Efficient linear reformulations for binary polynomial optimization problems

Sourour Elloumi, Zoé Verchère

Efficient linear reformulations for binary polynomial optimization problems

Sourour Elloumi[a], Zoé Verchère[b,*]

[a] *CNAM, CEDRIC, 292 Rue Saint-Martin, 75003, Paris, France*
[b] *UMA, ENSTA Paris, Institut Polytechnique de Paris, , 91120, Palaiseau, France*

Abstract

We consider unconstrained polynomial minimization problems with binary variables (BPO). These problems can be easily linearized, i.e., reformulated into a MILP in a higher dimensional space. Several linearizations are possible for a given BPO, depending on how each monomial is decomposed and replaced by additional variables and constraints. We focus on finding efficient linearizations that maximize the continuous relaxation bound of the resulting MILP. For this purpose, we introduce the notion of linearization patterns that allow us to model and enumerate the possible decompositions of a degree-d monomial. The assignment of a unique pattern to each monomial of BPO results in a reformulation of BPO into a MILP. Our method, called `MaxBound`, amounts to searching for an optimal association between monomials and patterns in the sense that it leads to a MILP with the best continuous relaxation bound. We show that this process can be formulated as a MILP which we denote by $(\widetilde{MB})$. We further highlight domination properties among the patterns that allow us to discard the dominated patterns and to decrease the size of $(\widetilde{MB})$. Another effect of these domination properties is that it now makes sense to search for a reformulation that requires as few additional variables as possible, based only on the non-dominated patterns. We call this reformulation method `ND-MinVar` and again show that it can be found by solving another MILP. We make an experimental study on degree 4 polynomials that compares the results of both methods and shows the advantages and disadvantages of each.

Keywords: Polynomial optimization, MINLP, Linear reformulation

*Corresponding author.

1. Introduction

Considering a positive integer n, we define the problem of minimizing a polynomial over the binary variables $x = (x_i)_{1 \leqslant i \leqslant n}$ as follows:

$$\text{(BPO)} \quad \begin{cases} \min\limits_{x} & \sum\limits_{M \in P} c_M \prod\limits_{i \in M} x_i & \text{(1a)} \\ & \text{s.t. } x \in \{0,1\}^n & \text{(1b)} \end{cases}$$

where P is a set of non-empty subsets M of $\{1, \ldots, n\}$ representing *monomials* and $c_M \in \mathbb{R}$, $M \in P$, are the corresponding monomial coefficients. We suppose w.l.o.g. that P contains all singletons $\{i\}$. The cardinality $|M|$ of monomial M is called the *degree* of M. The set P is called a *polynomial*, and its degree is the maximal degree of its monomials. Since x is binary, we have $x^2 = x$, hence no higher order powers appear and the objective function is a *multilinear* function of x. This objective function is sometimes called a *pseudo-Boolean* function in the literature, leading to the field sometimes being called pseudo-Boolean optimization [3].

Problem BPO is known to be NP-hard, even if the degree of P is two [24], where BPO reduces to the thoroughly studied *quadratic unconstrained binary optimization problem* (QUBO), see [21] for a survey. In this context, some methods aim at solving BPO by reducing it to a quadratic problem. Papers that present such methods focus on producing a quadratic reformulation with specific properties [5, 2], with some also discussing how to best solve the reformulated problem when needed [17]. Direct algorithms that do not involve linear or quadratic reformulations to solve BPO have also been studied [6, 11].

It is also known that BPO can be equivalently reformulated by a mixed integer linear problem [18]. A common linear reformulation is the *standard linearization*. For each monomial $M \in P$, one considers a binary variable X_M that will represent the value of the monomial, i.e., $X_M = \prod_{i \in M} x_i$. The number of additional variables X is precisely the number of non-singleton monomials in P. Standard linearization amounts to reformulating BPO into the following MILP:

$$
(\mathrm{SL}) \begin{cases} \min\limits_{X} \quad \sum\limits_{M \in P} c_M X_M & \\ \text{s.t.} & \\ \quad X_M \leq x_i & \forall\, M \in P \quad \forall\, i \in M. \\ \quad X_M \geq \sum\limits_{i \in M} (x_i - 1) + 1 & \forall\, M \in P \\ \quad X_M \geqslant 0 & \forall\, M \in P \\ \quad x_i \in \{0,1\} & \forall\, i \in \{1, \ldots, n\} \end{cases}
$$

One can check that in any feasible solution of SL, the relation $X_M = \prod\limits_{i \in M} x_i$ holds. Therefore, solving SL actually solves BPO. A known drawback of SL is that its continuous relaxation bound may be very weak, especially when P contains a large number of monomials. To overcome this drawback, it is of interest to study the set $X_{BP} = \{X \in \{0,1\}^{|P|} :\ X_M = \prod_{i \in M} x_i,\ \forall\, M \in P\}$.

Some papers use the standard linearization as a basis, and focus on finding new classes of valid inequalities for $conv(X_{BP})$ in order to achieve better computational results and/or to attain theoretical results through the study of the associated polyhedra. For example, reaching equality to $conv(X_{BP})$ is desirable since all extreme points of $conv(X_{BP})$ are integer, hence solving the linear relaxation of the problem over this polyhedron would give an integer solution, thus solving the integer problem immediately.

In [7], Crama and Rodríguez-Heck propose a class of valid inequalities they name "2-link" inequalities and study their impact on the resulting polyhedron, as well as in practice. In [12] and [13], Del Pia and Khajavirad study inequalities that generalize those used by Crama and Rodríguez-Heck. They also derive more general conclusions concerning the resulting polyhedron, and study the computational impact of these inequalities in [14]. More recent papers have uncovered yet more classes of valid inequalities and associated polyhedral results [10, 15].

Another approach lies in what we may call extended linear reformulations, where new variables are introduced in order to represent certain parts of monomials, or certain products of variables. Such techniques have been used in continuous polynomial optimization, as in RLT-POS and further developments [9, 8], and more recently as the base relaxation in a solver called RAPOSa [19]. In the case of binary variables, Hojny, Pfetsch and Walter [20] use such an approach to tackle BPO.

Our contribution

The general aim of the paper is to explore the set of extended linear reformulations in order to find a reformulation with a good lower bound or a good compromise between bound quality and size of the reformulation. We do not intend to provide the best possible bounds, and indeed better bounds can be achieved through cutting plane methods [15, 13, 14] or SDP relaxations [17]. Instead, we present a new idea centered around the concept of linearization patterns, and the idea of optimizing the choice of these patterns with a specific goal in mind, namely the quality of the resulting bound or the size of the resulting reformulation.

In Section 2, we give a definition for extended linear reformulations. To this end, we introduce linearization patterns for monomials, and linearization graphs for polynomials. We then define the linear reformulation of BPO corresponding to a linearization graph. In Section 3, we present a new method to obtain a linear reformulation, named `MaxBound`, which consists in finding the linear reformulation that yields the best continuous relaxation bound out of a wide class of possible reformulations. We show that this objective can be reached from the solution of a MILP where the decision variables help to choose a linearization pattern for any monomial $M \in P$. Next, we show a domination property between linearization patterns that allows us to reduce the size of the MILP. In Section 4, we introduce method `ND-MinVar` , the goal of which is to provide a linear reformulation based on the same reduced set of monomial linearization patterns while minimizing the total number of additional variables. This excludes dominated linearization patterns, including the standard linearization. We also introduce a bi-objective problem hybridizing `ND-MinVar` and `MaxBound`. Finally, in Section 5 we provide a detailed analysis of our computational experiments, comparing our new methods with each other and with the standard linearization, across multiple important parameters (root gap, computation time and more). Finally, in Section 6 we draw our conclusions.

2. Extended linear reformulations of polynomials of binary variables

In this section, we establish a formal setting for extended linear reformulations of BPO. We define linearization patterns and graphs, address the particular case of simple linearization patterns, and show that some known linear reformulations and theoretical results can fit in our setting.

2.1. Linearization patterns, graphs, and extended linear reformulations

We start by defining linearization patterns.

Definition 1 (Linearization pattern of a monomial)**.** *Let M be a non-empty monomial. A linearization pattern of M is a rooted acyclic directed graph $G_M = (V_M, A_M)$. Every vertex $v \in V_M$ is a non-empty subset of M. Graph G_M must satisfy the following:*

- *Root vertex: The unique root of G_M is M.*
- *Singleton leaves: $\forall i \in M$, $\{i\}$ is a leaf of G_M. There are no other leaves.*
- *Succession by strict inclusion: A_M is such that any non-leaf vertex v verifies $v = \bigcup_{t \in \delta^+(v)} t$ and $t \subsetneq v \; \forall v \in V_M, \; \forall t \in \delta^+(v)$.*

The idea behind this definition is that any monomial of a given degree can be linearized in several ways. These ways of linearizing a monomial do not depend on the monomial but on the structure of the linearization itself, hence why we call them patterns. A given linearization pattern can be applied to any monomial of the appropriate degree.

Next, we define the linearization graph of a polynomial, as well as the underlying concept of concatenation of linearization patterns and graphs.

Definition 2 (Linearization graph of a polynomial)**.** *Let $P = (M_1, \dots, M_{|P|})$ be a set of monomials. For each monomial M_i, a linearization pattern G_i is provided. The linearization graph of P is $G = \mathcal{C}(G_1, \dots, G_{|P|})$, where $\mathcal{C}$ is the linearization concatenation operator as defined below in Definition 3.*

Definition 3 (Concatenation of linearization patterns and graphs)**.** *Let M_1 and M_2 be two monomials, and G_1, G_2 their respective linearization patterns. Let $P = \{M_1, M_2\}$. Then, we can define a linearization graph of P, $G = \mathcal{C}(G_1, G_2) = (V_P, A_P)$, in the following manner: G_P is the union of G_1 and G_2, and any vertices that appear in both G_1 and G_2 if and only if the subgraph originating from it is identical in both graphs. If it is not the case, we arbitrarily number the vertices to distinguish them from one another.*
We may concatenate more than two linearization patterns by doing so sequentially. That is to say, for any k linearization patterns $\{G_1, G_2, \dots, G_k\}$, we have

$$\mathcal{C}(G_1, G_2, \dots, G_k) = \mathcal{C}(\mathcal{C}(G_1, G_2), \dots, G_k) = \cdots = \mathcal{C}(\mathcal{C}(G_1, \dots, G_{k-1}), G_k).$$

Now that we have defined linearization patterns for monomials, we can now see how they translate into a MILP formulation of BPO. We can define what this reformulation is, given a linearization pattern for each monomial $M \in P$. We do so in the following definition, in which we also give practical details on how to construct the MILP.

Definition 4 (Linear reformulation associated to a linearization graph.)**.** *Let P be the polynomial involved in the objective function of BPO. Let $G_P = (V_P, A_P)$ be a linearization graph of P. We introduce one variable X_v for every vertex v in $\bigcup_{1 \leqslant i \leqslant |P|} G_i$. The variables corresponding to leaves*

are binary.

The following mixed integer linear problem $(MILP_G)$*:*

$$(MILP_G) \quad \begin{cases} \min\limits_{X} \sum\limits_{M \in P} c_M X_M & & \text{(2a)} \\ s.t. \;\; X_v \leqslant X_s & \forall v \in V_P,\ \forall s \in \delta_G^+(v) & \text{(2b)} \\ \quad X_v \geqslant 1 + \sum\limits_{s \in \delta_G^+(v)} (X_s - 1) & \forall v \in V_P & \text{(2c)} \\ \quad X_v \in \{0,1\} & \forall v \in V_P,\ v \text{ is a leaf} & \text{(2d)} \\ \quad X_v \geqslant 0 & \forall v \in V_P,\ v \text{ is not a leaf} & \text{(2e)} \end{cases}$$

is the linear reformulation of BPO corresponding to the given linearization patterns of monomials $M \in P$*.*

Lemma 5. *Problems BPO and* $(MILP_G)$ *are equivalent.*

Proof. For any solution $\tilde{x}$ of BPO, we can build a solution $\tilde{X}$ of $(MILP_G)$ by setting $\tilde{X}_{\{i\}} = \tilde{x}_i$ for all $i \in \{1, \dots, n\}$. In doing so, the rest of $\tilde{X}$ is fixed, because for all linearization patterns, if the leaves take binary values, then all the vertices do so as well due to the nature of the constraints. This entails that for all monomials $M \in P$, we have $\tilde{X}_M = \prod\limits_{i \in M} \tilde{x}_i$. Therefore, for any solution of BPO, there exists a solution of $(MILP_G)$ of equal value.
Conversely, for any solution $\tilde{X}$ of $(MILP_G)$, we build a solution $\tilde{x}$ of BPO of equal value by setting $\tilde{x}_i = \tilde{X}_{\{i\}}$.

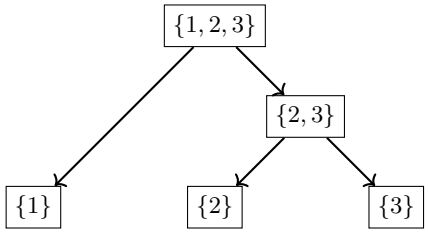
□

To illustrate these notions, we provide a small introductory example. Consider the function $f(x) = 3x_1x_2x_3 - 4x_2x_3x_4$. Taking after our notations, we have $P = \{\{1,2,3\},\{2,3,4\}\}$. Figures 1a and 1b show the linearization patterns chosen for monomials $\{1,2,3\}$ and $\{2,3,4\}$ respectively, while Figure 1c shows the linearization graph for P resulting from the concatenation of the linearization patterns of each monomial.

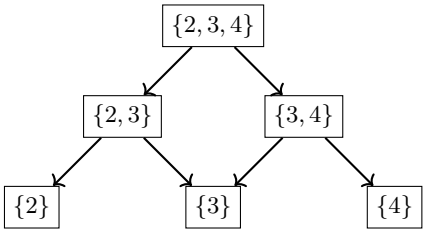
2.2. Simple linearization patterns

In many cases, our study will focus on linearization patterns that we call *simple*. We define them below, and give additional details in a corollary.

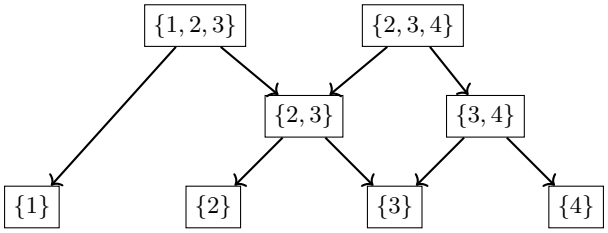
Definition 6 (Simple linearization pattern of a monomial)**.** *Let* M *be a monomial. A linearization pattern* $G_M = (V_M, A_M)$ *is simple if and only if for any non-leaf vertex* $v \in V_M$*, the set* $\{s \mid s \in \delta^+(v)\}$ *forms a partition of* v*.*



(a) A linearization pattern for monomial $\{1, 2, 3\}$.



(b) A linearization pattern for monomial $\{2, 3, 4\}$.



(c) The resulting linearization graph for $P = \{\{1, 2, 3\}, \{2, 3, 4\}\}$ after concatenation.

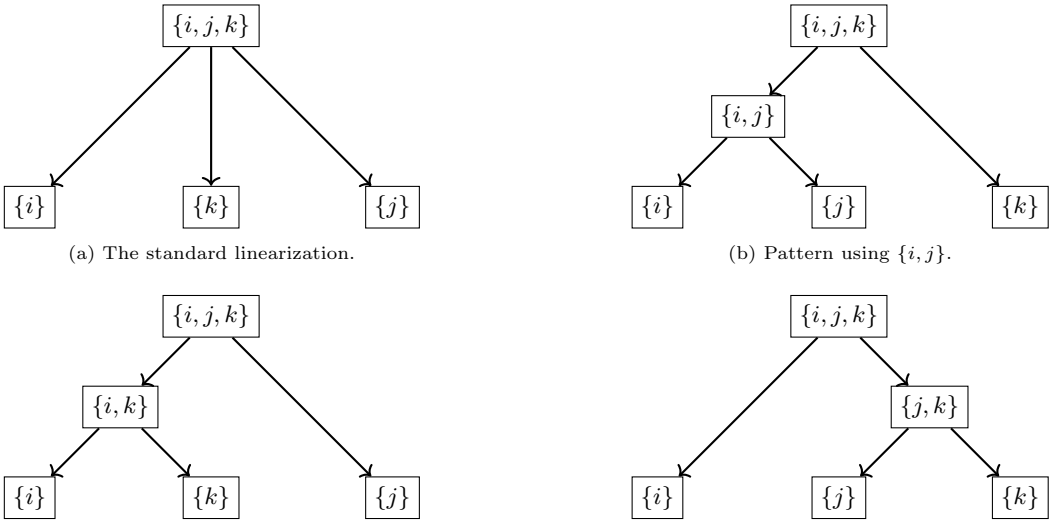
Figure 1: Example of linearization pattern and the resulting linearization graph after concatenation for $P = \{\{1, 2, 3\}, \{2, 3, 4\}\}$.

Corollary 7. *Simple linearization patterns Let M be a monomial, and let $G_M = (V_M, A_M)$ be a linearization pattern. The following propositions are equivalent.*

1. *G_M is a simple linearization pattern.*
2. *For any non-leaf vertex $v \in V_M$, the set $\{s \mid s \in \delta^+(v)\}$ forms a partition of v.*
3. *G_M is a rooted directed tree i.e. the undirected variant of G_M is acyclic.*

The simple linearization patterns are the only patterns we consider in the remainder of this paper. In order to fully describe them, we show how to count them for any degree d.

For a given degree d, we denote by N_d^s the number of simple linearization patterns that exist for a monomial of degree d. To calculate N_d^s, we must count all partitions of a monomial M of degree d into non-empty sets, and then, recursively account for the number of linearization patterns for the different partitions of M. For any such partition $\mathcal{I}$ of M, we associate a vector v of dimension $d-1$ where v_k denotes the number of subsets of cardinality k in $\mathcal{I}$. It holds that $\sum_{k=1}^{d-1} k v_k = d$. Let V_d be the set of $(d-1)$-vectors v satisfying the last equation. For example, $V_4 = \{(4, 0, 0), (2, 1, 0), (1, 0, 1), (0, 2, 0)\}$. Of course, a vector $v \in V_d$ may be associated to several different partitions. For a vector $v \in V_d$, the number of different partitions can be counted as follows : choose v_1 elements among the d elements of M, then choose $2v_2$ elements among the



(a) The standard linearization.

(b) Pattern using $\{i, j\}$.

(c) Pattern using $\{i, k\}$.

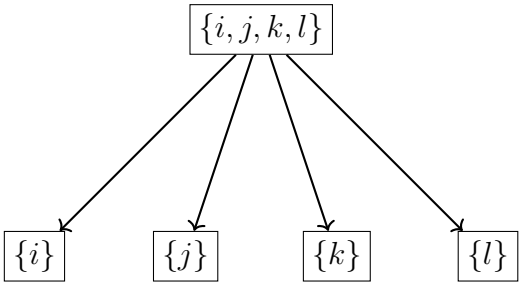
(d) Pattern using $\{j, k\}$.

Figure 2: Simple linearization patterns for a monomial of degree three.

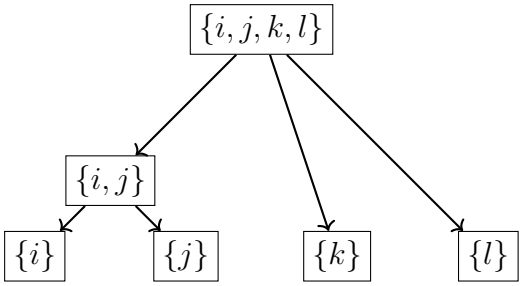
remaining elements and divide by the number of ways to assign these elements to v_2 subsets of cardinality 2, and so on until reaching $d-1$. Finally, the pattern is determined not only by the partition, but also by the pattern chosen by each subset of the partition. Therefore, we should multiply by $\prod_{k=1}^{d-1} (N_k)^{v_k}$, representing the number of possible pattern combinations for these subsets. Moreover, as this number is the number of possibilities for a given v, we need to sum across all possible $v \in V_d$. We set $N_1^s = 1$, since monomials of degree one are already linear terms. This gives the following expression.

$$N_d^s = d! \sum_{v \in V_d} \prod_{k=1}^{d-1} \frac{(N_k^s)^{v_k}}{(v_k)!(k!)^{v_k}}$$

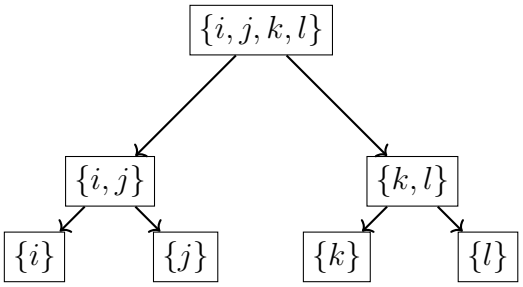
In Figure 2 we show all simple linearization patterns for monomials of degree 3. In Figure 3, we show all "types" of simple linearization patterns for monomials of degree 4.



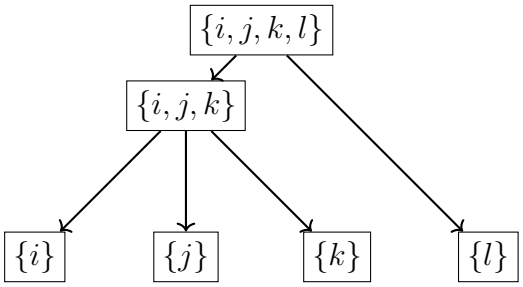
(a) Type 1: the standard linearization.



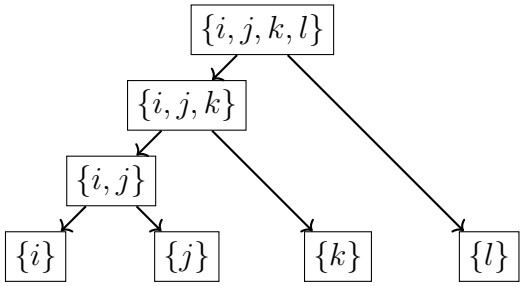
(b) Type 2: use of a sub-monomial of degree two. Six such patterns exist.



(c) Type 3: use of two sub-monomials of degree two. Three such patterns exist.



(d) Type 4: Use of a sub-monomial of degree three. Four such patterns exist.



(e) Type 5: Use of a sub-monomial of degree three and of one of degree four. Twelve such patterns exist.

Figure 3: Types of simple linearization patterns for a monomial of degree four.

2.3. Revisiting known approaches

Within this setting for linear reformulation, we may redefine existing linearization-based approaches. One such approach is the standard linearization. Let $M \in P$ be a monomial. Within our setting, the standard linearization pattern of M is a star graph with center M and branches $\{i\}$ for $i \in M$. As previously shown, Figure 3a represents the standard linearization pattern for an arbitrary monomial of degree four.

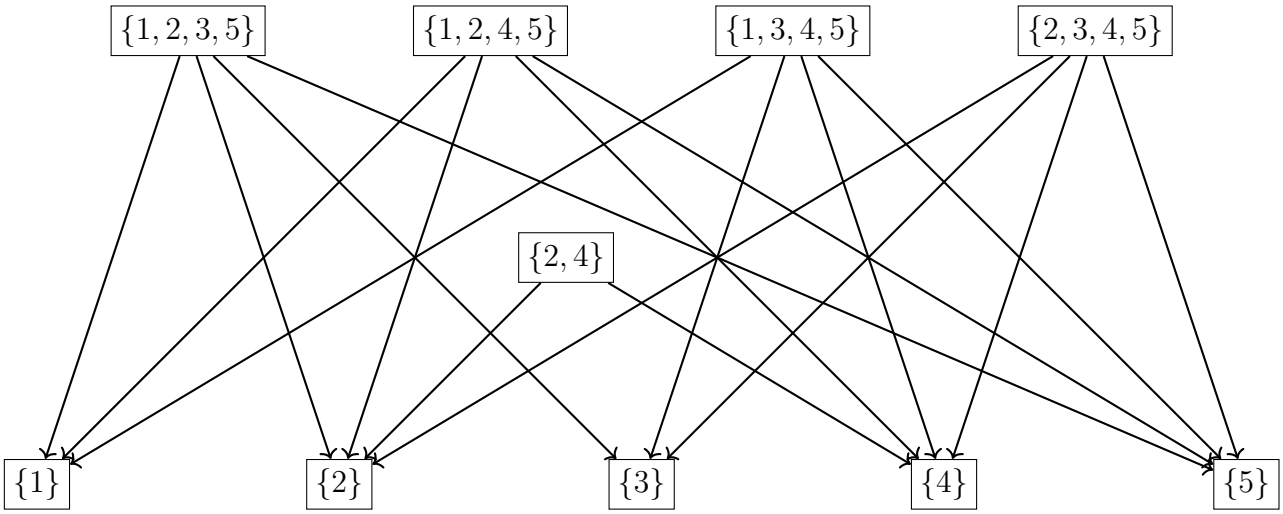
Let us introduce an example that will serve throughout the paper. The following problem is a small instance of BPO.

$$(Ex)\begin{cases} \min\limits_{x} & f(x) = 5.35\ x_2x_4 + 9.31\ x_1x_2x_3x_5 - 6.54\ x_1x_2x_4x_5 + 9.97\ x_1x_3x_4x_5 - 1.99\ x_2x_3x_4x_5 & \text{(3a)} \\ \text{s.t.} & x \in \{0,1\}^5 & \text{(3b)} \end{cases}$$

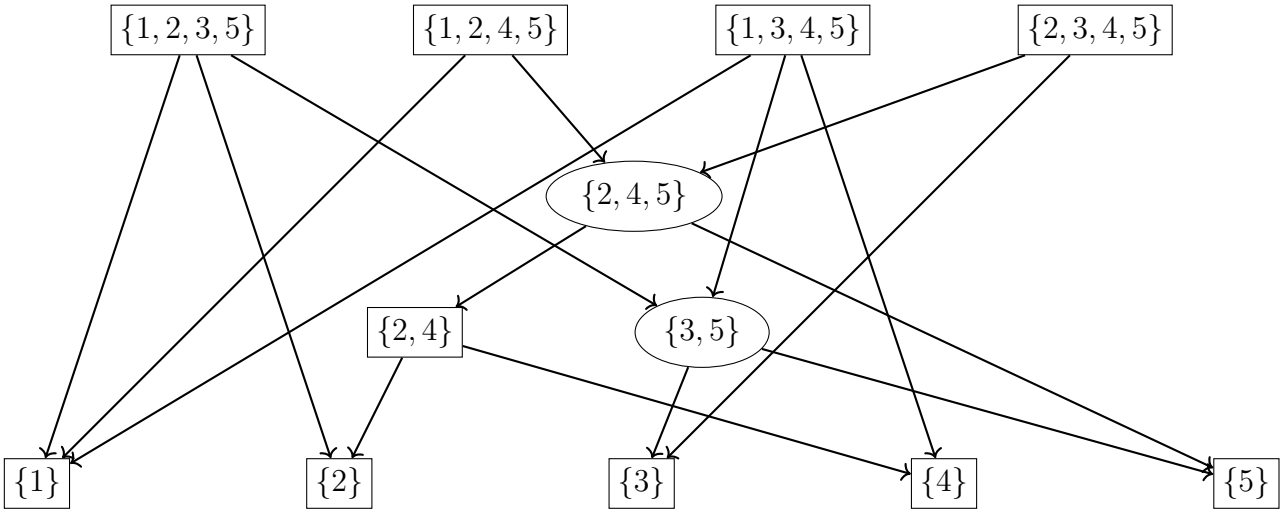
Figure 4 shows two linearization graphs corresponding to (Ex). The first is the standard linearization, the second is an arbitrarily created one. Vertices that correspond to monomials of the objective function are represented by a rectangle, as well as base variables. Any additional submonomials used are represented by ellipses.

The optimal value of (Ex) is -1.19. The standard linearization from Figure 4a gives a lower bound of -4.265, while the linear reformulation depicted in the graph in Figure 4b yields a bound of -2.385. This shows that extended linear reformulations can yield better bounds than the standard linearization.

Pushing this idea further, Hojny, Pfetsch and Walter [20] develop a method to build an extended linear reformulation of BPO in an as of yet unpublished paper. They give a direct definition of linearization graphs that is not based on our linearization patterns. However, their linearization graphs are very similar to graphs we may obtain by fusing linearization patterns together. Their work focuses on finding integer linearizations i.e. linear reformulations such that the polyhedron defined by the constraints, when projected onto the reduced space of variables $(X_M)_{M\in P}$ has only integer vertices. This property is desirable, since it ensures that no matter the coefficients $(c_M)_{M\in P}$ of BPO, solving the continuous relaxation of the linear reformulation yields an integer solution, thus solving BPO. It is also worth reminding that their main result, a characterization of instances for which an integer linear reformulation exists, is equivalent to a result about β-acyclic hypergraphs



(a) Linearization graph of the standard linearization for (Ex).



(b) Linearization graph of an arbitrary choice of linearization patterns for (Ex).

Figure 4: Two different linearization graphs for the polynomial in (Ex)

satisfying the running intersection property found in [12]. We show how some of their results translate in our setting.

Theorem 8 (Integer linear reformulations. Hojny et al. [20])**.** *Let P be a polynomial, and G a linearization graph of P. Then, G provides an integer linear reformulation if and only if the undirected version of G is acyclic.*

Hojny et al. [20] also give a method to construct a linearization graph $\hat{G}$ such that an integer linear reformulation exists if and only if $\hat{G}$ is acyclic. Graph $\hat{G}$ is built as follows: the vertices are all monomials of P and all monomials that result from the intersection of any two monomials of P. There is an arc from vertex M_1 to vertex M_2 if $M_2 \subset M_1$ and no other vertex M_3 satisfies $M_2 \subset M_3 \subset M_1$. The resulting graph can be interpreted as a linearization graph, though it may call on linearization patterns that are not simple. We call this method the HPW method, and the resulting linear reformulation the HPW linear reformulation.

In Figure 5, we show what linearization graph this method yields for the polynomial in (Ex). The linearization graph is not acyclic, hence there exists no integer extended linear reformulation for (Ex). The LP bound of the corresponding linear reformulation is -2.544, better than the standard linearization but slightly worse than that of Figure 4b.

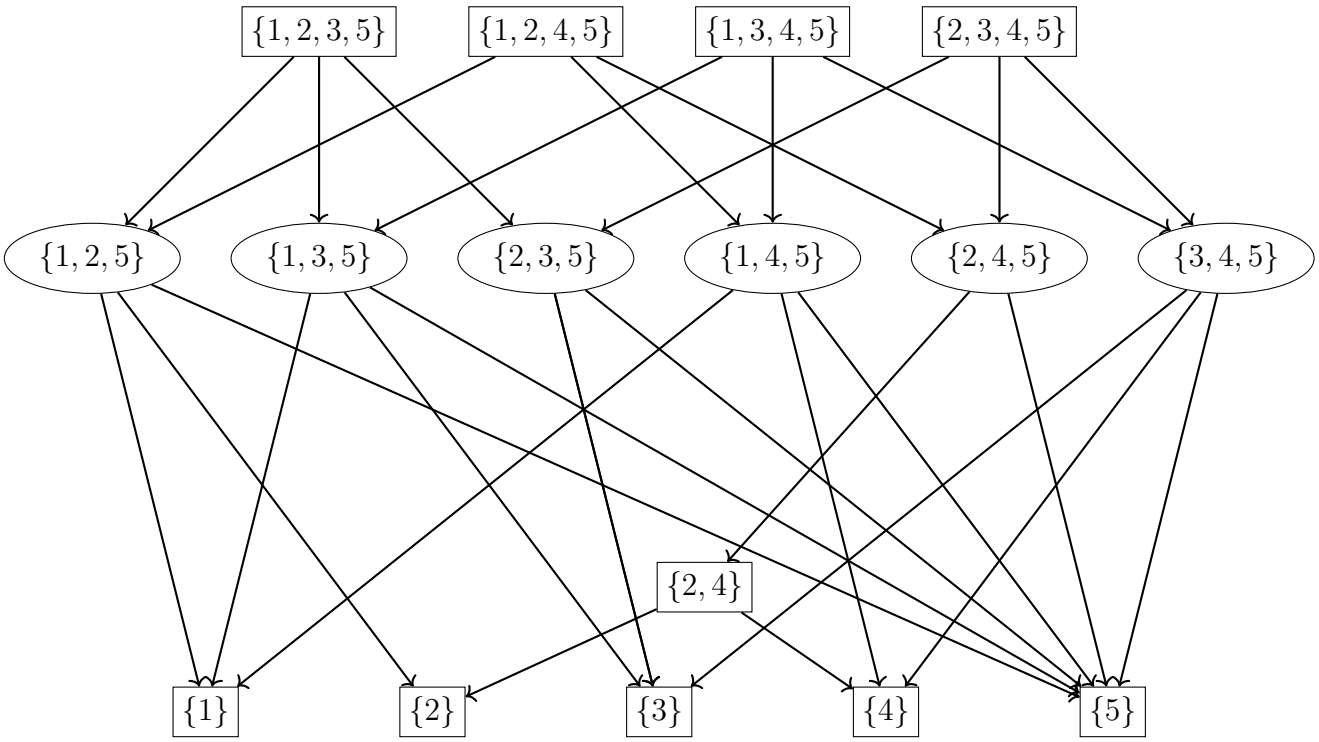


Figure 5: HPW linearization graph for (Ex).

3. `MaxBound`: Building efficient linear reformulations

In this section, we focus on finding efficient linear reformulations. To that end, we look for linear reformulation with a strong lower bound, in order to hasten the resolution process by cutting off branches that are not promising more quickly.

3.1. Obtaining the best bound by continuous relaxation

There are multiple different linear reformulations for a given instance of BPO. The bound by continuous relaxation is not the same across all possible linear reformulations. Hojny et al. [20] show that this is especially true when an integer linear reformulation exists, but this is almost never the case. However, it remains true that different linear reformulations can yield vastly different bounds. We can see this on problem (Ex), on which we have seen different extended linear reformulations produce different bounds. However, we have not yet been able to find a reformulation whose continuous relaxation closed the gap entirely. A question one may ask is that of finding a linear reformulation that yields the best possible bound. We aim to provide an answer to this question by developing a method named `MaxBound`, which selects one simple linearization pattern per monomial in order to maximize the value of the bound.

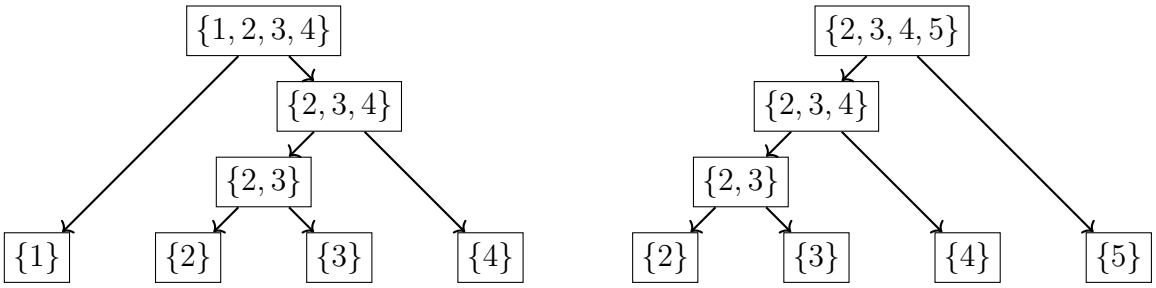
The idea at the heart of the `MaxBound` method is to introduce binary variables to represent the choice of a certain simple linearization pattern over the other possibilities. To do so, we arbitrarily number the simple linearization patterns for each degree, and we introduce variables u_M^α for all $M \in P$ and for all $\alpha \in \{1, \dots, N^s_{|M|}\}$.

$$u_M^\alpha = \begin{cases} 1 & \text{if linearization pattern number } \alpha \text{ is chosen for monomial } M \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

We also introduce variables X_M^α. For a given monomial M and a pattern number α, X_M^α is a variable that represents $\prod_{i \in M} x_i$, but is subject to linearization constraints according to pattern α. Using these new variables, we may linearize a monomial M by $\sum_{\alpha=1}^{N_{|M|}} u_M^\alpha X_M^\alpha$ as long as we ensure that $\sum_{\alpha=1}^{N_{|M|}} u_M^\alpha = 1$. In other words, we assign a unique simple linearization pattern to each monomial M. We can then maximize the value of the LP bound over the u variables. This yields problem (MB).

$$
(MB)\quad \begin{cases}
\max\limits_{u} \quad \min\limits_{X} \sum\limits_{M\in P} c_M \Big(\sum\limits_{\alpha=1}^{N^s_{|M|}} u^\alpha_M X^\alpha_M \Big) & \text{(5a)} \\
\qquad \text{s.t. } X^\alpha_v \text{ linearizes } v \text{ according to pattern } \alpha, \ \ \forall v \subseteq M \in P,\ \forall \alpha \in \{1,\dots,N^s_{|v|}\} & \text{(5b)} \\
\qquad\qquad 0 \leqslant X^\alpha_v \leqslant 1 \ \ \forall v \subseteq M \in P,\ \forall \alpha \in \{1,\dots,N^s_{|v|}\} & \text{(5c)} \\
\text{s.t.} \quad \sum\limits_{\alpha=1}^{N^s_{|M|}} u^\alpha_M = 1 \quad \forall M \in P & \text{(5d)} \\
\qquad u^\alpha_M \in \{0,1\} \qquad \forall M \in P,\ \forall \alpha \in \{1,\dots,N^s_{|M|}\} & \text{(5e)}
\end{cases}
$$

Concerning linearization variables X^α_v: there is one linearization variable for each subset of a monomial and every considered linearization of that subset. It is important to note that if a subset v appears in two different monomials M_1 and M_2, then the variables X^α_v that appear in some of the linearization patterns of M_1 are the same variables as those that appear in some of the linearization patterns of M_2. An example is given in Figure 6, with $M_1 = \{1,2,3,4\}$, $M_2 = \{2,3,4,5\}$ and $v = \{2,3,4\}$. Both the patterns depicted use the same linearization pattern for subset v, therefore in the resulting linear reformulation only one variable X^α_v is needed.



(a) A chosen pattern for $M_1 = \{1,2,3,4\}$.

(b) A chosen pattern for $M_2 = \{2,3,4,5\}$.

Figure 6: Chosen patterns for M_1 and M_2 both use the same pattern for subset $v = \{2,3,4\}$. In this case, only one variable X^α_v exists in the reformulated problem.

A generic form for (MB) is as follows. We denote by $\tilde{c}$ the modified coefficient vector where for every monomial M, coefficient c_M appears $N^s_{|M|}$ times, so that $\tilde{c}$ and u have the same dimension. We denote by $(\tilde{c} * u)$ the product of $\tilde{c}$ and u component-wise. Constraints (6b)-(6c) provide (5b)-(5c) in more compact notations. The number p is defined by $p = \sum\limits_{M\in P} N^s_{|M|}$, and $\mathbb{1}_p$ is the column vector of length p where every component is equal to one.

$$(MB)\begin{cases} \max\limits_{u} \quad \min\limits_{X} \ (\tilde{c} * u)^T X & \text{(6a)} \\ \qquad\quad \text{s.t. } LX \geqslant l & \text{(6b)} \\ \qquad\qquad\quad X \geqslant 0 & \text{(6c)} \\ \text{s.t.} \quad Au = \mathbb{1}_p & \text{(6d)} \\ \qquad\quad u \in \{0,1\}^p & \text{(6e)} \end{cases}$$

We denote by Y the dual variable associated to (6b). After using LP duality on the minimization problem embedded inside the maximization problem, we obtain a mixed-integer linear maximization problem.

$$(\widetilde{MB})\begin{cases} \max\limits_{u,Y} \quad l^T Y & \text{(7a)} \\ \text{s.t.} \quad L^T Y \leqslant (\tilde{c} * u) & \text{(7b)} \\ \qquad\quad Au = \mathbb{1}_p & \text{(7c)} \\ \qquad\quad Y \geqslant 0 & \text{(7d)} \\ \qquad\quad u \in \{0,1\}^p & \text{(7e)} \end{cases}$$

3.2. Domination among simple linearization patterns

Problem $(\widetilde{MB})$ contains binary variables u. For one monomial, there must be as many such variables as simple linearization patterns. This number of patterns grows quickly with the degree of the monomial, which means that solving problem $(\widetilde{MB})$ may prove very difficult to solve for instances of high size and/or degree, though it remains to see exactly where the limit may be. Thus, it is of interest to find a way to reduce the size of the space of possible reformulations, if possible without affecting the quality of the bound of the resulting reformulation.

The following result is a theorem first proven in [20]. Theorem 9 is a rewriting of this theorem, along with a proof that fits our formal setting. Corollary 10 addresses the particular case of the standard linearization, showing that the standard linearization is dominated by any other linearization using only simple linearization patterns, in terms of LP bound. Theorem 9 and

Corollary 11 indicate that we may consider a subset of simple linearization patterns without lowering the quality of the best bound found by `MaxBound`.

Theorem 9 (Domination among simple linearization patterns.)**.** *Let P be a polynomial, part of an instance of BPO. Let $G = (V, A)$ and $\tilde{G} = (\tilde{V}, \tilde{A})$ be two linearization graphs of P, in which the linearization pattern of every monomial is a simple linearization pattern.*
Assume that there exists a unique monomial of P, denoted M_0, such that G and $\tilde{G}$ are identical except for their respective subgraphs originating in M_0. More precisely, $\tilde{G}_{M_0}$ contains one more vertex than G_{M_0}, denoted by z. Its predecessor is denoted by $\hat{v}$. In short, in $\tilde{G}$, z linearizes part of the successors of $\hat{v}$ in G, and $\hat{v}$ represents the product of z and the rest of its successors in G.
Then, $v(RLP_{\tilde{G}}) \geqslant v(RLP_G)$, where (RLP_G) and $(RLP_{\tilde{G}})$ are the continuous relaxations of $(MILP_G)$ and $(MILP_{\tilde{G}})$, and where $v(RLP_G)$ and $v(RLP_{\tilde{G}})$ are their optimal values.

Proof. We begin by defining some notations needed throughout the proof. We denote by B and $\tilde{B}$ the feasible domains of problems (RLP_G) and $(RLP_{\tilde{G}})$ respectively.

Let $X = (X_v)_{v \in \tilde{V}}$ be a feasible solution to problem $(RLP_{\tilde{G}})$, that is to say $X \in \tilde{B}$: X satisfies the following constraints.

$$(\tilde{B}) \begin{cases} X_v \leqslant X_w & \forall v \in \tilde{V},\ \forall w \in \delta^+_{\tilde{G}}(v) \\ X_v \geqslant 1 + \sum\limits_{w \in \delta^+_{\tilde{G}}(v)} (X_w - 1) & \forall v \in \tilde{V},\ |v| \geqslant 2 \\ 0 \leqslant X_v \leqslant 1 & \forall v \in \tilde{V} \end{cases}$$

We will now show that the restriction of X to $(X_v)_{v \in V}$, that is to say the same set of variables without X_z, is such that $X \in B$.
B and $\tilde{B}$ are the same except when it comes to monomial M_0, where $\widetilde{V}_{M_0} = V_{M_0} \cup \{z\}$. This tells us, in particular, that $|z| \geqslant 2$, otherwise G_{M_0} and $\widetilde{G}_{M_0}$ would be linearizations of two distinct monomials. We now focus on the constraints pertaining to monomial M_0 and split them into different parts to highlight the places in which z appears. X satisfies the following.

$$(\widetilde{B}_{M_0}) \begin{cases} X_v \leqslant X_w & \forall v \in V_{M_0} \setminus \{\hat{v}\},\ \forall w \in \delta^+_{\tilde{G}}(v) & \text{(8a)} \\ X_{\hat{v}} \leqslant X_w & \forall w \in \delta^+_{\tilde{G}}(\hat{v}) \setminus \{z\} & \text{(8b)} \\ X_{\hat{v}} \leqslant X_z & & \text{(8c)} \\ X_v \geqslant 1 + \sum\limits_{w \in \delta^+_{\tilde{G}}(v)} (X_w - 1) & \forall v \in V_{M_0} \setminus \{\hat{v}\},\ |v| \geqslant 2 & \text{(8d)} \\ X_{\hat{v}} \geqslant 1 + \sum\limits_{w \in \delta^+_{\tilde{G}}(v) \setminus \{z\}} (X_w - 1) + (X_z - 1) & & \text{(8e)} \\ X_z \leqslant X_w & \forall w \in \delta^+_{\tilde{G}}(z) & \text{(8f)} \\ X_z \geqslant 1 + \sum\limits_{w \in \delta^+_{\tilde{G}}(z)} (X_w - 1) & & \text{(8g)} \\ 0 \leqslant X_v \leqslant 1 & \forall v \in V_{M_0} \setminus \{z\} & \text{(8h)} \\ 0 \leqslant X_z \leqslant 1 & & \text{(8i)} \end{cases}$$

We need to show that $(X_v)_{v\in V}$ also satisfies (B_{M_0}):

$$(B_{M_0})\quad \begin{cases} X_v \leqslant X_w & \forall v \in V_{M_0},\ \forall w \in \delta^+_G(v) & \text{(9a)}\\ X_v \geqslant 1 + \sum\limits_{w\in\delta^+_G(v)} (X_w - 1) & \forall v \in V_{M_0},\ |v| \geqslant 2 & \text{(9b)}\\ 0 \leqslant X_v \leqslant 1 & \forall v \in V_{M_0} & \text{(9c)} \end{cases}$$

In order to do so, we can combine certain inequalities together and deduce new inequalities satisfied by X. The ones of interest to us are specifically the ones in which z or a predecessor of z in $\tilde{G}$ appears, since they are the only differences between (B_{M_0}) and $(\widetilde{B}_{M_0})$. We first show that X also satisfies (9a).

$$(8c) + (8f) \implies X_{\hat{v}} \leqslant X_w \ \ \forall w \in \delta^+_{\tilde{G}}(z)$$

Together with (8b), we obtain $X_{\hat{v}} \leqslant X_w$ for all $w \in (\delta^+_{\tilde{G}}(\hat{v}) \setminus \{z\}) \cup \delta^+_{\tilde{G}}(z)$. However, the definitions of z and $\hat{v}$ imply that $(\delta^+_{\tilde{G}}(\hat{v}) \setminus \{z\}) \cup \delta^+_{\tilde{G}}(z) = \delta^+_G(\hat{v})$. We also consider that in (8a), $\forall v \in V_{M_0} \setminus \{\hat{v}\}$, we have $\delta^+_{\tilde{G}}(v) = \delta^+_G(v)$. In this way we get (9a).

We now show that X also satisfies (9b).

$$(8e)+(8g) \implies X_{\hat{v}} \geqslant 1+ \sum_{w\in\delta^+_{\tilde{G}}(\hat{v})\setminus z} (X_w-1)+ \sum_{w\in\delta^+_{\tilde{G}}(z)} (X_w-1) \ \ \forall v \in V_{M_0} \text{ such that } z \in \delta^+_{\tilde{G}}(v) \text{ and } |v| \geqslant 2$$

As before, realizing that $(\delta^+_{\tilde{G}}(\hat{v}) \setminus \{z\}) \cup \delta^+_{\tilde{G}}(z) = \delta^+_G(\hat{v})$ yields (9b) for $\hat{v}$. For the rest of the vertices of V_{M_0}, simply consider (8d). Since $\delta^+_{\tilde{G}}(v) = \delta^+_G(v)$ for all $v \in V_{M_0} \setminus \{\hat{v}\}$, we obtain (9b).

Finally, (9c) is trivially satisfied thanks to (8h).

Hence, the restriction of X to $X_V = (X_v)_{v\in V}$ is such that $X_V \in B_{M_0}$. Since B and $\tilde{B}$ are otherwise defined by identical inequalities, we have shown that for any solution $X \in \tilde{B}$, then the restriction of X to the vertices of G belongs to B. In other words, we have found that for any feasible solution to problem $(\widetilde{LP})$, it is possible to construct a solution to problem (LP) of equal value. Since BPO is a minimization problem, then (LP) and $(\widetilde{LP})$ are also minimization problems: this implies that $v(RLP_{\tilde{G}}) \geqslant v(RLP_G)$. □

Corollary 10 (Domination of the standard linearization by simple linearization patterns.)**.** *Let P be the polynomial appearing in an instance of BPO. Let $G = (V, A)$ be the standard linearization of P, and $\tilde{G} = (\tilde{V}, \tilde{A})$ be a linearization graph of P in which the linearization pattern of every monomial is a simple linearization pattern. Then, $v(RLP_{\tilde{G}}) \geqslant v(RLP_G)$.*

Proof. Consider a polynomial P, the standard linearization graph $G = \underset{M\in P}{\mathcal{C}}(G_M)$, and a linearization graph $\tilde{G} = \underset{M\in P}{\mathcal{C}}(\tilde{G}_M)$, where for every monomial $M \in P$, $\tilde{G}_M = (\tilde{V}_M, \tilde{A}_M)$ is a simple linearization pattern. Then, we can build a sequence of linearization graphs $(G^k)_{1\leqslant k\leqslant K}$ such that:

- $G^1 = G$

- $G^K = \tilde{G}$
- For all $k \in \{1, \ldots, K-1\}$, G^k and G^{k+1} respect the hypotheses of Theorem 9.

Building this sequence can be done algorithmically. Start at $G^1 = G$. Then, given $G^k = \underset{M \in P}{\mathcal{C}}(G^k_M)$, find one monomial M_0 such that $G^k_{M_0} \neq \tilde{G}_{M_0}$.

Create $G^{k+1} = \underset{M \in P}{\mathcal{C}}(G^{k+1}_M)$ such that for all $M \neq M_0$, $G^{k+1}_M = G^k_M$, and $G^{k+1}_{M_0}$ is $G^k_{M_0}$ augmented by one vertex that is in $\tilde{G}_{M_0}$ but not in $G^k_{M_0}$. This is possible by construction because for every monomial $M \in P$, the standard linearization pattern G_M has the fewest possible vertices.

Since the only vertices added using this procedure are vertices from $\tilde{G}_M$, the procedure stops once $G^K = \tilde{G}$ is reached. The sequence resulting from this procedure satisfies all three requirements by construction.

Therefore, using Theorem 9, we have

$$v(RLP_G) = v(RLP_{G^1}) \leqslant v(RLP_{G^2}) \leqslant \cdots \leqslant v(RLP_{G^{K-1}}) \leqslant v(RLP_{G^K}) = v(RLP_{\tilde{G}})$$

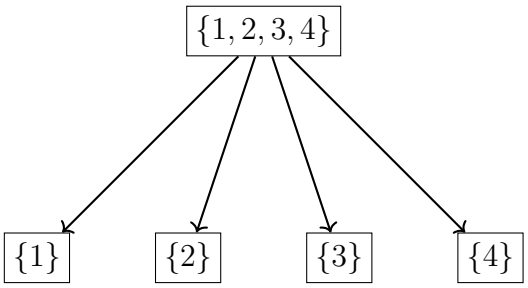
□

Corollary 11 (Reduction of the set of simple linearization patterns for maximizing the LP bound.)**.** *Let P be the polynomial appearing in an instance of BPO. Then for any monomial $M \in P$, we may consider only simple linearization patterns of the form $G_M = (V_M, A_M)$ such that the out-degree of every non-leaf vertex $v \in V_M$ is equal to two, without affecting the quality of the bound obtained by the* `MaxBound` *method.*

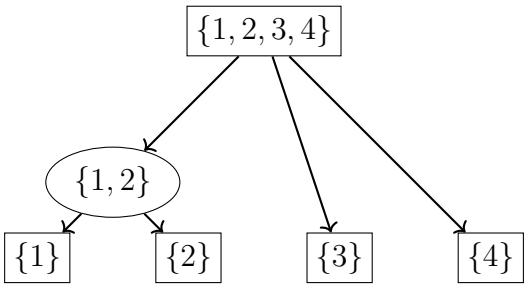
Proof. Consider a polynomial P and a linearization $G = \underset{M \in P}{\mathcal{C}}(G_M)$ where $G_M = (V_M, A_M)$ is a simple linearization of monomial M. Suppose that there exists $M_0 \in P$ such that $\exists\, v \in V_{M_0}$ such that v is not a leaf and $\delta^+(v) \neq 2$. First we remind that $\delta^+(v) \neq 1$, as the contrary would contradict Definition 6. Then, if $\delta^+(v) \geqslant 3$, then we may put forward a new simple linearization G'_{M_0} for M_0, and thus a new linearization of P, $G' = \mathcal{C}(\underset{M \in P \setminus \{M_0\}}{\mathcal{C}}(G_M), G'_{M_0})$, by introducing a new vertex corresponding to the union of the first two successors of v. We apply this procedure recursively until $\delta^+(v) = 2$. According to Theorem 9, each new linearization provides a bound at least as tight as the previous one. This proves the corollary. □

We provide an example in Figure 7, where the reduction procedure described in Corollary 11 is applied to the standard linearization of a monomial of degree four, $M = \{1, 2, 3, 4\}$. As the corollary indicates, we may remove dominated simple linearization patterns from the available choices in `MaxBound`. In the case of a monomial of degree three, this means removing the standard linearization from the available patterns. For a monomial of degree four, this means removing linearization patterns of types 1, 2 and 4 as indicated in Figure 3. This does not mean that a linear reformulation that uses the dominated patterns cannot be tight. It means that replacing each dominated pattern

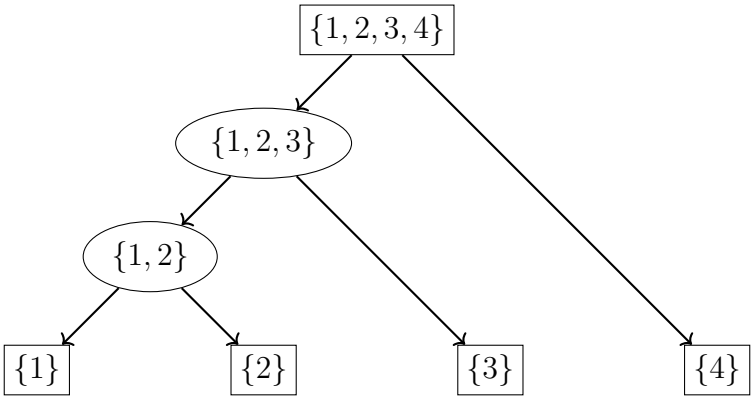
by one of the patterns that dominate it creates a reformulation that is at least as tight.



(a) Step 1: the standard linearization.



(b) Step 2: add vertex $\{1, 2\}$ to (a).



(c) Step 3: add vertex $\{1, 2, 3\}$ to (b). Every vertex has out-degree 2 or 0.

Figure 7: Successive linearizations for $M = \{1, 2, 3, 4\}$. (c) dominates (b), and (b) dominates (a).

Eliminating the dominated patterns leaves a number of non-dominated simple linearization patterns. For a monomial of degree d, we denote this number by N_d^{nd}.

In order to calculate N_d^{nd}, we must count the number of ways to split this monomial into two non-empty subsets, multiplied by the number of possibilities for these smaller subsets. We also set $N_1^{nd} = 1$, since monomials of degree one are already linear. Accounting for symmetry, we get the following formula.

$$N_d^{nd} = \frac{1}{2} \sum_{k=1}^{d-1} \binom{d}{k} N_k^{nd} N_{d-k}^{nd}$$

These two sequences $(N_d^s)_{d \geqslant 1}$ and $(N_d^{nd})_{d \geqslant 1}$ can be found on the Online Encyclopedia of Integer Sequences, and are the solution to Schröder's fourth problem and third problem, respectively [25].

Since every pattern is a binary variable in problem (MB), it is interesting to see how many such variables we can eliminate by considering only non-dominated patterns. We give a few values of N_d^s and N_d^{nd} for monomials of degree 3 to 8 in Table 1. We can see that both numbers quickly increase with degree. Nevertheless, it allows us to quantify the impact of Theorem 9 on `MaxBound`.

Degree d	N_d^s	N_d^{nd}	Reduction
3	4	3	25%
4	26	15	42%
5	236	105	55%
6	2 752	945	66%
7	39 208	10 395	73%
8	660 032	135 135	79%

Table 1: Values of N_d^s, N_d^{nd} for different values of d.

3.3. The compact `MaxBound` algorithm

As a conclusion to this section, we outline below the steps of the `MaxBound` method as an algorithm.

Algorithm 1: The `MaxBound` method

Input: An instance of BPO: a polynomial P and a coefficient vector c.
Output: An optimal solution of BPO and its value.
Reformulation step:
Solve $(\widetilde{MB})$, find an optimal or feasible solution $(\tilde{u}, \tilde{Y})$.
Use $\tilde{u}$ to deduce a linearization pattern G_M for each $M \in P$ accordingly.
Build G, a linearization graph of P, by concatenation: $G = \underset{M \in P}{\mathcal{C}}(G_M)$.
Resolution step:
Solve $(MILP_G)$ the linear reformulation of BPO, based on graph G.

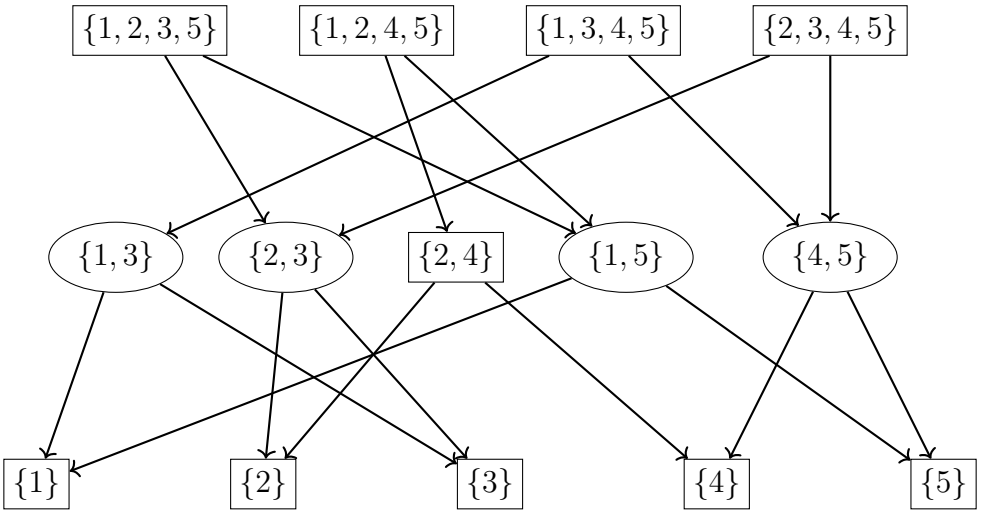


Figure 8: Linearization graph of (Ex) generated by `MaxBound`.

In applying this method to problem (Ex), by merging the linearization patterns chosen for each monomial, we get the linearization graph depicted in Figure 8. The corresponding extended linear reformulation has an LP bound of -1.723, and adds four new variables compared to the standard linearization.

4. `ND-MinVar`: Building linear reformulations with a minimum number of additional variables

In the previous section, we introduced a method to find an extended linear reformulation of BPO so that the continuous relaxation of that reformulation yields the best possible lower bound. It does so by picking one non-dominated simple linearization pattern per monomial. However, these non-dominated patterns introduce $d-1$ new variables, which is the most out of all simple linearization patterns. Therefore, it is likely that `MaxBound` introduces many more variables than simpler approaches such as the standard linearization. These additional variables may slow down the resolution.

In search of a different tradeoff, we introduce `ND-MinVar`, a method which seeks an extended linear reformulation of BPO that uses non-dominated simple linearization patterns, with the goal of introducing as few new variables as possible.

4.1. The `ND-MinVar` method

In [26], the authors use a MILP to reformulate BPO into a quadratic problem with as few new variables as possible, for polynomials of degrees three and four. Inspired by [26], we derive below a MILP to find a linear reformulation with the fewest additional variables. Given a polynomial P, this MILP contains the same variables u_M^α for all $M \in P$ and $\alpha \in \{1, \dots, N_{|M|}^{nd}\}$ defined by (4) and w_v^β for all $v \subseteq M \in P$ and $\beta \in \{1, \dots, N_{|v|}^{nd}\}$, such that:

$$w_v^\beta = \begin{cases} 1 & \text{if linearization pattern number } \beta \text{ of } v \text{ appears in the chosen} \\ & \text{linearization pattern of one of the monomials of } P \\ 0 & \text{otherwise.} \end{cases}$$

These definitions mean that $w_v^\beta = 1$ if and only if variable X_v^β appears in the resulting extended linear reformulation. Our goal is to minimize the number of variables needed, so we minimize the sum of variables w_v^β.

$$
(MV)\quad \begin{cases}
\min\limits_{u,w} \displaystyle\sum_{\substack{v\subseteq M\\ M\in P}} \sum_{\beta=1}^{N^{nd}_{|v|}} w_v^\beta & & \text{(10a)}\\
\text{s.t. } u_M^\alpha \leqslant w_v^\beta & \forall M\in P,\ \forall \alpha\in\{1,\dots,N^{nd}_{|M|}\}, \forall\ v \text{ such that pattern } \beta & \text{(10b)}\\
& \text{of } v \text{ appears in linearization pattern } \alpha \text{ of } M & \\
\displaystyle\sum_{\alpha=1}^{N^{nd}_{|M|}} u_M^\alpha = 1 & \forall M\in P & \text{(10c)}\\
u_M^\alpha\in\{0,1\} & \forall M\in P,\ \forall\ \alpha\in\{1,\dots,N^{nd}_{|M|}\} & \text{(10d)}\\
w_v^\beta\in\{0,1\} & \forall v\subseteq M\in P,\ \forall\ \beta\in\{1,\dots,N^{nd}_{|v|}\} & \text{(10e)}
\end{cases}
$$

Constraints (10b) ensure that if $u_M^\alpha = 1$, then for any vertices $v = (s, \beta)$ appearing in this chosen linearization, we must have $w_s^\beta = 1$.

In (MV), the w variables may be set to continuous non-negative instead of binary, as they will always take the minimum value they are allowed to and are constrained only by the values of the u variables, themselves binary.

The solution of problem (MV) provides us with a quadratization-based linear reformulation of BPO, introducing as few additional variables as possible. As with `MaxBound`, we outline the various steps of "the `ND-MinVar` method" in Algorithm 2.

Algorithm 2: The `ND-MinVar` method

Input: An instance of BPO: a polynomial P and a coefficient vector c.
Output: An optimal solution of BPO and its value.
Reformulation step:
Solve problem (MV), find an optimal or feasible solution $(\tilde{u}, \tilde{w})$.
Use $\tilde{u}$ to deduce a linearization pattern G_M for each $M \in P$ accordingly.
Build G, a linearization graph of P, by concatenation: $G = \underset{M\in P}{\mathcal{C}}(G_M)$.
Resolution step:
Solve the linear reformulation $(MILP_G)$ of BPO.

Applying `ND-MinVar` to the problem (Ex) generates the linearization graph shown in Figure 9. The corresponding extended linear reformulation has an LP bound of -2.385, and adds three variables compared to the standard linearization.

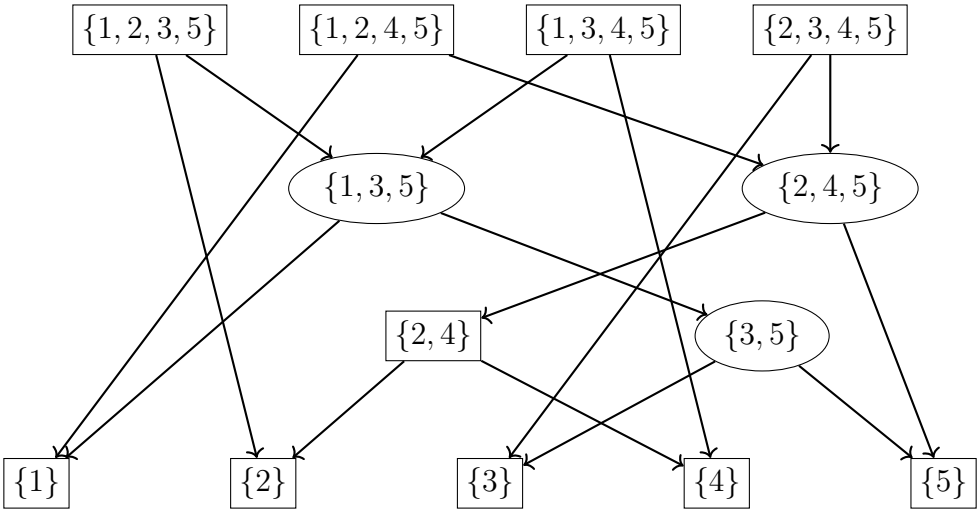


Figure 9: Linearization graph of (Ex) generated by `ND-MinVar`.

4.2. The bi-objective viewpoint

`MaxBound` and `ND-MinVar` both use the non-dominated simple linearization patterns. But as we can see in Theorem 9 and its Corollary, these patterns are the simple linearization patterns that introduce the most variables. It may happen that some of the picked patterns are not optimal, in the sense that one of the variables it introduces may not have any impact on the bound. For example, on problem (Ex), `ND-MinVar` generates an extended linearization with three new variables, while the linearization graph from Figure 4b adds only two new variables and yields the same LP bound.

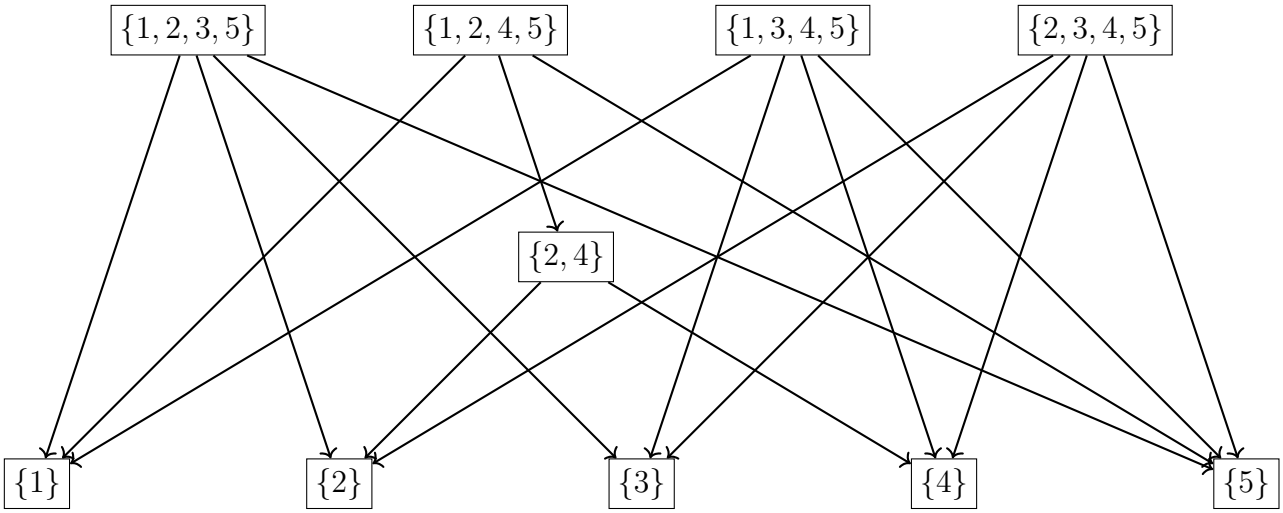
Thus, it is interesting to reintegrate the dominated patterns and try to maximize the bound and use as few variables as possible at the same time. Since problems (MB) and (MV) share the set of variables u, which are the choice of linearization pattern for all monomials, we can merge these problems into a bi-objective problem as follows.

$$
\begin{cases}
\text{Obj 1: } \min\limits_{u,w} \displaystyle\sum_{\substack{v\subseteq M\\ M\in P}} w_v & & (11a)\\
\text{Obj 2: } \max\limits_{u} & \min\limits_{X} \displaystyle\sum_{M\in P} c_M\Big(\sum_{\alpha=1}^{N^s_{|M|}} u^\alpha_M X^\alpha_M\Big) & (11b)\\
& \text{s.t. } X^\alpha_v \text{ linearizes } s \text{ according to pattern } \alpha \ \ \forall v\subseteq M\in P,\ \forall \alpha\in\{1,\dots,N^s_v\} & (11c)\\
& \qquad 0\leqslant X^\alpha_v\leqslant 1 \ \ \forall v\subseteq M\in P,\ \forall \alpha\in\{1,\dots,N^s_v\} & (11d)\\
\text{s.t. } w_t\geqslant u_v & \forall v,t \text{ such that } s_t\subseteq s_v \text{ and } s_v\in P & (11e)\\
\displaystyle\sum_{\alpha=1}^{N^s_{|v|}} u^\alpha_M = 1 & \forall M\in P & (11f)\\
u^\alpha_M\in\{0,1\} & \forall \alpha\in\{1,\dots,N^s_{|M|}\}\ \forall M\in P & (11g)\\
w_v\in\{0,1\} & \forall v\subseteq M\in P & (11h)
\end{cases}
$$

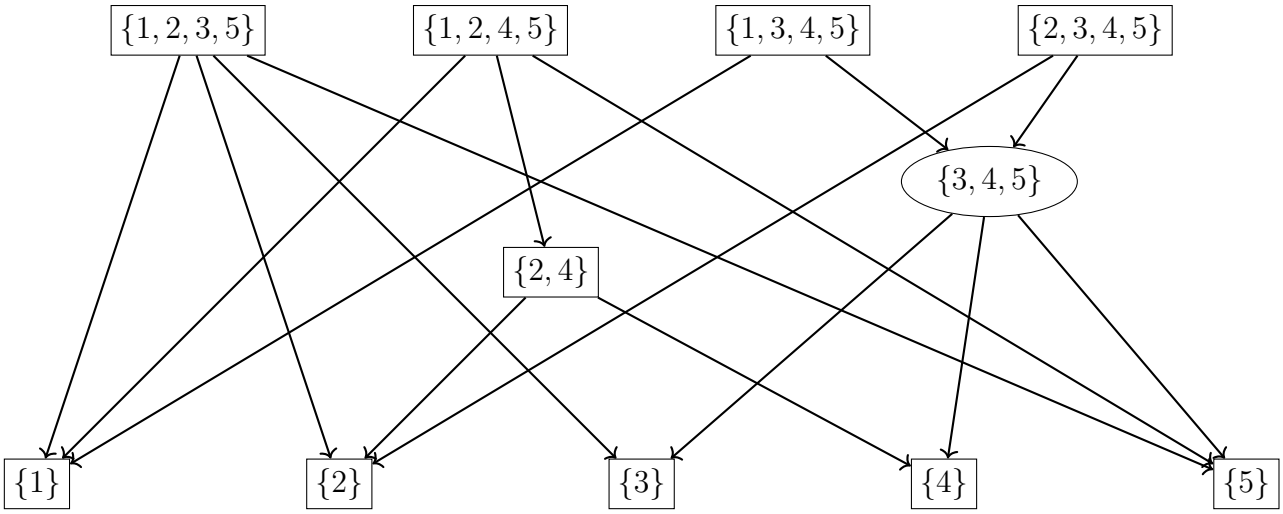
This problem can be solved via standard methods, since objective (11a) only takes integer values. We can find all Pareto-optimal solutions by solving the problem with objective (11b) with a constraint on the number of variables that can be used. We then strengthen this constraint each step until the problem becomes infeasible. This process may find better solutions than `MaxBound` or `ND-MinVar`, but it can be very computationally expensive.

On (Ex), this method gives two non-dominated solutions. Figure 10 shows the two linearization graphs corresponding to these solutions.

In the Table 2, we provide a summary of how all the different methods discussed so far work on (Ex), detailing the LP bound and total number of variables for each of the corresponding extended linear reformulations. "Non-dominated" 1 and 2 are the two solutions to the bi-objective problem described in this section, and the line "OPT" is just a reminder of the optimal value of (Ex) in binary variables.



(a) Non-dominated solution 1: no new variables, LP bound equal to -2.385.



(b) Non-dominated solution 2: one new variable, LP bound equal to -1.723.

Figure 10: Two linearization graphs corresponding to the non-dominated solutions of the bi-objective problem applied to (Ex).

Method	LP bound	Number of variables
Standard linearization	-4.265	10
HPW	-2.54	16
Figure 4b	-2.385	12
`MaxBound`	-1.723	14
`ND-MinVar`	-2.385	13
Non-dominated 1	-1.723	11
Non-dominated 2	-2.385	10
OPT	-1.19	-

Table 2: Characteristics of various extended linear reformulations on (Ex).

5. Computational experiments and analysis

We compare the different linear reformulation schemes introduced or reviewed in this paper: the standard linearization (SL), as well as `MaxBound` (MB) and `ND-MinVar` (MV) and described in Algorithm 1 and Algorithm 2. The reformulation step of the last two methods requires solving a MILP, which can take a long time to solve to optimality. We impose a time limit of thirty minutes on the reformulation step, as thirty minutes is usually enough to solve the problem or get a feasible solution of good quality for instances up to 40 variables and about 15,000 monomials. The time limit on the solving step is three hours.

We compare the values of the continuous relaxations of each linear reformulation, the best value and optimality gap obtained after the time limit of three hours, the number of explored nodes in the branch-and-bound tree and the time it takes to solve an instance to optimality when possible. We also give the total number of variables in the reformulated problem. In this way, we can try to draw conclusions as to what type of reformulation favors the global resolution of the problem, most notably whether our focus should be on designing reformulations that give the best lower bound, or whether the size of the reformulation should be more important.

5.1. Test instances

We use instances already tackled via quadratic convex reformulations in [17], and more recently in [14, 15] through linear reformulations and additional valid inequalities. These instances are known as *Bernasconi* or *LABS* instances, taken from [22]. These instances can be found online at either `https://www.minlplib.org` under the name *autocorr_bern*, or at `polip.zib.de` under the name *bernasconi*.

Furthermore, we generate instances with very diverse *densities* (different numbers of monomials for a given number of variables). In this endeavor, the Bernasconi instances are quite well-adapted as they have very varied densities. However, these instances have a rather specific structure, so we generated our own random instances with no specific structure and similar variety in density. We generate them in a similar way to [4]: the input is the number of variables n and the desired number of monomials $|P|$. We generate the monomials one after the other, by generating four integers in the set $\{0, \dots, n\}$. We then remove every zero and every number beyond its first appearance, in order to build monomials of varying degrees up to four, and to keep our instances multilinear since

$x_i^2 = x_i$. We associate a coefficient to every monomial, uniformly drawn as a real number from the interval $[-1, 1]$. We generated 33 instances, with $n \in \{20, 30, 40\}$, and $|P| = a \times n$ where $a \in \{2, 5, 10, 30, 50, 75, 100, 125, 150, 175, 200\}$. In our tables, these instances are given a generic name such as $R.n.a$, where R stands for random and n and a are the parameters used to generate them as previously explained.

To summarize, we have a total of 55 polynomials of degree 4, having up to 40 variables and about 15000 monomials.

5.2. Numerical results

We solve all mixed integer linear problems with Gurobi 9.1.2 [23]. All code is written in Julia 1.7.1 [1], using package JuMP 0.21.10 [16] for modelling purposes. All computations are run on a computer with a 256 GB RAM and Intel XEON W-2145 3.7 GHz processors.

5.2.1. Results of the reformulation step

We first compare results related to the reformulation step of the `ND-MinVar` and `MaxBound` methods, such as the time needed to compute the linear reformulation, the LP bound given by each reformulation, and the number of variables in each reformulation.

Tables 3 and 4 contain the results on the Bernasconi instances and the randomly generated instances, respectively. The first three columns give general information about the instance: Name, number of monomials $|P|$, and best known value called BKN. The number of variables is contained within the name of each instance (first number) and is therefore not repeated. The next two columns are dedicated to the time needed to compute the reformulation step of Algorithm 2 `ND-MinVar` and of Algorithm 1 `MaxBound`. The next three columns give the root gap yielded by each linear reformulation. The last two contain the number of variables present in each reformulation. Throughout all tables, short notations are used as follows: SL stands for standard linearization, MV stands for `ND-MinVar`, and MB for `MaxBound`.

From these results, we can observe that `MaxBound` and `ND-MinVar` produce far better bounds than the standard linearization across all instances. On randomly generated instances, the average gap given by the standard linearization is 518.9%, while the `ND-MinVar` and `MaxBound` methods yield average gaps of respectively 331.6% and 282.0%. On the Bernasconi instances, the differences between the methods is even clearer, with `ND-MinVar` cutting in half the standard linearization gap,

while `MaxBound` nearly divides it by three.

Let us now comment the number of variables in each linear reformulation (columns '# of variables' in Tables 3 and 4). As expected, the reformulation yielded by `MaxBound` always contains at least as many variables as `ND-MinVar`. It appears once more that this depends on instance structure, as the difference between the average number of variables is greater on randomly generated instances than on the Bernasconi instances. It is interesting to see that while gaps were more distant on the Bernasconi instances, the numbers of variables are closer on these instances and further apart on the randomly generated ones.

Table 3 also lets us see, in a way, the power that comes from Theorem 9. Indeed, on the Bernasconi instances, the structure is such that `ND-MinVar` yields a reformulation with exactly as many variables as the standard linearization. However, the set of linearization patterns allows for many more constraints linking these variables together, and the resulting gap is greatly reduced.

Instance			CPU time (s)		Root gap (%)			# of variables	
Name	$\|P\|$	BKN	MV	MB	SL	MV	MB	MV	MB
B.20.5	207	−416	1.1	1800	884.6	473.1	**312.8**	207	209
B.20.10	833	−2936	1.1	9.1	1428.6	750.5	**519.4**	833	850
B.20.15	1494	−5960	1.2	5.4	1564	841.2	**570.9**	1 494	1 502
B.25.6	407	−960	1.1	4.5	1116.7	606.7	**400**	407	411
B.25.13	1782	−8148	1.3	9	1518.5	771.7	**553.5**	1 782	1 819
B.25.19	3040	−14644	1.4	18.7	1636.3	877.5	**598.7**	3 040	3 162
B.25.25	3677	−10664	1.5	72.2	1659.3	955.2	**605.2**	3 677	3 799
B.30.4	223	−324	1.1	1800	633.3	348.2	**247.3**	223	223
B.30.8	926	−2952	1.2	19	1308.7	670.7	**473.4**	926	933
B.30.15	2944	−15744	1.4	21	1570.3	800.2	**573.4**	2 944	3 031
B.30.23	5376	−30460	1.8	52	1680.8	898.4	**615.7**	5 376	5 613
B.30.30	6412	−22888	1.9	69.7	1688	974.7	**616.2**	6 412	6 519
B.35.4	263	−384	1.1	1800	633.3	347.6	**247.2**	263	263
B.35.9	1381	−5108	1.2	19.8	1371.6	698.2	**497.7**	1 381	1 404
B.35.18	5002	−31160	1.7	33.3	1635.8	831.4	**598.8**	5 002	5 104
B.35.26	8347	−55288	2.4	64.3	1708.4	907.3	**626.4**	8 347	8 552
B.35.35	10252	−41068	2.7	117.7	1701.4	983.5	**621.3**	10 252	10 753
B.40.5	447	−936	1.1	1800	884.6	473.8	**311.9**	447	448
B.40.10	2053	−8248	1.3	21.8	1433.5	694.7	**521.4**	2 053	2 086
B.40.20	7243	−50576	2	184.8	1658.9	833.9	**607.5**	7 243	7 586
B.40.30	12690	−94872	3.2	137.3	1734.1	919.5	**636.3**	12 690	13 347
B.40.40	15384	−67964	3.5	513.3	1719.9	998.7	**628.6**	15 384	16 210
		Average	1.6	389.9	1416.9	757.6	**517.5**	4 108	4 265

Table 3: Reformulation step: CPU times, root gaps, numbers of variables of the different linear reformulations on the Bernasconi instances, up to 40 variables. In SL, # of variables equals $|P|$.

Instance			CPU time (s)		Root gap (%)			# of variables	
Name	$\|P\|$	BKN	MV	MB	SL	MV	MB	MV	MB
R.20.2	60	−6.05	1.1	1.3	24.3	18.1	**14.1**	94	103
R.20.5	120	−10.97	1.7	1800	66.5	40.5	**24.0**	175	208
R.20.10	220	−19.66	119.1	2.5	101.2	52.1	**38.7**	286	366
R.20.30	620	−27.56	1800	6.8	258.4	145.4	**135.1**	698	818
R.20.50	1018	−36.68	1800	12	366.8	208.3	**179.9**	1 087	1 314
R.20.75	1520	−36.21	1800	178.1	580	352.3	**291.5**	1 574	1 717
R.20.100	2018	−55.93	2.2	247.9	514.7	319.9	**242.3**	2 049	2 319
R.20.125	2516	−35.74	1.9	379.9	1026.5	687.5	**492.8**	2 535	2 728
R.20.150	3015	−34.63	1.9	1800	1287.7	888.4	**609.4**	3 029	3 449
R.20.175	3515	−54.97	2.5	1800	943.8	638.3	**456.2**	3 527	3 950
R.20.200	4010	−62.28	2.2	412.5	960	659.9	**467.2**	4 012	4 173
R.30.2	90	−6.67	1.1	1.3	4.2	0.8	**0**	155	179
R.30.5	180	−12.46	1.7	1800	108.4	74.8	**55.3**	293	337
R.30.10	330	−23.06	1800	22.8	147.3	98.1	**78.4**	481	593
R.30.30	929	−40.18	1800	22.8	302.8	185.5	**177.3**	1 142	1 487
R.30.50	1530	−58.44	1800	42	353.1	208.4	**199.9**	1 757	1 990
R.30.75	2279	−52.94	1800	64.9	591	372.7	**352.0**	2 514	2 806
R.30.100	3030	−59.54	1800	128.7	732.3	466.7	**422.6**	3 267	3 671
R.30.125	3780	−96.81	1800	867.8	587.6	349.4	**322.6**	3 986	4 340
R.30.150	4527	−84.92	1800	1164.2	800.4	508.6	**446.0**	4 724	5 599
R.30.175	5277	−115.81	1800	1800	686.7	423.5	**366.7**	5 466	6 041
R.30.200	6029	−106.88	1800	1800	869.7	563.1	**462.2**	6 193	6 605
R.40.2	120	−9.03	1.1	27.2	9.4	3.3	**1.7**	213	233
R.40.5	240	−22.92	1.4	1800	76.3	51.9	**37.9**	418	476
R.40.10	440	−30.59	1800	1800	138.1	89.5	**68.2**	707	841
R.40.30	1240	−46.88	1800	57.7	370.6	237.5	**227.5**	1 648	1 951
R.40.50	2038	−67.80	1800	77.9	420	261.2	**254.8**	2 500	2 810
R.40.75	3039	−88.82	1800	510	494.7	311.6	**298.8**	3 516	3 912
R.40.100	4040	−83.39	1800	190.8	713.6	454.3	**436.3**	4 548	4 988
R.40.125	5039	−106.36	1800	251.6	703.1	442.5	**423.6**	5 537	6 001
R.40.150	6037	−110.06	1800	456.2	826.4	525.8	**496.7**	6 527	7 100
R.40.175	7039	−107.21	1800	905.4	1020.2	648.6	**614.2**	7 631	8 209
R.40.200	8039	−119.75	1800	1054.1	1039.4	655.8	**610.9**	8 659	9 069
		Average	1150.8	651.5	518.9	331.6	**282.0**	2 756	3 042

Table 4: Reformulation step: CPU times, root gaps, numbers of variables of the different linear reformulations on randomly generated instances. In SL, the number of variables is equal to $|P|$.

We can see the influence of the polynomial density on the quality of the bounds: As shown in Figure 11, for a given number of variables, the relative gap grows significantly with the number of monomials, both in specially structured and non-structured instances. Striking examples can be found in our randomly generated instances, which we purposefully generated with great density differences in order to study this factor. On instance R.40.2, the gaps associated to the three methods are smaller than 10 %, while on instance R.40.200, all gaps are greater than 600 %.

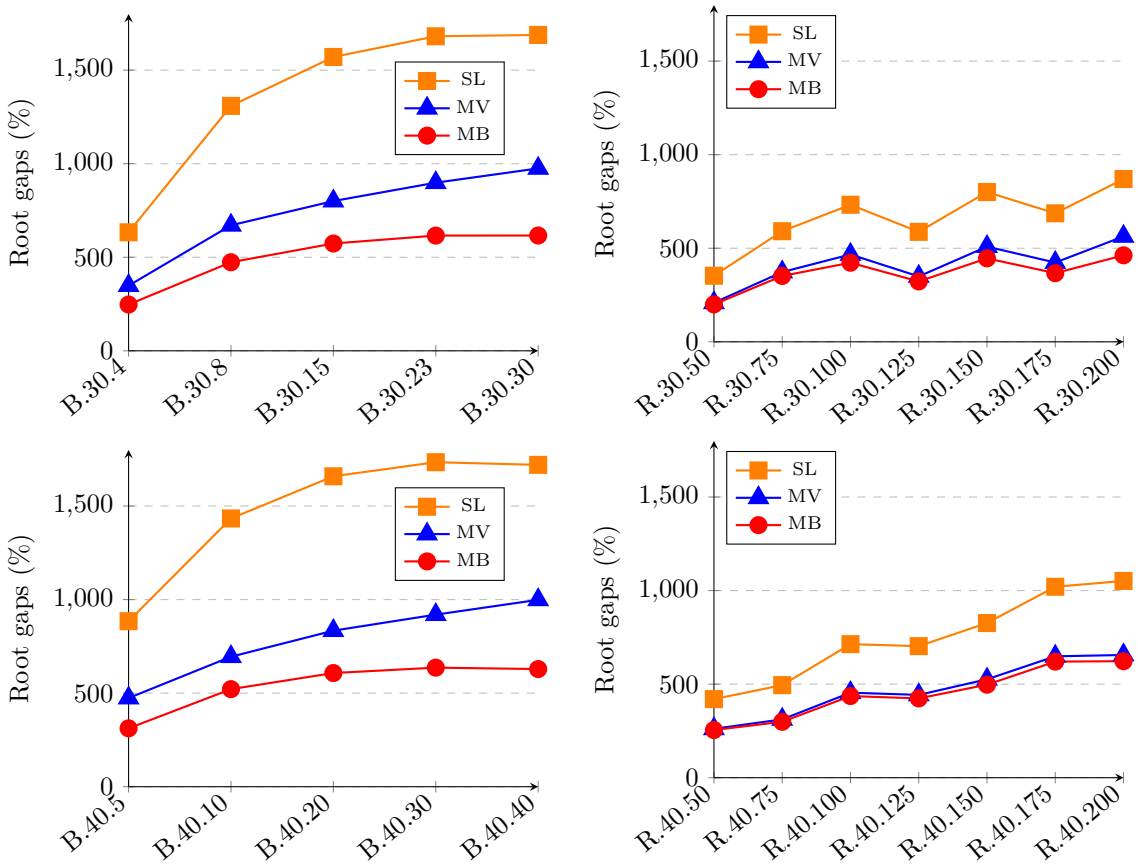


Figure 11: Root gaps (%) of all three methods on Bernasconi instances with 30 variables (top left) and 40 variables (bottom left), and randomly generated instances with 30 variables (top right) and 40 variables (bottom right).

We can also notice the influence of the structure of the instances. There is a clear difference in performance between randomly generated instances and the Bernasconi instances. We can find instances with the same number of variables and a similar number of monomials, but with different structure, and see that the performance of all three linear reformulations is greatly affected. As mentioned above, instances B.20.15 and R.20.75 provide a good comparison. In the case of the Bernasconi instances, this specific structure also has an effect on `ND-MinVar`. In fact, due to the structure of these instances, any solution is optimal to `ND-MinVar`. As a result, the `ND-MinVar` reformulation contains exactly the same number of variables as the standard linearization does. They only have different constraints.

Concerning the CPU time needed by the reformulation step: `MaxBound` and `ND-MinVar` require solving MILPs ($\widetilde{MB}$) and (MV). We can take a look at Tables 3 and 4 to better understand the effort needed to solve these preliminary MILPs. For `MaxBound`, in Column 5 of Table 4, we can see that it is not always easy to predict how easy it will be to solve ($\widetilde{MB}$). Part of these tables are represented in a more synthetic fashion in Figure 12. We do not include all the instances in these figures for visual clarity. Though the global tendency shows that the difficulty increases with the size of the problem, there are exceptions, most notably on instances such as B.30.4 and B.40.5 for Bernasconi instances, and R.30.5 and R.40.5 for randomly generated ones.

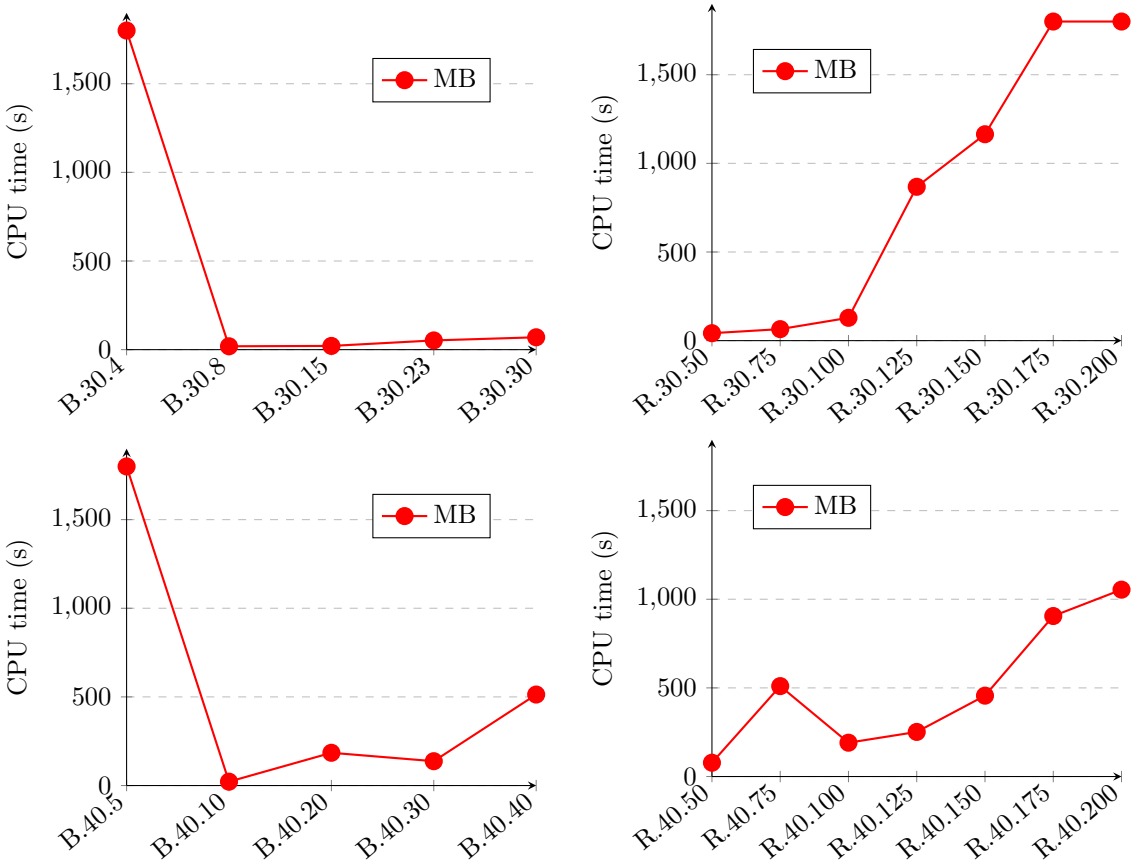


Figure 12: Time (s) needed to solve problem ($\widetilde{MB}$) on Bernasconi instances with 30 variables (top left) and 40 variables (bottom left), and randomly generated instances with 30 variables (top right) and 40 variables (bottom right), with a time limit of 1800s.

For `ND-MinVar`, a trend can be observed on instances R.20.2 to R.20.200: problem (MV) is hard to solve for a medium number of monomials, while being much easier to solve with either a lot of monomials, or very few of them. It is rather easy to understand why this problem is easy to solve on dense instances, as it is the same reason as for the Bernasconi instances: more monomials means that most, if not all necessary monomials already exist within the problem, and it makes it rather easy to find a reformulation that does not use any monomials that do not already exist. In the opposite case, on very sparse instances, it is easy to solve because there are not many intersections of size greater than or equal to two. Hence, we can generate all monomials corresponding to such intersections plus additional monomials to round out the reformulation, and probably reach a near-optimal solution quite easily. For instances with medium density, the problem becomes apparently harder to solve. A possible interpretation is that more intersections exist than in the sparse case, but not quite all of them, contrary to the dense case. In this case, for each monomial, the solver has to precisely evaluate how useful each possible smaller monomial would be to other monomials in the instance studied, and finding one of the best possible combinations is much more costly than in the sparse and dense cases.

In our randomly generated instances, we see this behavior very well on instances with 20 variables, but not so well on instances with 30 and 40 variables. This is to be expected, since we create instances where the number of monomials scales linearly with the number of variables, while the total number of possible monomials of degree smaller than four increases in $\mathcal{O}(n^4)$. More precisely, for n variables, we can create up to $\sum_{d=1}^{4} \binom{n}{d}$ such monomials. This means that our largest instances for each number of variables have very different densities. Instance R.20.200 is 64.6 % dense, while R.30.200 is only 18.8 % dense, and R.40.200 only 7.8 % dense. We can confirm the behavior observed on instances with 20 variables by generating a few instances of higher density for higher numbers of variables. Figure 13 gives a visual representation of the results on instances with 30 variables. The only thing that remains difficult to precisely determine is where the limit stands between so-called sparse, medium and dense instances.

It is also remarkable that it is generally for these instances of "medium" density that the `ND-MinVar` method yields a reformulation with an LP bound significantly closer to the one given by `MaxBound`. On these instances, with our chosen set of linearizations, finding the reformulation

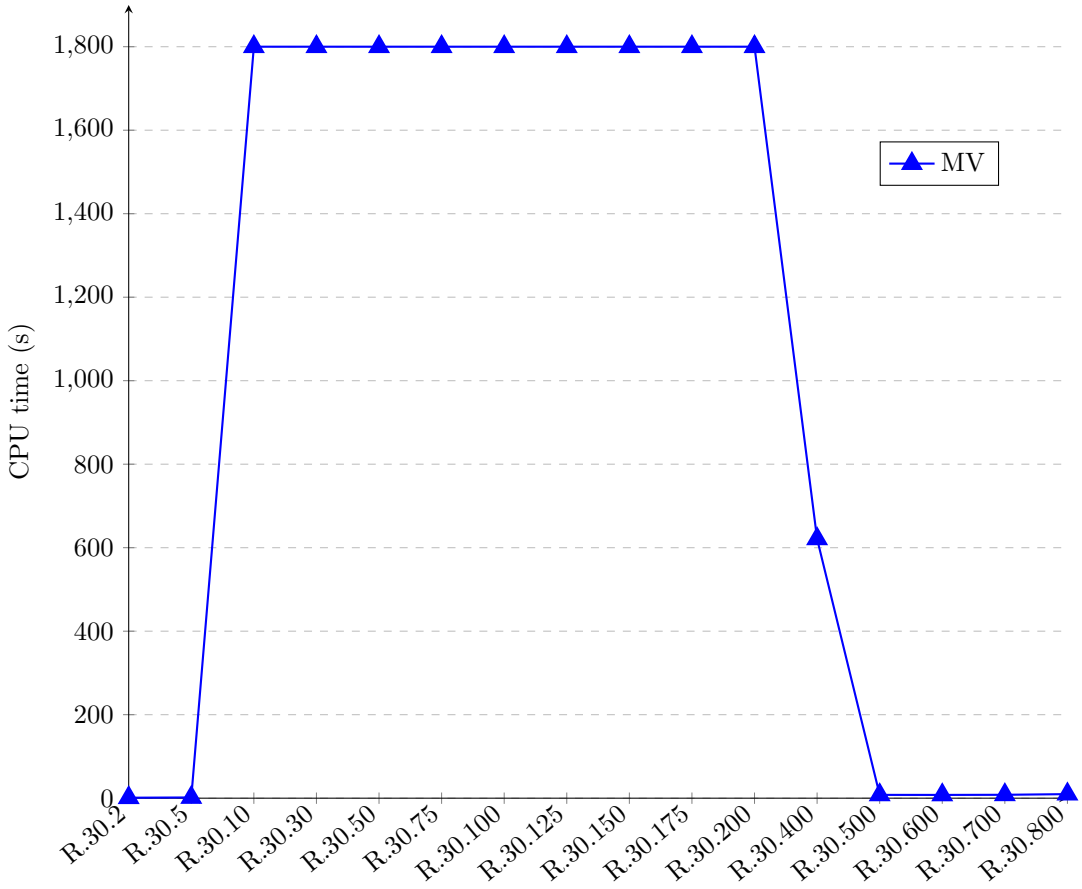


Figure 13: Time (s) needed to solve problem (MV), the problem of finding a linear reformulation with the fewest possible variables using non-dominated linearization patterns, on randomly generated instances with 30 variables (time limit: 1800s).

with the fewest variables and the one with the best LP bound may be similar goals.

5.2.2. Results of the resolution step

In this subsection we discuss the performance of all three linear reformulations when given to Gurobi 9.1.2 [23] with a time limit of three hours. Full results are given in Tables 5 and 6. We below discuss these results and provide a different representation of some of the data for better visualization.

A preliminary observation is that overall, given instances of similar sizes, all reformulations are much more efficient on the randomly generated instances, thus it is fair to say that these instances are easier to solve.

Concerning the resolution time and the number of nodes explored: on instances that were solved to optimality, we can compare the time needed by the different reformulations, as well as the number of explored nodes. The fastest method is the most efficient, but the number of nodes tell us additional information on how the reformulation behaves when given to the solver. We also take a look at the number of explored nodes per second. Bernasconi instances are hard to solve to optimality, so we will mostly turn to randomly generated instances for this part.

For most instances of small size, the difference in time between the three methods is small. Sometimes the standard linearization method remains the most efficient. However, when we turn to larger instances, `MaxBound` and `ND-MinVar` tend to perform better. Figure 14 shows this growing gap on a few randomly generated instances.

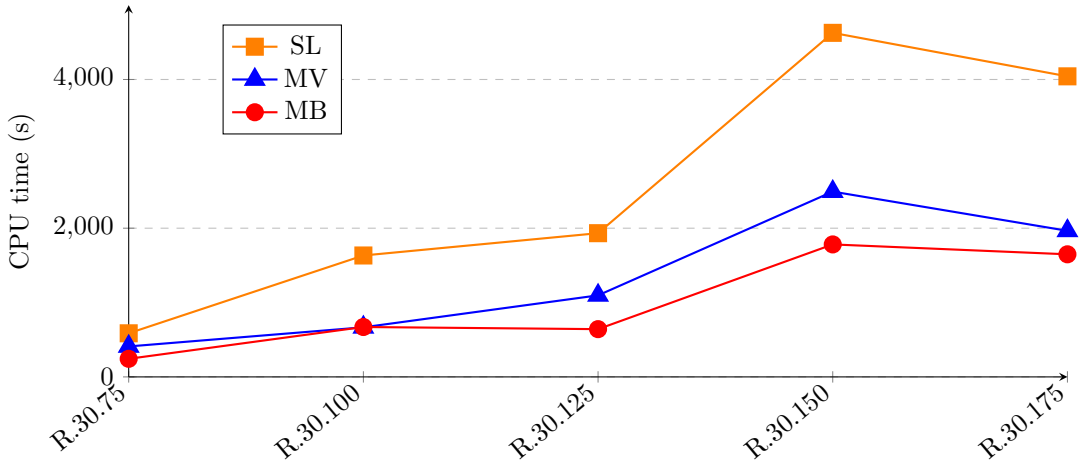


Figure 14: Time needed for the resolution step of the three linear reformulations on randomly generated instances with 30 variables.

Instance	Final gap (%)			CPU time (s)			Nodes explored		
Name	SL	MV	MB	SL	MV	MB	SL	MV	MB
B.20.5	0	0	0	**0.7**	1.1	1.3	11 913	6 049	20 618
B.20.10	0	0	0	**5.3**	8.6	7.9	0.1×10^6	0.1×10^6	0.1×10^6
B.20.15	0	0	0	**16.1**	25.7	23.4	0.1×10^6	0.1×10^6	0.1×10^6
B.25.6	0	0	0	**16.8**	28.1	19.5	0.8×10^6	1.4×10^6	0.9×10^6
B.25.13	0	0	0	174.8	**147.1**	185.4	2.1×10^6	1.6×10^6	1.8×10^6
B.25.19	0	0	0	**395.5**	530.7	481	2.4×10^6	2.4×10^6	1.9×10^6
B.25.25	0	0	0	**1672.3**	1857.2	1960.8	6.1	5.9	5.2
B.30.4	0	0	0	4.9	**2.4**	2.7	78 252	15 318	21 636
B.30.8	0	0	0	859.1	1209.9	**752.5**	24.1×10^6	29.7×10^6	18.2×10^6
B.30.15	0	0	0	6736.2	5887	**5693.6**	52.4×10^6	35.6×10^6	33.2×10^6
B.30.23	25.7	17	**13.4**	10800	10800	10800	31.2×10^6	27.3×10^6	24.6×10^6
B.30.30	132.8	74.9	**70.6**	10800	10800	10800	13.3×10^6	11.1×10^6	9.3×10^6
B.35.4	0	0	0	27.4	4.8	**3.8**	0.2×10^6	44 055	26 604
B.35.9	62	35.3	**22.9**	10800	10800	10800	127.7×10^6	174.3×10^6	173.3×10^6
B.35.18	99	56.3	**48.0**	10800	10800	10800	22.9×10^6	27.8×10^6	27×10^6
B.35.26	162.4	113	**88.5**	10800	10800	10800	6.6×10^6	7.2×10^6	6.8×10^6
B.35.35	396.1	223	**208.4**	10800	10800	10800	2.8×10^6	1.8×10^6	0.9×10^6
B.40.5	0	0	0	2706.5	**113.5**	128.3	19.9×10^6	1.3×10^6	1.6×10^6
B.40.10	80	102.8	**73.1**	10800	10800	10800	3.6×10^6	92×10^6	93.4×10^6
B.40.20	217.6	119.3	**95.0**	10800	10800	10800	10.2×10^6	13×10^6	11.6×10^6
B.40.30	401.1	347	**192.0**	10800	10800	10800	1.7×10^6	0.3×10^6	1.2×10^6
B.40.40	615.5	409.4	**321.3**	10800	10800	10800	523	0.2×10^6	780
Average	99.6	68.1	**51.5**	5482.5	5355.6	**5330.3**	14.9×10^6	19.7×10^6	18.7×10^6

Table 5: Resolution step: final gaps, CPU times and nodes explored on the Bernasconi instances, up to 40 variables. Time limit: 3 hours.

Instance	Final gap (%)			CPU time (s)			Nodes explored		
Name	SL	MV	MB	SL	MV	MB	SL	MV	MB
R.20.2	0	0	0	1.4	0.6	0.6	1	1	1
R.20.5	0	0	0	1.5	0.6	0.6	9	1	1
R.20.10	0	0	0	1.5	0.7	0.7	45	17	1
R.20.30	0	0	0	2.7	2.3	2.3	1 066	212	152
R.20.50	0	0	0	6.4	**4.5**	4.6	3 532	391	476
R.20.75	0	0	0	16.5	18.9	**13.3**	10 579	1 867	1 387
R.20.100	0	0	0	26.5	28.9	**24.2**	9 229	1 247	1 423
R.20.125	0	0	0	**58.6**	80.3	66	45 076	6 174	4 411
R.20.150	0	0	0	109.4	102.4	**87.5**	56 189	11 788	6 562
R.20.175	0	0	0	**96.0**	141.8	109.2	39 564	5 460	4 620
R.20.200	0	0	0	137.6	209.9	**110.1**	39 322	8 256	3 257
R.30.2	0	0	0	1.4	0.6	0.6	1	1	0
R.30.5	0	0	0	1.6	0.9	0.9	359	360	228
R.30.10	0	0	0	2.7	2.6	**1.6**	761	480	455
R.30.30	0	0	0	**28.8**	39.2	47.4	56 037	3 220	5 917
R.30.50	0	0	0	**84.4**	108.4	85.1	$0.1.0\times10^6$	9 825	11 442
R.30.75	0	0	0	586.5	409.7	**243.4**	0.7×10^6	0.2×10^6	0.1×10^6
R.30.100	0	0	0	1633	**667.0**	671.1	1.5×10^6	0.5×10^6	0.2×10^6
R.30.125	0	0	0	1932.9	1096.7	**642.3**	1.1×10^6	0.3×10^6	0.1×10^6
R.30.150	0	0	0	4627.2	2492	**1781.7**	2.7×10^6	0.9×10^6	0.2×10^6
R.30.175	0	0	0	4042.3	1964.9	**1648.8**	1.6×10^6	0.3×10^6	0.1×10^6
R.30.200	0	0	0	8028.1	3631.9	**3230.7**	3.4×10^6	1.3×10^6	0.2×10^6
R.40.2	0	0	0	1.4	0.6	0.6	1	1	1
R.40.5	0	0	0	1.8	**1.0**	1.1	144	269	249
R.40.10	0	0	0	10.7	**6.7**	7.3	2 657	1 944	1 389
R.40.30	0	0	0	1170.9	**467.1**	649.4	3.7×10^6	1.0×10^6	1.0×10^6
R.40.50	0	0	0	6698.5	4314.8	**1798.1**	11.4×10^6	3.4×10^6	1.0×10^6
R.40.75	61	0	0	10800	8408.1	**4626.6**	6.7×10^6	7.6×10^6	21.6×10^6
R.40.100	148.4	73.7	**55.1**	10800	10800	10800	4.3×10^6	6.3×10^6	4.5×10^6
R.40.125	171.4	85.2	**75.0**	10800	10800	10800	2.2×10^6	4.1×10^6	2.0×10^6
R.40.150	236.1	120.3	**105.4**	10800	10800	10800	1.5×10^6	2.8×10^6	1.1×10^6
R.40.175	369.8	210.4	**153.4**	10800	10800	10800	1.0×10^6	2.1×10^6	1.4×10^6
R.40.200	371.4	219.6	**162.1**	10800	10800	10800	0.7×10^6	1.0×10^6	0.7×10^6
Average	41.2	21.5	**16.7**	2851.8	2369.9	**2116.9**	1.3×10^6	1.0×10^6	0.4×10^6

Table 6: Resolution step: final gaps, CPU times and nodes explored on our randomly generated instances. Time limit: 3 hours.

The number of explored nodes and the rate at which they are explored are shown across a few instances in Figure 15. On randomly generated instances such as the ones shown in the figure, the hierarchy is quite clear: the standard linearization needs the most nodes to reach optimality, followed by `ND-MinVar` and `MaxBound` in that order. It is important to note that such a hierarchy is not reflected by the Bernasconi instances, which showcase great irregularities in the number of nodes needed to solve an instance.

In terms of nodes per second, the standard linearization reformulation yields the best node throughput, which is also expected since the formulation contains fewer variables and constraints than the others. The `MaxBound` reformulation takes the most time per node.

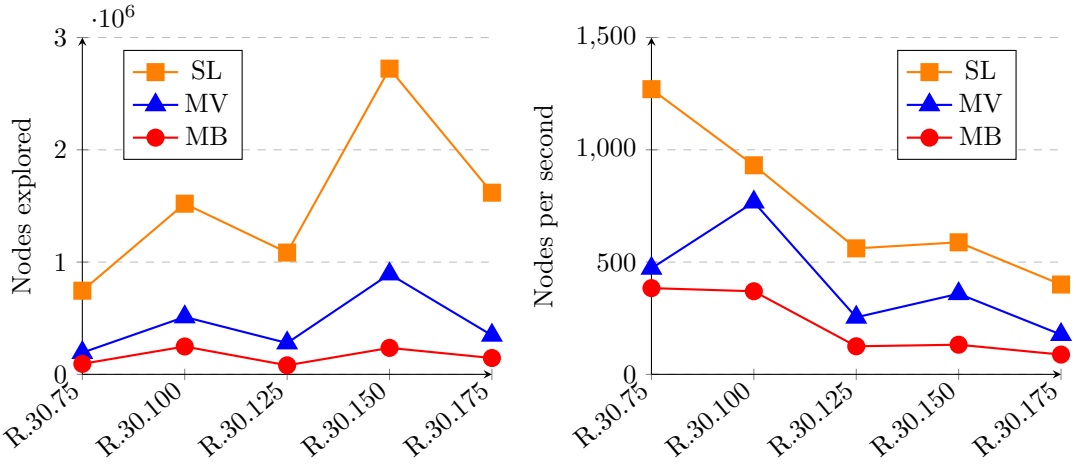


Figure 15: Nodes explored (left) and nodes per second (right) by all four methods on randomly generated instances with 30 variables.

Concerning final optimality gaps, Figure 16 shows the remaining gaps after 3 hours of computation time on instances with 40 variables (Bernasconi on the left, randomly generated on the right).

The general takeaway is that `MaxBound` outperforms the others on this criteria. `ND-MinVar` seems generally weaker than `MaxBound` but still performs better than the standard linearization.

5.3. Results of the bi-objective approach

The method used to solve the bi-objective optimization problem is not efficient enough to tackle the largest instances at our disposal. Indeed, while the method enables us to completely describe

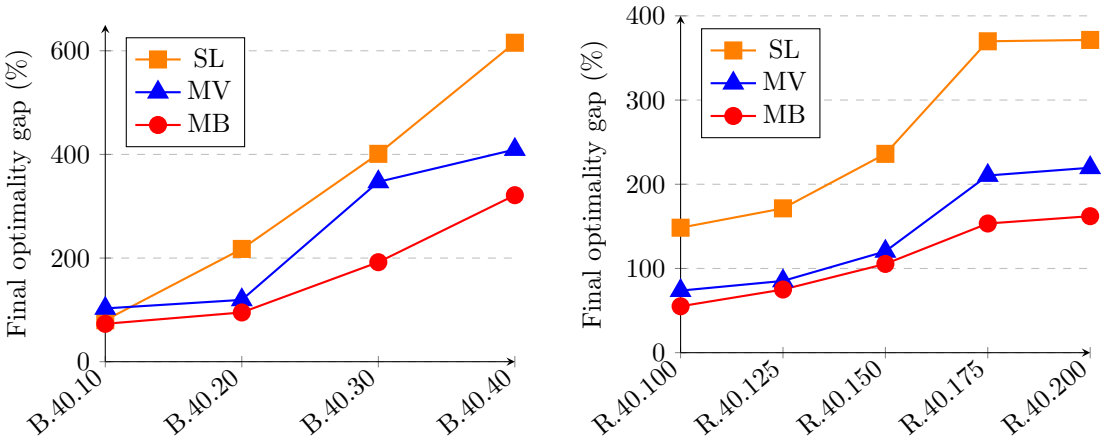


Figure 16: Final optimality gaps on Bernasconi instanes (left) and randomly generated instances (right) with 40 variables.

the set of Pareto-optimal solutions, tightening the constraint on the number of variables by just one each step is very slow. Therefore, in this section we focus on exploring the characteristics of the Pareto-optimal solutions rather than overall performance.

With this approach, we once again use all simple linearization patterns. Therefore, the choice of patterns corresponding to the standard linearization is a feasible solution of the bi-objective problem, containing the fewest possible variables. On the other end of the spectrum, `MaxBound` gives an extended linear reformulation with the best possible LP bound. Hence, the Pareto frontier exists in a rectangle as depicted in Figure 17.

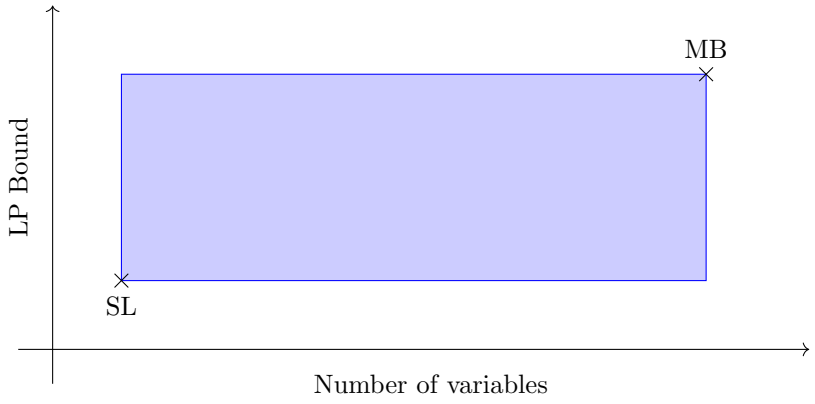


Figure 17: Domain of non-dominated solutions to the bi-objective problem.

An interesting question is the shape of the Pareto frontier. On (Ex), we know that there are two non-dominated solution points. In general, on randomly generated instances we seem to obtain many non-dominated solutions. We show the Pareto frontier for instance R.20.30 in Figure 18.

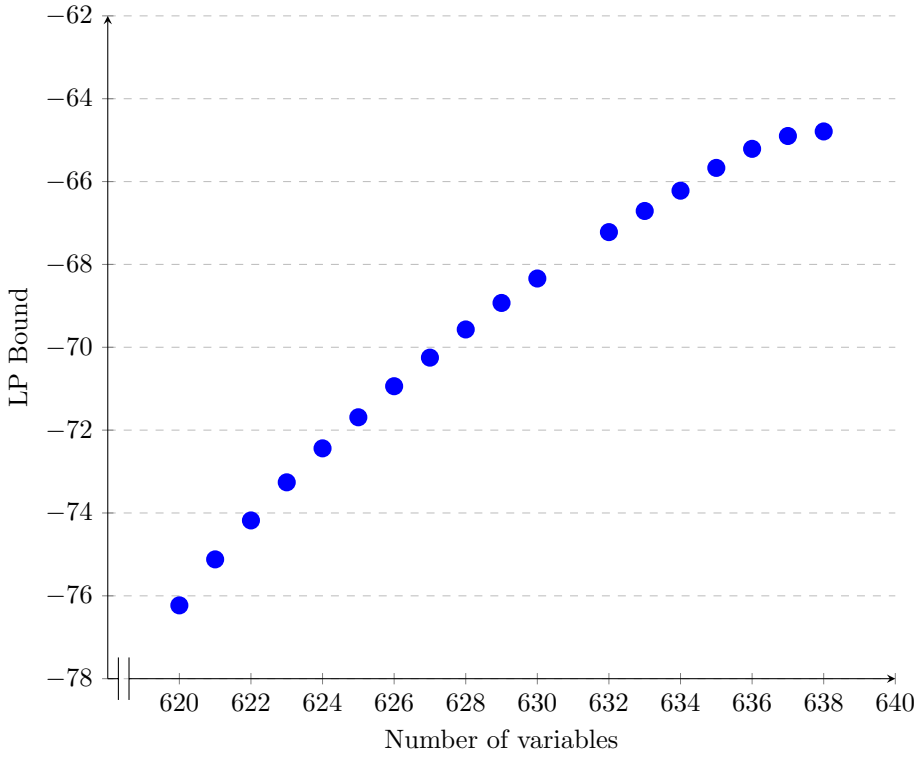
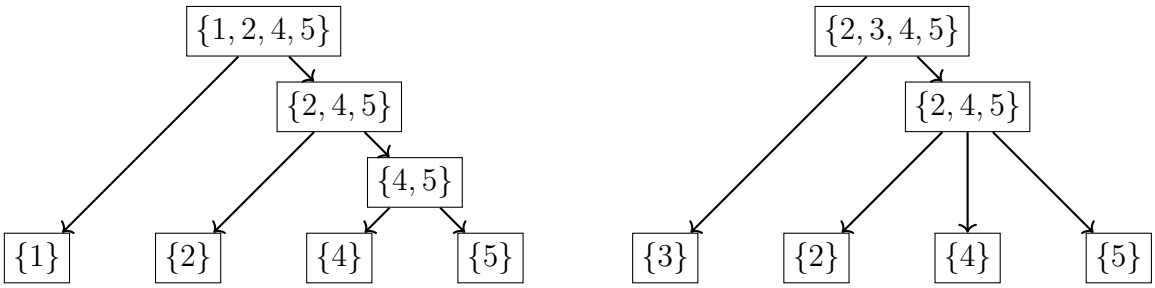


Figure 18: Set of non-dominated solutions to the bi-objective problem on instance R.20.30.

On the contrary, on the Bernasconi instances, in all cases where we were able to compute the set of non-dominated solutions, there exists exactly one non-dominated solution corresponding to the top left corner of the rectangle in Figure 17. An intuitive explanation for this fact is once again due to the structure of the instances. Indeed, these instances are such that for any monomial $M \in P$, then any subset $s \subset M$ also belongs to P. Hence, any linearization pattern uses only variables that correspond to monomials of P. Therefore, the only way to obtain more variables than the number of variables in the standard linearization is to decompose a monomial in multiple different ways. For example, if we consider monomials $\{1, 2, 4, 5\}$ and $\{2, 3, 4, 5\}$ and choose to use the linearization patterns as in Figure 19, then we obtain two different variables representing $\{2, 4, 5\}$. With this in mind, the results on the Bernasconi instances suggest that there is no interest to introduce two different variables in order to represent the same monomial.



(a) A linearization pattern for monomial $\{1, 2, 4, 5\}$.

(b) A linearization pattern for monomial $\{2, 3, 4, 5\}$.

Figure 19: A choice of linearization patterns for two monomials of problem (Ex).

6. Conclusion

We present two new methods to build linear reformulations to polynomial binary optimization problems. Our linear reformulations are based on a linearization graph, obtained by concatenation of linearization patterns of the monomials. Method `MaxBound` focuses on the quality of the LP bound of the linear reformulation. We show that obtaining a reformulation with the best possible bound can be derived from the solution of a MILP. We also show a domination property between linearization patterns that allows us to reduce the number of potential patterns, and therefore the size of this MILP. Method `ND-MinVar` considers the reduced set of patterns and focuses on finding a linear reformulation with as few additional variables as possible. It uses a different MILP to derive this linear reformulation.

We evaluate the performances of `MaxBound` and `ND-MinVar` and compare them to the standard linearization. We use two families of difficult instances, containing polynomials of degree 4: the Bernasconi instances and randomly generated instances. We observe that `MaxBound` drastically reduces the root gap compared to the standard linearization while adding a reasonable amount of variables, especially for the structured Bernasconi instances. On the other hand, `ND-MinVar` uses a smaller number of variables, and unexpectedly grants a significant improvement of the LP bound over the standard linearization. Finally, the bi-objective problem hybridizing `MaxBound` and `ND-MinVar` enables us to generate interesting linear reformulations, but is difficult to solve.

In theory, `MaxBound` and `ND-MinVar` can be applied to polynomials of higher degree, but the explosion of the number of linearization patterns would give rise to MILPs of huge size. In the future, we may look for ways to circumvent this issue, either by use of sophisticated mathematical programming methods such as decomposition methods, or by use of heuristics. Moreover, reformulations obtained using `MaxBound` or `ND-MinVar` may provide helpful information on the structure of a given instance of BPO and on which links between monomials can be exploited.

References

[1] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98, 2017.

[2] E. Boros, Y. Crama, and E. Rodríguez-Heck. Compact quadratizations for pseudo-boolean functions. *Journal of combinatorial optimization*, 39(3):687–707, 2020.

[3] E. Boros and P. L. Hammer. Pseudo-boolean optimization. *Discrete applied mathematics*, 123(1-3):155–225, 2002.

[4] C. Buchheim and C. D'Ambrosio. Monomial-wise optimal separable underestimators for mixed-integer polynomial optimization. *Journal of Global Optimization*, 67(4):759–786, 2017.

[5] C. Buchheim and G. Rinaldi. Efficient reduction of polynomial zero-one optimization to the quadratic case. *SIAM Journal on Optimization*, 18(4):1398–1413, 2008.

[6] Y. Crama, P. Hansen, and B. Jaumard. The basic algorithm for pseudo-boolean programming revisited. *Discrete Applied Mathematics*, 29(2-3):171–185, 1990.

[7] Y. Crama and E. Rodríguez-Heck. A class of valid inequalities for multilinear 0–1 optimization problems. *Discrete Optimization*, 25:28–47, 2017.

[8] E. Dalkiran and L. Ghalami. On linear programming relaxations for solving polynomial programming problems. *Computers & Operations Research*, 99:67–77, 2018.

[9] E. Dalkiran and H. D. Sherali. Rlt-pos: Reformulation-linearization technique-based optimization software for solving polynomial programming problems. *Mathematical Programming Computation*, 8(3):337–375, 2016.

[10] A. Del Pia and S. Di Gregorio. Chvátal rank in binary polynomial optimization. *INFORMS Journal on Optimization*, 3(4):315–349, 2021.

[11] A. Del Pia and S. Di Gregorio. On the complexity of binary polynomial optimization over acyclic hypergraphs. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2684–2699. SIAM, 2022.

[12] A. Del Pia and A. Khajavirad. The multilinear polytope for acyclic hypergraphs. *SIAM Journal on Optimization*, 28(2):1049–1076, 2018.

[13] A. Del Pia and A. Khajavirad. The running intersection relaxation of the multilinear polytope. *Mathematics of Operations Research*, 2021.

[14] A. Del Pia, A. Khajavirad, and N. V. Sahinidis. On the impact of running intersection inequalities for globally solving polynomial optimization problems. *Mathematical programming computation*, 12(2):165–191, 2020.

[15] A. Del Pia and M. Walter. Simple odd-cycle inequalities for binary polynomial optimization. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 181–194. Springer, 2022.

[16] I. Dunning, J. Huchette, and M. Lubin. Jump: A modeling language for mathematical optimization. *SIAM review*, 59(2):295–320, 2017.

[17] S. Elloumi, A. Lambert, and A. Lazare. Solving unconstrained 0-1 polynomial programs through quadratic convex reformulation. *Journal of Global Optimization*, pages 1–18, 2021.

[18] R. Fortet. Applications de l'algebre de boole en recherche opérationnelle. *Revue Française de Recherche Opérationnelle*, 4(14):17–26, 1960.

[19] B. González-Rodríguez, J. Ossorio-Castillo, J. González-Díaz, Á. M. González-Rueda, D. R. Penas, and D. Rodríguez-Martínez. Computational advances in polynomial optimization: Raposa, a freely available global solver. *Journal of Global Optimization*, pages 1–28, 2022.

[20] C. Hojny, M. E. Pfetsch, and M. Walter. Integrality of linearizations of polynomials over binary variables using additional monomials. *arXiv preprint arXiv:1911.06894*, 2019.

[21] G. Kochenberger, J.-K. Hao, F. Glover, M. Lewis, Z. Lü, H. Wang, and Y. Wang. The unconstrained binary quadratic programming problem: a survey. *Journal of Combinatorial Optimization*, 28(1):58–81, 2014.

[22] F. Liers, E. Marinari, U. Pagacz, F. Ricci-Tersenghi, and V. Schmitz. A non-disordered glassy model with a tunable interaction range. *Journal of Statistical Mechanics: Theory and Experiment*, 2010(05):L05003, 2010.

[23] G. Optimization. Llc.,"gurobi optimizer reference manual," 2021, LLC.

[24] M. Padberg. The boolean quadric polytope: some characteristics, facets and relatives. *Mathematical Programming*, 45:139–172, 1989.

[25] E. Schröder. Vier kombinatorische probleme. *Z. Math. Phys.*, 15:361–376, 1870.

[26] A. Verma and M. Lewis. Optimal quadratic reformulations of fourth degree pseudo-boolean functions. *Optimization Letters*, 14(6):1557–1569, 2020.