RC25318 (WAT1210-050) October 12, 2012
Mathematics

IBM Research Report

Global Optimization of Nonconvex Problems with Multilinear Intermediates

Xiaowei Bao
Sabre Airline Solutions

Aida Khajavirad
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 208
Yorktown Heights, NY 10598
USA

Nikolaos V. Sahinidis
Department of Chemical Engineering
Carnegie Mellon University

Mohit Tawarmalani
Krannert School of Management
Purdue University

Global optimization of nonconvex problems with multilinear intermediates

Xiaowei Bao · Aida Khajavirad ·
Nikolaos V. Sahinidis · Mohit Tawarmalani

Received: date / Accepted: date

Abstract We consider global optimization of nonconvex problems containing multilinear functions. It is well known that the convex hull of a multilinear function over a box is polyhedral, and the facets of this polyhedron can be obtained by solving a linear optimization problem (LP). When used as cutting planes, these facets can significantly enhance the quality of conventional relaxations in general-purpose global solvers. However, in general, the size of this LP grows exponentially with the number of variables in the multilinear function. To cope with this growth, we propose a graph decomposition scheme that exploits the structure of a multilinear function to decompose it to lower-dimensional components, for which the aforementioned LP can be solved very efficiently by employing a customized simplex algorithm. We embed this cutting plane generation strategy at every node of the branch-and-reduce global solver `BARON`, and carry out an extensive computational study on quadratically constrained quadratic problems, multilinear problems, and polynomial optimization problems. Results show that the proposed multilinear cuts enable `BARON` to solve many more problems to global optimality and lead to an average 60% CPU time reduction.

Keywords Multilinear functions · Global optimization · Convex envelope · Polyhedral relaxations

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

Xiaowei Bao
Sabre Airline Solutions, e-mail: `xiaoweib@gmail.com`.

Aida Khajavirad
Mathematical Sciences Department, IBM T. J. Watson Research Center, e-mail: `aidakhajavi@us.ibm.com`.

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

Mohit Tawarmalani
Krannert School of Management, Purdue University, e-mail: `mtawarma@purdue.edu`.

1 Introduction

We consider a nonconvex optimization problem, where the objective function and/or constraints contain multilinear subexpressions. A function $L(x_1, \ldots x_n)$ is said to be *multilinear* if its restriction to each variable x_j, $j = 1, \ldots, n$ is a linear function; i.e.,

$$L(x_1, \ldots, x_n) = \sum_{T \in \Omega} c_T \prod_{j \in T} x_j, \tag{1}$$

where Ω is a collection of subsets of $\{1, \ldots, n\}$, $c_T \in \mathbb{R} \setminus \{0\}$, and $x_j \in \mathbb{R}$. We focus on the case where $L(x)$ is defined over a box $x \in \mathcal{H} = \prod_{j=1}^n [l_j, u_j]$, which is a case of central importance in rectangular branch-and-bound algorithms. We further assume that $\mathcal{H}$ is bounded and has a nonempty interior; i.e., $-\infty < l_j < u_j < +\infty$ for all $j = 1, \ldots, n$. Throughout the paper, we refer to each product $\prod_{j \in T} x_j$ in (1) as a multilinear term.

Multilinear functions, including bilinear functions as special cases, are building blocks of many difficult nonconvex problems, such as quadratic programs (QPs), quadratically constrained quadratic programs (QCQPs), and polynomial programs. Building tight convex relaxations for multilinear functions has been the subject of extensive research by the mathematical programming community for over four decades now [19, 1,5,6,24,30,25,22,31,2]. It has been shown that the quality of these relaxations significantly affects the convergence rate of global optimization algorithms [2]. Given a nonconvex function $z = \phi(x)$ over a convex set $\mathcal{C}$, the tightest convex relaxation of the set $\mathcal{S} = \{(x, z) : z = \phi(x),\ x \in \mathcal{C}\}$ is obtained by replacing $\phi(x)$ by its convex and concave envelopes (cf. Chapter 2 of [35] for an exposition). Recall that the convex envelope of ϕ over $\mathcal{C}$, denoted by $\mathrm{conv}_{\mathcal{C}}\phi$, is the greatest convex underestimator of ϕ over $\mathcal{C}$. Similarly, the concave envelope of ϕ over $\mathcal{C}$ is the lowest concave function minorized by ϕ over $\mathcal{C}$, and we denote it by $\mathrm{conc}_{\mathcal{C}}\phi$. It is well known that the convex and concave envelopes of a multilinear function over a box are polyhedral and can be fully characterized by the extreme points of the box (cf. [24]). Let $\mathrm{vert}(\mathcal{H}) = \{v^i\}_{i \in I}$, $I = \{1, \ldots, 2^n\}$ denote the set of vertices of $\mathcal{H}$, and let λ_i be a convex multiplier associated with v^i for all $i \in I$. It follows that, for any $x \in \mathcal{H}$, the value of the convex envelope of $L(x)$ over $\mathcal{H}$ can be obtained by solving the following minimizing LP:

$$\mathrm{conv}_{\mathcal{H}} L(x) = \min_{\lambda} \left\{ \sum_{i \in I} \lambda_i L(v^i) : x = \sum_{i \in I} \lambda_i v^i,\ \sum_{i \in I} \lambda_i = 1,\ \lambda_i \geq 0, \forall i \in I \right\}. \tag{2}$$

Similarly, the value of the concave envelope of $L(x)$ over $\mathcal{H}$ is equal to the optimal value of the following maximizing LP:

$$\mathrm{conc}_{\mathcal{H}} L(x) = \max_{\lambda} \left\{ \sum_{i \in I} \lambda_i L(v^i) : x = \sum_{i \in I} \lambda_i v^i,\ \sum_{i \in I} \lambda_i = 1,\ \lambda_i \geq 0, \forall i \in I \right\}. \tag{3}$$

The above descriptions for the envelopes of a multilinear function grow exponentially with the number of variables (n), limiting their direct application in practice to multilinears with a small number of variables. In fact, it has been shown that finding the convex envelope of a multilinear function over the unit hypercube is an NP-hard problem, in general [5]. For several classes of multilinears with special structures, explicit characterizations of the envelopes are available in the literature. These results address bilinear terms [1] and trilinear terms [20,21] over a box, special forms of bilinear functions over the unit hypercube [24], special forms of multilinear functions over the unit hypercube and some discrete sets [30], bilinear and polynomial covering sets with certain sign restrictions and no upper bounds on variables [33], and submodular (supermodular) functions over a box and over various polyhedral subdivisions of a box [34,32]. For bounding a general multilinear function, however, a common practice is to utilize a termwise scheme in which each multilinear term is relaxed by a recursive application of bilinear envelopes [19,6,25,36]. This so-called *standard linearization* is simple to implement and has been employed in the general-purpose global solvers `BARON` [26], `LindoGlobal` [17], and `Couenne` [3]. Crama [6] derives necessary and sufficient conditions under which the standard linearization is equivalent to the convex hull relaxation. However, in general, the standard linearization can lead to very poor bounds and, for dense multilinears in particular, utilizing the convex hull relaxation has shown to be highly beneficial [2,18].

Bao et al. [2] consider the generation of multiterm polyhedral relaxations for nonconvex QCQPs. They show that adding the cuts corresponding to the facets of the envelopes of multiple bilinear terms at the root node of `BARON` significantly enhances the convergence rate of `BARON`'s branch-and-reduce algorithm. Motivated by this initial success, in this paper, we introduce a new class of cutting planes, namely, multilinear cutting planes for multiple multilinear terms based on the convex and concave envelopes of the multilinear function, and describe its implementation in a branch-and-bound algorithm for general nonconvex problems. With the goal of generating strong cuts that are cheap to compute, we propose a novel decomposition scheme that exploits the structure of a multilinear function to decompose it to lower-dimensional multilinears, for which the facets of the envelopes can be computed very efficiently by utilizing a customized simplex algorithm. To exploit the local information regarding bounds on variables for cut generation, as well as to utilize the multilinear cuts for local feasibility-based and optimality-based range reductions, we embed the proposed cutting plane generation technique at every node of the branch-and-reduce algorithm. Extensive computational results are presented for globally solving various sets of QCQPs, multilinear problems, and polynomial optimization problems. Results show that the incorporation of the multilinear cuts in `BARON` reduces the average CPU time and number of nodes in the search tree by 60% and 90%, respectively.

The remainder of the paper is structured as follows. We first review some basic material on constructing the convex envelopes of multilinear functions in Section 2. In Section 3, we describe the decomposition scheme and detail how the proposed cut generation algorithm is embedded in a branch-and-bound global solver. Section 4 describes computational experience with the algorithm and compares the proposed multiterm cuts with conventional termwise relaxations. Finally, conclusions are offered in Section 5.

2 Relaxation of multilinear functions and cutting plane generation

In this section, we review some preliminary material on constructing the envelopes of multilinear functions, and present a cutting plane generation scheme to generate a facet of the convex envelope that separates the epigraph of the convex envelope from a pre-specified point. Analogous results for the concave envelope can be established in a similar manner. While we focus on multilinear functions over hyper-rectangles, most of our results are valid for any function whose envelope over a polytope can be finitely generated (cf. [31] for an introduction to functions with polyhedral envelopes). For several classes of multilinears with special structures, explicit characterizations of the convex and/or concave envelopes are known [24,30,20,21,34], or it has been shown that the recursive factorable relaxation provides the convex envelope of the multilinear [6, 25,18]. In the following, we consider multilinear functions for which (i) no analytical description for the envelope is available and (ii) the envelope is strictly tighter than the term-wise recursive relaxation.

As we discussed in the previous section, the convex envelope of a multilinear function over a box can be constructed by solving the LP given by (2). It then follows that, using the dual representation of this LP, we can construct nonvertical facets of $\mathrm{conv}_{\mathcal{H}}L(x)$, as stated in the following theorem:

Theorem 1 *(Theorem 2.4 in [2]) Let $g(a,b)$ be an affine function, with $a \in \mathbb{R}^n$ and $b \in R$. Then, $z = {a^*}^T x + b^*$ defines a non-vertical facet of the convex envelope of the multilinear function $L(x)$ over $\mathcal{H} = \prod_{j=1}^n [l_j, u_j]$ if and only if (a^*, b^*) is a basic feasible solution of the following linear optimization problem:*

$$
\text{(F1)} \quad \begin{aligned} &\max\ g(a,b) \\ &\text{s.t.}\ \ a^T v^i + b \le L(v^i),\ \forall v^i \in \mathrm{vert}(\mathcal{H}) \\ &\qquad\ a \in \mathbb{R}^n,\ \ b \in \mathbb{R}. \end{aligned}
$$

Any affine underestimator of $L(x)$ is associated with a feasible solution of (F1), and an optimal solution of (F1) defines a facet of the epigraph of $\mathrm{conv}_{\mathcal{H}}L(x)$. The objective function $g(a,b)$ can be any affine function, which leaves the flexibility to generate proper facets for obtaining sharp relaxations. Based on this fact, we next present a scheme to generate strong cuts that separate a given point from the convex hull of a multilinear function.

Consider the set $\mathcal{S} = \{(x,z) : z = L(x),\ x \in \mathcal{H}\}$. We assume that $z = L(x)$ in an intermediate constraint that is generated after a factorable reformulation of a general nonconvex optimization problem (cf. [36]). Suppose that an initial relaxation of this nonconvex problem is constructed and let (x^*, z^*) denote the relaxation solution in the space of (x,z). Assume $z^* < L(x^*)$. If (x^*, z^*) is in the epigraph of the convex envelope of $L(x)$, then $z^* \ge \mathrm{conv}_{\mathcal{H}}L(x^*)$. Otherwise, there exists a facet $h(x)$ of $\mathrm{conv}_{\mathcal{H}}L(x)$ that cuts off the relaxation solution from the feasible region, i.e., $z^* < h(x^*)$. It then follows that, to find or examine the existence of such a facet, it suffices to solve the following LP:

$$
\text{(F2)} \quad \begin{aligned} &\max\ a^T x^* + b \\ &\text{s.t.}\ \ a^T v^i + b \le L(v^i),\ \forall v^i \in \mathrm{vert}(\mathcal{H}) \\ &\qquad\ a \in \mathbb{R}^n,\ \ b \in \mathbb{R}. \end{aligned}
$$

By Theorem 1, any basic feasible solution of (F2) defines a facet of $\mathrm{conv}_{\mathcal{H}}L(x)$. In addition, every feasible solution of (F2) denoted by $(\tilde{a}, \tilde{b})$, with an objective value greater than z^* yields a valid inequality $\tilde{a}^T x + \tilde{b} \le z$ that cuts off the relaxation solution (x^*, z^*). Defining the violation measure as $d = a^T x^* + b - z^*$, we conclude that an optimal solution of (F2) produces a facet of $\mathrm{conv}_{\mathcal{H}}L(x)$ that is violated by the relaxation solution by the largest amount. Moreover, if the optimal value of (F2) is less than or equal to z^*, then we conclude that (x^*, z^*) belongs to the epigraph of the convex envelope of $L(x)$. Similarly, we can examine whether (x^*, z^*) can be separated from the hypograph of the concave envelope of $L(x)$, by solving an analogous LP. Obviously, if $L(x^*) \leq z^*$ (resp. $L(x^*) \geq z^*$), then (x^*, z^*) is in the epigraph of $\text{conv}_{\mathcal{H}} L(x)$ (resp. hypograph of $\text{conc}_{\mathcal{H}} L(x)$). Therefore, one needs to solve at most one separation problem to generate the desired cutting plane.

These cutting planes capture the strength of the convex and concave envelopes of the entire multilinear function, while bypassing the requirement of describing the entire envelope, which may be impractical in terms of the computational cost and memory requirements for large problems. Furthermore, the cutting plane generation strategy is flexible and can be embedded in branch-and-bound algorithms with adjustable configurations, as discussed in the sequel.

3 Implementation in a branch-and-bound algorithm

In this section, we describe an efficient implementation of multilinear cutting planes introduced in Section 2. The implementation is integrated within the global solver `BARON`, which relies on a branch-and-reduce framework [26]. The main components of our implementation are:

(i) a recognition tool that identifies multilinear functions in the original formulation as well as hidden multilinear structures in a general nonconvex problem,
(ii) a novel decomposition scheme to construct a collection of low-dimensional dense components of a multilinear function to manage the size of the separation problem,
(iii) a customized simplex algorithm for solving the separation problem, and
(iv) an efficient cut generation scheme that is embedded at every node of the branch-and-reduce algorithm.

Next, we detail each of these four components.

3.1 Identification of multilinear structures in general nonconvex problems

To construct tight relaxations for general nonconvex problems, we identify multilinear functions that are present in the original problem, as well as intermediate multilinear structures that are introduced by the factorable reformulation. In the factorable programming module in `BARON`, each factorable expression is recursively decomposed into linear, logarithmic, exponential, monomial, and bilinear expressions. The linear expressions are stored in a sparse matrix data structure while the nonlinear expressions are stored in arrays linking the dependent and independent variables. Here, we provide a brief formal description of the parts of this recursive reformulation that we need in order to present the developments of the current paper. The reader is referred to [36] for additional details.

Consider a factorable optimization problem of the following form:

$$
\begin{aligned}
\text{(P)} \qquad \min \quad & f_0(x) \\
\text{s.t.} \quad & f_i(x) \leq 0, \ \forall i = 1, \ldots, q \\
& Ax \leq b \\
& x \in \mathbb{R}^n.
\end{aligned}
$$

We assume that all linear constraints of the above problem are embedded in $Ax \leq b$, where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, and $f_i(x)$, $i \in Q = \{0, 1, \ldots, q\}$ are nonlinear factorable functions. Algorithm 1 outlines a factorable reformulation of Problem (P) as implemented in `BARON`. In this algorithm, original variables are denoted by x, while intermediate variables are denoted by y. For notational simplicity, in the following, we denote by x all variables in the reformulated problem.

Algorithm 1 Factorable reformulation in `BARON`

Given a collection of nonlinear factorable functions $\mathcal{F} = \{f_i(x), x \in \mathbb{R}^n, i \in Q\}$.
Initialize the number of intermediate variables $j = 0$ and the list of intermediate relations $\mathcal{M} = \emptyset$.
For each function $f_i(x) \in \mathcal{F}$:
  If f_i is a basic univariate function (i.e., monomial, power, or logarithm), then
    update $j \leftarrow j + 1$ and add the univariate relation $y_j = f_i(x)$ to $\mathcal{M}$
  else if $f_i = g(x)/h(x)$, then:
    update $j \leftarrow j + 3$ and introduce the variables y_{j-2}, y_{j-1}, and y_j
    let $y_{j-2} = g(x)$ and $y_{j-1} = h(x)$; add $h(x)$ and $g(x)$ to $\mathcal{F}$
    add the bilinear relation $y_{j-2} = y_{j-1} y_j$ to $\mathcal{M}$
  else if $f_i = \prod_{k=1}^{l} g_k(x)$, then:
    for $k = 1$ to l,
      update $j \leftarrow j + 1$, let $y_j = g_k(x)$, and add $g_k(x)$ to $\mathcal{F}$
    end of for
    update $j \leftarrow j + 1$ and add the bilinear relation $y_j = y_{j-l} y_{j-l+1}$ to $\mathcal{M}$
    for $k = 3$ to l,
      update $j \leftarrow j + 1$ and add the bilinear relation $y_j = y_{j-1} y_{j-l+1}$ to $\mathcal{M}$
    end of for
  else if $f_i = \sum_{k=1}^{l} a_k g_k(x)$, then:
    for $k = 1$ to l,
      update $j \leftarrow j + 1$, let $y_j = g_k(x)$, and add $g_k(x)$ to $\mathcal{F}$
    end of for
    update $j \leftarrow j + 1$ and add the linear relation $y_j = \sum_{k=1}^{l} a_k y_{j-k}$ to $\mathcal{M}$
  else if $f_i = h(g(x))$, then:
    update $j \leftarrow j + 2$ and introduce the variables y_{j-1}, and y_j
    let $y_{j-1} = g(x)$ and $y_j = h(y_{j-1})$; add $g(x)$ and $h(y_{j-1})$ to $\mathcal{F}$
  end of if
end of for

As an example of this reformulation, consider the polynomial

$$f(x) = x_1^2 x_2 x_3 + x_1^2 x_2^2 + x_1 x_2^2 x_3. \tag{4}$$

Obviously, this function does not contain any multilinear subexpression. However, according to the factorable reformulation, auxiliary variables are first introduced for each monomial: $x_4 = x_1^2$, $x_5 = x_2^2$. Consider f in the augmented space: $f = x_4 x_2 x_3 + x_4 x_5 + x_1 x_5 x_3$. Now, we observe that $f(x)$, $x \in \mathbb{R}^5$ is indeed a multilinear function. As we will demonstrate in the next section, cutting planes for these lifted multilinears are very powerful for polynomial optimization problems.

Algorithms 2-4 contain the outline of our recognition approach. Given the factorable reformulation of an optimization problem as obtained by Algorithm 1, we start from intermediate linear relations of the form $\ell = \sum_k a_k x_k$ (see Algorithm 2). For auxiliary variables x_k that correspond to bilinear relations, we apply a recursive expansion to identify the multilinear functions (see Algorithm 3). By Algorithm 1, any multilinear function containing at least two terms corresponds to an intermediate linear relation in the reformulated problem. Thus, this simple technique identifies all multi-term multilinears. However, multilinears $L(x)$ that consist of a single term and appear in the form of $h(L(x))$, where h is a nonlinear function, are not recognizable by examining linear relations (see the last case in Algorithm 1). To capture such multilinears, we employ Algorithm 4, in which we start from nonlinear relations (i.e., fractions, monomials, powers, and logarithms) and apply a recursive decomposition to identify multilinear terms. For example, consider

$$f(x) = \log x_1 (x_2 x_3 x_4)^2 2^{x_2} + \exp(x_1 x_2 x_3 2^{x_2}).$$

The proposed recognition approach identifies the following multilinears: $L_1 = x_2x_3x_4$, $L_2 = x_1x_2x_3x_7$, and $L_3 = x_5x_6x_7$, where $x_5 = \log x_1$, $x_6 = (x_2x_3x_4)^2$, and $x_7 = 2^{x_2}$. The first two multilinears L_1 and L_2 are found by Algorithms 4, as they are composed by monomial and exponential functions, respectively. However, L_3 is identified by Algorithm 2, as it appears in a linear relation. It is important to note that, since `BARON`'s relaxations are already built in the lifted space [36], these intermediate multilinear cuts can significantly enhance the convergence rate of the branch-and-reduce algorithm. For simplicity, we chose not to employ product disaggregation; that is, for a bilinear relation of the form $f = x_4x_5$, where $x_4 = x_1 + x_2$ and $x_5 = x_2 + x_3$, we do not distribute the products to obtain the bilinear function $L = x_1x_2 + x_1x_3 + x_2x_3$. In addition, our algorithm does not capture multilinear functions that are implied by multiple constraints in the original space. For example, consider an optimization problem containing the following constraints:

$$\begin{cases} x_4 = x_1x_2 \\ x_5 = x_1x_3 \\ x_6 = x_2x_3 \\ x_4 + x_5 + x_6 \leq 1 \end{cases}$$

Our recognition algorithm does not identify any multilinear expression as the first three constraints contain a single bilinear term and the last constraint is linear. Clearly, the above set is equivalent to $x_1x_2 + x_1x_3 + x_2x_3 \leq 1$, which contains a multilinear function. However, for general nonconvex problems, there are often multiple ways for performing such substitutions and employing a naive scheme leads to the generation of a large number of multilinear functions, which may increase the overall computational cost of the global solver. In such cases, a more systematic approach is to consider a collection of constraints simultaneously and generate the facets of the convex hull of the entire set (cf. [32]). Convexifying multiple constraints is beyond the scope of this paper and is a subject of future research.

3.2 Decomposition algorithm

The computational cost of generating multilinear cutting planes depends on the number of variables in each multilinear function. The proposed cuts are most beneficial for relaxing multilinears that consist of many terms but have a small number of variables. In such cases, multilinear cuts are often significantly stronger than conventional termwise relaxations since they capture the interactions between different terms, and they are constructed efficiently since the corresponding separation problem has a reasonable size. For multilinear functions that do not naturally satisfy this property, one

Algorithm 2 Identifying multilinear structures in general nonconvex problems

Given a factorable reformulation of an optimization problem.
Initialize the list of multilinear functions $\mathcal{M} = \emptyset$ and a temporary list for storing potential multilinears $\mathcal{S} = \emptyset$.
Scan each intermediate linear relation $\ell_i = \sum_k a_k x_k$, $i \in I_\ell$:
  Initialize the multilinear function $L_i = 0$.
  For each intermediate variable x_k in ℓ_i:
    if x_k corresponds to a bilinear relation $x_k = x_i x_j$ with $k > \max\{i, j\}$, then
      iteratively decompose x_k to reconstruct the multilinear term $x_k = \prod_{t \in T} x_t$ using Algorithm 3. Let $L_i = L_i + a_k \prod_{t \in T} x_t$ and update the list $\mathcal{S}$.
    else
      store x_k for further investigation: $\mathcal{S} = \mathcal{S} \cup \{x_k\}$.
    end of if
  end of for
  If L_i contains at least three variables and L_i is unique, then store it $\mathcal{M} = \mathcal{M} \cup \{L_i\}$.

Algorithm 3 Identifying a multilinear term by recursive expansion of bilinear terms

Given a bilinear relation $x_k = x_i x_j$ and a list for storing potential multilinears $\mathcal{S}$.
Initialize the index set of variables in x_k: $T = \{i, j\}$. Let $n_t = |T|$ and $l = 1$.
Repeat:
  Consider the variable x_s associated with the lth element in T; Initialize $\xi = 0$.
  If x_s represents a bilinear relation $x_s = x_r x_w$ with $s > \max\{r, w\}$, then
    if x_r or x_w is already stored as a variable in x_k (i.e., $r \in T$ or $s \in T$), then
      do not decompose x_s, as it will result in a monomial function.
      add x_s to $\mathcal{S}$ for further investigation.
    else
      add x_r and x_w to the variables list and remove x_s:
        $s \leftarrow r$, $T = T \cup \{w\}$, and $n_t \leftarrow n_t + 1$.
        update $\xi = 1$.
    end of if
  else if x_s is an intermediate variable, then
    add x_s to $\mathcal{S}$ for further investigation.
  end of if
  if $\xi = 0$, then $l \leftarrow l + 1$.
Until $l \leq n_t$.

can devise a decomposition technique to construct lower-dimensional multilinears and then utilize Problem (FD) to generate strong cuts for subfunctions. Clearly, the quality of such relaxations depends on the decomposition strategy. With the objective of minimizing the overall cost of the branch-and-bound tree, in this section, we propose a decomposition technique that exploits the structure of multilinear functions to balance the trade-off between the cost of solving separation problems and the sharpness of resulting relaxations.

Consider a multilinear function $L(x) : \mathcal{H} \to \mathbb{R}$, as defined by (1). Let m denote the number of multilinear terms in $L(x)$. Suppose that $m \geq 2$. We will revisit the case where $L(x)$ has a single term but too many variables later. Ideally, one would like to break down $L(x)$ to multilinears in lower dimensions $L_\kappa(x_\kappa)$, $\kappa \in \mathcal{K}$ with $x_\kappa = [x_{\kappa 1}, \ldots, x_{\kappa q}]$ for some $2 \leq q < n$, while maintaining the tightness of the relaxation. An important instance is the case where the convex envelope of $L(x)$ is sum decomposable; i.e.,

$$\operatorname{conv}_{\mathcal{H}} L(x) = \sum_{\kappa \in \mathcal{K}} \operatorname{conv}_{\mathcal{H}^\kappa} L_\kappa(x_\kappa), \tag{5}$$

Algorithm 4 Identifying intermediate multilinear terms that are composed with other nonlinear functions

Given the list of potential multilinears $\mathcal{S}$ and a list for storing multilinears $\mathcal{M}$.
Initialize $l = 1$.
Repeat:
1. Let x_s be the lth element of $\mathcal{S}$:
2. Do while x_s is not a bilinear relation:
   - if $x_s = x_r^a$ or $x_s = a^{x_r}$ or $x_s = \log x_r$, then
     - $x_s \leftarrow x_r$.
   - else if $x_s = x_r/x_w$, then
     - add x_r and x_w to $\mathcal{S}$ for further investigation.
     - go to Step 5.
   - else if x_s represents a linear relation, then
     - go to Step 5.
   - end of if

   end of while
3. Iteratively decompose x_s to obtain the multilinear term $t(x)$ using Algorithm 3.
4. If $t(x)$ is unique and contains at least three variables, then add it to $\mathcal{M}$.
5. $l \leftarrow l + 1$.

Until $l \leq |\mathcal{S}|$.

where $\mathcal{K}$ is a partition of the set Ω as defined by (1), and $\mathcal{H}^\kappa$ is the projection of $\mathcal{H}$ onto the space of variables defined by x_κ. Furthermore, $L_\kappa(x_\kappa)$ denotes a multilinear subfunction containing one or more terms of $L(x)$. If $L(x)$ is separable, i.e., if we can write $L(x)$ as a sum of multilinears $L_\kappa(x_\kappa)$ such that $\{\mathcal{H}^\kappa : \kappa \in \mathcal{K}\}$ forms a partition of $\mathcal{H}$, then Condition (5) holds [8]. Thus, in the following, we assume that $L(x)$ is not separable. Rikun [24] derives the necessary and sufficient condition under which the convex envelope of addends is equal to the polyhedral convex envelope of their sum. In addition, he presents a number of special cases for which the aforementioned condition is easy to verify. Tardella [31] also studies the problem of sum decomposability of edge concave functions over polytopes and provides generalizations and alternative characterizations of the earlier results in [24,22]. We will make use of the following sufficient condition (Theorem 1.4 in [24]) as a preprocessing step in our decomposition algorithm.

Theorem 2 *([24]) Let P be a Cartesian product of polytopes, $P = P_0 \times P_1 \times \ldots \times P_k$, $P_i \in \mathbb{R}^{n_i}$, and let f_i be a continuous function defined on $P_0 \times P_i$, for all $i = 1, \ldots, k$. If each $f_i(x_0, x_i)$ is a concave function of x_0 when x_i is fixed and P_0 is a simplex, then*

$$\operatorname{conv}_P \left(\sum_{i=1}^{m} f_i(x_0, x_i) \right) = \sum_{i=1}^{m} \operatorname{conv}_{P_0 \times P_i} f_i(x_0, x_i)$$

As a special case of the above result, consider a multilinear function $L(x)$ over a box and let $n_0 = 1$. It follows that, if all multilinear subfunctions $L_\kappa(x_\kappa)$ depend on a single common variable x_0, then $L(x)$ is sum decomposable. We now present a slightly different characterization of this result. Let us define a graph representation for the multilinear function $L(x)$. Consider an undirected graph $G = (V, E)$, where V represents the index set of variables in $L(x)$, i.e., if $L(x)$ contains the variable x_i, then there exists a vertex i in G. Vertices i and j are connected by an edge e_{ij} if x_i and x_j appear in a common term in $L(x)$. Recall that a connected graph is said to be *biconnected* if it does not have any articulation point (i.e., a node whose removal disconnects the graph). Moreover, a biconnected component is a maximal biconnected subgraph. The following proposition provides a sufficient condition for decomposing a multilinear function without compromising the strength of the corresponding relaxation.

Proposition 1 *Let $G(V, E)$ be the graph representation of the multilinear function $L(x)$ defined over the box $\mathcal{H}$. Suppose that G is connected. Let G_κ, $\kappa \in \mathcal{K}$ be the biconnected components of G, and let $L_\kappa(x_\kappa)$, $x_\kappa \in \mathcal{H}^\kappa$ be the multilinear function corresponding to G_κ. Then*

$$\text{conv}_{\mathcal{H}} L(x) = \sum_{\kappa \in \mathcal{K}} \text{conv}_{\mathcal{H}^\kappa} L_\kappa(x_\kappa). \tag{6}$$

Proof We prove by induction on the number of biconnected components in G. If G is biconnected, then clearly (6) is valid. Suppose that (6) holds if G has k biconnected components. We now prove that Equation (6) is valid if G has $k+1$ biconnected components. We decompose G into two subgraphs at a certain articulation point; namely, the last articulation point x_s found in the depth first search tree of G. It follows that one subgraph denoted by G_{k+1} is biconnected, while the other subgraph, denoted by $G_{1,\ldots,k}$, has k biconnected components. Let $L_{k+1}(x_{k+1})$, $x_{k+1} \in \mathcal{H}^{k+1}$ and $L_{1,\ldots,k}(x_{1,\ldots,k})$, $x_{1,\ldots,k} \in \mathcal{H}^{1,\ldots,k}$ denote the multilinears associated with G_{k+1} and $G_{1,\ldots,k}$, respectively. Furthermore, the hyper-rectangle $\mathcal{H}^{1,\ldots,k}$ (resp. $\mathcal{H}^{k+1}$) is obtained by projecting $\mathcal{H}$ on the space of $x_{1,\ldots,k}$ (resp. x_{k+1}). By definition of the articulation point, there exists no edge between the two sets of vertices given by $V_1 = \{i : x_i \in x_{1,\ldots,k},\ i \neq s\}$ and $V_2 = \{i : x_i \in x_{k+1},\ i \neq s\}$. In other words, $L(x)$ does not contain a bilinear term $x_i x_j$ such that $i \in V_1$ and $j \in V_2$. Thus, we have the following decomposition for $L(x)$:

$$L(x) = L_{1,\ldots,k}(x_{1,\ldots,k}) + L_{k+1}(x_{k+1}).$$

Since by construction $G_{1,\ldots,k}$ and G_{k+1} have one shared vertex, the corresponding multilinears $L(x_{1,\ldots,k})$ and $L(x_{k+1})$ have a single variable in common. Hence, by Theorem 2 we have

$$\text{conv}_{\mathcal{H}} L(x) = \text{conv}_{\mathcal{H}^{1,\ldots,k}} L_{1,\ldots,k}(x_{1,\ldots,k}) + \text{conv}_{\mathcal{H}^{k+1}} L_{k+1}(x_{k+1}).$$

By the induction hypothesis, $\text{conv}_{\mathcal{H}^{1,\ldots,k}} L_{1,\ldots,k}(x_{1,\ldots,k}) = \sum_{j=1}^{k} \text{conv}_{\mathcal{H}^j} L_j(x_j)$. Thus, Equation (6) is valid for a graph with $k+1$ biconnected components. □

For instance, consider the bilinear function $L(x) = x_1x_2 - 2x_1x_3 + 0.5x_2x_3 - x_3x_4 + 3x_3x_5 - x_4x_5 + x_5x_6$. By the above result, $L(x)$ decomposes into three bilinears $L_1(x) = x_1x_2 - 2x_1x_3 + 0.5x_2x_3$, $L_2(x) = -x_3x_4 + 3x_3x_5 - x_4x_5$, and $L_3(x) = x_5x_6$. Clearly, Proposition 1 can be equivalently stated for the concave envelope of multilinear functions. To find the biconnected components of a graph, we employ the classical depth first search algorithm of Hopcroft and Tarjan [13] that runs in linear time. We remark that the connectivity assumption of G in Proposition 1 can be relaxed. It is simple to check that G is not connected when the corresponding multilinear function $L(x)$ is separable. Thus, finding the connected components of G is equivalent to finding nonseparable multilinear subfunctions of $L(x)$. Indeed, we employ this preprocessing step prior to searching for biconnected components, as shown in Algorithm 5.

In Algorithm 5, $\mathcal{L}$ denotes the set of multilinears for which Problem (FD) will be solved for cut generation, whereas $\mathcal{U}$ denotes the set of multilinears that should be decomposed into smaller functions using the techniques that will be discussed next.

Algorithm 5 Decomposing a multilinear function into components that are not sum-decomposable

Given a multilinear function $L(x)$ represented by an undirected graph $G(V,E)$ and $n_{\min}$.
1. Find all connected components of G: $\mathcal{G} = \{G_t : t \in T\}$.
   Let n_t be the number of nodes in G_t and let L_t be the corresponding multilinear.
   For each G_t:
      if $n_t < 3$, then
         update $\mathcal{G} = \mathcal{G} \setminus \{G_t\}$
      else if $n_t = 3$, then
         store L_t for cut generation $\mathcal{L} = \mathcal{L} \cup \{L_t\}$ and update $\mathcal{G} = \mathcal{G} \setminus \{G_t\}$
      end of if
   end of for
2. Find all biconnected components of each connected component G_t: $\mathcal{G}_t = \{G_{tk} : k \in K\}$.
   Let n_{tk} be the number of vertices of G_{tk} and let L_{tk} be the corresponding multilinear.
   For each G_{tk}:
      if $n_{tk} < 3$, then
         discard G_{tk}
      else if $n_{tk} \leq n_{\min}$, then
         store L_{tk} for cut generation: $\mathcal{L} = \mathcal{L} \cup \{L_{tk}\}$
      else
         add L_{tk} to the list of uncovered multilinears $\mathcal{U}$ for further decomposition
      end of if
   end of for

Moreover, $n_{\min}$ represents the threshold for further decomposition. Namely, if the number of variables in $L(x)$ does not exceed $n_{\min}$, we do not attempt to break it down to smaller multilinears but store it for cut generation.

Next, suppose that the n-dimensional multilinear $L(x)$ with $n > n_{\min}$ is not decomposable by means of Proposition 1. We are interested in identifying a collection of multilinear subfunctions $\{L_\kappa(x_\kappa) : \kappa \in \mathcal{K}\}$ for which generating the proposed cuts is, in some sense, more advantageous than generating the facets of the envelope of the entire multilinear $L(x)$. Namely, we address the trade-off between solving an expensive separation problem and the strength of the resulting relaxation by identifying lower-dimensional dense components of $L(x)$. This decomposition differs from the sum decomposability property in that (i) the strength of the relaxation is no longer maintained; in the current setting we are willing to compromise on the sharpness of cuts, in the hope of reducing the overall CPU time, and (ii) the equality $L(x) = \sum_{\kappa \in \mathcal{K}} L_\kappa(x_\kappa)$ is no longer satisfied, as we will allow some terms of $L(x)$ to appear in multiple multilinears $L_\kappa(x_\kappa)$, while some other terms may not be included in any of the subfunctions. We start by defining a *weighted* graph $G(V,E)$ for $L(x)$. Let the nodes and vertices of G be defined as before. We associate a weight w_{ij} with each edge e_{ij}, such that w_{ij} represents the number of occurrences of the bilinear term $x_i x_j$ in $L(x)$. Obviously, for a bilinear function we have $w_{ij} = 1$ for all $e_{ij} \in E$. However, for multilinear functions containing products of three or more variables, defining these weights is essential for our decomposition algorithm. In addition, we impose an upper bound on the number of variables $n_{\max}$ in each subfunction. As a result, the largest separation problem to be solved will involve no more than $2^{n_{\max}}$ variables. In each subfunction L_κ, we would like to include as many terms as possible, while satisfying the size constraint. Let $n_{\text{term}} = \max_i n_i$, where n_i denotes the number of variables in the ith term of the multilinear function. Suppose than $n_{\text{term}} \leq n_{\max}$. We motivate our decomposition approach by a simple example.

Example 1 Let

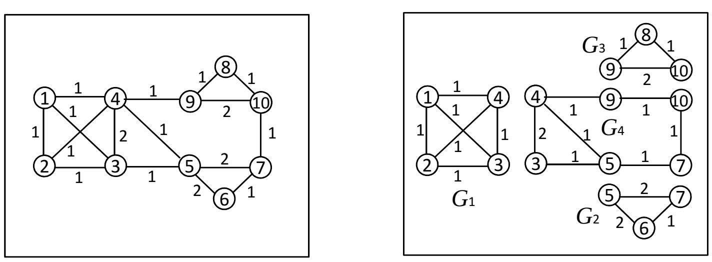
$$L(x_1,\ldots,x_{10}) = x_1x_2x_4 - x_1x_3 + x_2x_3 + x_3x_4 - x_3x_4x_5 + x_5x_6 + x_5x_6x_7 + x_5x_7 + x_4x_9 + x_7x_{10} + x_8x_9x_{10} - x_9x_{10},$$

and let $n_{\max} = 6$. Consider the following decomposition of $L(x)$:

$$\left.\begin{array}{l} L_1(x_1,x_2,x_3,x_4) = x_1x_2x_4 - x_1x_3 + x_2x_3 + x_3x_4 \\ L_2(x_5,x_6,x_7) = x_5x_6 + x_5x_6x_7 + x_5x_7 \\ L_3(x_8,x_9,x_{10}) = x_8x_9x_{10} - x_9x_{10} \\ L_4(x_3,x_4,x_5,x_7,x_9,x_{10}) = x_3x_4 - x_3x_4x_5 + x_5x_7 + x_4x_9 + x_7x_{10} - x_9x_{10} \end{array}\right\} \quad (7)$$

Denote by G_κ, $\kappa \in K$ the weighted graph associated with each subfunction L_κ. The main characteristics of the decomposition defined by (7) are (see Figure 1):

1. All terms in $L(x)$ are included in at least one subfunction L_κ, $\kappa \in K$.
2. The subfunctions L_κ, $\kappa \in K$ are not sum-decomposable (i.e., G_κ, $\kappa \in K$ is biconnected).
3. The number of multilinear terms that are included in more than one $L_\kappa(x_\kappa)$ is minimal; i.e., if such a term is removed from any of the associated subfunctions, the resulting multilinear has the same number of variables or is sum-decomposable.
4. The subfunctions $L_\kappa(x_\kappa)$, $x_\kappa \in \mathcal{H}^\kappa$ are maximal:
   - All multilinear terms in $L(x)$ whose variables are contained in $\mathcal{H}^\kappa$ are present in $L_\kappa(x_\kappa)$.
   - Denote by n_κ the number of variables in $L_\kappa(x_\kappa)$. If $n_\kappa < n_{\max}$, then construct an augmented multilinear $\tilde{L}_\kappa(x) = L_\kappa + \sum_l t_l(x)$, where $t_l(x)$ denotes a multilinear term in $L(x)$ that is not present in $L_\kappa(x)$. Suppose that $\tilde{L}_\kappa(x)$ has $\tilde{n}_k$ variables such that $n_\kappa < \tilde{n}_\kappa \le n_{\max}$. Then, two cases arise: (i) $\tilde{L}_\kappa(x)$ violates Condition 3, or (ii) the graph of $\tilde{L}_\kappa(x)$ is not biconnected, and has a biconnected component with the corresponding multilinear given by $L_\kappa(x)$.



(a) (b)

Fig. 1 The graph representation for multilinear functions in Example 1. The graph of $L(x)$ is shown in Fig 1(a) and the subgraphs of its decomposition defined by (7) are shown in Fig 1(b).

In Example 1, Condition 1 ensures that the set of subfunctions L_κ, $\kappa \in K$ provides a good cover for $L(x)$. Condition 2 simply states that a subfunction is stored for cut generation only if it is not sum-decomposable; otherwise, the decomposition scheme of Algorithm 5 should be employed as a post-processing step, as it reduces the computational cost of solving the separation problem without compromising the quality of resulting relaxations. Condition 3 avoids the generation of too many subfunctions by imposing restrictions on multilinear terms that appear in multiple subfunctions. Finally, Condition 4 ensures that each subfunction contains as many multilinear terms as possible while satisfying the constraint on the maximum number of variables in it. In general, constructing a decomposition that satisfies Conditions 1-4 is not a tractable task. Moreover, as we detail later, additional factors regarding the computational cost of solving the separation problems need to be considered as well. In the following, we propose an efficient heuristic that often yields decompositions for which the above conditions are (almost) satisfied. To this end, consider the following graph partitioning problem as defined in [16]:

Graph Partitioning: Given a weighted graph $G(V, E)$, partition the nodes of G into subsets no larger than $n_{\max}$, so as to minimize the total weight of the edges cut.

This so-called partitioning or clustering problem essentially decomposes a graph into dense subgraphs whose numbers of nodes do not exceed $n_{\max}$. The objective function of the above problem is often referred to as the *edge-cut*. A partition of $G(V, E)$, $|V| = n$ into ρ subgraphs is commonly represented by a partition vector p of length n, where p_j is equal to the partition element i, $i \in \{1, \dots, \rho\}$ to which the jth node belongs. For example, consider the graph G of Example 1. As shown in Figure 2(a), the optimal partitioning for G with $n_{\max} = 6$ is given by $p = [1, 1, 1, 1, 2, 2, 2, 2, 2, 2]$. We utilize such a partitioning of G to construct dense subfunctions of $L(x)$ as follows. Let $L'_i(x) = 0$ for all $i = 1, \dots, \rho$. Denote by $\mathcal{I}_k$ the index set of variables that are present in the kth multilinear term t_k in $L(x)$. If $p_j = i$ for all $j \in \mathcal{I}_k$ and some $i \in \{1, \dots, \rho\}$, then add t_k to the ith subfunction $L'_i(x) = L'_i(x) + t_k(x)$. Denote by G'_i the weighted graph associated with $L'_i(x)$. As we discussed before, we are interested in subfunctions that are not sum-decomposable. Hence, we employ Algorithm 5 to identify biconnected components of each subgraph G'_i and store the proper multilinears for cut generation. Applying this procedure to the multilinear function of Example 1, we first obtain the two subfunctions $L'_1(x) = x_1x_2x_4 - x_1x_3 + x_2x_3 + x_3x_4$ and $L'_2(x) = x_5x_6 + x_5x_6x_7 + x_5x_7 + x_7x_{10} + x_8x_9x_{10} - x_9x_{10}$. The graphs of L'_1 and L'_2 are shown in Figure 2(b). Since G'_2 is not biconnected, it is further decomposed into two smaller subgraphs, denoted by G_2 and G_3 in Figure 2(c). Thus, we store the three multilinears corresponding to the graphs shown in Figure 2(c) for cut generation. Observe that these subfunctions are the first three multilinears in the decomposition of $L(x)$ given by (7).

It is well known that the graph partitioning problem is NP-complete [9]. However, many heuristics have been developed to find good partitions at a reasonable cost (cf. [16, 15]). In particular, multilevel graph partitioning schemes reduce the size of the graph by collapsing vertices and edges, partition the smaller graph, and then uncoarsen it to construct a partition for the original graph [12]. A widely-used successful implementation of these methods is the `METIS` graph partitioner that provides high quality partitions for large graphs and is extremely fast [15]. It turns out that the execution time of our decomposition scheme is determined by the partitioning step. Thus, we utilize `METIS` to generate a number of high quality partitions and subsequently select a partition that yields multilinears for which the proposed cutting planes are cheap to compute. To partition a graph, `METIS` requires the number of partition elements as well as partition sizes as inputs arguments. However, it allows to search for partitions with different sizes by defining a load imbalance parameter. In our graph partitioning problem, the only constraint is an upper bound on the size of each partition element. The number of partition elements and their sizes are not known a priori. Thus, we employ `METIS` to solve our problem as follows. Denote by $\rho_{\min}$ and $\rho_{\max}$ a lower and an upper bound on the number of partition elements, respectively. For a graph with n nodes, we define $\rho_{\min} = \lceil n/n_{\max} \rceil$ and $\rho_{\max} = \lfloor n/n_{\min} \rfloor$, where $n_{\min}$ is as defined in Algorithm 5. For each number of partition elements ρ, we define a nominal partition size $s_\rho = \lceil n/\rho \rceil$, and then use `METIS` to obtain a partition with the minimum edge-cut by specifying a load imbalance $r_\rho = n_{\max}/s_\rho$, in order to explore all partitions within our size limits. Subsequently, we construct the corresponding set of multilinears, as described before. Let $\mathcal{D}_\rho$ be the set of multilinear functions associated with ρ and let $N_{\mathrm{LP}} = |\mathcal{D}_\rho|$. Denote by $\mathcal{N}_e$ the total number of multilinear terms in $\mathcal{D}_\rho$ that are not present in the cut generation list $\mathcal{L}$, and let δ denote the maximum number of variables in the multilinear functions in $\mathcal{D}_\rho$. For each partition, we define a *gain factor* Θ_ρ as follows:

$$\Theta_\rho = \frac{\mathcal{N}_e}{N_{\mathrm{LP}}^{\beta_1} \beta_2^{\delta}}, \tag{8}$$

where $0 < \beta_1 \leq 1 \leq \beta_2$. The above relation is designed to capture the trade-off between the strength of a relaxation (i.e., $\mathcal{N}_e$) and the cost of solving the corresponding separation problems (i.e., N_{LP} and δ). We select the partition with the largest gain factor; namely, we prefer partitions with higher-dimensional multilinears only if they provide a considerably stronger relaxation. As we will see in the next section, such an adaptive scheme is crucial for the performance of a global solver as dense problems significantly benefit from the cutting planes corresponding to the envelopes of high-dimensional multilinears (i.e., $10 \leq n \leq 15$), while for sparser problems the best performance is obtained by decompositions that consist of many low-dimensional multilinears (i.e., $4 \leq n \leq 8$). The outline of our approach is presented in Algorithm 6.

Algorithm 6 Identifying a subset of dense components of a multilinear function using graph partitioning

Given a multilinear function $L(x)$ with the weighted biconnected graph $G(V, E)$, and the set of parameters $\{\beta_1, \beta_2, n_{\min}, n_{\max}\}$.
Initialize the gain factor $\tilde{\Theta} = 0$, and the set of dense components $\mathcal{D} = \emptyset$.
Compute $\rho_{\min}$ and $\rho_{\max}$. Let $\rho = \rho_{\min}$.
Repeat
1. Use `METIS` to find an optimal partition of G into ρ subgraphs, with s_ρ and r_ρ as inputs.
2. Construct the multilinears $\mathcal{D}_\rho$ corresponding to this partitioning and compute N_{LP}, $\mathcal{N}_e$ and δ.
3. Compute the gain factor Θ_ρ for this partition.
   If $\Theta_\rho > \tilde{\Theta}$, then
      $\tilde{\Theta} \leftarrow \Theta_\rho$
      $\mathcal{D} \leftarrow \mathcal{D}_\rho$
   end if.
4. $\rho \leftarrow \rho + 1$.

Until $\rho \leq \rho_{\max}$.
Store the set of dense components $\mathcal{D}$ for cut generation: $\mathcal{L} = \mathcal{L} \cup \mathcal{D}$.

Algorithm 6 is not guaranteed to generate dense components. For example, consider $L(x) = \sum_{i=1}^{n-1} x_i x_{i+1} + x_1 x_n$, and let $n_{\max}$ be any number such that $n_{\max} < n$.

Then, it is simple to verify that, if we remove any term from $L(x)$, by Proposition 1, the resulting function decomposes into n bilinear terms. Clearly, this is due to the structure of $L(x)$ and is not a shortcoming of our algorithm. However, we can construct multilinears containing dense components that Algorithm 6 is not able to capture. Let $L(x) = x_1x_2x_3 + x_2x_4 + x_3x_4$ and $n_{\min} = n_{\max} = 3$. Clearly, $L_1(x) = x_1x_2x_3$ is a valid dense component. However, in Algorithm 6, the partition $p = [1, 2, 2, 2]$ is also optimal, which yields $\mathcal{D} = \emptyset$. In our implementation, for such cases, we randomly remove a few terms from the multilinear and then reapply Algorithm 6 to search for dense components. If after a few iterations no multilinear is found, then the algorithm terminates with $\mathcal{L} = \emptyset$.

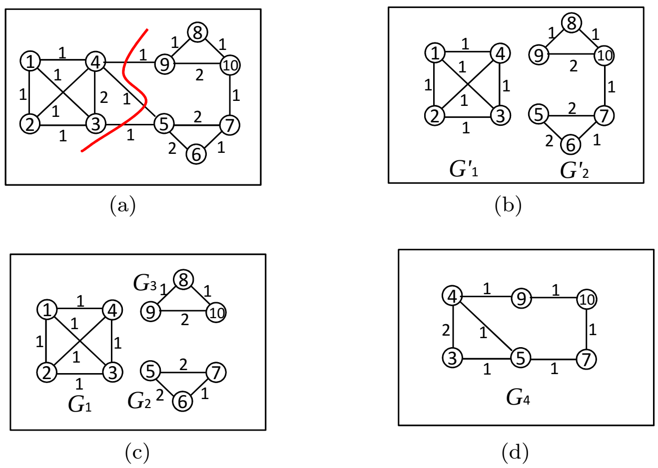


Fig. 2 Illustration of the proposed decomposition on the multilinear function $L(x)$ of Example 1. In Fig. 2(a), the graph of L is partitioned to two subgraphs. Using this partitioning, two low-dimensional multilinears are constructed whose graphs are shown in Fig. 2(b). The decomposition of these graphs into biconnected components is depicted in Fig. 2(c). To cover all of the terms in $L(x)$, a reduced multilinear is constructed with the graph shown in Fig. 2(d).

By employing the above procedure, we generated a collection of lower-dimensional multilinears that contain 75% of terms in the multilinear function of Example 1. Clearly, for dense multilinears, a simple application of this partitioning approach yields poor decompositions. For example, for a fully dense bilinear function $L(x)$ with $n = 20$ and $n_{\max} = 6$, the proposed method yields subfunctions that contain at most 24% of the bilinear terms in $L(x)$. Intuitively, if we remove a proper subset of the terms in $L(x)$ and reapply the same partitioning technique, we can capture additional dense subfunctions of $L(x)$. Next, we present an algorithm that formalizes this idea.

As before, denote by $\mathcal{L}$ the set of subfunctions stored for cut generation. For each term t_k in $L(x)$, let μ_k denote the number of multilinears in $\mathcal{L}$ in which t_k appears. In other words, μ_k represents the number of times that t_k has been covered so far. If $\mu_k > 0$ for some t_k, then we say that t_k is a covered term. In addition, if for two covered terms t_{k1} and t_{k2}, we have $\mu_{k1} > \mu_{k2}$, then we say that t_{k1} is covered more than t_{k2}. In the following, we present an algorithm to construct a reduced multilinear

$L_{\rm rd}(x)$ by removing a certain percentage of terms in $L(x)$ that are covered most. We denote by $\mathcal{T}_j$ the set of terms in which the variable x_j appears.

Algorithm 7 Constructing a reduced multilinear function by removing a specific subset of the multilinear terms that are already stored for cut generation

Given a multilinear function $L(x)$ with $n_{\rm cv}$ covered terms, and $\alpha \in (0, 1]$.
1. Let x_j be a covered variable if, for all terms t_k in $\mathcal{T}_j$, we have $\mu_k > 0$. Let $L_{\rm rd}$ be the multilinear obtained by removing all terms of $L(x)$ in which at least one covered variable appears. Let $n_{\rm rd}$ be the number of terms in $L_{\rm rd}$.
   If $n_{\rm rd} < \alpha n_{\rm cv}$, then
     quit
   else
     go to Step 2
   end if
2. Let $n_{\rm rs} = \alpha n_{\rm cv} - n_{\rm rd}$. Remove from $L_{\rm rd}$ the $n_{\rm rs}$ terms with largest μ values. In case of a tie, remove terms with a minimal number of common variables.

Let us apply Algorithm 7 to the multilinear of Example 1. The number of covered terms in this case is $n_{\rm cv} = 9$. We construct a reduced multilinear by removing at least 30% of covered terms in $L(x)$; i.e., $\alpha = 0.7$. It is simple to check that all terms containing x_1, x_2, x_6, and x_8 are covered by the subfunctions L_1, L_2, and L_3. After eliminating all such terms, we obtain $L_{\rm rd} = x_3x_4 - x_3x_4x_5 + x_5x_7 + x_4x_9 + x_7x_{10} - x_9x_{10}$, with $n_{\rm rd} = 6$. Since $6 < 0.7 \times 9$, no further reduction is required. The graph of $L_{\rm rd}$ is depicted in Figure 2(d). Since this graph is biconnected and $n_{\rm rd} \leq n_{\max}$, we let $L_4 = L_{\rm rd}$ and store L_4 for cut generation. Thus, Algorithm 7 generates the last subfunction of Example 1, which was not obtained after a single application of `METIS`. To simplify the presentation, in this example, we had $\beta_1 = \beta_2 = 1$; i.e., partitions with best cover were selected.

We are now in a position to present our complete decomposition scheme that is composed of an iterative application of Algorithms 5-7. The outline of our approach is shown in Algorithm 8. In this algorithm, the parameter γ_c controls the strength of the relaxation. By letting $\gamma_c = 1.0$, we require all terms in the original multilinear to be included in the decomposition. However, in many cases, such a choice for γ_c leads to the generation of many subfunctions with a large number of terms covered multiple times, which in turn slows down the convergence of the global solver. In our implementation, we found that $\gamma_c = 0.85$ provides reasonable decompositions for a wide range of problems.

Finally, let us revisit the case where the multilinear function contains a term $t_k = \prod_{j=1}^{n_k} x_j$ with $n_k > n_{\max}$. Clearly, there exist many different ways to rewrite t_k as a system of lower-dimensional multilinears. The relative strength of the resulting relaxations depends on the variable bounds and, in general, cannot be determined a priori. We employ a simple approach in which, prior to the application of Algorithm 8, we replace each term t_k with $n_k > n_{\max}$ by $d = 1 + \lceil (n - n_{\max})/(n_{\max} - 1) \rceil$ multilinear terms, given by $\tilde{t}_1 = \prod_{j=1}^{n_{\max}} x_j$, $\tilde{t}_{k+1} = \tilde{t}_k \prod_{j=l}^{u} x_j$, for all $k = 1, \ldots, d-1$, where $l = k(n_{\max} - 1) + 2$ and $u = \min\{(k+1)n_{\max} - k, n\}$.

Now that multilinears have been identified and decomposed, the separation problem must be solved to identify facets of their convex envelopes.

Algorithm 8 Constructing a collection of low-dimensional components of a multilinear function that yields strong and cheaply computable relaxations

Given a multilinear function $L(x)$ with m terms and $\gamma_c \in (0, 1)$.
Initialize the number of covered terms $\mathcal{N}_c = 0$ and the list of uncovered functions $\mathcal{U} = \emptyset$.
Decompose $L(x)$ to functions that are not sum-decomposable using Algorithm 5; update $\mathcal{N}_c$ and $\mathcal{U}$.
Repeat
  1. Select a multilinear L_i with m_i terms from the list of uncovered functions $\mathcal{U}$.
  2. Decompose L_i into lower-dimensional subfunctions using Algorithm 6 and update $\mathcal{N}_c$.
  4. Let $\mathcal{N}_e$ be the number of covered terms in L_i (i.e., those with $\mu > 0$).
   If $\mathcal{N}_e < \gamma_c m_i$, then
    Construct a reduced multilinear L_{rd} by removing a subset of covered terms from L_i using Algorithm 7.
    Decompose L_{rd} into components that are not sum-decomposable using Algorithm 5; update $\mathcal{N}_c$ and $\mathcal{U}$.
   end of if
Until $\mathcal{U}$ is nonempty and $\mathcal{N}_c < \gamma_c m$.

3.3 A customized simplex algorithm for solving the separation problem

The proposed cut generation scheme requires the solution of the LP given by (F2), the size of which grows exponentially in the number of variables of the multilinear function. We choose to solve the dual of (F2), given by:

$$
\begin{aligned}
\text{(FD)} \qquad \min_{\lambda} \ & \sum_{i \in I} \lambda_i L(v^i) \\
\text{s.t.} \ & \sum_{i \in I} \lambda_i v^i = x^* \\
& \sum_{i \in I} \lambda_i = 1 \\
& \lambda_i \geq 0, \forall i \in I,
\end{aligned}
$$

where $I = \{1, \ldots, 2^n\}$. The exponentially many variables of the above LP make it difficult to directly solve the entire problem. However, the number of constraints in Problem (FD), which is one more than the number of variables in the multilinear function, is usually not too large. Moreover, as detailed in Section 3.2, we employ a decomposition technique to control the size of this problem for larger multilinears. Thus, during the execution of a simplex-type algorithm, the size of the basis and basic feasible solutions of the above problem are relatively small and can be handled efficiently by modern computational techniques.

We solve Problem (FD) by a simplex algorithm whereby we choose the first nonbasic column associated with a negative reduced cost to enter the basis at each iteration (cf. [4] for details on the simplex algorithm). Let λ^0 denote the basic feasible solution of (FD) examined at a given iteration of the simplex algorithm. It is simple to verify that if

$$\sum_{i \in I} \lambda_i^0 L(v^i) \leq z^*,$$

then the relaxation solution (x^*, z^*) belongs to the epigraph of the convex envelope of $L(x)$. Thus, at any iteration of the simplex algorithm, if the above condition is satisfied, the algorithm terminates with the proof that a separating hyperplane does not exist.

In the following, we discuss some of the key features of our customized simplex algorithm.

3.3.1 Initialization

In order to start the simplex algorithm, we must first identify a basic feasible solution (BFS) of (FD). To obtain such a starting point, it suffices to express the relaxation solution x^* as a convex combination of $n+1$ affinely independent vertices of the box $\mathcal{H}$. The corresponding set of convex multipliers λ^0 is then a BFS of (FD). In [29], the author proposes a polynomial time algorithm to construct such a representation for a point that belongs to a polyhedral set. Bao et al. [2] present a similar approach for the special case where the point belongs to a box. In this paper, we propose to use a different initialization approach for reasons that will become clear shortly. Let e^k denote the kth unit vector in $\mathbb{R}^n$. Given any $x^* \in \mathcal{H} = \prod_{k=1}^{n} [l_k, u_k]$, let $\tilde{x}_k = (x^*_k - l_k)/(u_k - l_k)$, $k \in \{1, \ldots, n\}$. Denote by π a permutation of $\{1, \ldots, n\}$ such that $\tilde{x}_{\pi(1)} \geq \tilde{x}_{\pi(2)} \geq ... \geq \tilde{x}_{\pi(n)}$. It can be shown that the region defined by this set of inequalities is a simplex $\Delta_\pi \subset \mathcal{H}$, whose vertices are given by $\text{vert}(\Delta_\pi) = \{\nu^j : \nu^j = l + (u-l)\sum_{k=1}^{j-1} e^{\pi(k)},\ j = 1, \ldots n+1\}$. By construction, $x^* \in \Delta_\pi$. We can therefore express x^* as a convex combination of $\text{vert}(\Delta_\pi)$ as follows: $x^* = (1-\tilde{x}_{\pi(1)})\nu^1 + \sum_{j=2}^{n} (\tilde{x}_{\pi(j-1)} - \tilde{x}_{\pi(j)})\nu^j + \tilde{x}_{\pi(n)}\nu^{n+1}$. Thus, the convex multipliers associated with $\text{vert}(\Delta_\pi)$ are given by:

$$\tilde{\lambda}_1 = 1 - \tilde{x}_{\pi(1)},\ \tilde{\lambda}_j = \tilde{x}_{\pi(j-1)} - \tilde{x}_{\pi(j)},\ j = 2, \ldots, n,\ \tilde{\lambda}_{n+1} = \tilde{x}_{\pi(n)}.$$

Let $\mathcal{I} = \{i \in I : v^i = \nu^j, \text{ for some } j \in \{1, \ldots, n+1\}\}$ and, for each $i \in \mathcal{I}$, let $q(i)$ be equal to a j such that $v^i = \nu^j$. Then, the set λ^0 given by:

$$\lambda^0_i = \begin{cases} \tilde{\lambda}_{q(i)}, & \text{if } i \in \mathcal{I} \\ 0, & \text{otherwise,} \end{cases} \tag{9}$$

is a BFS of Problem (FD). As we discussed in Section 1, for certain types of multilinear functions, explicit characterizations of the envelopes are known. Ideally, we would like to solve Problem (FD), only if the facets of the convex envelope are not available in closed form. In particular, the authors of [34] prove that the convex envelope (resp. concave envelope) of a multilinear function whose restriction to the vertices of the box is submodular (resp. supermodular), is given by its Lovász extension. In [5], the author presents polynomial time algorithms to identify certain submodular/supermodular functions of the form (1). In addition, it can be shown that, for such functions, the set of convex multipliers defined by (9) are optimal for the envelope representation problem (see [34] for details). In our implementation, instead of including a preprocessing step to identify multilinears that satisfy the aforementioned conditions and avoiding the solution of an LP for constructing the envelope, we choose to utilize (9) as the starting point of the simplex algorithm, which is optimal for (FD) if $L(x)$, $x \in \text{vert}(\mathcal{H})$ is submodular.

3.3.2 Efficient storage and fast reduced cost computation

To solve the separation problem, all vertices of the box will be used by the simplex algorithm. Given a multilinear function with n variables, this requires the storage of $n2^n$ real numbers. To reduce the memory usage and speed up the computation of reduced costs, we utilize a one-to-one mapping from these vertices to the integer set $\mathcal{I} = \{0, 1, \ldots, 2^n - 1\}$, defined as follows. Given an integer $i \in \mathcal{I}$, consider its n-bit binary reflected Gray code representation [11], denoted by β_i. If the jth bit of β_i has a value equal to zero, then in the corresponding vertex, we have $v_j^i = l_j$; otherwise, $v_j^i = u_j$. Recall that a binary reflected Gray code is an encoding of numbers so that adjacent numbers have a single digit differing by one. For example, for $n = 3$ we order the vertices of the unit hypercube as follows $V = \{(0,0,0), (0,0,1), (0,1,1), (0,1,0), (1,1,0), (1,1,1), (1,0,1), (1,0,0)\}$. Now, consider two consecutive vertices v^i and v^{i+1} in set V. Assume that the Gray code representations of v^i and v^{i+1} differ in the jth bit. Define $\eta_i = j$, if $v_j^i = l_j$ and $\eta_i = -j$, if $v_j^i = u_j$; i.e., η_i represents a rule for obtaining v^{i+1} from v^i. For example, for the set V defined above, we have $\eta = [1, 2, -1, 3, 1, -2, -1]$. Hence, we can fully characterize the vertices of an n-dimensional box by an integer array of length $2^n - 1$, as opposed to $n2^n$ real numbers, which is required by the normal representation.

More importantly, ordering the vertices according to Gray code, as we discuss next, reduces the computational cost of the simplex algorithm considerably. Computing the reduced costs at each iteration of the simplex algorithm turns out to be one of the most time-consuming steps in solving the separation problem. At a given simplex iteration, denote by $\mathcal{N}$ the index set of nonbasic variables in (FD). Then, it can be shown that the reduced costs are given by $d_i = L(v^i) - (a^T v^i + b)$ for all $i \in \mathcal{N}$, where $a \in \mathbb{R}^n$, $b \in \mathbb{R}$ are as defined in (F2). Consider the subexpression $y_i = a^T v^i$ in d_i. Clearly, computing y_i requires n multiplications. As we described earlier, in our Gray code representation of vert($\mathcal{H}$), consecutive vertices correspond to adjacent vertices of the box. Now, consider two vertices v^i and v^{i+1} in the list. It is simple to check that $y_{i+1} = y_i - a_k(u_k - l_k)$, where $k = |\eta_i|$, and η is the integer vector relating the consecutive vertices, as defined above. Thus, assuming that y_i is given, it follows that y_{i+1} can be computed in $O(1)$. Since we may need to compute an exponential number of reduced costs to find a negative one at each iteration, this recursive method can speed up the algorithm significantly.

3.3.3 Preprocessing, LU factorization, and updates

In the course of the branch-and-bound algorithm, certain variables may be fixed at bounds due to branching and range reduction. This is most likely to happen in the case of integer variables. Such variables are eliminated in a preprocessing step in our implementation, thus reducing the size of the separation problem to be solved. The remaining problem (FD) is almost always fully dense and involves a relatively small number of rows. For this reason, we have built the implementation of the simplex algorithm on top of `LUMOD` [28], which provides routines for factorizing a dense basis matrix and updating a dense LU factorization. The `LUMOD` package allows for thousands of LU updates without refactorization and is highly efficient for the sizes of LP bases involved in the current application.

3.4 Cut generation

To exploit the local information regarding bounds on variables for cut generation, as well as to utilize the multilinear cuts for range reduction, we embed the proposed cutting plane generation scheme at every node of the branch-and-bound based solver

BARON. As we described in Section 2, for a general nonconvex problem containing multilinear structures, we can enhance the quality of an existing relaxation by solving Problem (FD). Namely, we obtain a facet $h(x)$ of the convex envelope of a multilinear function that separates the relaxation solution from the feasible region of the nonconvex problem (if such a facet exists). Now, suppose that we add $h(x)$ to the current relaxation and solve Problem (FD) to separate the optimal solution of the augmented relaxation from the feasible region. Clearly, we can continue this iterative procedure until no separating hyperplane can be generated. In addition to capturing the strength of the envelopes of multilinear functions, this cut generation technique bypasses the requirement of including exponentially many facets of the entire envelope, which is impractical for large problems. In fact, due to the cost of solving the separation problem, it is often highly beneficial to continue the iterative cut generation scheme only if the resulting cuts are sufficiently *deep*. Next, we describe our implementation in BARON.

At each node of the branch-and-bound tree, BARON first constructs and solves an initial outer-approximation of the problem based on a conventional factorable framework (see [37] for details). Subsequently, various classes of cutting planes are generated and added to the current relaxation iteratively, only if they reduce the size of the feasible region of the relaxation. These cuts are used locally in the sense that they are not passed on to the descendant nodes in the branch-and-bound tree.

We follow a similar approach to generate the multilinear cutting planes. Given a nonconvex factorable problem, recognition and decomposition algorithms of Sections 3.1 and 3.2 are employed prior to the initialization of the branch-and-bound tree. The resulting multilinear functions are stored in proper data structures and the required memory for storing the corresponding cutting planes is allocated. We do not include the multilinear cuts in the initial outer-approximation but consider them for the subsequent iterative cut generation. Suppose that the recognition and decomposition routines have identified n_L multilinear functions stored in list $\mathcal{L}$. At each node of the branch-and-bound tree, we utilize the multilinear cuts as follows. At a given cut generation iteration, we examine all multilinear functions $z = L_i(x)$ in $\mathcal{L}$. Denote by (x^*, z^*) the projection of the current relaxation solution to the (x, z) space. Using the techniques described in Section 2, we determine the type of the envelope (i.e., convex or concave) for cut generation. Suppose that a facet of the convex envelope is desired in this case. Next, we solve the separation problem using the simplex algorithm described in Section 3.3 to obtain a facet $h(x)$ of the convex envelope of $L_i(x)$. If $h(x^*) > z^*$, then we add $h(x)$ to the current relaxation. In addition, we compute the distance between the relaxation solution (x^*, z^*) and $h(x)$ to quantify the strength of the cut. If at the current iteration, the percentage of deep cuts does not exceed a predefined threshold, then the iterative multilinear cut generation algorithm terminates; that is, in the next round of BARON's cut generation module, no multilinear cut will be examined. Algorithm 9 presents the outline of our cut generation scheme.

4 Numerical Experiments

The purpose of this section is to demonstrate the computational benefits of incorporating multilinear cutting planes at every node of the branch-and-reduce global solver BARON [26,37]. To this end, we consider a variety of randomly generated test sets containing QCQPs, multilinear problems, and polynomial optimization problems. We solve these problems to global optimality using BARON 11.5 with and without mul-

Algorithm 9 Generating multilinear cutting planes at each node of the branch-and-bound tree

Given the relaxation solution, n_L multilinear functions stored in $\mathcal{L}$, parameters $\theta > 0$ and $\gamma_r \in (0, 1)$.
1. Initialize the list of cuts to be added to the relaxation $\mathcal{M} = \emptyset$, and the number of deep cuts $n_d = 0$.
2. For each multilinear function $z = L_i(x)$, in $\mathcal{L}$:
   Determine the type of the envelope (i.e., convex or concave).
   Solve the separation problem given by (FD) to obtain the cut $h(x)$:
     if $h(x)$ cuts off the relaxation solution, then
       add $h(x)$ to the relaxation $\mathcal{M} = \mathcal{M} \cup \{h(x)\}$
       compute the distance d between the relaxation solution and $h(x)$.
       if $d > \theta$, then $h(x)$ is a deep cut: $n_d \leftarrow n_d + 1$
     end of if
   end of for
3. If the number of deep cuts n_d is smaller than $\gamma_r n_L$, then this is the last round of multilinear cut generation.

Table 1 Default settings for the proposed cut generation scheme

Option	Description	Value
$n_{\min}$	Threshold for decomposing multilinears (Algorithms 5, 6)	4
$n_{\max}$	Maximum number of variables in multilinears (Algorithm 6)	15
β_1	Penalty parameter for the gain factor (Algorithm 6)	0.1
β_2	Penalty parameter for the gain factor (Algorithm 6)	1.25
α	Reduction parameter for the decomposition scheme (Algorithm 7)	0.6
γ_c	Covering parameter for the decomposition scheme (Algorithm 8)	85%
γ_r	Threshold to proceed to the next round of cut generation (Algorithm 8)	0.05
θ	Deep cut measure (Algorithm 9)	10^{-3}

tilinear cuts. In addition, we assess the efficiency of the proposed decomposition and cut generation schemes by examining a number of simpler strategies. After extensive experimentation, we set the main parameters of our cut generation scheme as listed in Table 1. Throughout this section, all problems are solved with relative/absolute optimality tolerance of 10^{-6} and a CPU time limit of 500 seconds. Other algorithmic parameters are set to the default settings of the `GAMS/BARON` distribution [27]. When comparing the performance of different algorithms, we say that a problem is *trivial*, if all algorithms take less than one second to solve it to global optimality. All tests were performed on a 64-bit Intel Xeon X5650 2.66Ghz processor using a development version of `BARON 11.5` [37] interfaced with `CPLEX 12.4` [14], `IPOPT 3.9.0` [38], `MINOS 5.5` [23] and `SNOPT 7.2.4` [10] for solving LP/NLP subproblems.

4.1 The test set

We consider a polynomial optimization problem of the form:

$$
\begin{aligned}
\text{(PL)} \qquad \min \quad & f_0(x) \\
\text{s.t.} \quad & f_i(x) \le b_i, \ \forall i = 1, \ldots, q \\
& x \in [0, 1]^n,
\end{aligned}
$$

where $b \in \mathbb{R}^q$, $f_i(x)$, $i = 0, 1, \ldots, q$ are multivariate polynomials

$$f_i(x) = \sum_{T \in \Omega_i} c_T \prod_{j \in T} x_j^{a_j},$$

Ω_i is a collection of subsets of $\{1, \ldots, n\}$, c_T, $T \in \Omega_i$ are nonzero real-valued coefficients, and the exponents a_j, $j \in T$ are positive integers. By convention, the degree of a monomial $\prod_{j \in T} x_j^{a_j}$ is the sum of its exponents, and the degree a polynomial function $f_i(x)$ is the largest degree of its monomials, i.e., $d_i = \max_T \sum_{j \in T} a_j$. Similarly, the degree of the polynomial optimization problem (PL) is defined as the largest degree of its polynomials $d = \max_i d_i$, $i = \{0, 1, \ldots, q\}$. If $d = 2$, then (PL) is a QCQP. In addition, by letting $q = 0$, we obtain a box-constrained optimization problem. In the following, we assume that $d_i > 1$ for all $i \in \{0, 1, \ldots, q\}$.

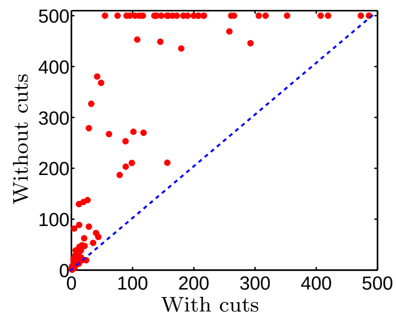
For a polynomial f_i with n variables and degree d_i, it can be shown that the maximum number of monomial terms is given by $W_i = \sum_{k=1}^{d_i} \binom{d_i}{k}\binom{n}{k}$, where $\binom{n}{k}$ is zero if $n < k$. Denote by w_i the number of nonlinear monomials in f_i. We associate a density with each polynomial f_i defined as $\nu_i = w_i/(W_i - n)$. Accordingly, when $\nu_i = \nu$ for all $i \in \{0, 1, \ldots, q\}$, we say that the density of Problem (PL) is given by ν. Throughout this section, we characterize a polynomial optimization problem by its degree (d), number of variables (n), number of constraints (q), and density (ν). We also consider multilinear optimization problems that can be obtained by replacing the polynomial functions of Problem (PL) by multilinear functions; i.e., $a_j = 1$ for all j. For multilinear problems, we adapt the terminology defined above for polynomials, by noting that in this case we have $W_i = \sum_{k=2}^{d_i} \binom{n}{k}$. For our numerical experiments, we generated three sets of problems:

- Set 1. Quadratic problems with $n \in \{20, 30, 40, 50\}$, and $\nu = \{0.25, 0.5, 0.75, 1.0\}$.
- Set 2. Polynomial problems of degree 3 with $(n, \nu) \in \{(10, 0.75), (15, 0.25), (20, 0.1)\}$, and multilinear problems of degree 3 with $(n, \nu) \in \{(10, 1.0), (15, 0.5), (20, 0.15)\}$.
- Set 3. Polynomial problems of degree 4 with $(n, \nu) \in \{(10, 0.25), (15, 0.05)\}$, and multilinear problems of degree 4 with $(n, \nu) \in \{(10, 1.0), (15, 0.15), (20, 0.02)\}$.

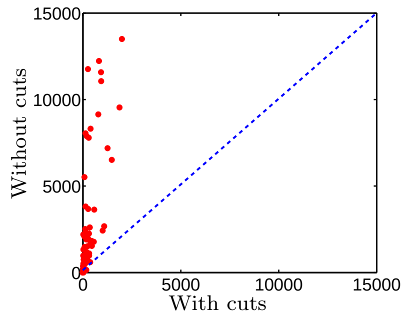
In all three sets, we let $q \in \{0, n/5, n/2, n\}$. For each combination of $\{d, n, q, \nu\}$, we generated five problem instances, where the problem data were randomly generated from uniform distributions: the polynomial coefficients c_T were generated in the range $[-1, 1]$, while the righthand side values b_i were generated in the range $[0, 100]$. Overall, our test sets contains 320 quadratic problems, 120 polynomial problems of degree three, and 100 polynomial problems of degree four. This collection is designed to examine the effect of multilinear cuts on small- to medium-sized problems with different sparsity characteristics, ranging from boxed-constrained problem to those that are highly constrained.

4.2 Comparisons with termwise relaxations

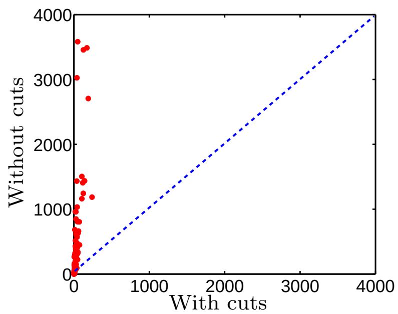
We solve the three test sets described in the previous subsection to global optimality using `BARON 11.5`, with and without multilinear cutting planes. For nonconvex problems of form (PL), `BARON`'s factorable bounds consist of supporting hyperplanes and affine envelopes for univariate monomials, along with a recursive application of bilinear envelopes to bound multilinear terms (see Algorithm 1). In order to construct stronger



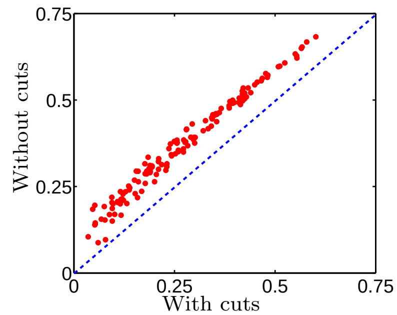
(a) CPU time (s)



(b) Iterations



(c) Memory



(d) Relative gap

Fig. 3 Performance of `BARON 11.5` with and without multilinear cutting planes for 320 randomly generated quadratic optimization problems. In Figs. 3(a)-3(c), nontrivial problems that are solved in less than 500 s are compared with respect to CPU time, number of iterations, and memory requirements. In Fig. 3(d), final relative gaps for problems that are not solvable within the time limit are compared.

relaxations, we employ the cut generation scheme of Section 3.4 to incorporate multilinear cutting planes in `BARON`. For quadratic (resp. multilinear) problems, these cuts correspond to bilinear (resp. multilinear) functions present in the original formulation, whereas, for polynomial problems multilinear cuts are generated for intermediate multilinears in the lifted space (see Equation (4) and the discussion that follows).

To compare the performance of the two algorithms, we consider the following factors: (i) execution time, (ii) total number of nodes in the branch-and-bound tree (iterations), (iii) maximum number of nodes stored in memory (memory), (iv) number of problems solved within the time limit, and (v) the normalized difference between the best lower bound (L) and upper bound (U) for problems that are not solved to global optimality within the time limit: $(U - L)/|L|$ (relative gap).

Computational results for the quadratic test set are presented in Figure 3. For a meaningful comparison, we eliminated trivial problems from the test set (61 instances). For those problems that are solvable within 500 s by at least one of the two algorithms (119 instances), incorporating multilinear cuts in `BARON` results in average reductions of 60% in CPU time, and 90% in number of iterations as well as in maximum number of nodes in memory. Furthermore, utilizing multilinear cuts leads to a 35% increase in the number of nontrivial problems that are solved to global optimality in less that 500 seconds.

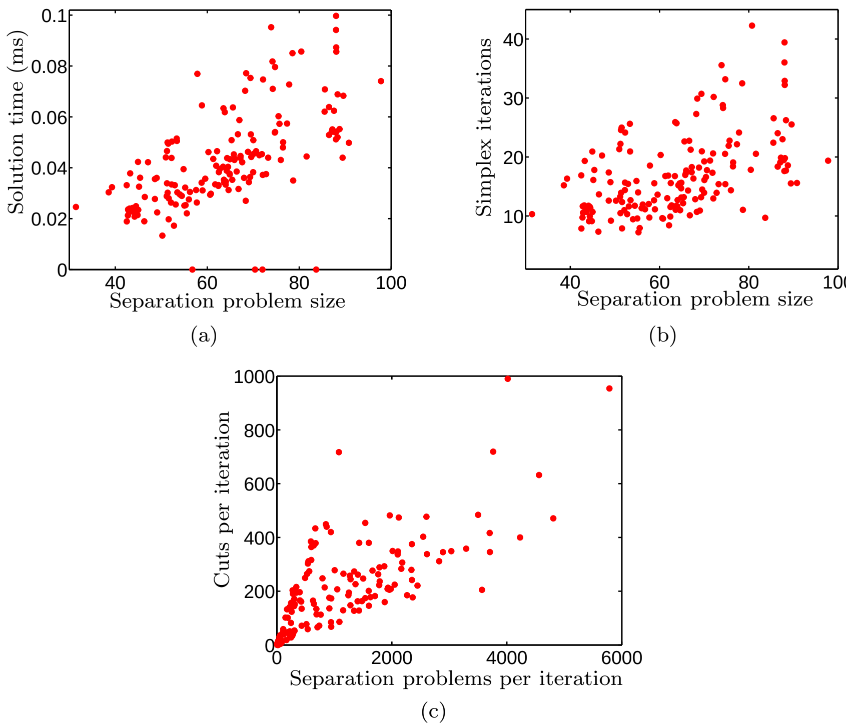


Fig. 4 Multilinear cutting plane statistics for quadratic optimization problems. In Fig. 4(a), average solution time (in milliseconds) versus average size of separation problems are shown for each problem in the test set. In Fig. 4(b), average number of simplex iterations for solving separation problems versus average size of separation problems is shown. In Fig. 4(c), the average number of separation problems solved per iteration of the branch-and-bound tree is compared with the average number of multilinear cutting planes generated per iteration.

The results of Figure 3 are analyzed in Table 2(a) to further quantify the effect of incorporating the multilinear cuts. The first line of Table 2(a) provides the percentage of nontrivial problems for which multilinear cutting planes lead to at least a factor of two improvement with respect to the total number of iterations, memory requirements, and CPU time. The subsequent lines of the table provides similar statistics for problems for which the algorithm was improved by at least 30% but no more than an order of magnitude, problems for which there was no significant performance change after addition of cutting planes, and problems for which there was some deterioration in performance because of cutting planes. As can be seen from Table 2(a), for more than 60% of quadratic optimization problems, multilinear cutting planes improve `BARON`'s execution time by at least an order of magnitude. Clearly, improvements in total and maximum number of nodes are more significant due to the time spent on generating the multilinear cuts. For this test set, it turns out that on average 5% of the overall time is spent on recognition and decomposition of multilinears, while 15% is spent on solving the separation problem to generate multilinear cuts. In Figure 4, we provide detailed statistics regarding the size of separation problems and the computational cost of the proposed simplex algorithm. In this test set, separation problems are small

Table 2 Effect of adding multilinear cutting planes to BARON

(a) Quadratic optimization problems

Effect of adding cuts	Iterations	Memory	CPU time
Better by a factor at least 2	84%	86%	61%
Between 30% and 100% better	3%	1%	22%
Difference smaller than 30%	11%	13%	17%
Between 30% and 100% worse	1%	0%	0%
Worse by a factor of at least 2	1%	0%	0%

(b) Third-order polynomial and multilinear optimization problems

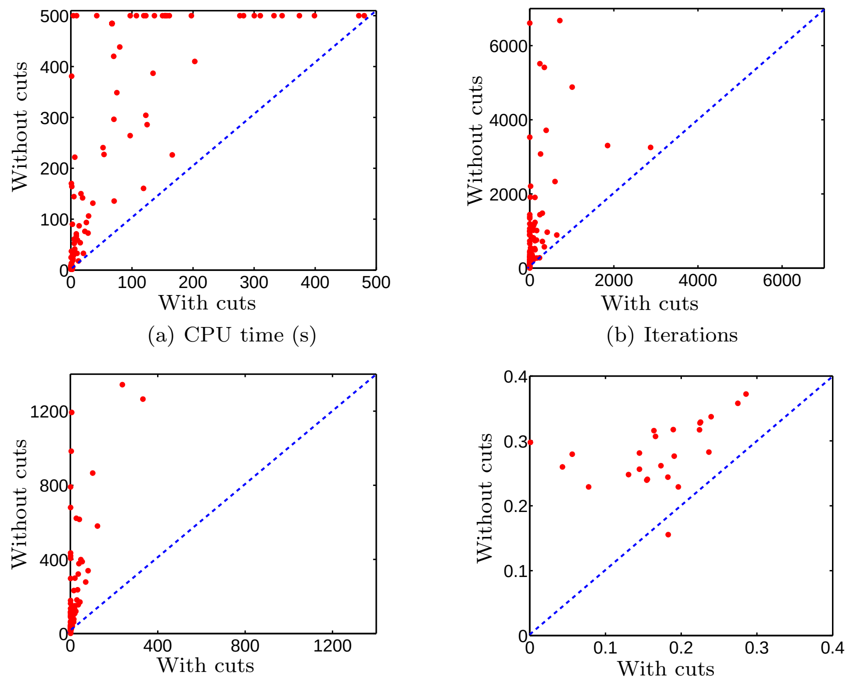
Effect of adding cuts	Iterations	Memory	CPU time
Better by a factor at least 2	93%	99%	82%
Between 30% and 100% better	4%	0%	14%
Difference smaller than 30%	3%	1%	3%
Between 30% and 100% worse	0%	0%	1%
Worse by a factor of at least 2	0%	0%	0%

(c) Fourth-order polynomial and multilinear optimization problems

Effect of adding cuts	Iterations	Memory	CPU time
Better by a factor at least 2	92%	98%	81%
Between 30% and 100% better	2%	2%	6%
Difference smaller than 30%	6%	0%	11%
Between 30% and 100% worse	0%	0%	2%
Worse by a factor of at least 2	0%	0%	0%

size LPs with $16 - 128$ variables, and are often solved after a few tens of simplex iterations, which takes less than 0.1 milliseconds. Moreover, on average, one fourth of the separation problems yield facets that strengthen the existing relaxation. We will provide further details on the performance of decomposition and cut generation schemes in the next subsection.

Next, we consider polynomial and multilinear problems of degree three. Our results are shown in Figure 5, and summarized in Table 2(b). In this case, we do not have any trivial problems. For 96 instances that are solvable within 500 s by at least one of the two algorithms, adding multilinear cuts results in average reductions of 70% in CPU time, 85% in number of iterations, and 90% in maximum number of nodes in memory. In addition, by utilizing multilinear cuts, we are able to increase the number of solvable problems by 33%. For this test set, on average 1.0% of the total time is spent on recognition and decomposition, while 45% is spent on cut generation. The computational effort to solve the separation problems, in terms of average number of simplex iterations and average solution time, is shown in Figure 8. Compared to the quadratic test set, the average size of separation problems is considerably larger. For LPs with 2^{15} variables, the simplex algorithm takes $10^3 - 5 \times 10^3$ iterations, which translates into solution times from 0.1 s to 1.0 s. For this test set, on average, 40% of the separation problems yield cutting planes that violate the existing relaxation. Generation of the multilinear cuts takes a significant fraction of the total CPU time of the algorithm but the derived cuts improve the performance of BARON by least an order of magnitude for more than 80% of the test problems (see Table 2(b)). Details on the



(a) CPU time (s)

(b) Iterations

(c) Memory

(d) Relative gap

Fig. 5 Performance of `BARON 11.5` with and without multilinear cutting planes for 120 randomly generated third-order polynomial optimization problems. In Figs. 5(a)-5(c) nontrivial problems that are solved in less than 500 s are compared with respect to CPU time, number of iterations and memory requirements. In Fig. 5(d), final relative gaps for problems that are not solvable within the time limit are compared.

relation between the dimension of multilinear cuts and the structure of the nonconvex problems are provided in the next subsection.

Finally, we examine the effect of multilinear cuts on polynomial and multilinear problems of degree four. Out of 100 problems, 64 instances are solvable to global optimality within 500 s, none of which is trivial. As can be seen from Figure 7 and Table 2(c), the proposed cuts significantly improve the performance of `BARON` for this test set. Specifically, we obtain average reductions of 70% in CPU time, 90% in number of iterations, and 95% in maximum number of nodes in memory. Moreover, the number of problems that are solvable within 500 s has increased from 25 to 63; i.e., an improvement of 152%. For this test set, on average 1.5% of the total time is spent on recognition and decomposition, while 55% is spent on cut generation. Figure 6 shows average characteristics of the separation problems solved in this test set. Similar to the third-order test set, solution times vary from a few milliseconds for multilinears with $n < 10$, to one second for multilinears with $n = 15$. In addition, for this collection, on average, approximately half of the separation problems provide cutting planes that reduce the size of the feasible region of the relaxation.

We should remark that in all three test sets, the decomposition time ranges from 0.001 s for smaller problems with a few constraints to 1.0-2.0 s for larger problems with many constraints. The relatively large value for the average ratio of decomposition time

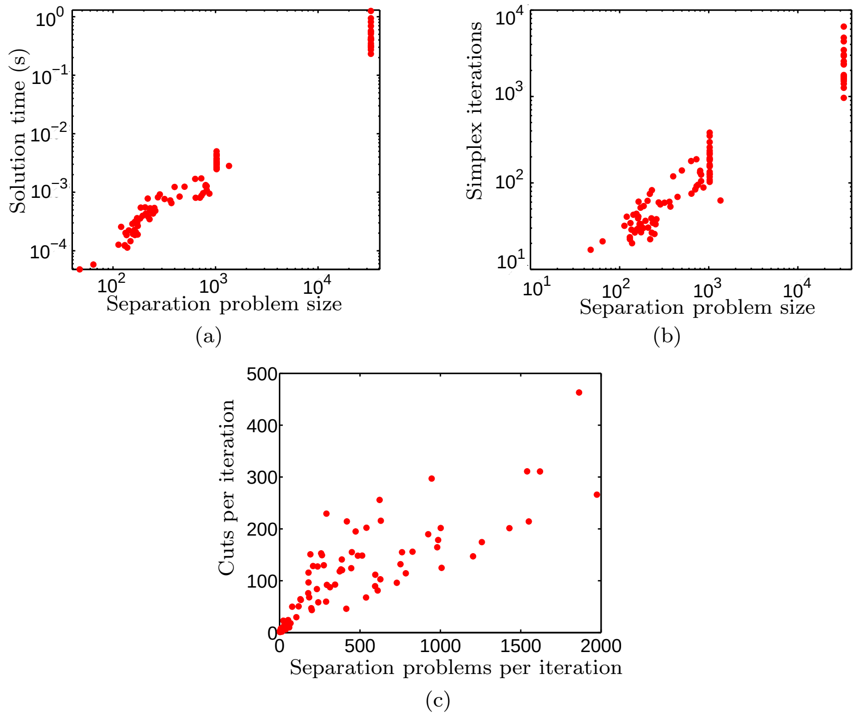


Fig. 6 Multilinear cutting plane statistics for third-order polynomial optimization problems. In Fig. 6(a), average solution time versus average size of separation problems are shown for each problem in the test set. In Fig. 6(b), average number of simplex iterations for solving separation problems versus average size of separation problems is shown. In Fig. 6(c), the average number of separation problems solved per iteration of the branch-and-bound tree is compared with the average number of multilinear cutting planes generated per iteration.

to total time for quadratic problems (i.e., 5%) is due to the presence of simple problems in this test set. These are problems that can be solved within a few seconds even without multilinear cuts. Results of Table 2 indicate that multilinear cuts are more effective for problems containing higher order multilinears. Moreover, in all three test sets, the strength of multilinear cuts degrades by increasing the number of constraints. This empirical observation is in agreement with theory, as the proposed cuts correspond to a single constraint, and finding tight relaxations for individual constraints does not necessarily provide a strong relaxation for the feasible region. However, our cut generation technique can be extended to convexify a collection of constraints whose convex hull can be finitely generated (cf. [32]).

4.3 Decomposition and cut generation schemes

The goals of this section are two fold. First, to demonstrate the key role of our decomposition algorithm in efficient solution of nonconvex problems with multilinear intermediates to global optimality. Second, compare the dynamic cut generation strategy proposed in Section 3.4 with static approaches that add cutting planes to the

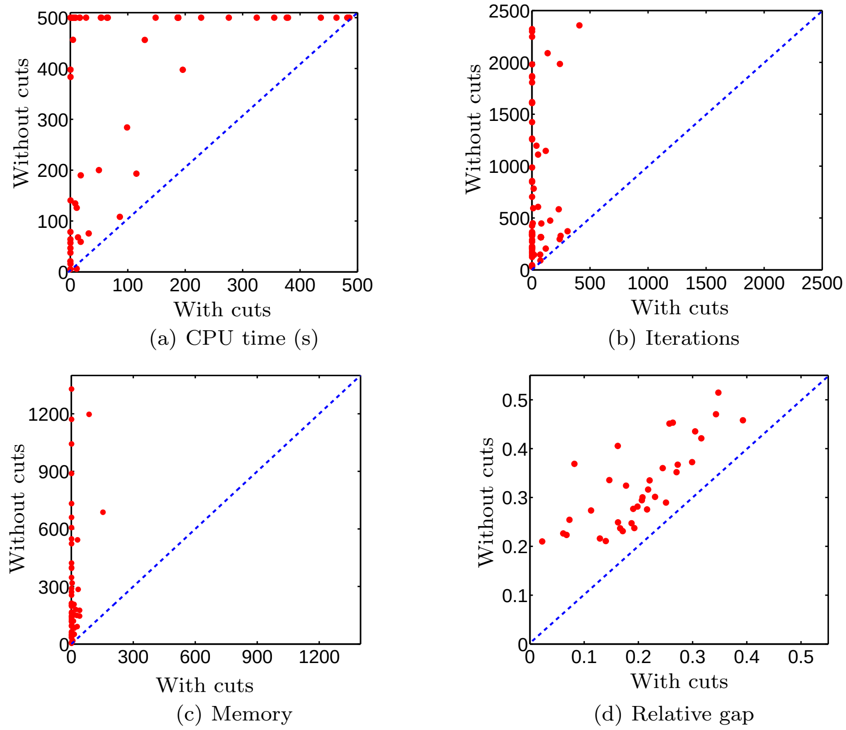


Fig. 7 Performance of `BARON 11.5` with and without multilinear cutting planes for 100 randomly generated fourth-order polynomial optimization problems. In Figs. 7(a)-7(c) nontrivial problems that are solved in less than 500 s are compared with respect to CPU time, number of iterations and memory requirements. In Fig. 7(d), final relative gaps for problems that are not solvable within the time limit are compared.

relaxation for a predefined number of rounds. To assess the performance of alternative algorithms, we make use of performance profiles, as described in [7]. In the following, we use the ratio of the CPU time of an algorithm versus the best time of all algorithms as the performance metric. Denote by $t_{p,s}$ the time that algorithm s takes to solve problem p. For comparisons, we define a time ratio as $r_{p,s} = t_{p,s}/\min\{t_{p,s} : s \in \mathcal{S}\}$, where $\mathcal{S}$ refers to the set of all algorithms. It follows that the distribution function of time ratio is given by:

$$\Phi_s(\tau) = \text{size}\{p \in \mathcal{P} : r_{p,s} \leq \tau\}/|\mathcal{P}|,$$

where $\mathcal{P}$ denotes the set of all problems.

The first experiment is designed to analyze the importance of decomposing multilinears to lower-dimensional components prior to solving the separation problem. We disable the decomposition algorithm and solve the three problem sets of previous section, while keeping all other parameters unchanged. Compared to the results with decomposition, for the quadratic test set, the average CPU time for nontrivial problems that are solvable within 500 s increases by 1645%, and the number of solvable problems degrades by 82%. Interestingly, for this test set, even `BARON` with no multilinear cuts wins over the no-decomposition strategy by a large margin. This significant

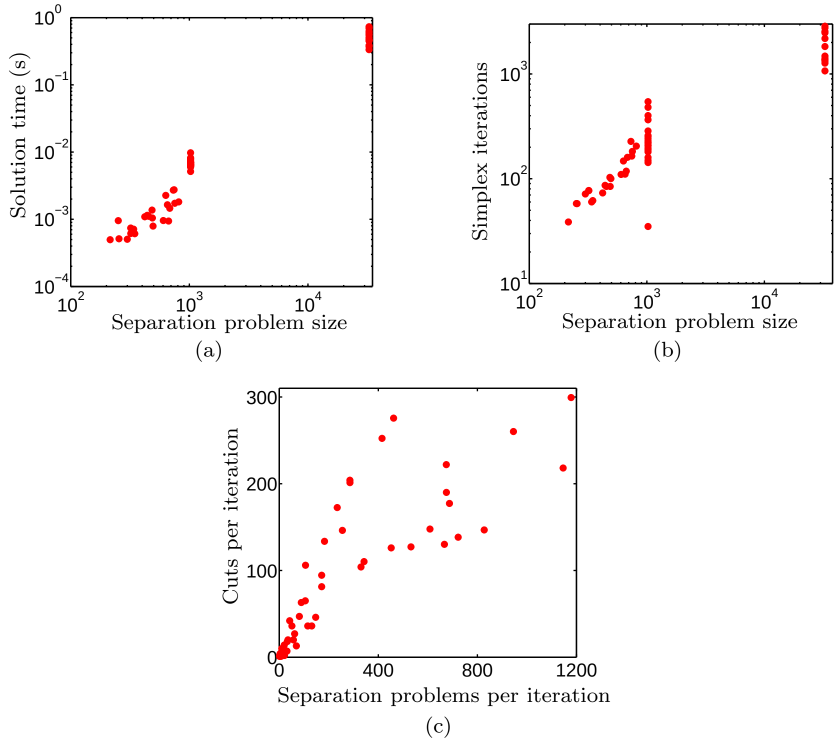
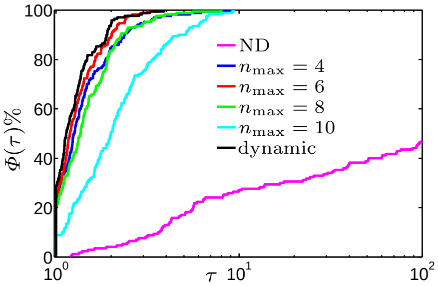


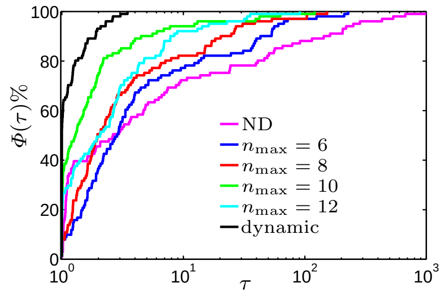
Fig. 8 Multilinear cutting plane statistics for fourth-order polynomial optimization problems. In Fig. 8(a), average solution time versus average size of separation problems are shown for each problem in the test set. In Fig. 8(b), average number of simplex iterations for solving separation problems versus average size of separation problems is shown. In Fig. 8(c), the average number of separation problems solved per iteration of the branch-and-bound tree is compared with the average number of multilinear cutting planes generated per iteration.

deterioration is due to the high computational cost of solving the separation problem for multilinears with more than 15 variables. In fact, for quadratics with more than 30 variables, the entire execution time is often spent on solving the first separation problem. For the polynomial problems of order three and four, disabling the decomposition algorithm leads to a CPU time increase of 415% and 150%, respectively. The latter degradations are less significant than that of the quadratic test set, as quadratic test problems have more variables and fewer terms; i.e., the case that benefits the most from decomposition (see Figure 9).

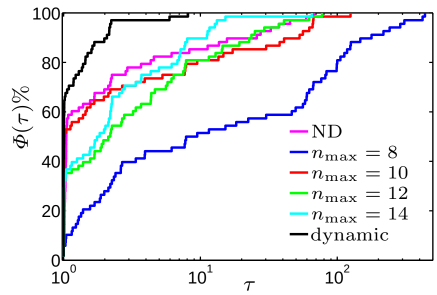
As we described in Section 3.2, to identify dense components of a multilinear, we employ an adaptive graph partitioning scheme in which we use the `METIS` solver to find a number good partitions, and subsequently choose the one with the largest gain factor, defined by (8). We now demonstrate the computational benefits of this dynamic scheme in comparison to a simpler approach in which the size of sub-multilinears is determined a priori. More precisely, in Step 2 of Algorithm 8, instead of using Algorithm 6 to generate dense components, we apply the following strategy. Let n be the number of variables in a multilinear function $L(x)$ and let $n_{\max}$ be the maximum number of variables in subfunctions. Let G denote the weighted graph associated with $L(x)$,



(a) Quadratic problems



(b) Third-order problems



(c) Fourth-order problems

Fig. 9 Effect of different decomposition strategies. Each figure shows the performance profiles of `BARON` with alternative decompositions schemes for a test set: ND refers to the algorithm without decomposition, $n_{\max}$ is the maximum number of variables in each multilinear, and "dynamic" refers to the proposed decomposition algorithm. The time ratio is denoted by τ with the distribution function denoted by $\Phi(\tau)$.

defined as before. We define the number of subgraphs of G as $\rho_0 = \lceil n/n_{\max} \rceil$, and the size of subgraphs as $s_\rho = n_{\max}$ for $\rho = 1, \ldots, \rho_0 - 1$, and $s_{\rho_0} = n - (\rho_0 - 1)n_{\max}$. Subsequently, we use `METIS` to find a partition with minimum edge-cut, and construct the resulting subfunctions as before. In addition, in Algorithm 5, we let $n_{\min} = n_{\max}$; i.e., any multilinear function corresponding to a biconnected component with no more than $n_{\max}$ nodes is stored for cut generation. We now show that the performance of this static approach highly depends on the value of $n_{\max}$, and a good choice for $n_{\max}$ depends on the structure of the problem.

Performance profiles for our three test sets with different values of $n_{\max}$ are shown in Figure 9. As Figures 9(a) indicates, for quadratic problems, the best performance is achieved for $4 \leq n_{\max} \leq 8$, and increasing the value of $n_{\max}$ beyond this range results in rapid deterioration. For this collection, the average CPU time attains its minimum value at $n_{\max} = 6$. As seen in Figure 9(b), for the third-order problems, $n_{\max} = 10$ significantly outperforms $n_{\max} = 6$ and, for larger values, the performance of the global solver degrades. Finally, Figure 9(c) shows that for the fourth-order collection, $n_{\max} = 8$ is a poor choice as it is dominated by the algorithm with no decomposition. In addition, even larger values for $n_{\max}$ do not lead to significant improvements for this test set. These results support our earlier discussion in Section 3.2, as the graphs of higher-order multilinears have a much larger density than the graphs of quadratics

Table 3 Size statistics for multilinear functions constructed by the proposed decomposition scheme. For each quantity, we report the average value over the entire set of problems in the corresponding collection with its standard deviation shown in parentheses

Test set	s_{min}	s_{ave}	s_{max}
Quadratic	3.3 (0.5)	5.6 (0.3)	6.8 (0.4)
Third order	7.2 (4.9)	9.3 (3.4)	11.2 (2.0)
Fourth order	9.2 (4.9)	11.0 (2.8)	11.7 (2.3)

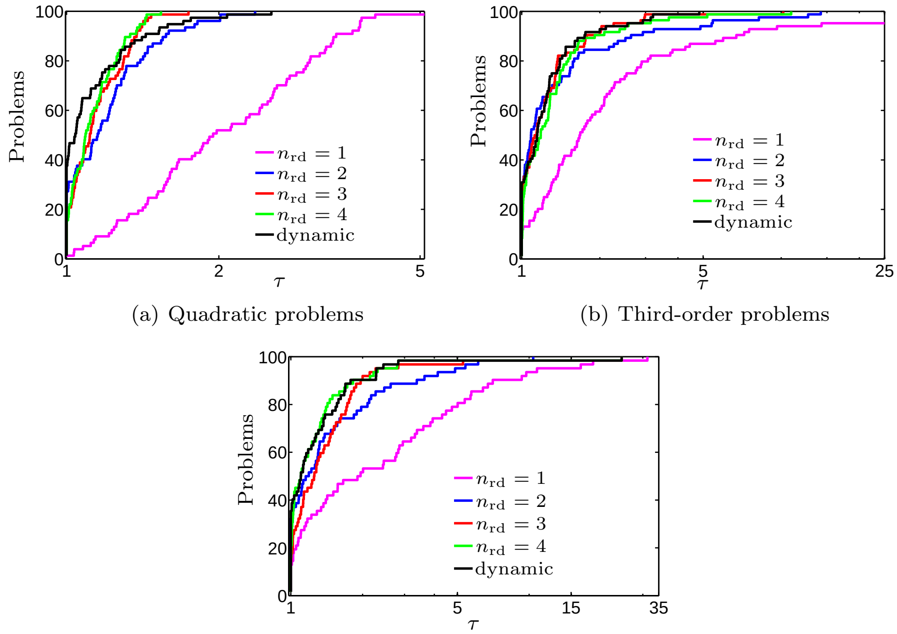
(density of a weighted graph is defined as the sum of its edge-weights). Hence, by allowing higher-dimensional subfunctions, we construct a much stronger relaxation. As can be seen from Figure 9, the adaptive decomposition scheme performs well across the three test sets, and in fact significantly outperforms all static algorithms for third- and fourth-order problems.

In Table 3, we provide the statistics for the size of multilinears obtained by the adaptive decomposition approach. Namely, for each problem, we present minimum (s_{min}), average (s_{ave}), and maximum (s_{max}) number of variables per multilinear and report the average value and the standard deviation for each quantity over the corresponding test set. The results of Table 3 are in complete agreement with the performance profiles of Figure 9. For the quadratic collection, the decomposition scheme constructs low-dimensional subfunctions, most of which have 4 to 6 variables, while for higher-order problems, most of the subfunctions contain 8 to 12 variables. Moreover, large standard deviations for third-order and fourth-order problems indicate that a fixed value for n_{max} is a poor choice, a conclusion that is confirmed by Figures 9(b) and 9(c).

Finally, let us revisit the cut generation scheme outlined in Algorithm 9. As described in Section 3.4, at each node of the branch and bound tree, `BARON` adds various classes of cutting planes to the relaxation in multiple rounds. Cuts are added only if they violate the relaxation, and if no such cutting planes exist at a given round, then the cut generator terminates. By `BARON`'s default setting, up to 4 rounds of cutting planes are allowed. In our implementation of multilinear cuts, we departed from this static strategy in order to be able to address potentially costly separation problems. We do this through a dynamic strategy as follows. Let n_{rd} denote the maximum allowable rounds of cut generation. To examine the efficiency of the proposed cut generation scheme, we solve the test sets using `BARON`'s static cut generation for $n_{\text{rd}} = 1, \ldots, 4$. Performance profiles of the alternative algorithms are depicted in Figure 10. Clearly, in all test sets, $n_{\text{rd}} = 1$ is dominated by other schemes. Among the static strategies, for the quadratic and for third-order problems $n_{\text{rd}} = 3, 4$ are preferable, and for the fourth-order collection $n_{\text{rd}} = 4$ is dominant. Figure 10 indicates that, dynamic cut generation is quite competitive with the best static approaches for all three test sets.

5 Conclusions

In this paper, we described an efficient implementation of multilinear cutting planes for global optimization of problems with multilinear intermediates. We developed a decomposition scheme that exploits the structure of a multilinear to identify low-dimensional multilinear subfunctions, whose convex hulls can be approximated closely within a reasonable computational effort. Moreover, we complemented this decomposition strategy



(a) Quadratic problems

(b) Third-order problems

(c) Fourth-order problems

Fig. 10 Effect of different cut generation strategies. Each figure shows the performance profiles of `BARON` with alternative cut generation schemes for a test set: n_{rd} denotes the number of cut generation rounds for static strategies, and "dynamic" refers to the proposed cut generation scheme. The time ratio is denoted by τ with the distribution function denoted by $\Phi(\tau)$.

by a customized simplex algorithm that features efficient initializations and tailored data structures, with dense LU factorization and updates. We incorporated the proposed cut generation scheme at every node of the branch-and-reduce global solver `BARON`. To demonstrate the efficiency of the proposed implementation, we considered various sets of test problems, including QCQPs, multilinear problems, and polynomial problems. Results show that multilinear cutting planes significantly accelerate the convergence rate of the branch-and-bound algorithm, and enable `BARON` to solve many more problems to global optimality. In particular, for a total number of 279 problems, the average CPU time and total number nodes in the branch-and-bound tree were reduced by 60% and 90%, respectively. Several extensions of this work are possible, as the developed approach provides building blocks for simultaneous convexification of a collection of multilinear functions, as well as convexification of general nonconvex functions with polyhedral envelopes.

References

1. F. A. Al-Khayyal and J. E. Falk. Jointly constrained biconvex programming. *Mathematics of Operations Research*, 8:273–286, 1983.
2. X. Bao, N. V. Sahinidis, and M. Tawarmalani. Multiterm polyhedral relaxations for nonconvex, quadratically-constrained quadratic programs. *Optimization Methods and Software*, 24:485–504, 2009.

3. P. Belotti. *COUENNE:* A user's Manual, 2009. Technical report, Lehigh University.
4. D. Bertsimas and J. N. Tsitsiklis. *Introduction to linear optimization*. Athena Scientific, 1997.
5. Y. Crama. Recognition problems in polynomials in 0−1 programming. *Mathematical Programming*, 44:139–155, 1989.
6. Y. Crama. Concave extensions for non-linear 0−1 maximization problems. *Mathematical Programming*, 61:53–60, 1993.
7. E. Dolan and J. More. Benchmarking optimization software with performance profiles. *Mathematical programming*, 91:201–213, 2002.
8. J. E. Falk and R. M. Soland. An algorithm for separable nonconvex programming problems. *Management Science*, 15:550–569, 1969.
9. M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete problems. In *Proceedings of the sixth annual ACM symposium on Theory of computing*, pages 47–63. ACM, 1974.
10. P. E. Gill, W. Murray, and M. A. Saunders. User's Guide for SNOPT 7.2.4: A FORTRAN Package for Large-Scale Nonlinear Programming. Technical report, University of California, San Diego and Stanford University, CA, 2008.
11. F. Gray. Pulse code communication. *U.S. patent no. 2,632,058*, 1953.
12. B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. In *Proceedings of the 1995 ACM/IEEE conference on Supercomputing*, page 28. ACM, 1995.
13. J. Hopcroft and R.Tarjan. Efficient algorithms for graph manipulation. *Communications of the ACM*, 16:372–378, 1973.
14. IBM. *CPLEX Optimizer*, 2011. `http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/`.
15. G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20:359–392, 1999.
16. B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal*, 49:291–307, 1970.
17. Y. Lin and L. Schrage. The global solver in the LINDO API. *Optimization Methods and Software*, 24:657–668, 2009.
18. J. Luedtke, M. Namazifar, and J. T. Linderoth. Some results on the strength of relaxations of multilinear functions. *Technical Report, Computer Sciences Department, University of Wisconsin-Madison*, 2010.
19. G. P. McCormick. Computability of global solutions to factorable nonconvex programs: Part I—Convex underestimating problems. *Mathematical Programming*, 10:147–175, 1976.
20. C. A. Meyer and C. A. Floudas. Trilinear monomials with positive or negative domains: Facets of the convex and concave envelopes. *Frontiers in Global Optimization, C.A. Floudas and P.M. Pardolos (eds.), Kluwer Academic Publishers*, 103:327–352, 2003.
21. C. A. Meyer and C. A. Floudas. Trilinear monomials with mixed sign domains: Facets of the convex and concave envelopes. *Journal of Global Optimization*, 29:125–155, 2004.
22. C. A. Meyer and C. A. Floudas. Convex envelopes for edge-concave functions. *Mathematical Programming*, 103:207–224, 2005.
23. B. A. Murtagh and M. A. Saunders. MINOS 5.5 User's Guide. Technical Report SOL 83-20R, Systems Optimization Laboratory, Department of Operations Research, Stanford University, CA, 1995.
24. A. D. Rikun. A convex envelope formula for multilinear functions. *Journal of Global Optimization*, 10:425–437, 1997.
25. H. S. Ryoo and N. V. Sahinidis. Analysis of bounds for multilinear functions. *Journal of Global Optimization*, 19:403–424, 2001.
26. N. V. Sahinidis. BARON: A general purpose global optimization software package. *Journal of Global Optimization*, 8:201–205, 1996.
27. N. V. Sahinidis and M. Tawarmalani. *BARON 10.3: Global Optimization of Mixed-Integer Nonlinear Programs,* User's Manual, 2012.
28. M. A. Saunders. *LUMOD:* Fortran software for updating dense LU factors. `http://www.stanford.edu/group/SOL/software/lumod.html`.
29. H. D. Sherali. A constructive proof of the representation theorem for polyhedral set based on fundamental definitions. *American Journal of Mathematical and Management Sciences*, 7:253–270, 1987.
30. H. D. Sherali. Convex envelopes of multilinear functions over a unit hypercube and over special discrete sets. *Acta Mathematica Vietnamica*, 22:245–270, 1997.

31. F. Tardella. Existence and sum decomposition of vertex polyhedral convex envelopes. *Optimization Letters*, 2:363–375, 2008.
32. M. Tawarmalani. Inclusion certificates and simultaneous convexification of functions. *Mathematical Programming*, submitted.
33. M. Tawarmalani, J-P. Richard, and K. Chung. Strong valid inequalities for orthogonal disjunctions and bilinear covering sets. *Mathematical Programming*, 124:481–512, 2010.
34. M. Tawarmalani, J-P. Richard, and C. Xiong. Explicit convex and concave envelopes through polyhedral subdivisions. *Mathematical Programming*, submitted.
35. M. Tawarmalani and N. V. Sahinidis. *Convexification and Global Optimization in Continuous and Mixed-Integer Nonlinear Programming: Theory, Algorithms, Software, and Applications*. Kluwer Academic Publishers, Dordrecht, 2002.
36. M. Tawarmalani and N. V. Sahinidis. Global optimization of mixed-integer nonlinear programs: A theoretical and computational study. *Mathematical Programming*, 99:563–591, 2004.
37. M. Tawarmalani and N. V. Sahinidis. A polyhedral branch-and-cut approach to global optimization. *Mathematical Programming*, 103:225–249, 2005.
38. A. Wächter and L. T. Biegler. On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106:25–57, 2006.