A Mixed-Integer Linear Programming Reduction of Disjoint Bilinear Programs via Symbolic Variable Elimination

Jihwan Jeong[1,3], Scott Sanner[1,3], and Akshat Kumar[2]

[1] University of Toronto, Toronto, Canada
[2] Singapore Management University, Singapore
[3] Vector Institute, Toronto, Canada
jihwan.jeong@mail.utoronto.ca, ssanner@mie.utoronto.ca, akshatkumar@smu.edu.sg

Abstract. A disjointly constrained bilinear program (DBLP) has various practical and industrial applications, e.g., in game theory, facility location, supply chain management, and multi-agent planning problems. Although earlier work has noted the equivalence of DBLP and mixed-integer linear programming (MILP) from an abstract theoretical perspective, a practical and exact closed-form reduction of a DBLP to a MILP has remained elusive. Such explicit reduction would allow us to leverage modern MILP solvers and techniques along with their solution optimality and anytime approximation guarantees. To this end, we provide the first constructive closed-form MILP reduction of a DBLP by extending the technique of symbolic variable elimination (SVE) to constrained optimization problems with bilinear forms. We apply our MILP reduction method to difficult DBLPs including XORs of linear constraints and show that we significantly outperform Gurobi. We also evaluate our method on a variety of synthetic instances to analyze the effects of DBLP problem size and sparsity w.r.t. MILP compilation size and solution efficiency.

Keywords: Bilinear programming · Symbolic variable elimination

1 Introduction

A disjointly constrained bilinear program (DBLP) is formally defined as follows

$$\begin{aligned} \min_{\mathbf{x},\mathbf{y}} \quad & f(\mathbf{x},\mathbf{y}) = \mathbf{c}^\top \mathbf{x} + \mathbf{x}^\top Q \mathbf{y} + \mathbf{d}^\top \mathbf{y} \\ \text{s.t.} \quad & \mathbf{a}_i^\top \mathbf{x} \le a_i \ \forall i \in I, \quad \mathbf{b}_j^\top \mathbf{y} \le b_j \ \forall j \in J \\ & x_k \ge 0 \ \forall k \in K; \ x_m \in \{0,1\} \ \forall m \in M \\ & y_l \ge 0 \ \forall l \in L; \ y_n \in \{0,1\} \ \forall n \in N, \end{aligned} \tag{1}$$

where I and J are the index sets of the linear constraints. K, L and M, N are those of continuous and binary variables, respectively. Let $n_x = |K| + |M|$ and $n_y = |L| + |N|$, then we have $Q \in \mathbb{R}^{n_x \times n_y}$, $\mathbf{c}, \mathbf{a}_i \in \mathbb{R}^{n_x}$, and $\mathbf{d}, \mathbf{b}_j \in \mathbb{R}^{n_y}$. The

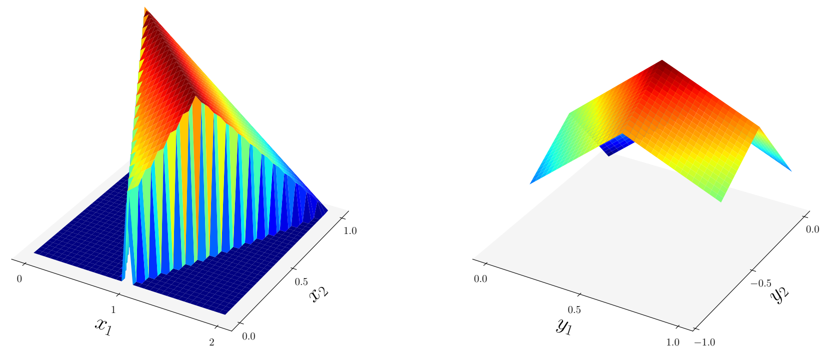


Fig. 1: The DBLP objective from Section 5 [14], evaluated on a range of values of $\mathbf{x}$ (left) and $\mathbf{y}$ (right). The piecewise linear structure hints at a MILP reduction.

disjointness property arises from the separation of linear constraints on $\mathbf{x}$ and $\mathbf{y}$. We define $\mathcal{X}$ and $\mathcal{Y}$ to be the feasible sets of $\mathbf{x}$ and $\mathbf{y}$ variables.

Historically, DBLPs have been used to formulate a variety of applications including uses in game theory, facility location, nonlinear multi-commodity network flows, dynamic assignment and production, risk management, and supply chain management [8–10, 13]. More recently, DBLPs have found applications in multi-agent planning problems [**?**], particularly when the transitions of different agents are assumed to be independent, which leads to disjoint constraints.

While Gurobi [4] can directly solve DBLPs to optimality since version 9.0 (based on spatial branching and a locally valid McCormick-based LP relaxation), it can only solve small DBLP instances when they use complex logical constraints (e.g., XORs of linear constraints). Given that logical constraints can be naturally encoded in a MILP, we conjecture (and later empirically show) that Gurobi can better solve such DBLPs when transformed to a MILP formulation.

Earlier work has shown that a DBLP is a concave minimization problem with a piecewise linear objective and linear constraints over one set of variables, say, $\mathbf{x}$ [5, 6]. To illustrate, Fig.1 shows a DBLP objective from Section 5 evaluated on a range of $\mathbf{x}$ and $\mathbf{y}$ values, where we clearly observe piecewise linear structure. Formally, consider $\min_{\mathbf{x},\mathbf{y}} f(\mathbf{x},\mathbf{y}) = \min_{\mathbf{x}} g(\mathbf{x})$ with

$$g(\mathbf{x}) := \min_{\mathbf{y}\in\mathcal{Y}} f(\mathbf{x},\mathbf{y}) = \min_{\mathbf{y}\in V(\mathrm{Conv}(\mathcal{Y}))} f(\mathbf{x},\mathbf{y}) = \mathbf{c}^\top\mathbf{x} + \min_{\mathbf{y}\in V(\mathrm{Conv}(\mathcal{Y}))} \big\{(\mathbf{d} + Q^\top\mathbf{x})^\top\mathbf{y}\big\},$$

where $V(\mathrm{Conv}(\mathcal{Y}))$ is the set of vertices of the convex hull of $\mathcal{Y}$. Theoretically, enumerating *all* vertices makes $g(\mathbf{x})$ piecewise linear and hence MILP-reducible, but a more compact and constructive MILP reduction has remained elusive.

In this work, we *extend symbolic variable elimination (SVE) [11] to bilinear expressions* and derive the *first* DBLP to MILP reduction that does not require enumeration of all vertices $V(\mathrm{Conv}(\mathcal{Y}))$. In addition to an investigation of the performance of our DBLP to MILP reduction on synthetic instances with varying size and sparsity, we demonstrate that *the Gurobi MILP solver applied to our DBLP reduction can outperform Gurobi's own bilinear solver for DBLPs.*

2 Reducing a DBLP to a MILP: A Worked Example

To foreshadow the general methodology that we explore in this paper, we first demonstrate how we can "deflate" a DBLP into a conditional DBLP by eliminating one variable from $\mathbf{y}$ at a time until the final result is a conditional LP, or a MILP. We proceed to show such deflation steps in close detail in Example 1.

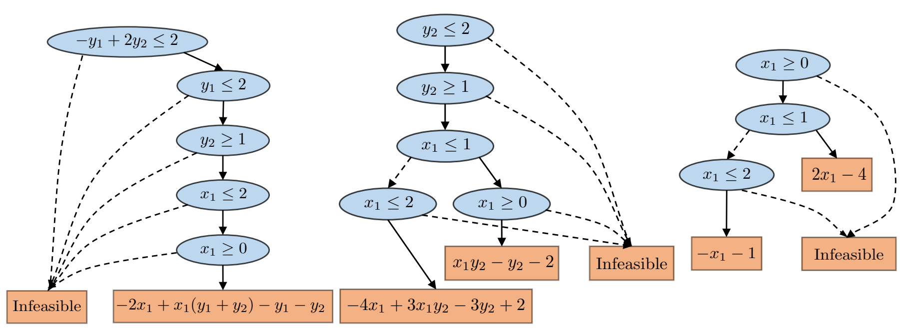
Example 1. Consider the following simple DBLP (Fig.2a):

$$\begin{aligned} \min_{x_1,y_1,y_2} \quad & -2x_1 + x_1(y_1 + y_2) - y_1 - y_2 \\ \text{s.t.} \quad & -y_1 + 2y_2 \le 2,\ y_1 \le 2,\ y_2 \ge 1,\ 0 \le x_1 \le 2 \end{aligned} \tag{2}$$

Our goal is to symbolically minimize out y_1 and y_2 so that we can obtain a reduced form over just x_1. To do this, we can view the $\min_{x_1,y_1,y_2}$ from the perspective of symbolic variable elimination [11] where we can "min-out" y_1 first. Observe that when y_1 is minimized, x_1 and y_2 are considered free variables, allowing us to treat the bilinear objective as linear in y_1. The minimum, therefore, must occur at a boundary value of y_1. We can easily obtain symbolic bounds on y_1 if we isolate it in the linear constraints. In this example, $-y_1 + 2y_2 \le 2$ and $y_1 \le 2$ are equivalent to $y_1^{lb} \le y_1 \le y_1^{ub}$ with $y_1^{ub} = 2$ and $y_1^{lb} = 2y_2 - 2$.

We now plug the two bounds on y_1 into the objective and compare resulting values. To that end, let $f^{ub}(x_1, y_2)$ and $f^{lb}(x_1, y_2)$ be the objective values when the upper and lower bound of y_1 is substituted in, respectively. That is,

$$\begin{aligned} f^{ub}(x_1, y_2) &= -2x_1 + x_1(2 + y_2) - 2 - y_2 = x_1 y_2 - y_2 - 2 \\ f^{lb}(x_1, y_2) &= -2x_1 + x_1[(2y_2 - 2) + y_2] - (2y_2 - 2) - y_2 = -4x_1 + 3x_1 y_2 - 3y_2 + 2 \end{aligned}$$



(a) DBLP in Example 1 (b) Conditional DBLP after eliminating y_1 (c) Final MILP

Fig. 2: Compact XADD [12] decision diagram representation of (2) in its (a) original form and after (b) y_1 and (c) y_2 are eliminated. Given values for x_1, y_1, and y_2, the XADD can be evaluated top-to-bottom. Oval constraints are decisions and the solid (dashed) edge is followed if the constraint evaluates to true (false). Leaf nodes provide the objective evaluation. In (c), once all $\mathbf{y}$ variables are symbolically eliminated, all constraints and leaves are linear leading to a conditional LP (=MILP).

In order to determine which bound on y_1 minimizes the objective, we can check if the difference $f^{ub}(x_1, y_2) - f^{lb}(x_1, y_2)$ is positive or negative:

$$f^{ub}(x_1, y_2) - f^{lb}(x_1, y_2) = (y_1^{ub} - y_1^{lb})(x_1 - 1) = (4 - 2y_2)(x_1 - 1) \tag{3}$$

Crucially in (3), the terms in the objective that do not have y_1 *always* cancel out, while the ones multiplied to y_1 remain. Hence, when we substitute in the boundary values of y_1 into the objective, the difference always has two factors: one *linear* factor of $\mathbf{x}$ and one *linear* factor of $\mathbf{y}$ (see the discussion in Section 4). If (3) is positive (or negative), $f^{lb}(x_1, y_2)$ is smaller (or greater) than $f^{ub}(x_1, y_2)$. Fortunately since $y_1^{ub} - y_1^{lb}$ should be nonnegative, we need only check if *linear* factor $(x_1 - 1)$ is negative (positive) to determine if the upper (lower) bound substitution is minimal. Then we can write a reduced *conditional* DBLP form (Fig.2b) with y_1 *eliminated*, *linear* conditions on $\mathbf{x}$, and a *bilinear* objective:

$$\begin{cases} (Case1)\ x_1 - 1 \le 0: & \min\limits_{x_1, y_2}\ f^{ub}(x_1, y_2) = x_1 y_2 - y_2 - 2 \\ & \text{s.t.}\quad 0 \le x_1 \le 1,\ 1 \le y_2 \le 2 \\ (Case2)\ x_1 - 1 > 0: & \min\limits_{x_1, y_2}\ f^{lb}(x_1, y_2) = -4x_1 + 3x_1 y_2 - 3y_2 + 2 \\ & \text{s.t.}\quad 1 < x_1 \le 2,\ 1 \le y_2 \le 2 \end{cases} \tag{4}$$

As a technical note, we need to symbolically guarantee $y_1^{ub} \ge y_1^{lb}$, which simplifies to $y_2 \le 2$ and is shown added to the above constraints.

Now that we've eliminated y_1, we can proceed to eliminate y_2. For *Case1*, we can minimize out y_2 in the same way as we've done for y_1. Firstly, the bounds are $y_2^{lb} = 1$ and $y_2^{ub} = 2$. By substituting these boundary values to $f^{ub}(x_1, y_2)$ and comparing the results, we get an LP $\min\limits_{0 \le x_1 \le 1} 2x_1 - 4$. Similarly, *Case2* gives us another LP, $\min\limits_{1 < x_1 \le 2} -x_1 - 1$. Fig.2c exemplifies the compact representation of this conditional LP. We can replace the case conditions with binary variables, reducing the overall problem of (2) to an optimization problem with a piecewise linear objective and linear constraints, which can be expressed as a MILP.

Example 1 illustrates that we can obtain a concrete MILP model by symbolically minimizing out one set of variables from a DBLP (e.g., $\mathbf{y}$) yielding a reduced MILP optimization problem over $\mathbf{x}$, which can be easily implemented and efficiently solved by off-the-shelf MILP solvers such as Gurobi. Substituting the optimal $\mathbf{x}$ in the original DBLP reduces to a MILP over $\mathbf{y}$ that is easily solved to obtain the corresponding $\mathbf{y}$. To move beyond this example and provide a fully automated reduction of an arbitrary DBLP to a MILP, we will need a general symbolic procedure to automate this reasoning, which we provide next.

3 Symbolic Calculus with Case Representation

Now that we have worked through a specific example, we proceed to show how the generic procedure for converting a DBLP to a MILP can be achieved through the symbolic case representation and case calculus [3, 12] (this section) with a novel extension to support symbolic variable elimination (SVE) [11] for continuous minimization operations with bilinear forms (Section 4). Subsequently in Section 5, we present empirical analysis.

3.1 Case Representation

We assume that all symbolic functions can be represented in *case form* [3, 12]:

$$f = \begin{cases} \phi_1 : & f_1 \\ \vdots & \vdots \\ \phi_k : & f_k \end{cases} \tag{5}$$

Here, ϕ_i (a *partition*) are logical formulae, which can include arbitrary logical $(\wedge, \vee, \neg)$ combinations of linear inequalities $(\geq, >, \leq, <)$. We assume that the set of conditions $\{\phi_1, \ldots, \phi_k\}$ disjointly and exhaustively partition the domain of the variables such that f is well-defined. We call ϕ_i "disjointly linear" if it consists only of either $\mathbf{x}$ or $\mathbf{y}$. We restrict f_i (a *function value*) to be linear or bilinear in $\mathbf{x}$ and $\mathbf{y}$. Further, we restrict ϕ_i to be disjointly linear if f has bilinear f_i. These restrictions are in place such that we can represent an arbitrary DBLP in case form in Section 4.

Henceforth, we refer to functions with linear ϕ_i and f_i as linear piecewise linear (LPWL). Functions with disjointly linear ϕ_i and bilinear f_i are dubbed as disjointly linear piecewise bilinear (LPWB). Later, we discuss that in order for SVE of a DBLP to remain closed-form, it is critical that the procedural reduction of the original case function always produces an LPWB or LPWL function.

We remark that the DBLP in Example 1 can be easily rewritten in case form

$$f = \begin{cases} [-y_1 + 2y_2 \leq 2] \wedge [y_1 \leq 2] \wedge [y_2 \geq 1] \wedge [0 \leq x_1 \leq 2] : & -2x_1 + x_1(y_1 + y_2) - y_1 - y_2 \\ \text{otherwise} : & \infty \end{cases}$$

where any finite value for f satisfying the first case (the feasible set) will always be chosen over ∞ in the other partition (infeasibility), since we want $\min_{x_1, y_1, y_2} f$.

3.2 Basic Case Operators

One of the most simple case operations on f in (5) is a *unary operation* such as scalar multiplication $c \cdot f$ $(c \in \mathbb{R})$ or negation $-f$. This operation is simply applied to the function value f_i for every partition ϕ_i. We can also define *binary operations* between two case functions by taking the cross-product of the logical partitions from the two case statements and performing the operation on the resulting paired partitions.[4] For example, the "cross-sum" $\oplus$ of two cases is:

$$\begin{cases} \phi_1 : & f_1 \\ \phi_2 : & f_2 \end{cases} \oplus \begin{cases} \psi_1 : & g_1 \\ \psi_2 : & g_2 \end{cases} = \begin{cases} \phi_1 \wedge \psi_1 : & f_1 + g_1 \\ \phi_1 \wedge \psi_2 : & f_1 + g_2 \\ \phi_2 \wedge \psi_1 : & f_2 + g_1 \\ \phi_2 \wedge \psi_2 : & f_2 + g_2 \end{cases}$$

[4] Only the case operations that we actually use for SVE of a DBLP are introduced.

Likewise, we perform $\ominus$ by subtracting function values per each pair of partitions. Observe that LPWL and LPWB functions are closed under $\oplus$ and $\ominus$.

Next, we define symbolic *case min(max)* between two case functions as:

$$\text{casemin}\left(\begin{cases} \phi_1 : & f_1 \\ \phi_2 : & f_2 \end{cases}, \begin{cases} \psi_1 : & g_1 \\ \psi_2 : & g_2 \end{cases} \right) = \begin{cases} \phi_1 \wedge \psi_1 \wedge \boldsymbol{f_1 > g_1} : & g_1 \\ \phi_1 \wedge \psi_1 \wedge \boldsymbol{f_1 \leq g_1} : & f_1 \\ \phi_1 \wedge \psi_2 \wedge \boldsymbol{f_1 > g_2} : & g_2 \\ \phi_1 \wedge \psi_2 \wedge \boldsymbol{f_1 \leq g_2} : & f_1 \\ \vdots & \vdots \end{cases} \tag{6}$$

wherein the resulting partitions also include the comparison of associated function values f_i and g_j to determine $\min(f_i, g_j)$ (highlighted in bold). casemin of more than two case functions is straightforward since the operator is associative. Crucially, LPWL functions are closed under casemin (max), but LPWB functions are not because $f_i \leq g_j$ can be bilinear or jointly linear.

Another important symbolic operation is *symbolic substitution*. This operation takes a set σ of variables and their substitutions, e.g., $\sigma = \{y/(x_1 + x_2), z/(x_1 - x_2)\}$ where the LHS of '/' represents the substitution variable and the RHS of '/' is the expression being substituted in. Then, we write the substitution operation on f_i with σ as $f_i\sigma$. Then the operation follows:

$$f = \begin{cases} \phi_1 : & f_1 \\ \vdots & \vdots \\ \phi_k : & f_k \end{cases}, \quad f\sigma = \begin{cases} \phi_1\sigma : & f_1\sigma \\ \vdots & \vdots \\ \phi_k\sigma : & f_k\sigma \end{cases} \tag{7}$$

In this paper, we will only substitute linear expressions of $\{y_j\}_{j \neq i}$ variables into y_i, which clearly preserves the LPWL and LPWB properties.

In the next section, we show that the procedural reduction of a DBLP to a MILP only involves the application of the case operations that preserve an LPWB form, which eventually reduces to an LPWL form (equivalent to a MILP).

4 Symbolic Reduction of a DBLP to a MILP

Having introduced the case form and its basic operations in Section 3, we first note that the DBLP in (1) can be written in case form. That is, (1) is equivalent to $\min_{\mathbf{x},\mathbf{y}} \; f_{DBLP}(\mathbf{x}, \mathbf{y})$ where

$$f_{DBLP}(\mathbf{x}, \mathbf{y}) = \begin{cases} \phi(\mathbf{x}) \wedge \psi(\mathbf{y}) : & \mathbf{c}^\top \mathbf{x} + \mathbf{x}^\top Q \mathbf{y} + \mathbf{d}^\top \mathbf{y} \\ \neg(\phi(\mathbf{x}) \wedge \psi(\mathbf{y})) : & \infty \end{cases} \tag{8}$$

with $\phi(\mathbf{x}) := [\mathbf{x} \in \mathcal{X}]$, $\psi(\mathbf{y}) := [\mathbf{y} \in \mathcal{Y}]$. Note how the feasible set of the DBLP is encoded as a partition and the objective as its function value. Also, observe that $\phi(\mathbf{x}) \wedge \psi(\mathbf{y})$ is disjointly linear, so $f_{DBLP}(\mathbf{x}, \mathbf{y})$ is an LPWB function.

We have seen in Example 1 that we get a MILP out of a DBLP via symbolic minimization of $\mathbf{y}$ variables. In general, if the result of SVE of $\mathbf{y}$ from an arbitrary LPWB function can be shown to be equivalent to an LPWL function, we effectively reduce a DBLP to a MILP. However, existing symbolic min operators [16] fall short of dealing with LPWB functions, since none of them can handle bilinear function values. In the sequel, we show that we can always factorize the bilinear expressions appearing during the SVE of $\mathbf{y}$ variables into one factor in $\mathbf{x}$ and the other in $\mathbf{y}$. This in turn makes LPWB functions closed under the SVE operations. With this, we prove that a DBLP can be reduced to a MILP.

4.1 Symbolic Minimization of Linear Piecewise Linear Functions

To see why existing approaches fail to symbolically optimize variables in closed-form when it comes to LPWB functions, we first consider the symbolic min operator for LPWL functions [16].[5] This operator differs from casemin in that the former optimizes a symbolic function w.r.t. decision variables, whereas the latter compares multiple symbolic functions as in (6). Example 2 illustrates the application of the symbolic min operator to an LPWL function.

Example 2. Let $f(x_1, x_2)$ be a symbolic function of $x_1, x_2 \in [0, 10]^2$ as below:

$$f(x_1, x_2) = \begin{cases} x_1 + x_2 \geq 1 : & 3x_1 + 2x_2 \\ x_1 + x_2 < 1 : & -3x_1 + x_2 \end{cases} \tag{9}$$

As in Example 1, we can view the $\min_{x_1,x_2}$ from the perspective of symbolic variable elimination, and we write it as $\min_{x_2} \min_{x_1} f(x_1, x_2)$. When x_1 is being minimized out, we can treat x_2 as a symbolic free variable. Then,

$$\min_{x_2} \min_{x_1} f(x_1, x_2) = \min_{x_2} \left[\min_{x_1} \begin{cases} \phi_1(x_1, x_2) : & f_1(x_1, x_2) \\ \phi_2(x_1, x_2) : & f_2(x_1, x_2) \end{cases} \right]$$

$$= \min_{x_2} \left[\min_{x_1} \operatorname*{casemin}_{i=\{1,2\}} \begin{cases} \phi_i(x_1, x_2) : & f_i(x_1, x_2) \\ \neg\phi_i(x_1, x_2) : & \infty \end{cases} \right] \tag{10}$$

$$= \min_{x_2} \left[\operatorname*{casemin}_{i=\{1,2\}} \min_{x_1} \begin{cases} \phi_i(x_1, x_2) : & f_i(x_1, x_2) \\ \neg\phi_i(x_1, x_2) : & \infty \end{cases} \right] \tag{11}$$

where ϕ_i and f_i are defined as per (9). (10) follows since partitions are disjoint. The commutative property gives (11). As a result, $\min_{x_1} f(x_1, x_2)$ is equivalent to minimizing out x_1 from "$\{\phi_i : \ f_i$" for all i, followed by casemin of the results.

Now in order to compute $\min_{x_1} \left\{ \phi_i(x_1, x_2) : \ f_i(x_1, x_2) \right.$, we make three important observations: (a) a partition ϕ_i and domain bounds on x_1 prescribe the lower and upper bounds over the variable, $x_1^{lb,i}$ and $x_1^{ub,i}$ respectively; (b) since f_i is linear in x_1, either $x_1^{lb,i}$ or $x_1^{ub,i}$ will evaluate to the minimum (ties broken arbitrarily); and (c) if there is a subset of conditionals in ϕ_i that are independent of x_1, denoted as $\phi_i^{\perp x_1}$, it should still be satisfied after the min operation.

[5] This operator has been introduced firstly in [16] and later in more detail in [7]. However, we include the result here for completeness and to better illustrate our extension to handling bilinear function values in Section 4.2.

For example, from $\phi_1(x_1, x_2) = [x_1 + x_2 \geq 1]$,

$$x_1^{lb,1} = \text{casemax}(1 - x_2, 0) = \begin{cases} x_2 \geq 1 : & 0 \\ x_2 < 1 : & 1 - x_2 \end{cases} \tag{12}$$

In general, a domain bound (e.g., $x_1 \geq 0$) and each conditional (e.g., $[x_1 + x_2 \geq 1]$) of a partition can contribute at most one lower bound *candidate*, and $x_1^{lb,i}$ is the casemax among the candidates. Similarly, we get $x_1^{ub,i}$ as the casemin among candidates, which in this case is simply $x_1^{ub,1} = 10$. From these bounds, we additionally impose a set of constraints such that $x_1^{lb,i} \leq x_1^{ub,i}$ is ensured at all times, which are added to $\phi_i^{\perp\!\!\!\perp x_1}$. In this example, these are $[0 \leq 10]$ and $[1 - x_2 \leq 10]$, which trivially hold true, and so we set $\phi_1^{\perp\!\!\!\perp x_1} = true$.

With these bounds, it remains to determine the minimum value by substituting $x_1^{lb,i}$ and $x_1^{ub,i}$ into x_1 in f_1 and performing casemin. For $i = 1$, we have:[6]

$$\begin{aligned} \min_{x_1} \begin{cases} \phi_1(x_1, x_2) : & f_1(x_1, x_2) \\ \neg\phi_1(x_1, x_2) : & \infty \end{cases} &= \text{casemin}(f_1\sigma_1^{ub}, f_1\sigma_1^{lb}) \oplus \begin{cases} \phi_1^{\perp\!\!\!\perp x_1} : & 0 \\ \neg\phi_1^{\perp\!\!\!\perp x_1} : & \infty \end{cases} \\ &= \text{casemin}\left(30 + 2x_2, \begin{cases} x_2 \geq 1 : & 2x_2 \\ x_2 < 1 : & 3 - x_2 \end{cases}\right) \\ &= \begin{cases} x_2 \geq 1 : & 2x_2 \\ x_2 < 1 : & 3 - x_2 \end{cases} \end{aligned} \tag{13}$$

where $\sigma_1^{lb} = \{x_1/x_1^{lb,1}\}$ and $\sigma_1^{ub} = \{x_1/x_1^{ub,1}\}$.

If we follow the same procedure for $\phi_2(x_1, x_2)$ and $f_2(x_1, x_2)$, we get below:

$$\min_{x_1} \begin{cases} \phi_2(x_1, x_2) : & f_2(x_1, x_2) \\ \neg\phi_2(x_1, x_2) : & \infty \end{cases} = \begin{cases} x_2 \geq 1 : & x_2 \\ x_2 < 1 : & -3 + 4x_2 \end{cases} \tag{14}$$

Finally, we take casemin of (13) and (14), which becomes

$$g(x_2) := \min_{x_1} f(x_1, x_2) = \begin{cases} x_2 \geq 1 : & x_2 \\ x_2 < 1 : & -3 + 4x_2 \end{cases} \tag{15}$$

Note that x_1 has been *eliminated* from $f(x_1, x_2)$ in (15). The same procedure can be repeated for the elimination of x_2.

4.2 Symbolic Minimization of Disjointly Linear Piecewise Bilinear Functions

Example 2 highlights the key operations entailed in symbolic minimization of an LPWL function. However for DBLPs, the step in (13) would compare *bilinear* expressions, leading to a case function with bilinear or jointly linear partitions, preventing naively applying the same symbolic manipulations. Despite these bilinear expressions, Proposition 1 affirms that we can still perform SVE of one

[6] Note the way we enforce $\phi_1^{\perp\!\!\!\perp x_1}$ by the cross-sum operation.

set of variables from the DBLP, which eventually gives rise to an LPWL function. This in turn can be modeled as a MILP by introducing binary indicator variables.

Firstly, we formally define an LPWB function $f(\mathbf{x}, \mathbf{y})$ for $\mathbf{x} \in \mathbb{R}^{n_x}, \mathbf{y} \in \mathbb{R}^{n_y}$:

$$f(\mathbf{x}, \mathbf{y}) = \begin{cases} \phi_1(\mathbf{x}) \wedge \psi_1(\mathbf{y}) : & f_1(\mathbf{x}, \mathbf{y}) = \mathbf{c}_1^\top \mathbf{x} + \mathbf{x}^\top Q_1 \mathbf{y} + \mathbf{d}_1^\top \mathbf{y} \\ \vdots & \vdots \\ \phi_n(\mathbf{x}) \wedge \psi_n(\mathbf{y}) : & f_n(\mathbf{x}, \mathbf{y}) = \mathbf{c}_n^\top \mathbf{x} + \mathbf{x}^\top Q_n \mathbf{y} + \mathbf{d}_n^\top \mathbf{y} \end{cases} \tag{16}$$

where $\mathbf{c}_i \in \mathbb{R}^{n_x}$, $\mathbf{d}_i \in \mathbb{R}^{n_y}$, $Q_i \in \mathbb{R}^{n_x \times n_y}$, and $\phi_i(\mathbf{x})$ and $\psi_i(\mathbf{y})$ are conjunction of linear inequalities in $\mathbf{x}$ and $\mathbf{y}$.[7] Note that f_{DBLP} is a special case of (16). Proposition 1 establishes that LPWB functions are closed under symbolic min and eventually become LPWL, which uses the following result from Lemma 1.

Lemma 1. *Consider the symbolic substitution operations into bilinear* $f_i(\mathbf{x}, \mathbf{y})$ *with* $\sigma^j = \{y_1 / l^j(\mathbf{y}_{2:n_y})\}$*, where* $\mathbf{y}_{2:n_y} = \{y_2, \ldots, y_{n_y}\}$*,* $l^{ub}(\mathbf{y}_{2:n_y})$ *and* $l^{lb}(\mathbf{y}_{2:n_y})$ *are linear. Then,* $\text{casemin}(f_i(\mathbf{x}, \mathbf{y})\sigma^{ub}, f_i(\mathbf{x}, \mathbf{y})\sigma^{lb})$ *is an LPWB function.*

Proof. Define $h : \mathbb{R}^{n_x \times (n_y - 1)} \mapsto \mathbb{R}$ as $h(\mathbf{x}, \mathbf{y}_{2:n_y}) := f_i(\mathbf{x}, \mathbf{y})\sigma^{ub} - f_i(\mathbf{x}, \mathbf{y})\sigma^{lb}$. If $h \geq 0$, we select $f_i(\mathbf{x}, \mathbf{y})\sigma^{lb}$ as the casemin; otherwise, $f_i(\mathbf{x}, \mathbf{y})\sigma^{ub}$ is selected. In other words, we get a case function with bilinear partitions and bilinear values:

$$\text{casemin}(f_i(\mathbf{x}, \mathbf{y})\sigma^{ub}, f_i(\mathbf{x}, \mathbf{y})\sigma^{lb}) = \begin{cases} h(\mathbf{x}, \mathbf{y}_{2:n_y}) \geq 0 : & f_i(\mathbf{x}, \mathbf{y})\sigma^{lb} \\ h(\mathbf{x}, \mathbf{y}_{2:n_y}) < 0 : & f_i(\mathbf{x}, \mathbf{y})\sigma^{ub} \end{cases} \tag{17}$$

However, $h(\mathbf{x}, \mathbf{y}_{2:n_y})$ can always be factorized into two factors where each factor is linear in *either* $\mathbf{x}$ or $\mathbf{y}_{2:n_y}$. That is,

$$h(\mathbf{x}, \mathbf{y}_{2:n_y}) = \Big(l^{ub}(\mathbf{y}_{2:n_y}) - l^{lb}(\mathbf{y}_{2:n_y})\Big)\Big[[\mathbf{d}_i]_1 + \sum_{r=1}^{n_x} x_r [Q_i]_{r,1}\Big] \geq 0 \tag{18}$$

since the terms in $f_i(\mathbf{x}, \mathbf{y})$ that do not include y_1 cancel out. Finally, we get

$$\begin{cases} [l^{ub}(\mathbf{y}_{2:n_y}) - l^{lb}(\mathbf{y}_{2:n_y}) \geq 0] \wedge [[\mathbf{d}_i]_1 + \sum_{r=1}^{n_x} x_r [Q_i]_{r,1} \geq 0] : & f_i(\mathbf{x}, \mathbf{y})\sigma^{lb} \\ [l^{ub}(\mathbf{y}_{2:n_y}) - l^{lb}(\mathbf{y}_{2:n_y}) < 0] \wedge [[\mathbf{d}_i]_1 + \sum_{r=1}^{n_x} x_r [Q_i]_{r,1} < 0] : & f_i(\mathbf{x}, \mathbf{y})\sigma^{lb} \\ [l^{ub}(\mathbf{y}_{2:n_y}) - l^{lb}(\mathbf{y}_{2:n_y}) \geq 0] \wedge [[\mathbf{d}_i]_1 + \sum_{r=1}^{n_x} x_r [Q_i]_{r,1} < 0] : & f_i(\mathbf{x}, \mathbf{y})\sigma^{ub} \\ [l^{ub}(\mathbf{y}_{2:n_y}) - l^{lb}(\mathbf{y}_{2:n_y}) < 0] \wedge [[\mathbf{d}_i]_1 + \sum_{r=1}^{n_x} x_r [Q_i]_{r,1} \geq 0] : & f_i(\mathbf{x}, \mathbf{y})\sigma^{ub} \end{cases} \tag{19}$$

which has disjointly linear partitions and bilinear values, hence an LPWB.

Now, we present the main result in Proposition 1.

Proposition 1 (Symbolic minimization of LPWB functions). *Let* $g(\mathbf{x})$ *denote the result of symbolic minimization of* $f(\mathbf{x}, \mathbf{y})$ *over* $\mathbf{y}$ *variables, which we assume to be well-defined. That is,*

$$g(\mathbf{x}) := \min_{\mathbf{y}} f(\mathbf{x}, \mathbf{y}) \tag{20}$$

Then, it follows that $g(\mathbf{x})$ *is an LPWL function of* $\mathbf{x}$*.*

[7] A function value can be ∞, which implies that the corresponding partition is infeasible (see Fig.2c).

Proof. The proof relies on inductive reasoning as we show how each y_i can be eliminated in turn yielding an LPWB closed-form and ultimately a final LPWL form once all $\mathbf{y}$ have been eliminated.

Firstly, similar to (11), we note $\min_{\mathbf{y}} f(\mathbf{x}, \mathbf{y})$ is equivalent to the following:

$$\min_{y_{n_y},\dots,y_2} \left[\operatorname*{casemin}_{i=\{1,\dots,n\}} \min_{y_1} \begin{cases} \phi_i(\mathbf{x}) \wedge \psi_i(\mathbf{y}) : & f_i(\mathbf{x}, \mathbf{y}) \\ \neg\phi_i(\mathbf{x}) \vee \neg\psi_i(\mathbf{y}) : & \infty \end{cases} \right] \tag{21}$$

For the ith partition, $\psi_i(\mathbf{y})$ and the generic domain bounds over y_1 specify the upper and lower bounds of y_1, denoted as $y_1^{ub,i}$ and $y_1^{lb,i}$, respectively. Notice that $y_1^{ub,i}$ and $y_1^{lb,i}$ are LPWL functions of $\mathbf{y}_{2:n_y}$. We now substitute the bounds in the place of y_1, followed by casemin to determine a smaller value, which gives:

$$\begin{aligned} g_i(\mathbf{x}, \mathbf{y}_{2:n_y}) := & \min_{y_1} \begin{cases} \phi_i(\mathbf{x}) \wedge \psi_i(\mathbf{y}) : & f_i(\mathbf{x}, \mathbf{y}) \\ \neg\phi_i(\mathbf{x}) \vee \neg\psi_i(\mathbf{y}) : & \infty \end{cases} \\ = & \operatorname{casemin}\Big(f_i(\mathbf{x}, \mathbf{y})\sigma_i^{ub}, f_i(\mathbf{x}, \mathbf{y})\sigma_i^{lb} \Big) \oplus \begin{cases} \phi_i(\mathbf{x}) \wedge \psi_i^{\perp y_1}(\mathbf{y}_{2:n_y}) : 0 \\ \neg\big(\phi_i(\mathbf{x}) \wedge \psi_i^{\perp y_1}(\mathbf{y}_{2:n_y})\big) : & \infty \end{cases} \end{aligned} \tag{22}$$

where $\sigma_i^{ub} = \{y_1/y_1^{ub,i}\}$ and $\sigma_i^{lb} = \{y_1/y_1^{lb,i}\}$.

The second term in (22) ensures that the conditionals *independent* of y_1 in $[\phi_i(\mathbf{x}) \wedge \psi_i(\mathbf{y})]$ hold true, which are not accounted for in $y_1^{ub,i}$ and $y_1^{lb,i}$. $\psi_i^{\perp y_1}(\mathbf{y}_{2:n_y})$ also includes a set of conditionals that require $y_1^{ub,i} \geq y_1^{lb,i}$ for all pairs of function values. Naturally, we use ∞ as the value of an infeasible partition such that it will be ignored in later steps since we are minimizing.

Now, we have that $g_i(\mathbf{x}, \mathbf{y}_{2:n_y})$ is an LPWB function. To see this, denote the casemin in (22) as $m(\mathbf{x}, \mathbf{y}_{2:n_y})$. A partition of $m(\mathbf{x}, \mathbf{y}_{2:n_y})$ is conjunction of a partition from $y_1^{ub,i}$, say the jth, and another from $y_1^{lb,i}$, say the kth; the corresponding function value is casemin$(f_i\{y_1/l_j^{ub}(\mathbf{y}_{2:n_y})\},\ f_i\{y_1/l_k^{lb}(\mathbf{y}_{2:n_y})\})$, with l_j^{ub} and l_k^{lb} denoting the function values from $y_1^{ub,i}$ and $y_1^{lb,i}$, respectively. Then for this partition, we clearly see we get disjointly linear partitions and bilinear function values as per Lemma 1. This analysis can be extended to all other partitions and function values of $m(\mathbf{x}, \mathbf{y}_{2:n_y})$, and hence $g_i(\mathbf{x}, \mathbf{y}_{2:n_y})$ is LPWB $\forall i$.

Finally, we note (21) becomes

$$\begin{aligned} \min_{\mathbf{y}} f(\mathbf{x}, \mathbf{y}) &= \min_{y_{n_y},\dots,y_2} \left[\operatorname*{casemin}_{i=\{1,\dots,n\}} g_i(\mathbf{x}, \mathbf{y}_{2:n_y}) \right] \\ &= \operatorname*{casemin}_{i=\{1,\dots,n\}} \left[\min_{y_{n_y},\dots,y_3} \left(\min_{y_2} g_i(\mathbf{x}, \mathbf{y}_{2:n_y}) \right) \right] \end{aligned} \tag{23}$$

where (23) follows since min and casemin are commutative. Then, we see that the inner-most minimization is essentially SVE of y_2 of an LPWB function. Hence, we can repeat the elimination procedure until all $\mathbf{y}$ variables are minimized out, at which point we get a sequence of casemin applied to an LPWL function of $\mathbf{x}$. Since an LPWL function is closed under the casemin operator, we will get an LPWL function, $g(\mathbf{x})$ in closed-form.

Corollary 1. *The DBLP in* (1) *is equivalent to a MILP.*

Proof. The DBLP can be represented in case form as in (8), which is an LPWB function. Hence, $\min_{\mathbf{x},\mathbf{y}} f_{DBLP}(\mathbf{x},\mathbf{y})$ can be represented as $\min_{\mathbf{x}} g_{DBLP}(\mathbf{x})$ where $g_{DBLP}(\mathbf{x}) := \min_{\mathbf{y}} f_{DBLP}(\mathbf{x},\mathbf{y})$ is an LPWL function (Proposition 1). Therefore, the DBLP is equivalent to the minimization problem with piecewise linear objective and linear constraints, which is equivalent to a MILP.

We remark that maintaining a case representation of a DBLP or its LPWL equivalent with explicit partitions can be prohibitively expensive. Hence, in practice we use Extended Algebraic Decision Diagrams (XADDs) [12] (example in Fig.2) to compactly represent the case statement and perform operations.

5 Empirical Analysis

In this section, we evaluate the proposed novel reduction of a DBLP to a MILP on various test problems. First, we present the problem constrained with XORs of linear constraints in which the proposed approach outperformed Gurobi (9.5.0). Then, we explore empirical characteristics of the MILP reduction on general DBLPs using a set of randomized test instances. Specifically, we analyze the effects of the problem size and sparsity on the MILP reduction and its solution efficiency. We use the XADD for practical implementation of case functions, and we ported the original XADD implementation in Java to our own in Python. Generated MILPs are then solved using Gurobi. All experiments were done on a Linux machine with a 2.90GHz processor.[8]

Problems with XOR Conditional Constraints Consider the following DBLP involving XOR ($\veebar$) combinations of constraints as motivated by [15]:

$$
\begin{aligned}
\min \quad & \mathbf{c}^\top \mathbf{r} + \mathbf{r}^\top Q\mathbf{y} + \mathbf{d}^\top \mathbf{y} + c_z z, \quad \text{where} \qquad (24)\\
r_i = \quad & \begin{cases}
\big[[x_{3i-2} \geq x_{3i-1}] \veebar [x_{3i-1} \geq x_{3i}]\big] \wedge [z \geq 0]: & \max(x_{3i-1}, x_{3i}) - \min(x_{3i-1}, x_{3i})\\
\big[[x_{3i-2} \geq x_{3i-1}] \veebar [x_{3i-1} \geq x_{3i}]\big] \wedge [z \leq 0]: & \min(x_{3i-1}, x_{3i}) - \max(x_{3i-1}, x_{3i})\\
\neg\big[[x_{3i-2} \geq x_{3i-1}] \veebar [x_{3i-1} \geq x_{3i}]\big] \wedge [z \geq 0]: & \min(x_{3i-2}, x_{3i-1}) - \max(x_{3i-2}, x_{3i-1})\\
\neg\big[[x_{3i-2} \geq x_{3i-1}] \veebar [x_{3i-1} \geq x_{3i}]\big] \wedge [z \leq 0]: & \max(x_{3i-2}, x_{3i-1}) - \min(x_{3i-2}, x_{3i-1})
\end{cases}\\
\text{s.t.} \quad & \mathbf{b}_j^\top \mathbf{y} \leq b_j, \quad \forall j = 1, \ldots, 15\\
& x_i \in [-10, 10], \; r_j \in [-20, 20], \; y_k \in [-10, 10]\\
& i = 1, \ldots, 3n, \quad j = 1, \ldots, n, \quad k = 1, \ldots, 15.
\end{aligned}
$$

Here, $\mathbf{c}, \mathbf{r} \in \mathbb{R}^n$, $\mathbf{x} \in \mathbb{R}^{3n}$, $c_z, z \in \mathbb{R}$ and $\mathbf{b}_j, \mathbf{y} \in \mathbb{R}^{15}$. Observe that r_i in the objective is determined based on an XOR conditional expression involving $x_{3i-2}, x_{3i-1}, x_{3i}$ and a linear constraint of z $\forall i = 1, \ldots, n$. The feasible region over $\mathbf{y}$ is independently constructed by randomly generating $\mathbf{b}_j$ and b_j $\forall j$, and we also randomly generate the coefficients $(c_z, \mathbf{c}, \mathbf{d}, Q)$ in the objective. We eliminate $\mathbf{x}$ from (24) and solve the resulting MILP using Gurobi for the remaining variables.

[8] SVE runs on a single processor, but Gurobi made use of all 16 available cores.

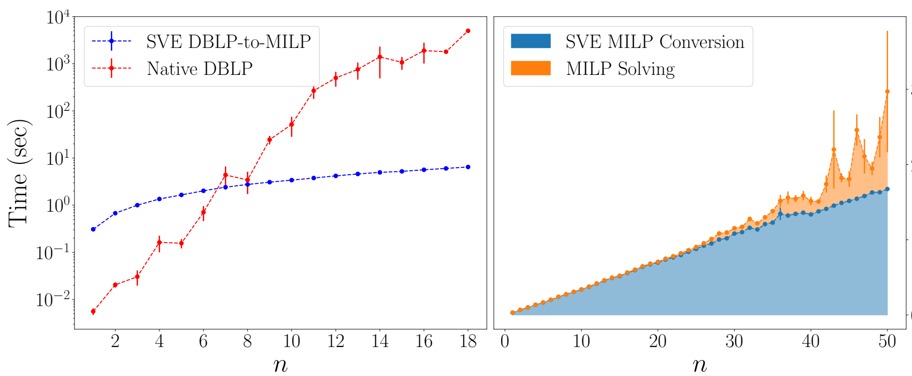


Fig. 3: **Left**: Runtime comparison of the Native DBLP form (using Gurobi's bilinear solver) and the SVE DBLP-to-MILP conversion (using Gurobi's MILP solver) vs. n (number of variables in XOR problem). Unlike Native DBLP whose time complexity appears exponential in n, SVE DBLP-to-MILP appears linear in n (nb. logarithmic y-axis). **Right**: Breakdown of total runtime of the SVE DBLP-to-MILP solution separated into SVE Conversion time and Gurobi MILP solve time. While SVE scales linearly in n, the MILP step takes a larger fraction of time as n increases (nb. linear y-axis and extended range of n on the x-axis, which only SVE DBLP-to-MILP can solve).

Note that this problem structure is particularly advantageous for the symbolic framework since each r_i can be compactly represented in XADD with only a small number of decision variables and the XOR constraints are sparse. In Fig.3, we compare the runtime performance of our approach against that of Gurobi. For each n, we generated 5 instances with different random seeds and plot the mean and its standard error. As the runtime grows *exponentially* for Gurobi, it quickly becomes impossible to solve problems with $n \geq 15$ ($n_x \geq 45$) within the given time limit of 5000 seconds. However, the solution time increases linearly in the number of variables for the symbolic approach, and we solve the problem with $n_x = 150$ within 30 seconds. In other words, we have effectively reformulated a DBLP that Gurobi cannot practically solve in its native form to the one that Gurobi can solve as a MILP!

Randomized Test Problems with Different Sizes and Sparsity Now, we scrutinize the proposed approach on some general DBLP test problems. For the first set of experiments, we follow [14] for systematic generation of test problems with certain properties. In particular, they suggested a two-step method in which smaller DBLP problems are first constructed, which are then additively combined. Furthermore, the underlying structure of the problem is then concealed by random transformations on the decision variables using Householder matrices [1]. 5 instances with different random transformation matrices are constructed for each configuration (n_x, n_y) and we report the average and standard error.

In Table.1, we evaluate the impact of how balanced a problem is on computational complexity by fixing the total number of variables while altering (n_x, n_y)

Table 1: Time and space complexity for balanced and imbalanced problems. For every fixed number of total variables $(12, 16, 20, 24)$, the results for an imbalanced $(n_x > n_y)$ and a balanced $(n_x = n_y)$ are reported. Observe that imbalanced problems are easier to solve and more compact to encode than their balanced counterparts.

$n_x + n_y$	n_x	n_y	Time (Symbolic)	Time (MILP)	# XADD Nodes	# Cont var (MILP)	# Bin var (MILP)	# Constr (MILP)
12	8	4	4.35 ± 0.01	0.01 ± 0.00	44	35	16	55
	6	6	16.35 ± 0.17	0.04 ± 0.00	76	54	18	75
16	10	6	44.67 ± 0.17	0.04 ± 0.00	114	75	23	103
	8	8	121.45 ± 1.09	0.88 ± 0.02	214	140	24	168
20	12	8	391.87 ± 3.22	0.71 ± 0.01	318	185	30	221
	10	10	959.65 ± 66.15	28.84 ± 2.81	622	388	30	423
24	16	8	313.38 ± 1.65	0.18 ± 0.00	536	295	32	335
	12	12	5886.00 ± 142.75	356.23 ± 11.17	1840	1122	36	1164

such that one instance has $n_x = n_y$ whereas $n_x > n_y$ for the other ($\mathbf{y}$ is eliminated). We have compared four sets of problem instances with varying total numbers of variables, i.e., $12, 16, 20, 24$. For each total number of variables, balanced and imbalanced instances are compared. We can see that it is in general much easier to solve imbalanced problems, which turn out to be more compact to encode as well. As the number of total variables increases, we observe that the discrepancy in the complexity between an imbalanced and its balanced counterpart widens.

Notably, the number of binary variables only rises at a moderate rate, whereas the numbers of continuous variables and constraints increase along with the size of the MILP reduction. This suggests that the case representation of the MILP equivalent of a given DBLP turns out to have a structure similar to a tree. For this type of problem, the computational gain attributed to using XADD can rather be small, and therefore we observe fast increases in complexity with the problem size. On the other hand, for types of problems we present in (24) and Fig.4, the SVE step can be efficiently done even for larger problems. Finally, note also that regardless of n_y, the running times for the optimal MILP solution remain very small.

In order to better understand the solution efficiency with regard to the number of variables and the sparsity of the problem, we created other sets of random test problems. Concretely, the goals are to examine (a) whether the increase in the number of symbolically eliminated variables has greater impact than the increase in the total number of variables in solution efficiency and (b) the effects of the sparsity of coefficients $(\mathbf{a}_i, \mathbf{b}_j, Q)$. For these problems, we generate feasible and bounded problems with 30 constraints $(n_a = n_b = 15)$. 5 instances generated with different random seeds are used per each experiment configuration, and we plot the average and its standard error.

For (a), we symbolically eliminate $\mathbf{y}$ and compare two sets: one with $n_x = 8$ and n_y from 4 to 9, and the other with $n_y = 4$ and n_x increased from 8 to 13. This way, when we increment the total number of variables by 1, it is only for the first set that the number of symbolically eliminated variables increases. In Fig.4, we see that the time requirements for solving problems with fixed n_y (solid) have virtually remained consistent regardless of the total number of variables. On the other hand, the runtimes for the symbolic solution with increasing n_y have seen a huge jump at $n_x + n_y = 16$ and they are generally on the increase along with the number of variables (dashed). On the contrary, the final sizes of the MILP reduction — in terms of the number of nodes in XADD, the number of binary and continuous variables, and the number of constraints — have shown only mild increasing patterns.

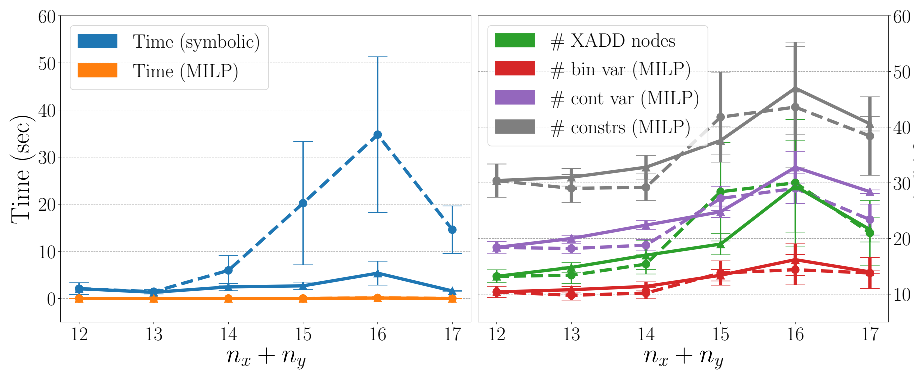


Fig. 4: Time and space complexity as the total number of variables increases. Here, $\mathbf{y}$ is symbolically minimized. The dashed lines correspond to the case of increasing n_y, whereas the solid lines represent the case of increasing n_x.

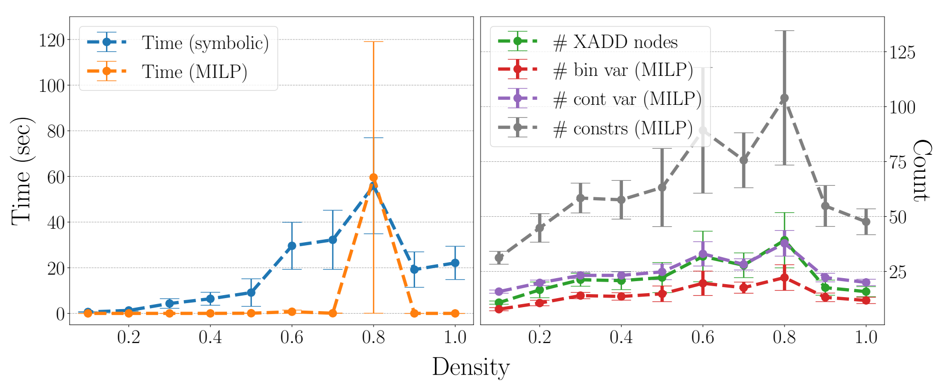


Fig. 5: Time and space complexity as the sparsity of $Q, \{\mathbf{a}_i\}_{i=1}^{n_a}, \{\mathbf{b}_j\}_{j=1}^{n_b}$ changes

For (b), we vary the density parameter used in the generation of the coefficient matrices ($\mathbf{a}, \mathbf{b}$, Q) from 0.1 to 1.0 (full matrices) and record the time and space complexity thereof. The numbers of variables are set to $(n_x, n_y) = (8, 4)$ and we eliminate $\mathbf{y}$ variables. Fig.5 shows a general trend where the MILP reduction becomes increasingly expensive as the density of the coefficient matrices rises. However, the complexity peaks at the density 0.8, and the instances with denser coefficients turn out to be easier to solve. Typically, instances that take longer symbolic compilation running times tend to result in XADDs with more nodes. Hence, it appears that sparse forms have few constraints leading to smaller encodings and solution times, while the highest density problems likely have redundant (implied) constraints that the XADD can eliminate also leading to smaller encodings and solution times.

To sum up, we have seen that there are types of DBLP problems that cannot be solved by Gurobi within a reasonable amount of time in their native form. We are able to solve such problems by solving the MILP equivalent of a DBLP which can be obtained via SVE. Using various test problems, we have also examined the efficiency of the proposed approach. In particular, we have observed that imbalanced problems are much easier to solve with SVE than their balanced counterparts with the same numbers of decision variables. Although it generally takes longer to solve a larger DBLP, there exists a set of problems with which we do not see much increase in solution time as the number of variables increases. These sorts of problems can benefit the most from our symbolic approach. Finally, we have seen that sparse instances can be more compactly represented via XADD, leading to smaller runtimes, while the densest form can be solved relatively easily as well.

6 Conclusion and Future Work

We proposed a novel use of symbolic variable elimination (SVE) for reducing one optimization problem (DBLP) to another (MILP) exactly in closed-form. We showed this methodological innovation involves extending existing SVE operations to work with *bilinear* forms. As a result, we were able to provide the *first exact constructive* MILP reformulation of DBLPs by proving that all symbolic operations involved remain closed-form. Empirically, we saw this reduction enables solving DBLPs with complex logical constraints to optimality, which are unsolvable in their native form.

As future work, we note that it is possible to extend our methodology to disjointly constrained *multilinear* programs (DMLPs), which will further broaden the applicability of our method to multi-agent decision-making problems [2].

Longer term, we hope that this work inspires the use of (and further research into) SVE as a technique for manipulating and reducing constrained optimization problems into alternative forms more amenable for use with highly efficient and optimal off-the-shelf solvers.

References

1. Audet, C., Hansen, P., Jaumard, B., Savard, G.: A symmetrical linear maxmin approach to disjoint bilinear programming. Mathematical Programming **85**(3), 573–592 (1999)
2. Becker, R., Zilberstein, S., Lesser, V., Goldman, C.V.: Transition-independent decentralized markov decision processes. In: Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems. p. 41–48. AAMAS '03, Association for Computing Machinery, New York, NY, USA (2003). https://doi.org/10.1145/860575.860583, https://doi.org/10.1145/860575.860583
3. Boutilier, C., Reiter, R., Price, B.: Symbolic dynamic programming for first-order MDPs. In: IJCAI-01. pp. 690–697. Seattle (2001)
4. Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual (2021), https://www.gurobi.com
5. Horst, R., Pardalos, P., Van Thoai, N.: Introduction to Global Optimization. Nonconvex Optimization and Its Applications, Springer US (1995), https://books.google.ca/books?id=w6bRM8W-oTgC
6. Horst, R., Tuy, H.: Global Optimization: Deterministic Approaches. Springer Berlin Heidelberg (2013)
7. Jeong, J., Jaggi, P., Sanner, S.: Symbolic dynamic programming for continuous state mdps with linear program transitions. In: Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI-21). Online (2021)
8. Konno, H.: A Bilinear Programming: Part II. Applications of Bilinear Programming. Technical Report (1975)
9. Nahapetyan, A.G.: Bilinear programming: applications in the supply chain management, pp. 282–288. Springer US, Boston, MA (2009)
10. Rebennack, S., Nahapetyan, A., Pardalos, P.M.: Bilinear modeling solution approach for fixed charge network flow problems. Optimization Letters **3**(3), 347–355 (2009). https://doi.org/10.1007/s11590-009-0114-0, https://doi.org/10.1007/s11590-009-0114-0
11. Sanner, S., Abbasnejad, E.: Symbolic variable elimination for discrete and continuous graphical models. Proceedings of the AAAI Conference on Artificial Intelligence **26**(1), 1954–1960 (Sep 2012)
12. Sanner, S., Delgado, K.V., de Barros, L.N.: Symbolic dynamic programming for discrete and continuous state mdps. In: Proceedings of the 27th Conference on Uncertainty in AI (UAI-2011). Barcelona (2011)
13. Sherali, H.D., Alameddine, A.: A new reformulation-linearization technique for bilinear programming problems. Journal of Global Optimization **2**(4), 379–410 (1992)
14. Vicente, L.N., Calamai, P.H., Júdice, J.J., J, J.J.: Generation of disjointly constrained bilinear programming test problems. Computational Optimization and Applications **1**, 299–306 (1992)
15. Ye, Z., Say, B., Sanner, S.: Symbolic bucket elimination for piecewise continuous constrained optimization. In: CPAIOR. pp. 585–594 (2018), https://doi.org/10.1007/978-3-319-93031-2_42
16. Zamani, Z., Sanner, S., Fang, C.: Symbolic dynamic programming for continuous state and action mdps. In: Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI-12). Toronto, Canada (2012)