Optimization and Engineering manuscript No.
(will be inserted by the editor)

Pyomo.GDP: an ecosystem for logic based modeling and optimization development

Qi Chen · Emma S. Johnson · David E. Bernal · Romeo Valentin · Sunjeev Kale · Johnny Bates · John D. Siirola · Ignacio E. Grossmann

Received: date / Accepted: date

Abstract We present three core principles for engineering-oriented integrated modeling and optimization tool sets— intuitive modeling contexts, systematic computer-aided reformulations, and flexible solution strategies—and describe how new developments in Pyomo.GDP for Generalized Disjunctive Programming (GDP) advance this vision. We describe a new logical expression system implementation for Pyomo.GDP allowing for a more intuitive description of logical propositions. The logical expression system supports automated reformulation of these logical constraints to linear constraints. We also describe two new logic-based global optimization solver implementations built on Pyomo.GDP that exploit logical structure to avoid "zero-flow" numerical difficulties that arise in nonlinear network design problems when nodes or streams disappear. These new solvers also demonstrate the capability to link to external libraries for expanded functionality within an integrated implementation. We present these new solvers in the context of a flexible array of solution paths available to GDP models. Finally, we present results on a new library of GDP models demonstrating the value of multiple solution approaches.

Keywords mathematical programming · generalized disjunctive programming · MINLP

Q. Chen · D.E. Bernal · R. Valentin · S. Kale · J. Bates · I.E. Grossmann
Center for Advanced Process Decision-Making, Department of Chemical Engineering, Carnegie Mellon University, Pittsburgh, PA 15213
Tel.: +1-412-268-3642
E-mail: grossmann@cmu.edu

E.S. Johnson · J.D. Siirola
Sandia National Laboratories, Albuquerque, NM 87123

1 Introduction

Mathematical programming is a powerful tool for tackling a wide range of challenges in the chemical process industries, with applications that include process design, planning, scheduling, and operations (Trespalacios and Grossmann 2014). To address these challenges, we must translate the conceptual problem into a mathematical formulation, apply a solution algorithm, and then interpret the results. The critical difficulty in this optimization process is the formulation of a tractable mathematical programming model that adequately describes all relevant problem logic. In general, optimization problems can involve decision-making in both continuous and discrete domains, e.g. selecting a coolant flowrate or among various coolant options, as well as nonlinear relationships between the decision variables. Traditionally, these problems are formulated as Mixed-Integer Nonlinear Programming (MINLP) models.

$$\begin{aligned} \min obj = {} & f(x,y) \\ s.t. \quad & g(x,y) \leq 0 \\ & h(x,y) = 0 \\ & x \in X \subseteq \mathbb{R}^n \\ & y \in Y \subseteq \mathbb{Z}^m \end{aligned} \tag{MINLP}$$

MINLP models are known to be NP-hard (Sahinidis 2019), and therefore modelers attempting to solve them often must apply domain-specific knowledge to arrive at a tractable modeling and solution strategy. It is often necessary to trade-off model fidelity with solution speed or to develop custom solution algorithms that exploit known model structure. However, the exact strategy needed to solve a given problem is not necessarily known a priori.

As a result, each layer of the optimization process (see Figure 1) can require multiple iterations of model development, as tradeoffs are explored between modeling detail and tractability. For a given conceptual problem (stage 1), different simplifying assumptions may be made in the physical, chemical, or process models (stage 2). For example, in a planning or scheduling problem, the choice of a continuous or discrete time representation can impact tractability (Floudas and Lin 2004). In process synthesis, high-level aggregate models may be preferable for solution targeting or rough approximations, but more rigorous models may be required to obtain a solution to support capital purchase decisions (Chen and Grossmann 2017). With a selected set of physical, chemical, and/or process models, the next step is translating them into a set of mathematical and logical relations among the decision and state variables (stage 3). These mathematical or logical models may involve differential equations or logical propositions that are not readily supported by modern optimization solvers. Therefore, they must be further converted into algebraic models, such as MINLP (stage 4). However, there is no unique model formulation for encoding problem logic as an MINLP, and no general technique for determining which formulations may result in a tractable model (Grossmann and Trespalacios 2013). Finally, if applying a custom solution algorithm (stage

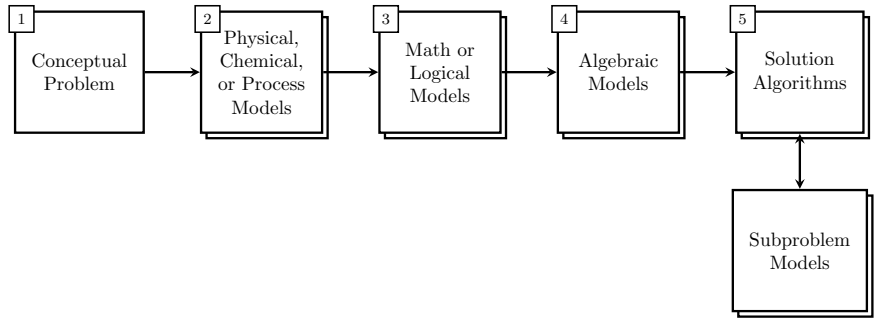


Fig. 1: Mathematical programming solution process

5), the modeler may be required to generate several additional models to provide, for example, the subproblems for a decomposition based solution scheme. For challenging problems, modelers may need to iterate several times between stages of the optimization process, leading to significant modeling effort. Many of the conversions from one stage to another still require manual translation by the modeler, increasing engineering costs and introducing potential sources of error.

To remedy this deficiency, we envision new integrated modeling and optimization toolsets based on three core principles:

1. Users should be able to express problem logic at a high level, using syntax intuitive to their domain of expertise.
2. High-level models should be supported by systematic computer-aided conversions and reformulations to tractable mathematical programming models.
3. Flexible solution strategies allowing for tailored approaches should be available to find high-quality model solutions.

Recent literature advances enable us to realize these toolsets by introducing new theoretical techniques and computational tools. First introduced by Raman and Grossmann in 1994 (Raman and Grossmann 1994), Generalized Disjunctive Programming (GDP) provides a high-level modeling construct that allows the intuitive specification of logical propositions and disjunctive relationships between groupings of constraints. For example, in distillation column design, the existence or absence of a particular tray is a disjunction: we impose equilibrium relations when a tray exists and pass-through constraints when a tray is absent (Barttfeld et al. 2003). Traditional disjunctive programming (Balas 1979, 2018) introduces logical disjunctions to linear programming and inspired improved cutting planes for integer programming, namely the lift-and-project cutting planes. Generalized disjunctive programming is the extension of the theory to include Boolean variables and nonlinear variable relationships. The general form for a GDP model can be seen in (GDP).

$$
\begin{aligned}
\min\, obj = {} & f(x,z) \\
s.t. \quad & Ax + Bz \leq d \\
& g(x,z) \leq 0 \\
& \bigvee_{i \in D_k} \begin{bmatrix} Y_{ik} \\ M_{ik}x + N_{ik}z \leq e_{ik} \\ r_{ik}(x,z) \leq 0 \end{bmatrix} \quad k \in K \\
& \Omega(Y) = True \\
& x \in X \subseteq \mathbb{R}^n \\
& Y \in \{True, False\}^p \\
& z \in Z \subseteq \mathbb{Z}^m
\end{aligned}
\tag{GDP}
$$

In (GDP), we minimize a quantity obj defined by an objective function $f(x,z)$, assumed without loss of generality to be linear. The optimization is performed subject to linear constraints $Ax + Bz \leq d$, nonlinear constraints $g(x,z) \leq 0$, and a set of disjunctions $k \in K$. Each of the disjunctions is comprised of a logical-OR relationship between disjuncts $i \in D_k$. A disjunction k represents a major discrete decision and each disjunct i in the disjunction represents a selection alternative. Selection of each of these alternatives, in turn, is indicated by the Boolean variable Y_{ik}, and may be described by both linear relationships $M_{ik}x + N_{ik}z \leq e_{ik}$ and nonlinear relationships $r_{ik}(x,z) \leq 0$. Logical propositions $\Omega(Y) = True$ on the overall model may also be present. These often establish an exclusive-OR or "at most one" relationship between disjuncts. Continuous variables x are assumed to be closed and bounded in the set X, Boolean variables Y take values of True or False, and integer variables z are closed and bounded in the set Z.

Critically, GDP modeling provides for well-defined standardized methods to convert a set of logical relations to an algebraic model. The two classical reformulation strategies are the big-M reformulation (BM) (Nemhauser and Wolsey 1988; Raman and Grossmann 1994) and hull reformulation (HR) (Balas 1985; Grossmann and Lee 2003) approaches. We refer the reader to an excellent review paper for details on these reformulations (Trespalacios and Grossmann 2014). BM provides an MINLP that does not require the introduction of additional variables, leading to a comparatively smaller problem size and potentially reduced solution time; however, HR yields an MINLP with a tighter continuous relaxation, which may reduce the number of iterations required for a solution algorithm to converge. More recently, Trespalacios and Grossmann (2016) describe a cutting-plane algorithm to construct an MINLP formulation tighter than BM, but without requiring as many new variables and constraints as HR. Other advanced reformulations are also possible (Bonami et al. 2015). These alternatives allow the modeler to systematically evaluate formulation options. Therefore, modelers can choose to express their problem logic as a GDP, and then select among known methods to arrive at an MINLP formulation. This additional layer of

abstraction eases the modeling burden (Chen and Grossmann 2019a) because the modeler can express problem logic without needing to simultaneously convert it to a specific algebraic formulation.

The formulation as a GDP also provides access to a variety of powerful decomposition algorithms developed over the years, including logic-based outer approximation (LOA) (Türkay and Grossmann 1996) and logic-based branch and bound (LBB) (Lee and Grossmann 2000). These algorithms exploit the availability of user-provided information via the GDP logical structure to decompose difficult problems. Many times, this logical information is obscured during the conversion to MINLP and then subsequently inferred by solver preprocessing routines (Belotti et al. 2016). By using a direct GDP algorithm, all relevant logical information is retained. LOA has been shown to improve convergence compared to traditional Outer Approximation (OA) or Generalized Benders Decomposition (GBD) (Türkay and Grossmann 1996).

However, to fully realize the benefits of GDP modeling, effective computational tools must be developed, allowing for modelers to easily perform the reformulations and other model transformations needed to implement the advanced solution schemes described in the literature. Algebraic modeling platforms such as GAMS (Brook et al. 1988), AIMMS (Bisschop and Roelofs 2006), AMPL (Fourer et al. 2003), Pyomo (Hart et al. 2017), and JuMP (Dunning et al. 2017) provide modelers with the ability to express optimization problems as a set of variables, equalities, and inequalities in a syntax reminiscent of mathematical notation, rather than requiring a direct interface to solver scripts. They are therefore an indispensable tool in the optimization process. To date, only two widely used algebraic modeling languages feature support for GDP (Chen et al. 2018): GAMS and Pyomo. GAMS is a long-standing algebraic modeling environment that offers GDP modeling via their extended mathematical programming framework (GAMS Development Corp 2020). After specification of the model, the GDP is automatically reformulated using BM or HR and solved as an MINLP by the LOGMIP 2.0 solver (Vecchietti and Grossmann 1997). Pyomo is an algebraic modeling platform written as an open-source software library within the higher-level programming language Python, providing GDP functionality using the `Disjunction` and `Disjunct` modeling objects. Modeling syntax for Pyomo.GDP is given in Appendix A.

In support of the core modeling principles, once a user has formulated a model in Pyomo.GDP, a powerful set of automated modeling manipulations becomes available to obtain a model solution, illustrated in Figure 2. The classical reformulation approaches to MINLP, BM and HR, remain available if the user wishes to leverage modern commercial MINLP codes or newer prototype strategies (Bernal et al. 2018). However, other advanced MINLP reformulations may also be applied, such as the hybrid BM/HR cutting plane approach (Trespalacios and Grossmann 2016). Moreover, preservation of logical structure in the GDP model allows for operations such as basic steps (Ruiz and Grossmann 2012) as well as the application of advanced logic-based solution algorithms (Chen et al. 2018).

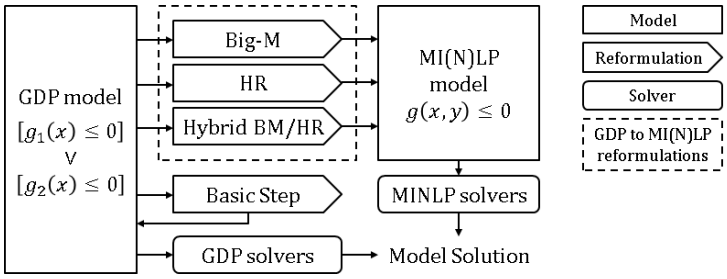


Fig. 2: GDP model solution strategies (Chen et al. 2018)

In this paper, we present Pyomo.GDP as an extensible ecosystem supporting development of new methods and tools for logic-based modeling and optimization. This modeling ecosystem both empowers practitioners to create more intuitive models for mathematical programming and enables algorithm developers to prototype advanced solution techniques that make use of high-level information about the problem logical structure. By providing this set of capabilities in an open-source and transparent modeling platform, Pyomo.GDP aims to promote accessibility of advanced algorithms and tools for optimization in support of the three previously described core optimization principles. In the following sections, we present new contributions in Pyomo.GDP. In Section 2, we introduce a new prototype logical expression system for Pyomo. In Section 3, we describe revised transparent implementations of GDP reformulations to MINLP. In Sections 4-6, we introduce new logic-based direct GDP solver implementations. We then introduce the application of these logical modeling and solution strategies on a set of case studies in section 7. Finally, we draw conclusions in section 8.

2 Logical expression system

As a complement to Pyomo.GDP, an extension was recently added to support a logical expression system within the Pyomo modeling language. The logical expression system allows for the inclusion of propositional logic in GDP models without requiring the user to manually convert the logical constraints to algebraic form. These statements may stipulate, for example, that the selection of reactor I (Y_I) requires the use of separator II (Y_{II}) or separator III (Y_{III}): $Y_I \implies Y_{II} \vee Y_{III}$. While this statement may equivalently be expressed using the algebraic relation, $-y_I + y_{II} + y_{III} \geq 0$, directly doing so obfuscates the original problem logic and violates our proposed core modeling principles. Furthermore, preservation of the original logical structure allows for its future use with logical inference platforms. Therefore, we implement the capability to convert logical constraints into Conjunctive Normal Form (CNF) and thereafter to linear inequalities using systematic approaches described in

the literature (Raman and Grossmann 1991), and previously implemented as part of LOGMIP (Vecchietti and Grossmann 1997).

To express the logical propositions, various modeling objects of the logical expression system are used. Table 1 lists the supported logical operations and their usage. `BooleanValue` objects are used to represent Boolean values and expressions, with `BooleanConstant` for logical literals (i.e. Python `True`/`False`) and `BooleanVar` for representing Boolean variables. To maintain backwards compatibility, auto-generated indicator variables for `Disjunct` objects remain `binary`; however, we support the ability to link a `BooleanVar` to its indicator counterpart (see Listing 1).

Listing 1: Associating Boolean variables with disjunct indicators.

```
for i in m.disjunct:
    m.Y[i].set_binary_var(m.disjunct[i].indicator_var)
```

The indexed `BooleanVar` can now be used to express logical propositions using `LogicalConstraint` objects. To express the logical relationship given in Equation (1), we can write the code given in Listing 2.

$$(Y_1 \vee Y_2) \implies (Y_3 \wedge \neg Y_4 \wedge (Y_5 \vee Y_6)) \tag{1}$$

Listing 2: Example of logical proposition posed as logical constraint.

```
m.proposition = LogicalConstraint(
    expr=(m.Y[1] | m.Y[2]).implies(
        m.Y[3] & Not(m.Y[4]) & (m.Y[5] | m.Y[6])))
```

After logical propositions in a GDP model have been specified, it may be advantageous to convert them to algebraic form. We can make use of the `logical_to_linear` transformation implemented in Pyomo.GDP (see Listing 3).

Listing 3: Applying the logical-to-linear transformation. The second line is optional and prints out the auto-generated constraints.

```
TransformationFactory('core.logical_to_linear').apply_to(m)
m.logic_to_linear.pprint() # pretty-print auto-generated constraints
```

After a solution for the GDP model is obtained, the Boolean variable values may be automatically updated from the solution by invoking Listing 4.

Listing 4: Updating the Boolean variable values from the solution values of their associated binary variables (see Listing 1).

```
update_boolean_vars_from_binary(m)
```

Deeper integration of the logical expression system into Pyomo remains underway. However, the present work is nevertheless openly available online, functional, and demonstrates a step towards improving the expressive capabilities of Pyomo.GDP.

Table 1: Supported logical operators

Operator	Usage	Description
Not ($\neg$)	`~m.Y[1]`	Negation of Y_1
And ($\wedge$)	`m.Y[1] & m.Y[2]`	Conjunction of Y_1 and Y_2
Or ($\vee$)	`m.Y[1] | m.Y[2]`	Disjunction of Y_1 and Y_2
Xor ($\oplus$)	`m.Y[1] ^ m.Y[2]`	Exclusive disjunction of Y_1 and Y_2
If ($\Longrightarrow$)	`m.Y[1] >> m.Y[2]`	Y_1 implies Y_2
Iff ($\Longleftrightarrow$)	`m.Y[1] == m.Y[2]`	Y_1 if and only if Y_2
Exactly N	`Exactly(1, m.Y[:])`	Exactly 1 of Y_i is True
At Most N	`AtMost(2, m.Y[:])`	At most 2 of Y_i are True
At Least N	`AtLeast(3, m.Y[:])`	At least 3 of Y_i are True

3 Transparent reformulations

As previously introduced, the traditional solution approach for GDP models involves a reformulation to an algebraic MINLP model. Chen et al. (2018) briefly discussed the automated implementation of the BM, HR, and hybrid BM/HR reformulations in Pyomo.GDP. Here, we elaborate on that conference paper and describe recent implementation changes that improve transparency and flexibility of the automated reformulations. The new implementation also seamlessly supports hierarchical modeling (Friedman et al. 2013), including the solution of reformulated model sub-blocks.

The conventional view on algebraic reformulations is to apply a given reformulation to an entire GDP model, and then to immediately send it to the MINLP solver. This is the approach supported by LOGMIP, and is also possible using Pyomo.GDP (see Listing 5).

Listing 5: Basic usage of the GDP to MINLP reformulations.

```
# Option 1: BM reformulation
TransformationFactory('gdp.bigm').apply_to(m)
# Option 2: HR reformulation
TransformationFactory('gdp.hull').apply_to(m)
# Option 3: Hybrid BM/HR reformulation
TransformationFactory('gdp.cuttingplane').apply_to(m)
```

However, advanced modelers often wish to adopt a mixed approach, reformulating some disjunctions with one strategy and others with a different strategy. They also frequently wish to manipulate or interrogate the model after reformulation. Pyomo.GDP supports these advanced use cases, as the user retains programmatic access to the Pyomo model object after the transformation. Indeed, the hybrid BM/HR cutting plane approach itself is an automated example of such advanced use. These transformations are fully compatible with the logical expression system, provided that the logical expressions are transformed first. Listing 6 also demonstrates how to obtain a transformed model copy using `create_using` rather than the in-place reformulations performed by `apply_to`.

Listing 6: Obtaining a transformed model copy using the BM reformulation.

```
transformed_model = TransformationFactory('gdp.bigm').create_using(m)
```

For mixed reformulation strategies, we support a `targets` argument to the transformations, allowing for a list of Pyomo components to be specified. These can either be Pyomo `Block`, `Disjunct`, or `Disjunction` objects. For `Block` and `Disjunct` targets, all contained disjunctions are transformed. `Disjunction` objects are reformulated individually.

Transformations deactivate the `Disjunct` and `Disjunction` objects on the model and create private `Block` objects (on the parent block of the `Disjunction`) storing the transformed components. We provide a public API for mapping between the original modeling components and the products of their transformation, described in Listing 7.

Listing 7: Mapping functions between original and reformulated Pyomo.GDP model objects.

```
"""
The following functions are applicable for both BM and HR
"""
xfrm = TransformationFactory('gdp.bigm')
# Given a transformation Block, returns its source Disjunct
xfrm.get_src_disjunct(transBlock)
# Given a transformed XOR (or OR) constraint, returns its source Disjunction
xfrm.get_src_disjunction(xor_constraint)
# Given a transformed constraint, returns its source constraint
xfrm.get_src_constraint(transformedConstraint)
# Given a source constraint, returns its transformed counterparts
xfrm.get_transformed_constraints(srcConstraint)
# Given a Disjunct object, get its transformation block
xfrm_block = disjunct.transformation_block()
# Given a Disjunction object, get its algebraic constraint counterpart
xor_constr = disjunction.algebraic_constraint()

"""
The following functions are only applicable for HR
"""
xfrm = TransformationFactory('gdp.hull')
# Given a variable and a Disjunct, returns its disaggregated counterpart.
xfrm.get_disaggregated_var(original_var, disjunct)
# Given a disaggregated variable, return its source variable.
xfrm.get_src_var(disaggregated_var)
# Given a variable and disjunction, return its
# corresponding disaggregation constraint.
xfrm.get_disaggregation_constraint(original_var, disjunction)
# Given a variable, return the constraint for its variable bounds.
xfrm.get_var_bounds_constraint(original_var)
```

The helper functions are defined for the `Transformation` object returned by `TransformationFactory()`. All transformed `Disjunct` objects also contain a pointer to their corresponding transformation block, and all transformed `Disjunction` objects contain a pointer to the algebraic `OR` or generalized `XOR` constraint. An advanced user can interrogate the transformed model using these functions.

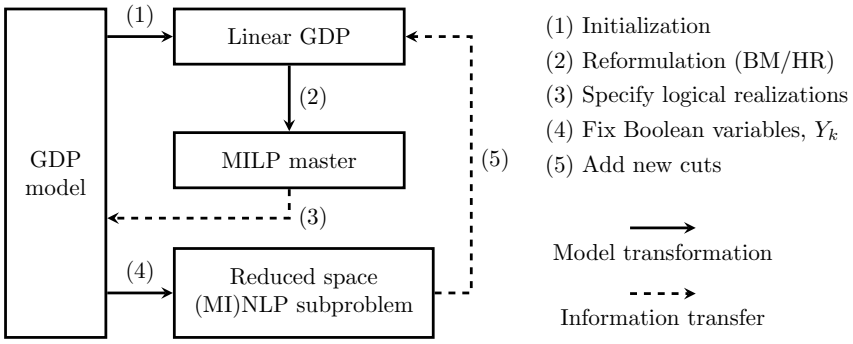


Fig. 3: GDPopt solver implementation of the logic-based outer approximation algorithm

With the public API, we offer an open invitation for integration of these tools within more elaborate solution strategies.

4 GDPopt-LOA: logic-based outer approximation

Rather than reformulate to MINLP, advanced decomposition algorithms that operate directly on the GDP model are also available in Pyomo.GDP, implemented in the GDPopt solver (Chen et al. 2018). One such algorithm is logic-based outer approximation (Türkay and Grossmann 1996) (LOA). The first implementation of LOA was found in the LOGMIP 1.0 solver (Vecchietti and Grossmann 1997), using modeling extensions of the GAMS algebraic modeling language; however, as of LOGMIP 2.0 in GAMS version 23.7 (July 2011), LOA is no longer supported in LOGMIP. Therefore, GDPopt-LOA is the only working modern implementation of LOA, built as a meta-solver in Pyomo.GDP, with a free and open-source code base[1]. Note that despite being itself an open-source code, GDPopt is capable of utilizing commercial solvers, e.g. BARON (Tawarmalani and Sahinidis 2005) and Gurobi (Gurobi Optimization Inc 2016), as sub-solvers in the algorithm. GDPopt-LOA invocation is illustrated in Listing 8.

Listing 8: Example of GDPopt-LOA invocation in Pyomo.GDP.

```
results = SolverFactory('gdpopt').solve(m, tee=True)
```

The LOA algorithm exploits the logical structure of a GDP model to decompose its solution into a sequence of master problems and subproblems. At its core, LOA separates the selection of new logical realizations (or major discrete decisions) from the evaluation and optimization of the full nonlinear

[1] GDPopt is released as part of Pyomo: https://www.github.com/Pyomo/pyomo

$$\min obj = f(x,z) \tag{l-GDP.1}$$
$$s.t. \quad Ax + Bz \leq d \tag{l-GDP.2}$$
$$g(x^l,z^l) + \nabla g(x^l,z^l)\begin{bmatrix} x - x^l \\ z - z^l \end{bmatrix} \leq 0, \quad l \in \{1,\ldots,L\} \tag{l-GDP.3}$$
$$\bigvee_{i \in D_k} \begin{bmatrix} Y_{ik} \\ M_{ik}x + N_{ik}z \leq e_{ik} \\ r_{ik}(x^l,z^l) + \nabla r_{ik}(x^l,z^l)\begin{bmatrix} x - x^l \\ z - z^l \end{bmatrix} \leq 0, \quad l \in L_{ik} \end{bmatrix} \quad k \in K \tag{l-GDP.4}$$
$$\Omega(Y) = True \tag{l-GDP.5}$$
$$x \in X \subseteq \mathbb{R}^n \tag{l-GDP.6}$$
$$Y \in \{True, False\}^p \tag{l-GDP.7}$$
$$z \in Z \subseteq \mathbb{Z}^m \tag{l-GDP.8}$$

description for each logical realization. Figure 3 illustrates the major steps involved in the LOA algorithm. From the original GDP model, an initialization procedure (1) generates an initial linear GDP approximation, (l-GDP). The classical initialization procedure requires an evaluation of the constraint value and gradient for each nonlinear constraint, obtained by solving nonlinear subproblems in a set-covering procedure (see Section 4.3).

This linear GDP is reformulated (2) to MILP via an automatic model transformation in Pyomo.GDP. By default, the BM reformulation is used, although the HR or other advanced strategies (Chen et al. 2018) are also possible. The resulting MILP (MILP-BM) represents a linear approximation of the full-space logical problem, and the solution to the master problem represents (3) a candidate logical realization.

By (4) fixing the Boolean variables, Y_{ik}, to the values corresponding to the candidate logical realization proposed by the master problem, a nonlinear subproblem (rsMINLP^l) is obtained for iteration l. Due to the logical structure of the GDP model, this nonlinear subproblem can be solved in a reduced space—that is, only those constraints described in active disjuncts ($Y_{ik} = True$) are included in the subproblem; the constraints corresponding to unenforced logical realizations are simply omitted from the subproblem model. This not only results in a smaller subproblem in comparison to the analogous subproblem obtained from a corresponding MINLP representation of the full-space problem, but also improves subproblem robustness due to the avoidance of "zero-flow" singularities, discussed below.

After solving the nonlinear subproblem in reduced space, information from its solution is used to augment the linear GDP in the form of additional linear constraints (5), improving the quality of of the linear approximation in the master problem. These cuts outer-approximate the nonlinear functions using a first-order Taylor approximation centered at the variable solution values. In

$$\min obj = f(x,z) \tag{MILP-BM.1}$$
$$s.t. \quad Ax + Bz \leq d \tag{MILP-BM.2}$$
$$g(x^l,z^l) + \nabla g(x^l,z^l)\begin{bmatrix} x - x^l \\ z - z^l \end{bmatrix} \leq 0, \quad l \in \{1,\ldots,L\} \tag{MILP-BM.3}$$
$$M_{ik}x + N_{ik}z \leq e_{ik} + BM_{ik}(1 - y_{ik}), \quad i \in D_k, k \in K \tag{MILP-BM.4}$$
$$r_{ik}(x^l,z^l) + \nabla r_{ik}(x^l,z^l)\begin{bmatrix} x - x^l \\ z - z^l \end{bmatrix} \leq BM_{ik}(1 - y_{ik}), \quad i \in D_k, k \in K, l \in L_{ik} \tag{MILP-BM.5}$$
$$\sum_{i \in D_k} y_{ik} \geq 1, \quad k \in K \tag{MILP-BM.6}$$
$$\hat{\Omega} y \leq \hat{d} \tag{MILP-BM.7}$$
$$x \in X \subseteq \mathbb{R}^n \tag{MILP-BM.8}$$
$$y \in \{0,1\}^p \tag{MILP-BM.9}$$
$$z \in Z \subseteq \mathbb{Z}^m \tag{MILP-BM.10}$$

$$\min obj^l = f(x,z) \tag{rsMINLP^l.1}$$
$$s.t. \quad Ax + Bz \leq d \tag{rsMINLP^l.2}$$
$$g(x,z) \leq 0 \tag{rsMINLP^l.3}$$
$$\left.\begin{matrix} M_{ik}x + N_{ik}z \leq e_{ik} \\ r_{ik}(x,z) \leq 0 \end{matrix}\right\} \text{ if } Y^l_{ik} = True, \quad k \in K \tag{rsMINLP^l.4}$$
$$x \in X \subseteq \mathbb{R}^n \tag{rsMINLP^l.5}$$
$$z \in Z \subseteq \mathbb{Z}^m \tag{rsMINLP^l.6}$$
(rsMINLP^l.7)

step (5), an integer "no-good" cut is also added so that the same solution is not explored multiple times.

Assuming minimization, solutions to the master problem provide a lower bound on the remaining feasible logical realizations at each iteration (as we outer approximate the feasible region). The best feasible solution obtained from the reduced space subproblems provides an upper bound on the objective value. Termination of the algorithm occurs when the lower bound at an iteration converges to or crosses over the upper bound, indicating that the set of remaining unexplored logical realizations does not contain a better solution. An infeasible master problem indicates that no logical realizations remain to be explored, equivalent to a lower bound of $obj^{LB} = \infty$. Convergence of LOA is checked after both the master problem and reduced space subproblem solutions.

4.1 GDP models with integer variables

We note that though a standard GDP formulation (see Trespalacios and Grossmann 2014) does not include discrete variables in addition to Boolean variables, GDPopt-LOA supports the solution of such hybrid models (Vecchietti and Grossmann 1997) given by Problem (GDP) through one of two strategies. The default strategy is to handle only logical realizations (Boolean variables) in the master problem, and to solve an MINLP as the reduced space subproblem. An alternative strategy is to address the additional discrete variables in the master problem, as had been implemented in LOGMIP (Vecchietti and Grossmann 1997). The subproblem would, therefore, remain an NLP, but more master iterations may be required, as different values of these embedded discrete variables would be treated as different logical realizations. The trade-off is therefore between combinatorial complexity in the master problem and the subproblems. Both approaches are supported in GDPopt, allowing the user to select the most suitable for their problem.

4.2 GDP models with nonlinear equality constraints

We also note that to handle nonlinear equality constraints, the outer approximation with equality relaxation (OA/ER) cut generation approach from Kocis and Grossmann (1989) is applied. In ER, each equality is relaxed to an inequality based on the sign of its Lagrangean multiplier at each iteration. If the equality can be equivalently relaxed as a convex inequality, then global optimality guarantees are preserved. However, for non-convex GDP models, that is, instances of model (GDP) where $g(x)$ and/or $r_{ik}(x)$ include non-convex functions, the first-order Taylor approximation may not rigorously outer-approximate the feasible region of the problem. As a heuristic, the augmented penalty (AP) approach described by (Viswanathan and Grossmann 1990) is used by default. This introduces slack variables $0 \leq s_{ik}$ and $0 \leq \sigma$ on violations of the generated AP/OA/ER cuts, which are penalized via a linear term added to the objective function. As a result, the following modifications to (l-GDP) are required:

1. Generated linear cuts on the global constraints $g(x, z)$ in Equation (l-GDP.3) must be relaxed: $g(x^l, z^l) + \nabla g(x^l, z^l) \begin{bmatrix} x - x^l \\ z - z^l \end{bmatrix} \leq \sigma$
2. Generated OA cuts on the nonlinear constraints within each disjunct of Equation (l-GDP.4) must be relaxed: $r_{ik}(x^l, z^l) + \nabla r_{ik}(x^l, z^l) \begin{bmatrix} x - x^l \\ z - z^l \end{bmatrix} \leq s_{ik}$
3. The augmented penalty term must be added to the objective function (l-GDP.1): $\min obj = f(x, z) + \rho\sigma + \sum_{i \in D_k, k \in K} \rho s_{ik}$

By default, a penalty parameter of $\rho = 1000$ is used. These values may be tuned through solver options arguments to GDPopt-LOA. If the modeler knows that a given problem is convex, the slack values should be set to zero: $\sigma = s_{ik} = 0$.

The use of reduced space nonlinear subproblems allows GDPopt to avoid numerical difficulties associated with the deactivation of disjuncts. In network flow problems such as chemical flowsheet synthesis, these deactivated disjuncts often involve setting flow through a disappearing node (or process unit) to zero. When performance equations for these nodes include functions like $\log(x)$, $x^a, a \in (0, 1)$, or $1/x$, nonlinear solver stability may suffer as a flow variable x approaches zero, exacerbated by numerical noise. The absence of flow also creates a singularity that results in degeneracy in variables that become irrelevant, such as component concentrations. Because there is zero flow, any value of the concentration is valid in the context of a solution. However, these degenerate variables may participate nonlinearly in expressions that become poorly conditioned for certain variable values. We refer to these numerical difficulties as "zero-flow" singularities.

In this work, we also present several novel enhancements to the LOA algorithm implemented in GDPopt version 20.05.23, including a more efficient set-covering initialization variant, the extension of GDPopt-LOA to guarantee global optimality, and the use of tighter disjunctive variable bounds for relaxations.

4.3 GDPopt-LOA initialization

LOA initialization generates the starting linear approximation of the full-space logical problem. By default, GDPopt features an enhanced version of the set-covering initialization approach described by Türkay and Grossmann (1996, Appendix D), whereby enough nonlinear subproblems are solved such that each disjunct is active in at least one feasible subproblem. Therefore, after the set-covering initialization, each disjunct has at least one set of linear approximations for its nonlinear constraints. However, the literature approach has two major shortcomings: first, the iterative set-covering routine may generate several infeasible subproblems, stalling the initialization procedure; second, the original routine may evaluate more logical configurations than is necessary, as disjuncts with no nonlinear constraints do not require a covering.

GDPopt-LOA features an improved set-covering algorithm (pictured in Figure 4 that includes the linear constraints of the master problem when solving the set-covering problem (SCOV-BM) to determine the next logical realization to evaluate in the initialization. These constraints include logical relations encoded in the logical prepositions of the original GDP model. With their inclusion, the set-covering master problem is more likely to generate feasible logical realizations.

The initialization algorithm begins with the iteration counter $l = 1$ and definition of the sets DC^0 and DN^0, the initial set of covered disjuncts and disjuncts needing to be covered, respectively: $DC^0 = \varnothing$ and $DN^0 = \{(i, k) \mid i \in$

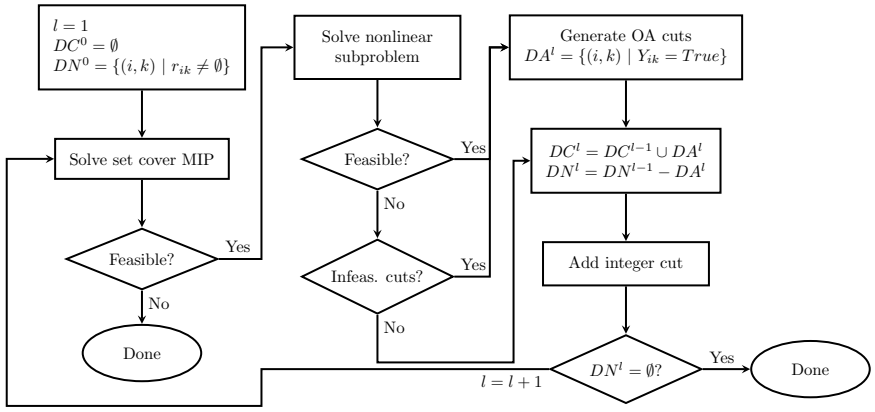


Fig. 4: Set covering initialization procedure

$$\max obj^l_{cover} = (|DC^{l-1}| + 1)\sum_{ik} y_{ik} + \sum_{ik} y_{ik} \tag{SCOV-BM.1}$$

$$s.t. \quad Ax + Bz \leq d \tag{SCOV-BM.2}$$

$$\hat{\Omega} y \leq \hat{d} \tag{SCOV-BM.3}$$

$$M_{ik}x + N_{ik}z \leq e_{ik} + BM_{ik}(1 - y_{ik}) \quad k \in K \tag{SCOV-BM.4}$$

$$\sum_{i \in D_k} y_{ik} \geq 1 \quad k \in K \tag{SCOV-BM.5}$$

$$x \in X \subseteq \mathbb{R}^n \tag{SCOV-BM.6}$$

$$y \in \{0,1\}^p \tag{SCOV-BM.7}$$

$$z \in Z \subseteq \mathbb{Z}^m \tag{SCOV-BM.8}$$

D_k and $k \in K$ and $r_{ik} \neq \varnothing\}$. Note that GDPopt-LOA reduces the number of nonlinear subproblems required for the set-covering initialization by seeking to cover only those disjuncts with nonlinear constraints ($r_{ik} \neq \varnothing$). For network design problems (such as conceptual process design) where the formulation is a disjunction between presence and absence of a potential node, this can significantly reduce the number of initialization subproblems needed. The set-covering master problem (SCOV-BM) provides a set of active disjuncts Y_{ik}. if (SCOV-BM) yields an infeasible result, then the set-covering initialization concludes. If this occurs on the first iteration ($l = 1$), then the GDP model is infeasible, as the feasible region of (SCOV-BM) is a relaxation of the original model (GDP). If a feasible result is obtained, then the reduced space subproblem (rsMINLP^l) corresponding to the active disjuncts is solved. If (rsMINLP^l) for the given Y_{ik} is infeasible, then OA cuts may nevertheless be generated, or the step may be skipped. Note that a feasible solution is not required to generate OA cuts, but as a heuristic, they are assumed to be of better quality when generated at optimal subproblem solution points. By

default, GDPopt-LOA does not generate OA cuts at infeasible subproblem solution values. If OA cuts are generated, then a set of newly covered disjuncts $DA^l = \{(i,k) \mid Y_{ik} = True\}$ is defined; otherwise, $DA^l = \varnothing$. The sets DC and DN are then updated to reflect the newly covered disjuncts, and an integer cut is added to exclude the logical realization from subsequent runs. If the set of disjuncts remaining to be covered is empty, then the initialization concludes. Otherwise, the iteration counter is advanced and another set covering MIP is solved to repeat the process.

Other initialization approaches are also possible with GDPopt-LOA. In some cases, the relaxation resulting from excluding all nonlinear constraints is sufficient as the initial master problem, effectively skipping the initialization step. This may result in an unbounded master problem, in which case GDPopt will emit a warning and enforce an arbitrary lower bound of $obj^{LB} = -1 \times 10^{15}$ to the problem to generate an arbitrary candidate logical realization. Custom initialization options may also be added by amending the GDPopt code, but even without modifying the code base, the user is also able to specify their own sequence of logical realizations to evaluate. Note that initialization of variable values (particularly for the subproblems) is also possible with GDPopt, and may even utilize tightened bounds, described in Section 5.1.

5 GDPopt-GLOA: Global optimization

The logic-based outer approximation algorithm in GDPopt-LOA is guaranteed to converge to the optimal solution of convex GDP problems—problems where the feasible region of the continuous relaxation of all the disjuncts and global constraints is convex (Grossmann and Lee 2003). Unfortunately, this is not the case for many relevant GDP models, which include non-convex functions (e.g. from mixing). There is a set of algorithms that can guarantee convergence to the globally optimal solution of non-convex GDPs (Ruiz and Grossmann 2017). Among these algorithms is an extension of the LOA algorithm, which we refer to as Global Logic-based Outer Approximation (GLOA) (Lee and Grossmann 2001). GDPopt-GLOA offers the first automated GLOA implementation, with usage given in Listing 9.

Listing 9: Example of GDPopt-GLOA invocation in Pyomo.GDP.

```
results = SolverFactory('gdpopt').solve(
    m, tee=True, strategy="GLOA", nlp_solver="baron")
```

While GLOA is described by Lee and Grossmann (2001) and Bergamini et al. (2005), the previous implementations require the user to manually provide required reformulations and the template for the linearization of each nonlinear expression. For larger models, this quickly becomes an onerous burden. GDPopt offers automatic generation and updating of the necessary subproblem models for GLOA. To guarantee global optimality in GLOA, two conditions must be imposed on the base LOA algorithm: first, the nonlinear

subproblems must be solved to global optimality, and second, the outer approximation cuts added to the linear GDP must be rigorous. That is, they must not exclude unexplored feasible regions of the original GDP model. To satisfy the first condition, the user must specify a global optimization solver to the GDPopt-GLOA `nlp_solver` optional argument. Note that global optimization solvers offer convergence to an ϵ-optimal solution (Boukouvala et al. 2016). The quality of the optimality guarantee provided by GDPopt is thus dependent upon the quality of the solution provided by the nonlinear subproblem solver. Note that if the GDP formulation includes integer variables, the subproblem will be an MINLP, and the `minlp_solver` optional argument is used instead, as seen in Listing 10.

Listing 10: Example of GDPopt-GLOA invocation in Pyomo.GDP.

```
results = SolverFactory('gdpopt').solve(
    m, tee=True, strategy="GLOA", minlp_solver="baron")
```

To ensure that the outer approximation linearizations are rigorous, GDPopt-GLOA makes use of convex and/or concave envelope values computed using MC++, a library in C++ allowing for computations and operations on factorable functions (Chachuat 2019), including multivariate McCormick relaxations (Tsoukalas and Mitsos 2014). Using the ctypes Python library, an interface was developed to bridge Pyomo expressions in Python with MC++ data types in C/C++. With the Pyomo-MC++ interface, nonlinear expressions may be queried for their convex under-estimator and concave over-estimator values and corresponding subgradients at the variable values from the GLOA subproblem solution. This information is used to generate two cutting planes that rigorously outer approximate the feasible region (see Figure 5), strengthening the linear GDP.

After installation, usage of the MC++ interface is straightforward. A `McCormick` class provides a wrapper for any valid Pyomo expression: `mc_expr = McCormick(pyomo_expr)`. This wrapper object can then be queried. `mc_expr.convex()` yields the value of the convex underestimator using the current Pyomo variable values. The Pyomo-MC++ interface demonstrates the capability in Pyomo.GDP to integrate external libraries for the creation of advanced algorithmic implementations.

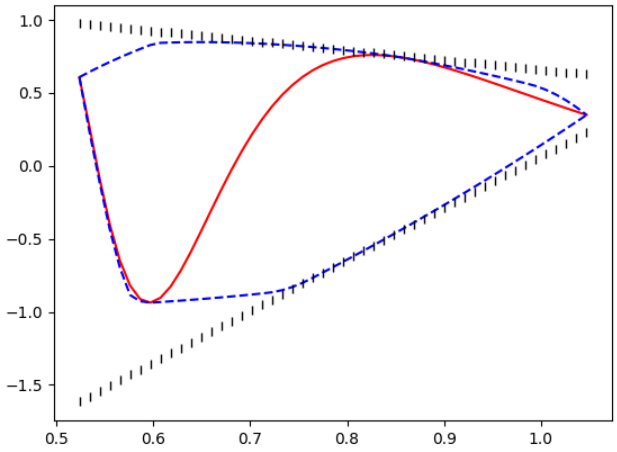


Fig. 5: Example of outer approximation cutting planes (vertical hashes) generated for the nonlinear function $f(x) = cos(x^2)sin(x^{-3})$ (red line) with $\frac{\pi}{6} \leq x \leq \frac{\pi}{3}$ and $x^* = \frac{\pi}{4}$. Blue dashed lines denote the envelope values sampled across the x domain.

5.1 Disjunctive variable bounds tightening

$$\min \ z \tag{DVB.1}$$

$$s.t. \quad \frac{1}{e} \leq x \leq e \tag{DVB.2}$$

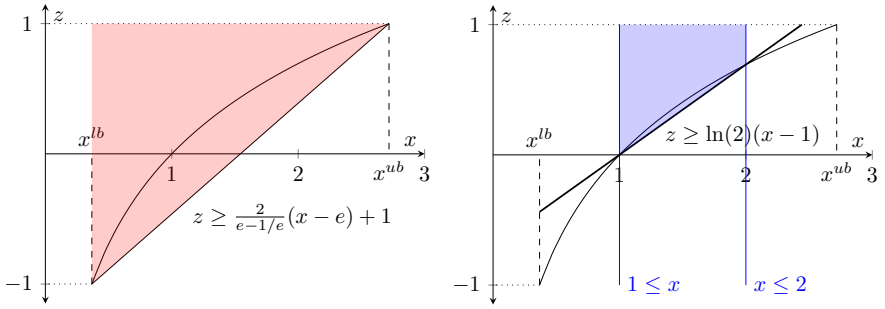
$$\begin{bmatrix} Y \\ z \geq \ln(x) \\ 1 \leq x \leq 2 \end{bmatrix} \vee \begin{bmatrix} \neg Y \\ z \geq x - 1 \end{bmatrix} \tag{DVB.3}$$

$$-1 \leq z \leq 1 \tag{DVB.4}$$

$$z, x \in \mathbb{R} \tag{DVB.5}$$

$$Y \in \{True, False\} \tag{DVB.6}$$

The quality of the MC++ envelopes for GLOA—and thus, the quality of the generated supporting hyperplanes—depends upon the tightness of the variable bounds. Range reduction has been shown to be an important tool in global optimization (Puranik and Sahinidis 2017). In GDPopt-GLOA, we can exploit logical structure to implement range reduction within the logical scopes defined by disjuncts. Use of GDPopt-GLOA with disjunctive variable bounding is denoted GDPopt-GLOA-DVB. To illustrate its impact, consider Problem (DVB). To generate the linear approximation (l-GDP), we must outer-approximate the nonlinear inequality $z \geq \ln(x)$ in the first disjunct of (DVB.3). Figure 6



(a) Global bounds $x^{lb} = \frac{1}{e}$, $x^{ub} = e$ (b) In disjunctive scope, $x \geq 1$ and $x \leq 2$.

Fig. 6: Rigorous linear approximations of $z \geq ln(x)$ with and without disjunctive scope variable range reduction.

illustrates the difference between the quality of two possible outer approximations for the nonlinear expression. If the linear underestimator is generated based on the global bounds of x in (DVB.2), then we obtain the constraint $z \geq \frac{2}{e-1/e}(x-e)+1$, pictured in Figure 6a. However, within the disjunctive context of $Y = True$, tighter variable bounds of $1 \leq x \leq 2$ exist for x. Therefore, a tighter linear underestimator of $z \geq \ln(2)(x-1)$ may be derived, pictured in Figure 6b.

In Pyomo.GDP, these disjunctive variable bounds may be automatically computed using either feasibility-based bounds tightening (FBBT) or optimality-based bounds tightening (OBBT). To execute this computation, a subroutine traverses the logical structure of the GDP model and determines the effective variable bounds within each disjunctive scope, storing them as a mapping on each disjunct. By doing so, we support models featuring nested disjunctions, as in Equation (3). These mappings provide enhanced variable bounds to tighten the MC++ envelopes used to generate outer approximation cuts. Use of disjunctive variable bounds in GDPopt-GLOA is given by Listing 11.

Listing 11: Example of GDPopt-GLOA-DVB invocation in Pyomo.GDP.

```
results = SolverFactory('gdpopt').solve(
    m, tee=True, strategy="GLOA",
    nlp_solver="baron", minlp_solver="baron",
    calc_disjunctive_bounds=True)
```

GDPopt provides a powerful new direct logic-based solution alternative for GDP models with flexibility for customization and a transparent implementation. Taking advantage of disjunctive structure, it is able to avoid nonlinear zero-flow numerical difficulties related to deactivation of nodes/paths in a network design problem. GDPopt also demonstrates the ability to integrate many different tools within a cohesive algorithmic implementation.

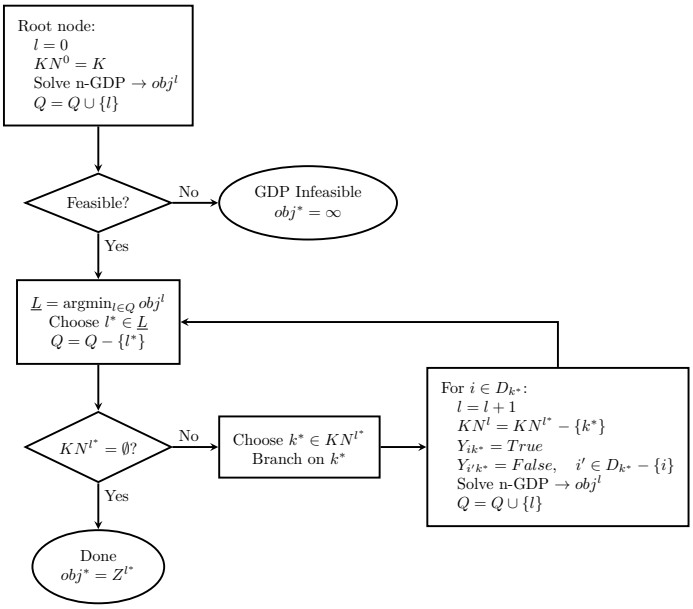


Fig. 7: GDPopt-LBB algorithm flow diagram

6 GDPopt-LBB: logic-based branch and bound

Given the higher-level description supported by GDP modeling, the user provides a hierarchy for decision-making encoded in the disjunctions, which can be exploited by solution methods. One way to take advantage of this is by using the logical variables in the GDP as the branching variables in a branch-and-bound approach. This was one of the first tailor-made disjunctive algorithms, logic-based branch-and-bound, proposed by Beaumont (1990) and generalized for GDPs by Grossmann and Lee (2003). We present GDPopt-LBB, a new implementation of the logic-based branch-and-bound algorithm, which directly solves models in Pyomo.GDP. Note that our implementation differs from that described by Grossmann and Lee (2003), as we preserve integrality at our solution nodes. Observe that our root node remains a nonlinear GDP rather than an NLP as in (Grossmann and Lee 2003). Usage of GDPopt-LBB is given in Listing 12.

Listing 12: Example of GDPopt-LBB invocation in Pyomo.GDP.

```
results = SolverFactory('gdpopt').solve(
    m, tee=True, strategy="LBB", minlp_solver="baron")
```

The algorithm implemented in GDPopt-LBB is summarized in Figure 7, involving the global optimization of a succession of subproblems. GDPopt-LBB begins by setting the node counter $l = 0$ and initializing the set $KN^0 = K$

$$\min obj^l = f(x,z) \tag{n-GDP.1}$$
$$s.t. \quad Ax + Bz \leq d \tag{n-GDP.2}$$
$$g(x,z) \leq 0 \tag{n-GDP.3}$$
$$\left.\begin{array}{c} M_{ik}x + N_{ik}z \leq e_{ik} \\ r_{ik}(x,z) \leq 0 \end{array}\right\} \quad \text{if } Y_{ik} = True, \quad k \in KB^l \tag{n-GDP.4}$$
$$\bigvee_{i \in D_k} \begin{bmatrix} Y_{ik} \\ M_{ik}x + N_{ik}z \leq e_{ik} \\ \Psi_{ik}(Y) = True \end{bmatrix} \quad k \in KN^l \tag{n-GDP.5}$$
$$\Omega(Y) = True \tag{n-GDP.6}$$
$$x \in X \subseteq \mathbb{R}^n \tag{n-GDP.7}$$
$$Y \in \{True, False\} \tag{n-GDP.8}$$
$$z \in \{0,1\}^m \tag{n-GDP.9}$$

corresponding to the set of unbranched disjunctions. Initially, all disjunctions are unbranched. At the root node, the subproblem (n-GDP) is a relaxation of the original model (GDP) in which all nonlinear constraints involved only in disjunctions are omitted. In general, (n-GDP) is still a nonlinear GDP model due to the inclusion of global nonlinear constraints $g(x,z)$. However, relaxation of the decision-dependent nonlinear constraints allows this algorithm, like GDPopt, to avoid nonlinear zero-flow numerical issues. Note that for all nodes l, the set of branched disjunctions may be defined $KB^l = K \setminus KN^l$.

At each iteration, the node on the queue Q with the minimum objective value is dequeued. If multiple nodes share the same minimum objective value, then the node with the fewest unbranched disjunctions $|KN^l|$ and that was most recently encountered, successively, will be selected. Note that the objective value of each evaluated node in Q gives a lower bound for its children. The minimum objective value of all nodes in Q therefore gives a global lower bound on (GDP). If the selected node has no unbranched disjunctions, then an upper bound to (GDP) has been found matching this global bound and the algorithm concludes.

Otherwise, an arbitrary unbranched disjunction $k^* \in KN^{l^*}$ is selected and branching takes place. In its current implementation, GDPopt-LBB assumes that disjunctions take the exactly-one relationship described in section 1. For each disjunct $i \in D_{k^*}$, the iteration counter advances by one, the set of unbranched disjuncts is updated, and the indicator variable Y_{ik^*} is set to $True$. For all other disjuncts in the disjunction, $i' \in D_{k^*} \setminus \{i\}$, the indicator variable $Y_{i'k^*}$ is set to $False$. The solution to (n-GDP) with the fixed logical realizations at this node obj^l is registered and added to the queue Q. The algorithm then repeats with selection of a new minimum objective value node from the queue.

Note that the structure of (n-GDP) ensures that only nonlinear constraints that apply globally and those corresponding to the $True$ disjunct of branched disjunctions exist in the problem. For unbranched disjunctions, only the linear

constraints and logical propositions are considered. Therefore, the nonlinear subproblems in GDPopt-LBB are also solved in reduced space, as nonlinear constraints for the branched $False$ disjuncts do not appear. Note that while modern MINLP solvers like BARON also attempt to solve subproblems in reduced space through sophisticated means of detecting logical structure, explicit specification of this structure in Pyomo.GDP enables logic-based solvers like GDPopt-LBB to do so more easily and reliably. The use of reduced space subproblems in GDPopt-LBB simplifies the BM/HR reformulation of (n-GDP) to MINLP and allows the algorithm to leverage advances in modern MINLP solvers. Note that spatial branch and bound is handled by the MINLP solver.

During the branching process, many infeasible nodes may be encountered, e.g. ones that violate logical propositions in (GDP). To more quickly prune out these nodes, we have developed a new interface between Pyomo.GDP models and satisfiability modulo theory solvers, such as Z3 (de Moura and Bjørner 2008). The expressions and variables of the Pyomo.GDP model are translated into a form consistent with the SMT-LIB standard, described in (Barrett et al. 2017). Via the python bindings for the Z3 solver, a check for satisfiability is performed. If the node is proved unsatisfiable, then it is given an objective value of $obj^l = \infty$. Usage of this interface involves invocation of the `satisfiable()` function, which takes as an argument the Pyomo.GDP model and returns `True` if the model is feasible, `False` if the model is proven to be infeasible, and `None` if satisfiability could not be determined.

Note that many enhancements are possible for GDPopt-LBB, including implementation of a more sophisticated branching strategy. In the worst case, GDPopt-LBB can give an exhaustive enumeration of the logical decision space. However, even in its present form, GDPopt-LBB provides a useful decomposition strategy. At a high-level, its approach is to iteratively consider the alternatives of each major discrete decision (disjunction) in the model and provides another flexible, open-source solution alternative for addressing difficult GDP models.

7 Case studies

To validate the proposed modeling ecosystem, we present its application to a set of 39 discrete-continuous optimization problems. A summary of these case studies and their characteristics are provided in Table 2. Where permitted, models have been released as part of the new open-source GDPlib model library. The instances are separated into four main sections. The first section includes many smaller-scale examples, including five which do not contain nonlinear constraints. Many of these GDP models are sourced from the Pyomo.GDP examples directory, freely available online. The heat exchanger network synthesis (`HENS_*`), modular processing network (`Mod_*`), and `Biofuel` models are described by Chen and Grossmann (2019c). These are larger instances, with moderate nonlinear complexity. Chen and Grossmann (2019b) describe the stranded gas processing (`Gas_*`) model variants, which feature a

multi-period network planning problem. These models have high combinatorial complexity, but only one or two nonlinear constraints. The final section includes three larger process synthesis problems, with greater nonlinear complexity, including multilinear terms and power functions. Rawlings et al. (2019) describe the `Kaibel` column design model, Türkay and Grossmann (1996) details the methanol synthesis (`MeOH`) model, and the membrane cascade design model (`Memb`) is a proprietary IDAES (Institute for the Design of Advanced Energy Systems) model.

The solution strategies tested against the models include those presented in the paper, along with the DICOPT 2.0 (Bernal et al. 2019), BARON (Tawarmalani and Sahinidis 2005), and SCIP (Vigerske and Gleixner 2018) MINLP solvers via the Pyomo-GAMS interface using GAMS 31.1.1. DICOPT is a local MINLP solver based on AP/ER/OA, and BARON and SCIP are global MINLP solvers based on convexification and spatial branch-and-bound (Kronqvist et al. 2019). A summary of the different strategies and their symbols are presented in Table 3. For subsolvers, DICOPT, BARON, and SCIP were allowed to use any supported GAMS-linked codes. For GDPopt (LOA and GLOA), CPLEX is used as the MILP solver. GDPopt-LOA uses IPOPT (Wächter and Biegler 2006) and CONOPT (Drud 1994) as NLP subproblem solvers, and DICOPT for MINLP subproblems. GDPopt-GLOA uses both BARON and SCIP as nonlinear subproblem solvers (NLP and MINLP). GDPopt-LBB uses BARON as its subsolver.

7.1 Motivating example

To illustrate the effect of zero-flow singularities introduced earlier, we first describe the nine-process problem, a new addition to the model library above, inspired by the eight-process problem from literature (Duran and Grossmann 1986). The superstructure for the problem is shown in Figure 8, and the model equations are given as Problem (9PP) in Appendix B. In this example, one process from each stage is to be selected, with the objective to maximize profit (here, expressed as minimization of negative cost). The product value per unit quantity is ten times that of the raw material, with a maximum flow rate of 50 units in any one stream.

The nine process problem has 63 constraints (8 nonlinear), 3 disjunctions, and 34 variables, of which 9 are Boolean and 25 are continuous. Attempting to apply the traditional reformulations to MINLP via BM or HR is not possible due to zero-flow issues, such as the asymptotic behavior of the natural logarithm function. Selecting an arbitrary big-M value is also not effective. DICOPT, SCIP, and BARON all incorrectly report an infeasible model or converge to the wrong value after transformation using various big-M values. By avoiding the zero-flow issues, GDPopt-LOA (using CPLEX and BARON as subsolvers) successfully converges after 6 master iterations and less than two seconds with the optimal solution of 36.62, pictured in Figure 9. Solving Problem (9PP) using GDPopt-GLOA reveals this to be the globally optimal

Table 2: GDP model library problems. Headings: vars = variables, Bool = Boolean variables, bin = binary variables, int = integer variables, cont = continuous variables, cons = constraints, nl = nonlinear constraints, disj = disjuncts, disjtn = disjunctions.

Name	vars	Bool	bin	int	cont	cons	nl	disj	disjtn
8PP	44	12	0	0	32	52	5	12	5
9PP	34	9	0	0	25	63	8	9	3
9PPnex	43	18	0	0	25	41	8	18	9
CLAY	30	18	0	0	12	48	24	18	6
BS	8	6	0	0	2	6	6	6	3
LeeEx1	6	3	0	0	3	6	3	3	1
Ex633	6	4	0	0	2	12	12	4	2
HENS_ncvx	17	9	0	0	8	31	12	9	3
strip8	129	112	0	0	17	120	0	112	28
strip4	33	24	0	0	9	28	0	24	6
rxn2	5	2	0	0	3	9	1	2	1
batchp	287	138	0	0	149	601	1	18	9
disease	1250	52	0	0	1198	831	0	52	26
jobshop	10	6	0	0	4	9	0	6	3
purchasing	498	216	0	0	282	762	0	216	72
stickies	418	176	0	0	242	685	18	176	91
Spectralog	128	60	0	0	68	158	8	60	30
Positioning	56	50	0	0	6	30	25	50	25
HENS_conv	214	24	0	0	190	250	36	24	32
HENS_int_sing	313	27	0	96	190	265	24	27	45
HENS_int_mult	262	36	0	36	190	262	24	36	44
HENS_int_opt	274	48	0	36	190	322	36	48	44
HENS_disc_sing	3077	27	2080	0	970	6006	12	27	33
HENS_disc_mult	1114	24	300	0	790	1486	12	24	44
HENS_disc_opt	1138	48	300	0	790	2122	36	48	44
Mod_grow	488	2	0	363	123	486	1	2	1
Mod_dip	488	2	0	363	123	486	1	2	1
Mod_decay	488	2	0	363	123	486	1	2	1
Mod_dist	2224	26	0	718	1480	1387	22	26	13
Mod_qtr	1314	42	0	486	786	672	36	42	21
Biofuel	36804	516	0	4320	31968	12884	12	516	252
Gas_100	8816	158	0	2520	6138	14925	1	158	96
Gas_250	8816	158	0	2520	6138	14925	1	158	96
Gas_500	8816	158	0	2520	6138	14925	1	158	96
Gas_small	11698	160	0	5040	6498	14927	2	160	96
Gas_large	11698	160	0	5040	6498	14927	2	160	96
Kaibel	3605	178	0	0	3427	5715	2124	178	100
MeOH	285	8	0	0	277	429	55	8	4
Memb	1339	6	0	0	1333	1944	779	6	3

solution, after 10 iterations in four seconds. Similarly, GDPopt-LBB converges after 34 iterations in three seconds. Problem (9PP) therefore demonstrates how logic-based decomposition solvers such as GDPopt can offer a more robust solution approach by exploiting disjunctive structure in the problem.

Table 3: Tested solution strategies

Symbol	Subsolvers	Description
DICOPT-BM	GAMS Defaults	BM + local MINLP solver
DICOPT-HR	GAMS Defaults	HR + local MINLP solver
BARON-BM	GAMS Defaults	BM + global MINLP solver
BARON-HR	GAMS Defaults	HR + global MINLP solver
SCIP-BM	GAMS Defaults	BM + global MINLP solver
SCIP-HR	GAMS Defaults	HR + global MINLP solver
GDPopt-LOA	CPLEX & IPOPT	Local GDP solver
GDPopt-GLOA	CPLEX & BARON	Global GDP solver
GDPopt-GLOA-DVB	CPLEX & BARON	GLOA with disjunctive variable bounding
GDPopt-LBB	BARON	Global GDP solver

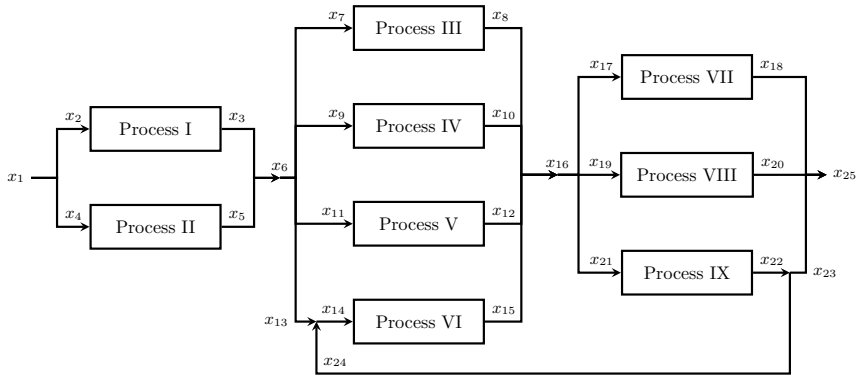


Fig. 8: Nine process problem superstructure

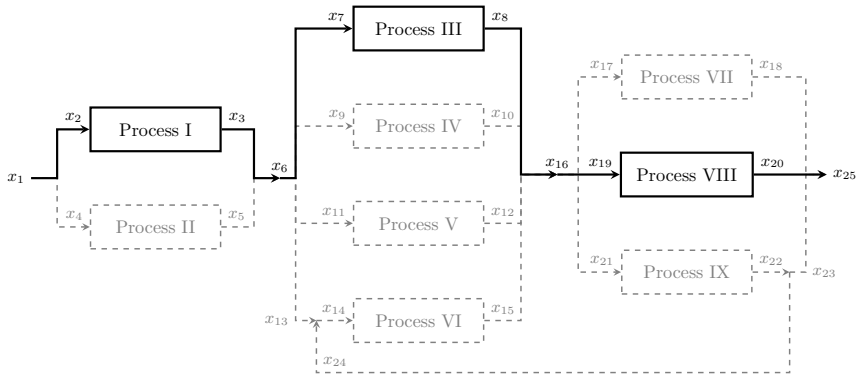


Fig. 9: Nine process problem globally optimal solution. Objective value: -36.62. Optimal flow values: $x_1 = x_2 = 3.85$, $x_3 = x_6 = x_7 = 1.58$, $x_8 = x_{16} = x_{19} = 3.91$, and $x_{20} = x_{25} = 4.05$.

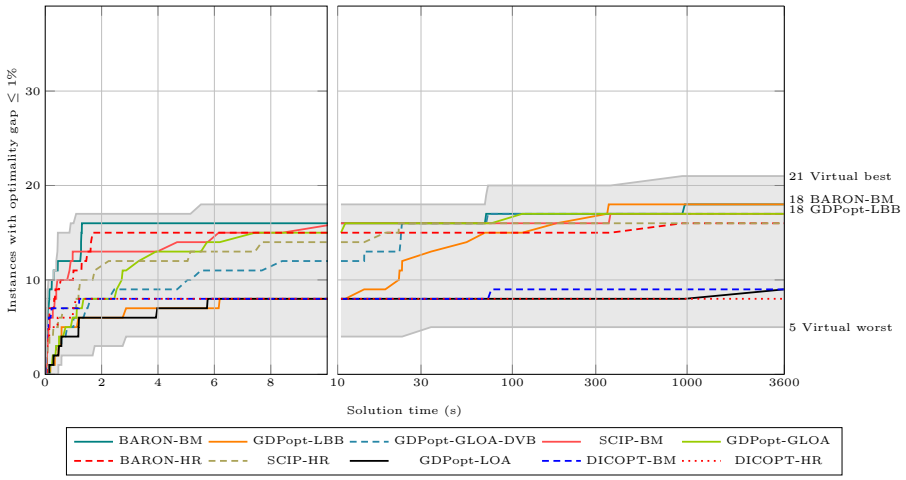


Fig. 10: Time required to converge to global optimality ($\leq 1\%$ gap)

7.2 Results

Each model-solver combination was executed on a machine running Ubuntu 16.04 LTE with 12 cores across two sockets (Intel Xeon X5650 @ 2.66 GHz) with 128GB physical memory, using a batch execution scheduler allocating 1 process and 16GB memory to each run. We evaluate three different measures of solution quality: time to global optimality, time to best known solution, and time to acceptable solution. For a solution strategy is regarded to achieve global optimality if it is able to converge to the known global optimal solution with at most 1% optimality gap. To achieve the "best known" solution, a strategy must report a feasible solution within 1% of the best known solution value. To achieve an "acceptable" solution, a feasible solution must be found within 10% of the best known solution value. The time limit for each run is set at 1 hour (3600 seconds); however, depending upon the implementation, some solvers may slightly exceed the limit while performing execution cleanup steps. The effect of this on the results is expected to be limited, as in the case of global optimality, most solvers plateaued before 1000 seconds, well before the time limit. In the other two cases, a quality feasible solution is often obtained long before bound convergence can be established. Figures 10, 11, and 12 illustrate the number of instances over time for which the optimal solution, the best known solution, and an acceptable solution were obtained for each solver, respectively. Figure 13 summarizes the total number of instances solved within the time limit. The lines denoted by "Virtual best" indicate the hypothetical performance of a composite solver that represents the best solution time for each model instance across all tested solvers.

Of the 39 model instances, 21 were solved to ϵ-tolerance of global optimality by at least one solver, 32 were solved to the best known solution, and

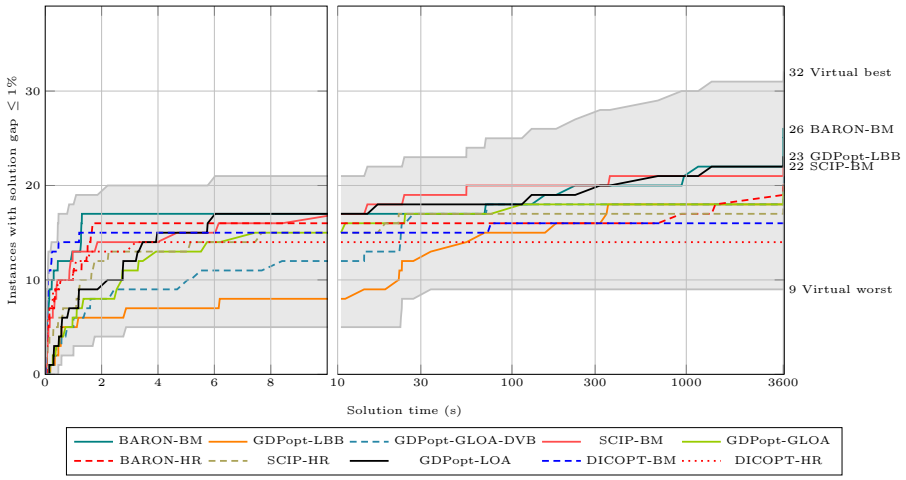


Fig. 11: Time required to converge to within 1% of the best known solution

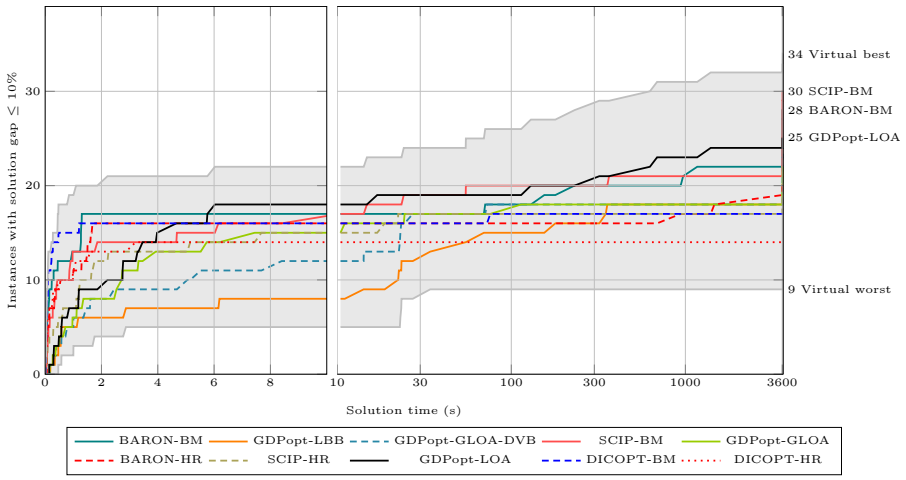


Fig. 12: Time required to converge to within 10% of the best known solution.

34 were solved to an acceptable solution. For global optimality, most tested solution strategies exhibited similar performance, able to solve around 19 instances. BARON-BM, SCIP-BM, GDPopt-GLOA, and GDPopt-GLOA-DVB all successfully converged for 19 model instances from the GDP library. However, they were able to do so for different problems. Of the library problems solved to global optimality, the logic-based decomposition strategies were collectively able to converge for all but `HENS_conv`, which only BARON-BM and BARON-HR were able to converge. On the other hand, only the logic-based decomposition algorithms were able to converge to the optimal solution for

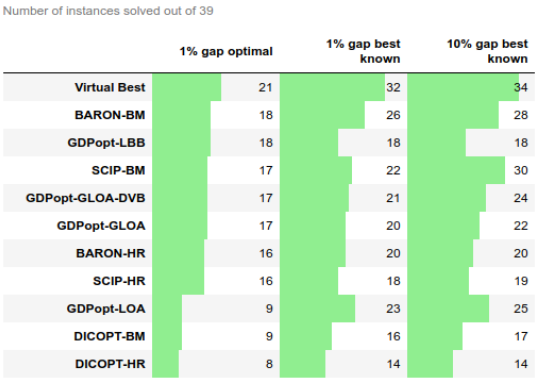


Fig. 13: Number of instances solved to each level of solution quality.

`9PP` and `9PPnex`. Note that GDPopt-LOA and DICOPT could solve at most the 10 convex GDP models to global optimality.

BARON-BM, GDPopt-LOA, and SCIP-BM were most successful at obtaining the best known feasible solutions, at 26, 23, and 22 instances, respectively. Here, a more pronounced gap exists between the performance of the best individual solver and the performance of the entire solution suite taken together.

For obtaining an "acceptable" solution within 10% of the best known objective value, the results closely mirror those of obtaining the best known objective value, with most solution strategies able to identify 2 or 3 more acceptable solutions. The outlier is SCIP-BM, which identifies "acceptable" solutions for 30 instances. Next is BARON-BM, with "acceptable" solutions found for 28 instances. In general, the global MINLP solvers did well on the `Gas_100`, `Gas_250`, and `Gas_500` problems, with only one nonlinear constraint. On the other hand, the direct logic-based decomposition algorithms outperformed MINLP solvers for `9PP`, `9PPnex`, and `Kaibel`, problems with greater nonlinear complexity.

For all three classes of solution quality, we observe that no individual solver is able to address all problems as effectively as the "Virtual best" composite solver. This validates the core principle that flexible solution strategies should be available for effective optimization frameworks.

8 Conclusions

In this work, we have presented a vision for integrated modeling and optimization toolsets based on the three core principles of intuitive high-level modeling syntax, systematic reformulations, and flexible solution strategies. We have described how Pyomo.GDP advances this vision by providing high-level modeling constructs to express problem logic, as well as a new logical expression system implementation. These GDP models then can be subject to a range of

automated, yet transparent transformations as part of a flexible array of solution approaches. We also demonstrate how Pyomo.GDP provides an ecosystem of modeling tools that facilitates the development of new tailored algorithms, such as GDPopt-LOA, GDPopt-GLOA, and GDPopt-LBB, that can even integrate with externally developed tools. Using a new library of GDP models, we demonstrate the importance of these new logic-based solver implementations towards a solution suite capable of handling a variety of challenging problems. The results also demonstrate the benefit of a flexible array of solution strategies available, as no one individual solver outperforms all others across a range of model instances. Note that no manual reformulation of the GDP models, once formulated, is necessary in order to take advantage of strategies in the presented Pyomo.GDP solution suite.

9 Acknowledgements

We graciously acknowledge funding from the U.S. Department of Energy, Office of Fossil Energy's Crosscutting Research Program through the Institute for the Design of Advanced Energy Systems (IDAES). We also recognize software development assistance from Zhifei Yuliu, supervised by Qi Chen, on the final logical expression system implementation.

Conflict of interest

The authors declare that they have no conflict of interest.

References

Balas E (1979) Disjunctive Programming. Annals of Discrete Mathematics 5:3–51, DOI 10.1016/S0167-5060(08)70342-X, URL http://linkinghub.elsevier.com/retrieve/pii/S016750600870342X

Balas E (1985) Disjunctive Programming and a Hierarchy of Relaxations for Discrete Optimization Problems. SIAM Journal on Algebraic Discrete Methods 6(3):466–486, DOI 10.1137/0606047, URL http://epubs.siam.org/doi/abs/10.1137/0606047

Balas E (2018) Disjunctive Programming. Springer International Publishing, Cham, DOI 10.1007/978-3-030-00148-3, URL http://link.springer.com/10.1007/978-3-030-00148-3

Barrett C, Fontaine P, Tinelli C (2017) The SMT-LIB Standard: Version 2.6. Tech. rep., Department of Computer Science, The University of Iowa

Barttfeld M, Aguirre PA, Grossmann IE (2003) Alternative representations and formulations for the economic optimization of multicomponent distillation columns. Computers & Chemical Engineering 27(3):363–383, DOI 10.1016/S0098-1354(02)00213-2, URL http://www.sciencedirect.com/science/article/pii/S0098135402002132

Beaumont N (1990) An algorithm for disjunctive programs. European Journal of Operational Research 48(3):362–371, DOI 10.1016/0377-2217(90)90419-C

Belotti P, Bonami P, Fischetti M, Lodi A, Monaci M, Nogales-Gómez A, Salvagnin D (2016) On handling indicator constraints in mixed integer programming. Computational Optimization and Applications 65(3):545–566, DOI 10.1007/s10589-016-9847-8

Bergamini ML, Aguirre P, Grossmann I (2005) Logic-based outer approximation for globally optimal synthesis of process networks. Computers & Chemical Engineering 29(9):1914–1933, DOI 10.1016/j.compchemeng.2005.04.003, URL http://www.sciencedirect.com/science/article/pii/S0098135405001006

Bernal DE, Chen Q, Gong F, Grossmann IE (2018) Mixed-Integer Nonlinear Decomposition Toolbox for Pyomo (MindtPy). Computer Aided Chemical Engineering 44(1):895–900, DOI 10.1016/B978-0-444-64241-7.50144-0

Bernal DE, Vigerske S, Trespalacios F, Grossmann IE (2019) Improving the performance of DICOPT in convex MINLP problems using a feasibility pump. Optimization Methods and Software 0(0):1–20, DOI 10.1080/10556788.2019.1641498, URL https://www.tandfonline.com/doi/full/10.1080/10556788.2019.1641498https://doi.org/10556788.2019.1641498

Bisschop J, Roelofs M (2006) AIMMS - User's Guide

Bonami P, Lodi A, Tramontani A, Wiese S (2015) On mathematical programming with indicator constraints. Mathematical Programming 151(1):191–223, DOI 10.1007/s10107-015-0891-4, URL http://dx.doi.org/10.1007/s10107-015-0891-4http://link.springer.com/10.1007/s10107-015-0891-4

Boukouvala F, Misener R, Floudas CA (2016) Global optimization advances in Mixed-Integer Nonlinear Programming, MINLP, and Constrained Derivative-Free Optimization, CDFO. European Journal of Operational Research 252(3):701–727, DOI 10.1016/j.ejor.2015.12.018, URL http://dx.doi.org/10.1016/j.ejor.2015.12.018

Brook A, Kendrick D, Meeraus A (1988) GAMS, a user's guide. ACM SIGNUM Newsletter 23(3-4):10–11, DOI 10.1145/58859.58863

Chachuat B (2019) MC++ (version 2.0): Toolkit for Construction, Manipulation and Bounding of Factorable Functions. URL https://omega-icl.github.io/mcpp/

Chen Q, Grossmann I (2017) Recent Developments and Challenges in Optimization-Based Process Synthesis. Annual Review of Chemical and Biomolecular Engineering 8(1):249–283, DOI 10.1146/annurev-chembioeng-080615-033546, URL http://www.annualreviews.org/doi/10.1146/annurev-chembioeng-080615-033546

Chen Q, Grossmann I (2019a) Modern Modeling Paradigms Using Generalized Disjunctive Programming. Processes 7(11):839, DOI 10.3390/pr7110839, URL https://www.mdpi.com/2227-9717/7/11/839

Chen Q, Grossmann IE (2019b) Economies of Numbers for A Modular Stranded Gas Processing Network: Modeling And Optimization. In: Proceedings of the 9th International Conference on Foundations of Computer-Aided Process Design, pp 257–262, DOI 10.1016/B978-0-12-818597-1.50041-2, URL https://linkinghub.elsevier.com/retrieve/pii/B9780128185971500412

Chen Q, Grossmann IE (2019c) Effective Generalized Disjunctive Programming Models for Modular Process Synthesis. Industrial & Engineering Chemistry Research 58(15):5873–5886, DOI 10.1021/acs.iecr.8b04600, URL http://pubs.acs.org/doi/10.1021/acs.iecr.8b04600

Chen Q, Johnson ES, Siirola JD, Grossmann IE (2018) Pyomo.GDP: Disjunctive Models in Python. In: Eden MR, Ierapetritou MG, Towler GP (eds) Proceedings of the 13th International Symposium on Process Systems Engineering, Elsevier B.V., San Diego, pp 889–894, DOI 10.1016/B978-0-444-64241-7.50143-9, URL https://linkinghub.elsevier.com/retrieve/pii/B9780444642417501439

Drud AS (1994) CONOPTA Large-Scale GRG Code. ORSA Journal on Computing 6(2):207–216, DOI 10.1287/ijoc.6.2.207, URL http://pubsonline.informs.org/doi/abs/10.1287/ijoc.6.2.207

Dunning I, Huchette J, Lubin M (2017) JuMP: A Modeling Language for Mathematical Optimization. SIAM Review 59(2):295–320, DOI 10.1137/15M1020575, URL http://arxiv.org/abs/1508.01982{%}0Ahttp://dx.doi.org/10.1137/15M1020575https://epubs.siam.org/doi/10.1137/15M1020575, 1508.01982

Duran MA, Grossmann IE (1986) An outer-approximation algorithm for a class of mixed-integer nonlinear programs. Mathematical Programming 36(3):307, DOI 10.1007/BF02592064

Floudas CA, Lin X (2004) Continuous-time versus discrete-time approaches for scheduling of chemical processes: A review. Computers and Chemical Engineering 28(11):2109–2129,

DOI 10.1016/j.compchemeng.2004.05.002
Fourer R, Gay DM, Kernighan BW (2003) AMPL: A Modeling Language for Mathematical Programming, 2nd edn
Friedman Z, Ingalls J, Siirola JD, Watson JP (2013) Block-oriented modeling of superstructure optimization problems. Computers and Chemical Engineering 57:10–23, DOI 10.1016/j.compchemeng.2013.04.008, URL http://dx.doi.org/10.1016/j.compchemeng.2013.04.008
GAMS Development Corp (2020) Disjunctive programming. URL https://www.gams.com/latest/docs/UG_EMP_DisjunctiveProgramming.html
Grossmann IE, Lee S (2003) Generalized Convex Disjunctive Programming: Nonlinear Convex Hull Relaxation. Computational Optimization and Applications 26(1):83–100, DOI 10.1023/A:1025154322278, URL http://link.springer.com/10.1023/A:1025154322278
Grossmann IE, Trespalacios F (2013) Systematic modeling of discrete-continuous optimization models through generalized disjunctive programming. AIChE Journal 59(9):3276–3295, DOI 10.1002/aic.14088, URL http://doi.wiley.com/10.1002/aic.14088
Gurobi Optimization Inc (2016) Gurobi Optimizer reference manual
Hart WE, Laird CD, Watson JP, Woodruff DL, Hackebeil GA, Nicholson BL, Siirola JD (2017) Pyomo Optimization Modeling in Python, Springer Optimization and Its Applications, vol 67, 2nd edn. Springer International Publishing, Cham, DOI 10.1007/978-3-319-58821-6, URL http://link.springer.com/10.1007/978-3-319-58821-6
Kocis GR, Grossmann IE (1989) Computational Experience With Dicopt Solving Minlp Problems in Process Systems-Engineering. Computers & Chemical Engineering 13(3):307–315, DOI 10.1016/0098-1354(89)85008-2
Kronqvist J, Bernal DE, Lundell A, Grossmann IE (2019) A review and comparison of solvers for convex MINLP, vol 20. Springer US, DOI 10.1007/s11081-018-9411-8, URL https://doi.org/10.1007/s11081-018-9411-8
Lee S, Grossmann IE (2000) New algorithms for nonlinear generalized disjunctive programming. Computers & Chemical Engineering 24(9-10):2125–2141, DOI 10.1016/S0098-1354(00)00581-0, URL http://www.sciencedirect.com/science/article/pii/S0098135400005810
Lee S, Grossmann IE (2001) A global optimization algorithm for nonconvex generalized disjunctive programming and applications to process systems. Computers & Chemical Engineering 25(11-12):1675–1697, DOI 10.1016/S0098-1354(01)00732-3, URL http://linkinghub.elsevier.com/retrieve/pii/S0098135401007323
de Moura L, Bjørner N (2008) Z3: An Efficient SMT Solver. pp 337–340, DOI 10.1007/978-3-540-78800-3_24, URL http://link.springer.com/10.1007/978-3-540-78800-3{_}24
Nemhauser GL, Wolsey LA (1988) Integer and combinatorial optimization. Wiley, New York
Puranik Y, Sahinidis NV (2017) Domain reduction techniques for global NLP and MINLP optimization. Constraints 22(3):338–376, DOI 10.1007/s10601-016-9267-5, 1706.08601
Raman R, Grossmann I (1991) Relation between MILP modelling and logical inference for chemical process synthesis. Computers & Chemical Engineering 15(2):73–84, DOI 10.1016/0098-1354(91)87007-V, URL https://linkinghub.elsevier.com/retrieve/pii/009813549187007V
Raman R, Grossmann I (1994) Modelling and computational techniques for logic based integer programming. Computers & Chemical Engineering 18(7):563–578, DOI 10.1016/0098-1354(93)E0010-7
Rawlings ES, Chen Q, Grossmann IE, Caballero JA (2019) Kaibel column: Modeling, optimization, and conceptual design of multi-product dividing wall columns. Computers & Chemical Engineering 125:31–39, DOI 10.1016/j.compchemeng.2019.03.006, URL https://linkinghub.elsevier.com/retrieve/pii/S0098135419300250
Ruiz JP, Grossmann IE (2012) A hierarchy of relaxations for nonlinear convex generalized disjunctive programming. European Journal of Operational Research 218(1):38–47, DOI 10.1016/j.ejor.2011.10.002
Ruiz JP, Grossmann IE (2017) Global optimization of non-convex generalized disjunctive programs: a review on reformulations and relaxation techniques. Journal of Global Optimization 67(1-2):43–58, DOI 10.1007/s10898-016-0401-0

Sahinidis NV (2019) Mixed-integer nonlinear programming 2018. Optimization and Engineering (0123456789), DOI 10.1007/s11081-019-09438-1, URL http://link.springer.com/10.1007/s11081-019-09438-1

Tawarmalani M, Sahinidis NV (2005) A polyhedral branch-and-cut approach to global optimization. Mathematical Programming 103:225–249, DOI 10.1007/s10107-005-0581-8

Trespalacios F, Grossmann IE (2014) Review of Mixed-Integer Nonlinear and Generalized Disjunctive Programming Methods. Chemie Ingenieur Technik 86(7):991–1012, DOI 10.1002/cite.201400037, URL http://doi.wiley.com/10.1002/cite.201400037

Trespalacios F, Grossmann IE (2016) Cutting Plane Algorithm for Convex Generalized Disjunctive Programs. INFORMS Journal on Computing 28(2):209–222, DOI 10.1287/ijoc.2015.0669, URL http://pubsonline.informs.org/doi/10.1287/ijoc.2015.0669

Tsoukalas A, Mitsos A (2014) Multivariate McCormick relaxations. Journal of Global Optimization 59(2-3):633–662, DOI 10.1007/s10898-014-0176-0, URL http://link.springer.com/10.1007/s10898-014-0176-0

Türkay M, Grossmann IE (1996) Logic-based MINLP algorithms for the optimal synthesis of process networks. Computers & Chemical Engineering 20(8):959–978, DOI 10.1016/0098-1354(95)00219-7, URL http://www.sciencedirect.com/science/article/pii/0098135495002197{%}5Cnhttp://linkinghub.elsevier.com/retrieve/pii/0098135495002197

Vecchietti A, Grossmann IE (1997) LOGMIP: a disjunctive 0-1 nonlinear optimizer for process systems models. In: PSE '97-ESCAPE 7, Pergamon Press Ltd, vol 21, pp S427–S432

Vigerske S, Gleixner A (2018) Scip: global optimization of mixed-integer nonlinear programs in a branch-and-cut framework. Optimization Methods and Software 33(3):563–593, DOI 10.1080/10556788.2017.1335312

Viswanathan J, Grossmann IE (1990) A combined penalty function and outer-approximation method for MINLP optimization. Computers and Chemical Engineering 14(7):769–782, DOI 10.1016/0098-1354(90)87085-4

Wächter A, Biegler LT (2006) On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Mathematical Programming 106(1):25–57, DOI 10.1007/s10107-004-0559-y, URL http://link.springer.com/10.1007/s10107-004-0559-y

A Pyomo.GDP Modeling Syntax

Modeling syntax for Pyomo.GDP uses `Disjunct` and `Disjunction` modeling objects. `Disjunct` objects represent a grouping of constraints in a logical context, and `Disjunction` objects describe the OR relationship between these logical groupings. Note that by default, Pyomo.GDP `Disjunction` objects enforce an "exactly one" relationship among the selection of the disjuncts (generalization of exclusive-OR). That is, exactly one of the `Disjunct` indicator variables should take a `True` value.

The example below illustrates the Pyomo.GDP syntax available to express the disjunction (2) between the selection of two units in a process network. In the example, Y_1 and Y_2 are the Boolean variables corresponding to the selection of process units 1 and 2, respectively. The continuous variables x_1, x_2, x_3, x_4 describe flow in and out of the first and second units, respectively. If a unit is selected, the nonlinear equality in the corresponding disjunct enforces the input/output relationship in the selected unit. The final equality in each disjunct forces flows for the absent unit to zero.

$$\begin{bmatrix} Y_1 \\ \exp(x_2) - 1 = x_1 \\ x_3 = x_4 = 0 \end{bmatrix} \bigvee \begin{bmatrix} Y_2 \\ \exp\left(\frac{x_4}{1.2}\right) - 1 = x_3 \\ x_1 = x_2 = 0 \end{bmatrix} \tag{2}$$

Expanded syntax: more descriptive Pyomo.GDP expanded syntax provides more clarity in the declaration of each modeling object (see Listing 13). Assuming the `ConcreteModel` object and variables have been defined, lines 1 and 5 declare the `Disjunct` objects corresponding to selection of unit 1 and 2, respectively. Lines 2 and 6 define the input-output relations for each unit, and lines 3-4 and 7-8 enforce zero flow through the unit that is not selected. Finally, line 9 declares the logical disjunction between the two disjunctive terms.

Listing 13: Expanded syntax for programmatic specification of disjunctions.

```
1 m.unit1 = Disjunct()
2 m.unit1.inout = Constraint(expr=exp(m.x[2]) - 1 == m.x[1])
3 m.unit1.no_unit2_flow1 = Constraint(expr=m.x[3] == 0)
4 m.unit1.no_unit2_flow2 = Constraint(expr=m.x[4] == 0)
5 m.unit2 = Disjunct()
6 m.unit2.inout = Constraint(expr=exp(m.x[4] / 1.2) - 1 == m.x[3])
7 m.unit2.no_unit1_flow1 = Constraint(expr=m.x[1] == 0)
8 m.unit2.no_unit1_flow2 = Constraint(expr=m.x[2] == 0)
9 m.use_unit1or2 = Disjunction(expr=[m.unit1, m.unit2])
```

The indicator variables for each disjunct Y_1 and Y_2 are automatically generated by Pyomo.GDP, accessible via `m.unit1.indicator_var` and `m.unit2.indicator_var`.

Compact syntax: more concise For advanced users, a more compact syntax is also available (see Listing 14), taking advantage of the ability to declare disjuncts and constraints implicitly. When the `Disjunction` object constructor is passed a list of lists, the outer list defines the disjuncts and the inner list defines the constraint expressions associated with the respective disjunct.

Listing 14: Compact syntax for programmatic specification of disjunctions.

```
m.use1or2 = Disjunction(expr=[
    # First disjunct
    [exp(m.x[3])-1 == m.x[2],
     m.x[4] == 0, m.x[5] == 0],
    # Second disjunct
    [exp(m.x[5]/1.2)-1 == m.x[4],
     m.x[2] == 0, m.x[3] == 0]])
```

A.1 Nested disjunctions

Pyomo.GDP also supports modeling with nested disjunctions, as given in Equation (3). The inner disjunctions $q \in \mathcal{K}_{ik}$ describe a logical-OR relationship between disjuncts $p \in \mathcal{D}_q$. Selection of the inner disjunct indicated by the Boolean variable Y_{pq} enforces the corresponding linear constraints $M_{pq}x + N_{pq}z \leq e_{pq}$ and nonlinear constraints $r_{pq}(x, z) \leq 0$. Nested disjunction logic often describes discrete choices that are contingent upon another decision. For example, in chemical process synthesis, a choice between reactor models A, B, or C might be contingent upon existence of the reactor. The selection of a reactor model would therefore be a nested disjunction within the parent disjunction between existence or absence of the reactor.

$$\bigvee_{i \in D_k} \begin{bmatrix} Y_{ik} \\ M_{ik}x + N_{ik}z \leq e_{ik} \\ r_{ik}(x,z) \leq 0 \\ \bigvee\limits_{p \in \mathcal{D}_q} \begin{bmatrix} Y_{pq} \\ M_{pq}x + N_{pq}z \leq e_{pq} \\ r_{pq}(x,z) \leq 0 \end{bmatrix} \quad q \in \mathcal{K}_{ik} \end{bmatrix} \quad k \in K \tag{3}$$

Assuming that the correct sets are defined, Listing 15 gives the Pyomo.GDP code necessary to represent a nested disjunction as described by Equation (3).

Listing 15: Declaring nested disjunctions in Pyomo.GDP.

```
# Define outer disjuncts
@m.Disjunct(m.IK_matches)
def outer_disjunct(disj_ik, i, k):
    # Define constraints for outer disjunct Y_ik here
    disj_ik.cons = Constraint() # user-defined

    # Define the inner disjuncts
    @disj_ik.Disjunct(m.PQ_matches[i, k])
    def inner_disjunct(disj_pq, p, q):
        # Define constraints for inner disjunct Y_pq here
        disj_pq.cons = Constraint() # user-defined

    # Declare disjunctions between inner disjuncts
    @disj_ik.Disjunction(m.Kik)
    def inner_disjunction(disj_ik, q):
        return [disj_pq for disj_pq in disj_ik.inner_disjunct[:, q]]

# Declare disjunctions between outer disjuncts
@m.Disjunction(m.K)
def outer_disjunction(m, k):
    return [disj_ik for disj_ik in m.outer_disjunct[:, k]]
```

B 9PP GDP Model Formulation

$$\min Z = -10x_{25} + x_1 \tag{9PP.1}$$

$$s.t. \quad x_1 = x_2 + x_3 \tag{9PP.2}$$

$$\begin{bmatrix} Y_1 \\ x_2 = \exp(x_3 - 1) \\ x_4 = x_5 = 0 \end{bmatrix} \vee \begin{bmatrix} Y_2 \\ x_5 = \ln(x_4 + 1) \\ x_2 = x_3 = 0 \end{bmatrix} \tag{9PP.3}$$

$$x_3 + x_5 = x_6 \tag{9PP.4}$$

$$x_6 = x_7 + x_9 + x_{11} + x_{13} \tag{9PP.5}$$

$$\begin{bmatrix} Y_3 \\ x_8 = 2\ln(x_7) + 3 \\ x_7 \geq 0.2 \\ x_9 = x_{10} = x_{11} = 0 \\ x_{12} = x_{14} = x_{15} = 0 \end{bmatrix} \vee \begin{bmatrix} Y_4 \\ x_{10} = 1.8\ln(x_9 + 4) \\ x_7 = x_8 = x_{11} = 0 \\ x_{12} = x_{14} = x_{15} = 0 \end{bmatrix} \vee \begin{bmatrix} Y_5 \\ x_{12} = 1.2\ln(x_{11}) + 2 \\ x_{11} \geq 0.001 \\ x_7 = x_8 = x_9 = 0 \\ x_{10} = x_{14} = x_{15} = 0 \end{bmatrix} \vee \begin{bmatrix} Y_6 \\ x_{15} = x_{23}\sqrt{x_{14} - 3} \\ 5 \leq x_{14} \leq 20 \\ x_7 = x_8 = x_9 = 0 \\ x_{10} = x_{11} = x_{12} = 0 \end{bmatrix} \tag{9PP.6}$$

$$x_{14} = x_{13} + x_{23} \tag{9PP.7}$$

$$x_8 + x_{10} + x_{12} + x_{15} = x_{16} \tag{9PP.8}$$

$$\begin{bmatrix} Y_7 \\ x_{18} = 0.9x_{17} \\ x_{19} = x_{20} = 0 \\ x_{21} = x_{22} = 0 \end{bmatrix} \vee \begin{bmatrix} Y_8 \\ x_{20} = \ln(x_{19}^{1.5}) + 2 \\ x_{19} \geq 1 \\ x_{17} = x_{18} = 0 \\ x_{21} = x_{22} = 0 \end{bmatrix} \vee \begin{bmatrix} Y_9 \\ x_{22} = \ln(x_{21} + \sqrt{x_{21}}) + 1 \\ x_{21} \geq 4 \\ x_{17} = x_{18} = 0 \\ x_{19} = x_{20} = 0 \end{bmatrix} \tag{9PP.9}$$

$$x_{22} = x_{23} + x_{24} \tag{9PP.10}$$

$$x_{18} + x_{20} + x_{24} = x_{25} \tag{9PP.11}$$

$$0 \leq x_i \leq 50 \quad i \in \{1, 2, \ldots, 25\} \tag{9PP.12}$$

$$x \in \mathbb{R}^{25} \tag{9PP.13}$$

$$Y_i \in \{True, False\} \quad i \in \{1, 2, \ldots, 9\} \tag{9PP.14}$$