, 32)$
clay0205m	$(40, 40, 0, 40, 40)$
clay0303m	$(36, 36, 0, 36, 36)$
clay0304m	$(48, 48, 0, 48, 48)$
clay0305m	$(60, 60, 0, 60, 60)$
enpro48pb	$(2, 2, 0, 13, 0)$
enpro56pb	$(2, 2, 0, 12, 0)$
ex1223a	$(5, 2, 0, 6, 0)$
ex1223b	$(5, 5, 0, 12, 5)$
ex1223	$(5, 5, 0, 12, 5)$
ex4	$(26, 26, 0, 125, 2)$
fac1	$(1, 0, 1, 2, 0)$
fac2	$(1, 0, 1, 3, 0)$
fac3	$(1, 0, 1, 3, 0)$
gams01	$(111, 0, 1, 10, 0)$
hybriddynamic_fixed	$(1, 0, 1, 11, 0)$
immun	$(1, 0, 1, 6, 0)$
netmod_dol1	$(1, 0, 1, 6, 0)$
netmod_dol2	$(1, 0, 1, 6, 0)$
netmod_kar1	$(1, 0, 1, 4, 0)$
netmod_kar2	$(1, 0, 1, 4, 0)$
nvs03	$(2, 0, 1, 2, 0)$
nvs10	$(3, 0, 1, 2, 0)$
pedigree_ex1058	$(1, 1, 0, 28, 0)$
pedigree_ex485_2	$(1, 1, 0, 28, 0)$
pedigree_ex485	$(1, 1, 0, 28, 0)$
pedigree_sp_top4_250	$(1, 1, 0, 58, 0)$
pedigree_sp_top4_300	$(1, 1, 0, 74, 0)$
pedigree_sp_top4_350tr	$(1, 1, 0, 17, 0)$
pedigree_sp_top5_200	$(1, 1, 0, 54, 0)$
pedigree_sp_top5_250	$(1, 1, 0, 58, 0)$
portfol_classical050_1	$(1, 1, 0, 50, 0)$
portfol_classical200_2	$(1, 1, 0, 200, 0)$
risk2bpb	$(1, 0, 1, 3, 0)$
slay04h	$(1, 0, 1, 8, 0)$
slay04m	$(1, 0, 1, 8, 0)$
slay05h	$(1, 0, 1, 10, 0)$
slay05m	$(1, 0, 1, 10, 0)$
slay06h	$(1, 0, 1, 12, 0)$
slay06m	$(1, 0, 1, 12, 0)$
slay07h	$(1, 0, 1, 14, 0)$
slay07m	$(1, 0, 1, 14, 0)$
slay08h	$(1, 0, 1, 16, 0)$
slay08m	$(1, 0, 1, 16, 0)$
slay09h	$(1, 0, 1, 18, 0)$
slay09m	$(1, 0, 1, 18, 0)$
slay10h	$(1, 0, 1, 20, 0)$
slay10m	$(1, 0, 1, 20, 0)$
squfl010-025	$(1, 0, 1, 250, 0)$
squfl010-040	$(1, 0, 1, 400, 0)$
squfl010-080	$(1, 0, 1, 800, 0)$
squfl015-060	$(1, 0, 1, 900, 0)$
squfl015-080	$(1, 0, 1, 1200, 0)$
squfl020-040	$(1, 0, 1, 800, 0)$
squfl020-050	$(1, 0, 1, 1000, 0)$
squfl020-150	$(1, 0, 1, 3000, 0)$
squfl025-025	$(1, 0, 1, 625, 0)$
squfl025-030	$(1, 0, 1, 750, 0)$
squfl025-040	$(1, 0, 1, 1000, 0)$
squfl030-100	$(1, 0, 1, 3000, 0)$
squfl030-150	$(1, 0, 1, 4500, 0)$
squfl040-080	$(1, 0, 1, 3200, 0)$
st_e14	$(5, 5, 0, 12, 5)$
st_miqp1	$(1, 0, 1, 5, 0)$
st_miqp2	$(1, 0, 1, 2, 0)$
st_miqp4	$(1, 0, 1, 3, 0)$
st_miqp5	$(1, 0, 1, 2, 0)$
stockcycle	$(1, 0, 1, 48, 0)$
st_test1	$(1, 0, 1, 4, 0)$
st_test2	$(1, 0, 1, 5, 0)$
st_test3	$(1, 0, 1, 5, 0)$
st_test4	$(1, 0, 1, 2, 0)$
st_test5	$(1, 0, 1, 7, 0)$
st_test6	$(1, 0, 1, 10, 0)$
st_test8	$(1, 0, 1, 24, 0)$
st_testgr1	$(1, 0, 1, 10, 0)$
st_testgr3	$(1, 0, 1, 20, 0)$
st_testph4	$(1, 0, 1, 3, 0)$
synthes2	$(4, 0, 1, 3, 0)$
synthes3	$(5, 2, 0, 6, 1)$
tls12	$(12, 12, 0, 144, 0)$
tls2	$(2, 2, 0, 4, 0)$
tls4	$(4, 4, 0, 16, 0)$
tls5	$(5, 5, 0, 25, 0)$
tls6	$(6, 6, 0, 36, 0)$
tls7	$(7, 7, 0, 49, 0)$
unitcommit1	$(1, 0, 1, 240, 0)$
unitcommit_200_100_1_mod_8	$(1, 0, 1, 4662, 0)$
unitcommit_200_100_2_mod_8	$(1, 0, 1, 4639, 0)$
unitcommit_50_20_2_mod_8	$(1, 0, 1, 1152, 0)$
watercontamination0202	$(1, 0, 1, 4017, 0)$
watercontamination0303	$(1, 0, 1, 4521, 0)$

B Computational Results

Table 9 Summary of collections of type $C_i, i = 1, 2, 3$ in instances in test set TS_c. The second column reports the number of instances containing at least one collection of the type mentioned in the first column. In the last column, the first sub-column corresponds to the number of instances (out of the number of instances mentioned under the second column) in which at least 50% of the total number of variables are found to be semi-continuous. The second sub-column shows the number of instances in which the total number of semi-continuous variables is less than 10%.

		# inst. with semi-continuous variables	
type	# inst.	$\geq$ 50%	$\leq$ 10%
C_1	194	151	9
C_2	132	41	5
C_1 and C_2	220	203	0
C_1 and C_3	194	154	7
C_2 and C_3	132	43	5
C_1 and C_2 and C_3	220	208	0

Table 10 (Top) Comparison of *qg* and methods (M) on 15 instances in TS_{ps} that are solved by both the techniques. (Bottom) Performance on ten instances that are solved by both, but at least one of the techniques took more than ten seconds.

	time		nodes	
Method (M)	*qg*	rel.	*qg*	rel.
qgsep	78.24	0.32	1213.03	0.42
qgprsep	78.24	0.12	1213.03	0.12

	time		nodes	
Method (M)	*qg*	rel.	*qg*	rel.
qgsep	251.01	0.22	4418.17	0.31
qgprsep	251.01	0.07	4418.17	0.06

Table 11 (Top) Comparison of *qg* and methods (M) on instances solved by both, but at least one method took more than 100 seconds. (Bottom) Similar comparisons for instances solved by both methods, but at least one took more than 500 seconds.

# solved	time		nodes	
by both	*qg*	rel.	*qg*	rel.
7	654.78	0.15	10605.48	0.27
5	1454.86	0.01	15711.76	0.02

# solved	time		nodes	
by both	*qg*	rel.	*qg*	rel.
4	2389.38	0.04	27068.51	0.15
4	2389.38	0.01	27068.51	0.01

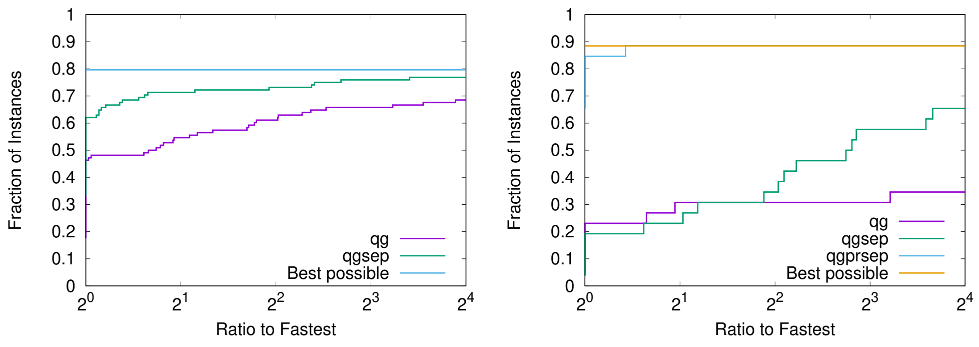


Fig. 3 Performance profiles comparing solution times of *qg* and *qgsep* (on left), and of *qg*, *qgsep*, and *qgprsep* (on right).